CS 455/555
Intro to Networks and Communications

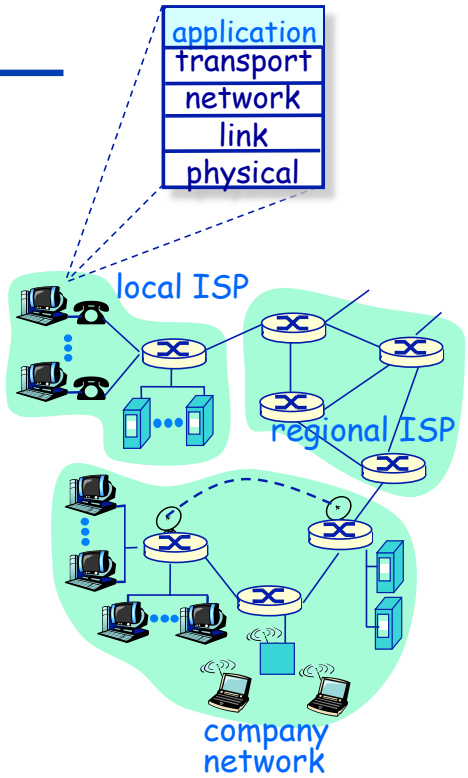
Applications & Application-Layer Protocols: The Web & HTTP

Dr. Michele Weigle
Department of Computer Science
Old Dominion University
mweigle@cs.odu.edu

http://www.cs.odu.edu/~mweigle/CS455-S13

Application-Layer Protocols
Outline

- The architecture of distributed systems
  - » Client/Server computing
  - » P2P computing
  - » Hybrid (Client/Server and P2P) systems
- The programming model used in constructing distributed systems
  - » Socket programming
- Example client/server systems and their application-layer protocols
  - » The World-Wide Web (HTTP)
  - » Reliable file transfer (FTP)
  - » E-mail (SMTP & POP)
  - » Internet Domain Name System (DNS)



The Web & HTTP

Outline

- Terminology (KR 2.2.1)
- HTTP protocol
  - » message format (KR 2.23)
  - » non-persistent and persistent connections (KR 2.2.2)
  - » pipelining
- Authentication
- Cookies (KR 2.2.4)
- Web caches (KR 2.2.5-2.2.6)
- Security (KR 8.1-8.3)

Application-Layer Protocols

The Web

- User agent (client) for the Web is called a *browser*
  - » Google Chrome
  - » Mozilla Firefox
  - » Apple Safari
  - » MS Internet Explorer
- Server for the Web is called a *Web server*
  - » Apache (public domain)
  - » MS Internet Information Server (IIS)

Application-Layer Protocols

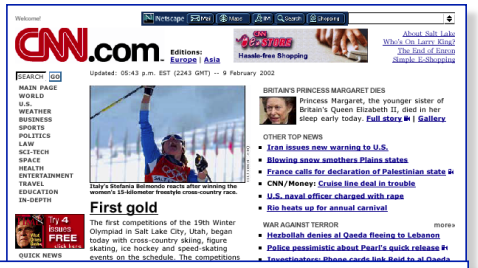
Web terminology

- Web page:
  - » Addressed by a **URL**
  - » Consists of "objects"
- Most Web pages consist of:
  - » Base HTML page
  - » Embedded objects

Application-Layer Protocols

Web terminology

- Web page:
  - » Addressed by a **URL**
  - » Consists of "objects"
- Most Web pages consist of:

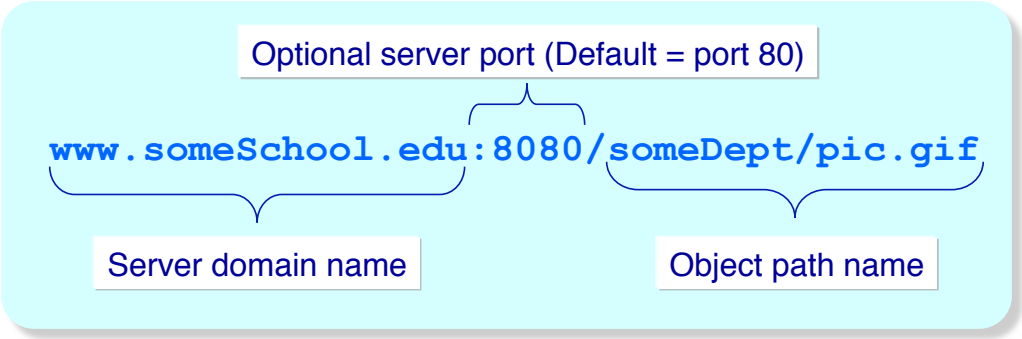



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html lang="en">
<head>
   <meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
   <title>CNN.com</title>
   <meta http-equiv="refresh" content="1800; URL=http://www.cnn.com/?">
   <link rel="StyleSheet" href="http://i.cnn.net/cnn/virtual/2001/style/main.css" type="text/css">
   <script language="JavaScript1.1" src="http://i.cnn.net/cnn/virtual/2000/code/main.js"
         type="text/javascript"> </script>
   <script language="JavaScript1.1" type="text/javascript"> </script>
<script language="JavaScript1.1" src="http://ar.atwola.com/file/adsWrapper.js"></script>
<style type="text/css"></style>
<script language="JavaScript">document.adoffset=0</script>
</head>

<body class="cnnMainBody" bgcolor="#FFFFFF">

<a name="top_of_page"></a>
  :
  :
```

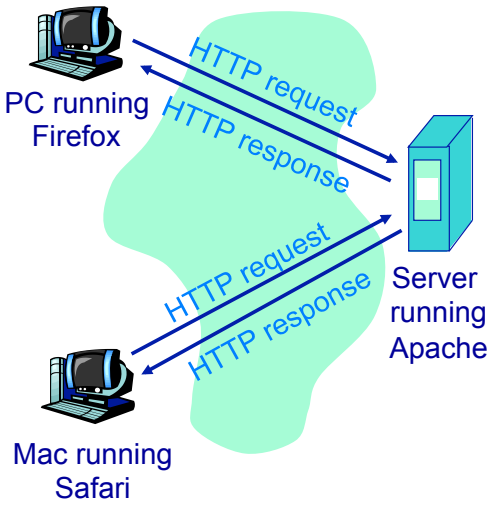
Web Terminology
URLs (Universal Resource Locators)



- URL components
  - » Server address
  - » (Optional port number)
  - » Path name

Web Terminology
The Hypertext Transfer Protocol (HTTP)

- Web's application layer protocol
- Client/server model
  - » *client:* browser that requests, receives, "displays" Web objects
  - » *server:* Web server sends objects in response to requests
- HTTP/1.0: RFC 1945
- HTTP/1.1: RFC 2616



The Hypertext Transfer Protocol

HTTP Overview

- HTTP uses TCP sockets
  - » Browser initiates TCP connection to server (on port 80)
- HTTP messages (application - layer protocol messages) exchanged between browser and Web server
- HTTP/1.0: RFC 1945
  - » One request/response interaction per connection
- HTTP/1.1: RFC 2616
  - » Persistent connections
  - » Pipelined connections
- HTTP is "stateless"
  - » Server maintains no information about past browser requests

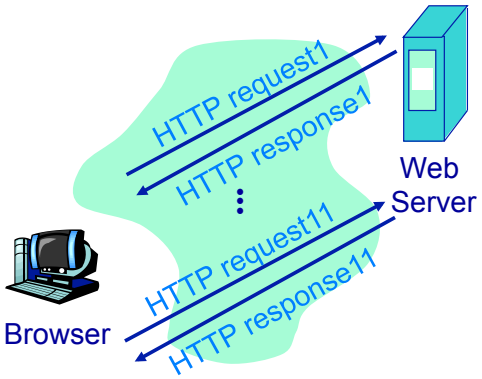
aside

- Protocols that maintain "state" are complex!
  - » Past history (state) must be maintained
  - » If server or client crashes, their views of "state" may be inconsistent and must be reconciled

The Hypertext Transfer Protocol

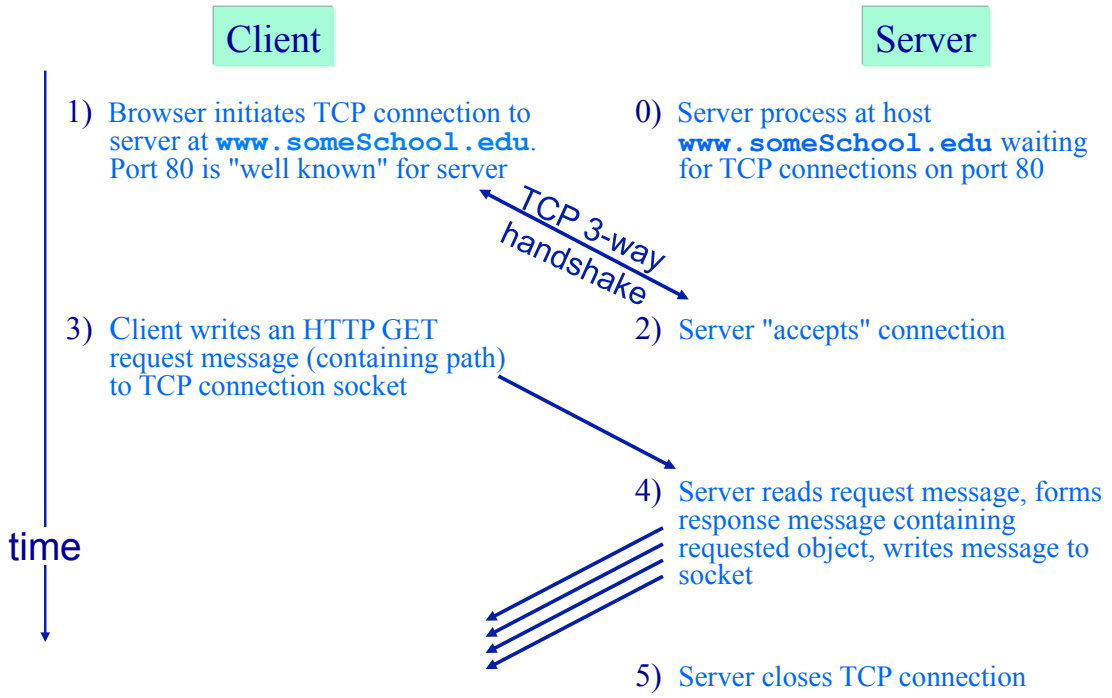
HTTP example

- User enters URL `www.someSchool.edu/someDept/home.index`
  - » Referenced object contains HTML text and references 10 JPEG images
- Browser sends an HTTP "GET" request to the server `www.someSchool.edu`
- Server will retrieve and send the HTML file
- Browser will read the file and sequentially make 10 separate requests for the embedded JPEG images



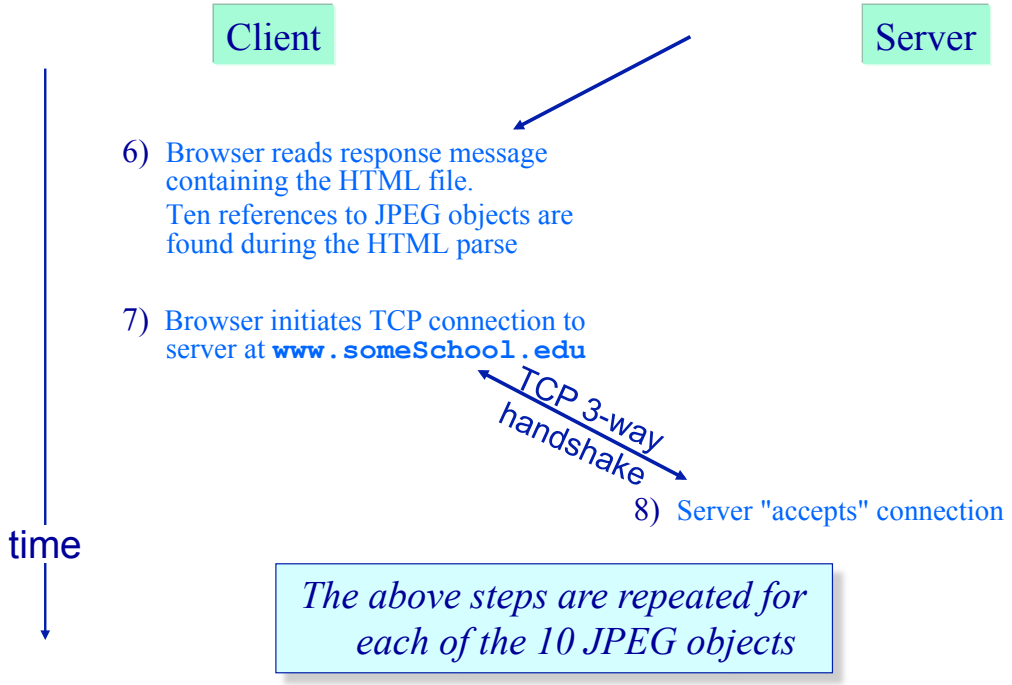
HTTP 1.0 Example

URL `www.someschool.edu/someDept/home.index`



HTTP 1.0 Example

URL `www.someschool.edu/someDept/home.index`



The Web & HTTP

Outline

- Terminology (KR 2.2.1)
- HTTP protocol
  - » message format (KR 2.23)
  - » non-persistent and persistent connections (KR 2.2.2)
  - » pipelining
- Authentication
- Cookies (KR 2.2.4)
- Web caches (KR 2.2.5-2.2.6)
- Security (KR 8.1-8.3)

The Hypertext Transfer Protocol

HTTP message format

- Two types of HTTP message formats: ***request*** and ***response***
  - » ASCII (human-readable format)
- HTTP request message:
  - » Request line
  - » Optional header lines
  - » Present only for some methods (*e.g.*, POST)

```
method <SP> path <SP> version <CR><LF>
header field name ":" value <CR><LF>
                ⋮
header field name ":" value <CR><LF>
<CR><LF>
entity body
```

HTTP Message Format

Chrome and Safari request examples

- How does Chrome process:

  http://www.cs.odu.edu:8080/~mweigle/ ?

```
GET /~mweigle/ HTTP/1.1
Host: www.cs.odu.edu:8080
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2)
AppleWebKit/537.17 (KHTML, like Gecko) Chrome/24.0.1312.56
Safari/537.17
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Cookie: SESSID=TkZCQjBROTI2NjY4; __utma=35744766....
```


HTTP Message Format

Chrome and Safari request examples

- How does Safari process:

  http://www.cs.odu.edu:8080/~mweigle/ ?

```
GET /~mweigle/ HTTP/1.1
Host: www.cs.odu.edu:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2)
AppleWebKit/536.26.17 (KHTML, like Gecko) Version/6.0.2
Safari/536.26.17
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: __utma=256360345.1846808915.1...
```

HTTP Message Format

General response message format

- Response messages
  - » ASCII (human-readable format)
- Message structure:
  - » Status line
  - » Optional header lines
  - » Requested object, error message message, etc.

```
version <SP> code <SP> phrase <CR><LF>
header field name ":" value <CR><LF>
        ⋮
header field name ":" value <CR><LF>
<CR><LF>
entity body
```


HTTP Message Format

Telnet example

Connect to HTTP server port →

Telnet output

Type GET command **plus blank line**

HTTP response status line →

HTTP response headers plus blank line

Object content

Telnet output →

```
% telnet www.cs.odu.edu 80
Trying 128.82.4.2...
Connected to xenon.cs.odu.edu.
Escape character is '^]'.
GET /~mweigle/files/foo.txt HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 30 Jan 2013 01:44:23 GMT
Server: Apache/2.2.17 (Unix) PHP/5.3.5 ...
Last-Modified: Thu, 19 May 2011 19:23:43 GMT
ETag: "5c-4a3a5f178cdd0"
Accept-Ranges: bytes
Content-Length: 92
Connection: close
Content-Type: text/plain

This test file is stored in the UNIX file
system at
/home/mweigle/public_html/files/foo.txt
Connection closed by foreign host.
```

HTTP Message Format

Telnet example (2)

Connect to HTTP server port →

Telnet output

Type GET command **plus blank line**

HTTP response status line →

HTTP response headers plus blank line

Telnet output →

```
% telnet www.msnbc.com 80
Trying 65.55.53.235...
Connected to www.msnbc.com.
Escape character is '^]'.
HEAD /notexist.html HTTP/1.0

HTTP/1.1 404 NotFound
Cache-Control: private
Content-Length: 52017
Content-Type: text/html; charset=utf-8
Server: Microsoft-IIS/7.5
X-AspNet-Version: 2.0.50727
X-Powered-By: ASP.NET
Date: Wed, 30 Jan 2013 02:00:20 GMT
Connection: close

Connection closed by foreign host.
```


HTTP Message Format

HTTP response status codes

- Sample response codes:

  `200 OK`
  - » Request succeeded, requested object later in this message

  `301 Moved Permanently`
  - » Requested object moved, new location specified later in this message (Location:)

  `400 Bad Request`
  - » Request message not understood by server

  `404 Not Found`
  - » Requested document not found on this server

  `505 HTTP Version Not Supported`

HTTP Message Format

Typical Request and Response Headers

Request headers

```
Connection: Keep-Alive
User-Agent: Mozilla/4.74 [en] (WinNT; U)
Host: www.cs.odu.edu:8080
Accept: image/gif, image/x-xbitmap, image/jpeg,
        image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: SITESERVER=ID=8a064b785a043146e4599174a3d970
```

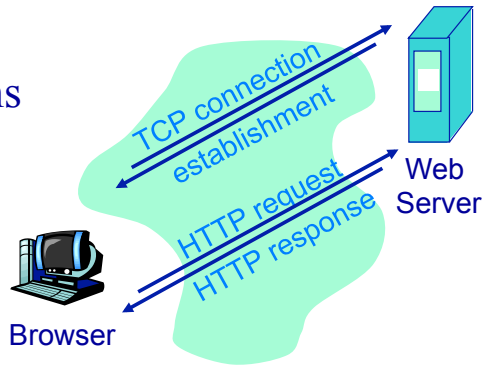
Response headers

```
Date: Fri, 02 Feb 2001 19:10:11 GMT
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
Last-Modified: Tue, 30 Jan 2001 21:48:14 GMT
ETag: "1807135e-67-3a77369e"
Accept-Ranges: bytes
Content-Length: 103
Connection: close
Content-Type: text/plain
```


HTTP Protocol Design

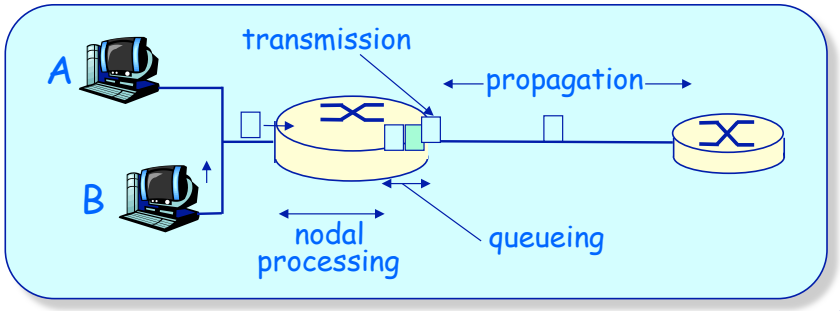
Non-persistent connections

- The default browser/server behavior in HTTP/1.0 is for the connection to be closed after the completion of the request
  - » Server parses request, responds, and closes TCP connection
  - » The **`Connection: keep-alive`** header allows for persistent connections
- With non-persistent connections *at least* 2 RTTs are required to fetch every object
  - » 1 RTT for TCP handshake
  - » 1 RTT for request/response

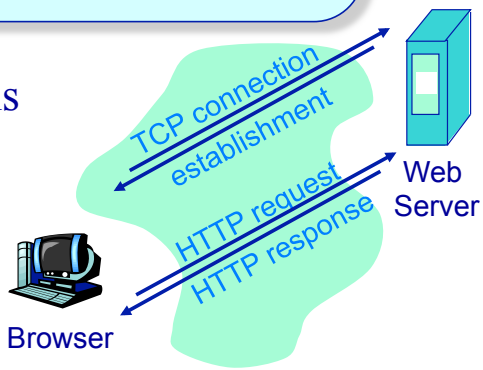


Non-Persistent Connections

Performance



- With non-persistent connections *at least* 2 RTTs are required to fetch every object
  - » 1 RTT for TCP handshake
  - » 1 RTT for request/response



HTTP Protocol Design

Persistent *v.* non-persistent connections

- Non-persistent
  - » HTTP/1.0
  - » Server parses request, responds, and closes TCP connection
  - » At least 2 RTTs to fetch every object
- Persistent
  - » Default for HTTP/1.1 (negotiable in 1.0)
  - » Client sends requests for multiple objects on one TCP connection
  - » Server, parses request, responds, parses next request, responds...
  - » Fewer RTTs

Non-Persistent vs. Persistent Connections

Performance Example

- A base page with five embedded images located on the same web server
- How many TCP connections to download the page with non-persistent connections?
  - » How many round-trip times (RTTs)?
- How many TCP connections to download the page with persistent connections?
  - » How many RTTs?

Non-Persistent vs. Persistent Connections

Performance Example

- A base page with five embedded images located on the same web server
- How many TCP connections to download the page with *non-persistent* connections?
  - » 1 TCP connection to download the base page
  - » 1 TCP connection to download each of the 5 embedded images
  - » Total: **6 TCP connections**
- How many round-trip times (RTTs)?
  - » Each TCP connection requires a handshake: 1 RTT
  - » Once connection is setup, it takes 1 RTT to download an object (send HTTP request, receive HTTP response)
  - » Can only download one object per connection (*non-persistent),* so 2 RTTs per connection
  - » Total: 2 RTTs * 6 connections = **12 RTTs**

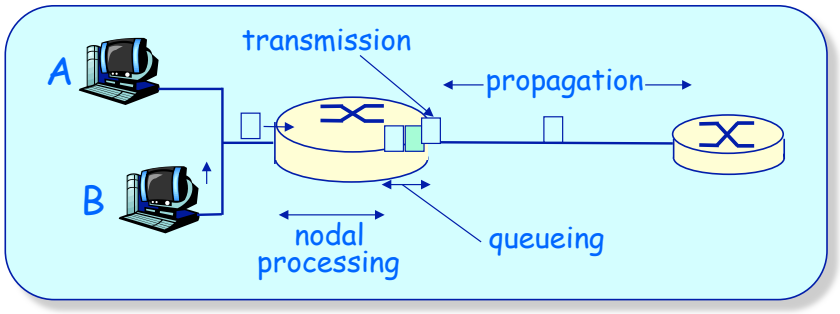
Non-Persistent vs. Persistent Connections

Performance Example

- A base page with five embedded images located on the same web server
- How many TCP connections to download the page with *persistent* connections?
  - » 1 TCP connection to download the base page
  - » Since all 5 embedded images are at the same web server as the base page, no more TCP connections are needed
  - » Total: **1 TCP connection**
- How many round-trip times (RTTs)?
  - » Each TCP connection requires a handshake: 1 RTT
  - » Once connection is setup, it takes 1 RTT to download an object (send HTTP request, receive HTTP response)
  - » There are 6 objects (base page + 5 images) to download
  - » Total: 1 RTT (TCP handshake) + 6 RTTs (download all objects) = **7 RTTs**

Non-Persistent Connections

Performance



- Example: A 1 KByte base page with five 1.5 KByte embedded images coming from the West coast on an OC-48 link
  - » 1 RTT for TCP handshake = 50 *ms*
  - » 1 RTT for request/response = 50 *ms*
- Page download time with non-persistent connections?
- Page download time with a persistent connection?

Non-Persistent vs. Persistent Connections

Your Turn

- A base page with 3 embedded images located on the same web server
  - » RTT from client to web server is 50 ms
  - » slowest link on the path is 2 Mbps
    - ❖ other links are high-speed, so transmission delay is negligible
  - » base page is 1000 bytes
  - » each image is 3000 bytes

- What is the total time (propagation + transmission delays) needed to download the entire web page with non-persistent HTTP connections?

Non-Persistent vs. Persistent Connections

Your Turn

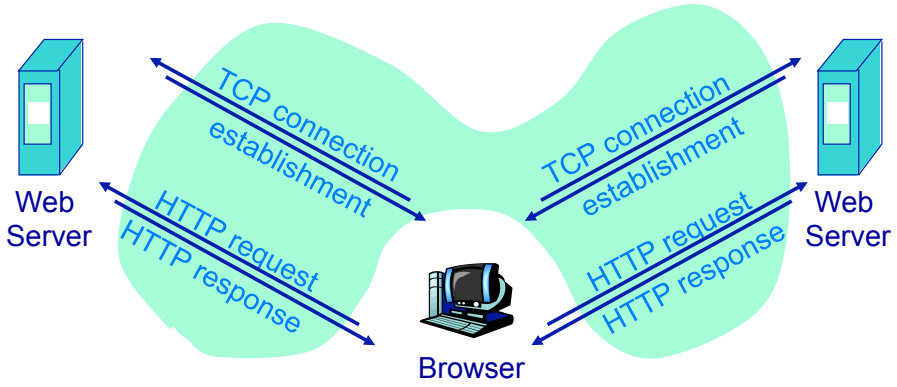
- A base page with 3 embedded images located on the same web server
  - » RTT from client to web server is 50 ms
  - » slowest link on the path is 2 Mbps
    - ❖ other links are high-speed, so transmission delay is negligible
  - » base page is 1000 bytes
  - » each image is 3000 bytes

- What is the total time (propagation + transmission delays) needed to download the entire web page with *persistent* HTTP connections?

Non-Persistent Connections

Parallel connections

- To improve performance a browser can issue multiple requests in parallel to a server (or servers)
  - » Server parses request, responds, and closes TCP connection



- Page download time with parallel connections?
  - » 2 parallel connections =
  - » 4 parallel connections =

Persistent Connections

Persistent connections with pipelining

Persistent without pipelining:

- Client issues new request only when previous response has been received
- At least one RTT for each embedded object

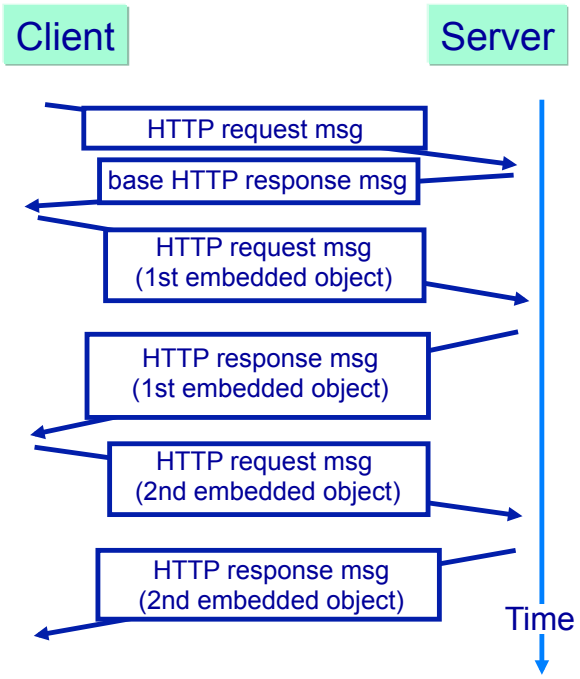
Persistent with pipelining:

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

Persistent Connections

Without Pipelining

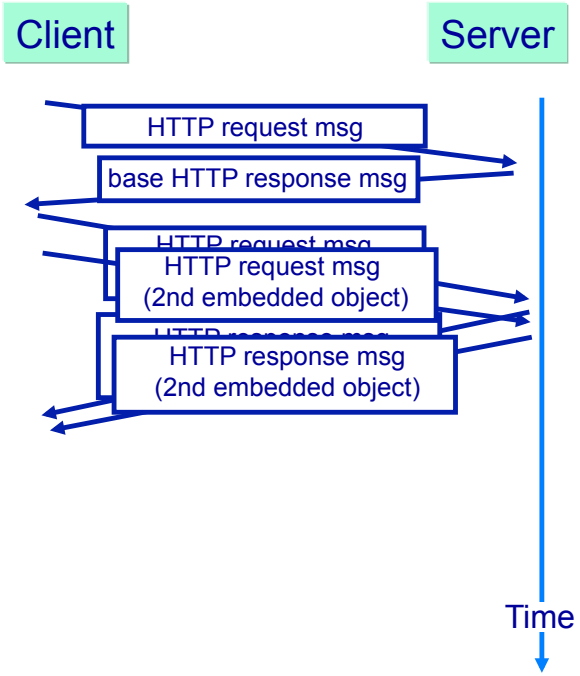
- Client issues new request only when previous response has been received
- At least one RTT for each embedded object



Persistent Connections

With Pipelining

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



The Web & HTTP

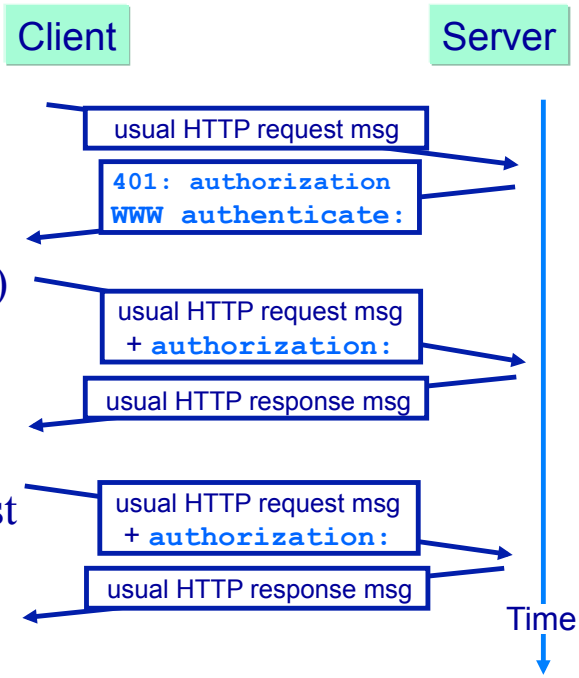
Outline

- Terminology (KR 2.2.1)
- HTTP protocol
  - » message format (KR 2.23)
  - » non-persistent and persistent connections (KR 2.2.2)
  - » pipelining
- Authentication
- Cookies (KR 2.2.4)
- Web caches (KR 2.2.5-2.2.6)
- Security (KR 8.1-8.3)

HTTP User-Server Interaction

Authentication

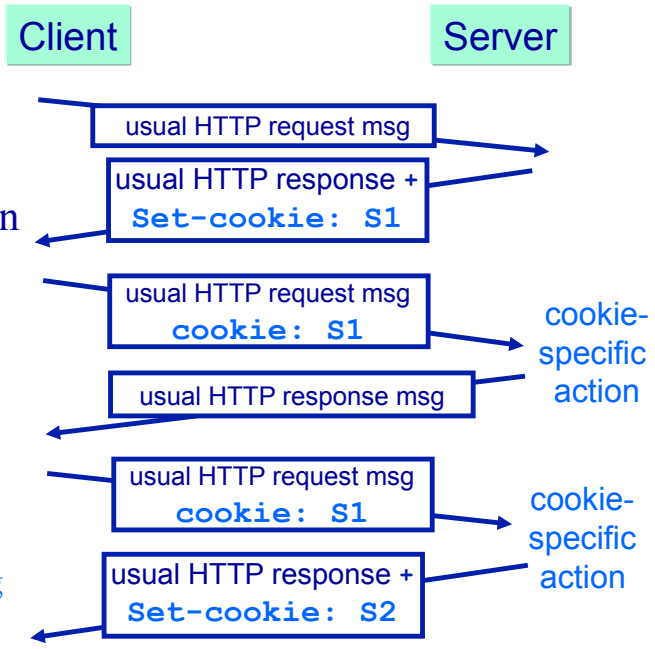
- Problem: How to limit access to server documents?
  - » Servers provide a means to require users to authenticate themselves
- HTTP includes a header tag for user to specify name and password (on a GET request)
  - » If no authorization presented, server refuses access, sends `WWW authenticate:` header line in response
- Stateless: client must send authorization for each request
  - » A stateless design
  - » (But browser may cache credentials)



HTTP User-Server Interaction

Cookies

- Server sends "cookie" to browser in response message
  `Set-cookie:<value>`
- Browser presents cookie in later requests to same server
  `cookie: <value>`
- Server matches cookie with server-stored information
  - » Provides authentication
  - » Client-side state maintenance (remembering user preferences, previous choices, …)



The Web & HTTP

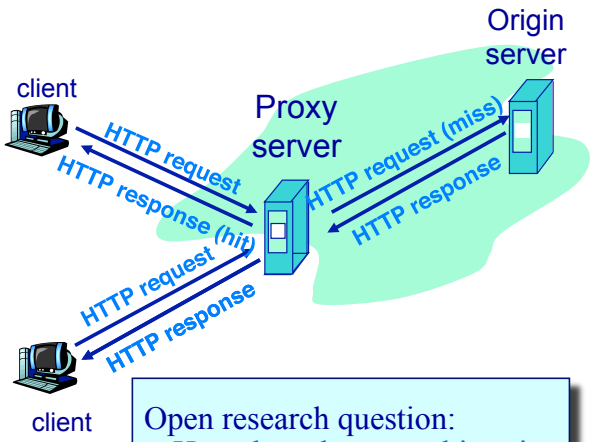
Outline

- Terminology (KR 2.2.1)
- HTTP protocol
  - » message format (KR 2.23)
  - » non-persistent and persistent connections (KR 2.2.2)
  - » pipelining
- Authentication
- Cookies (KR 2.2.4)
- Web caches (KR 2.2.5-2.2.6)
- Security (KR 8.1-8.3)

Caching on the Web

Web caches (Proxy servers)

- Web caches are used to satisfy client requests without contacting the origin server
- Users configure browsers to send all requests through a shared *proxy* server
  - » Proxy server is a large cache of web objects
- Browsers send *all* HTTP requests to proxy
  - » If object in cache, proxy returns object in HTTP response
  - » Else proxy requests object from origin server, then returns it in HTTP response to browser

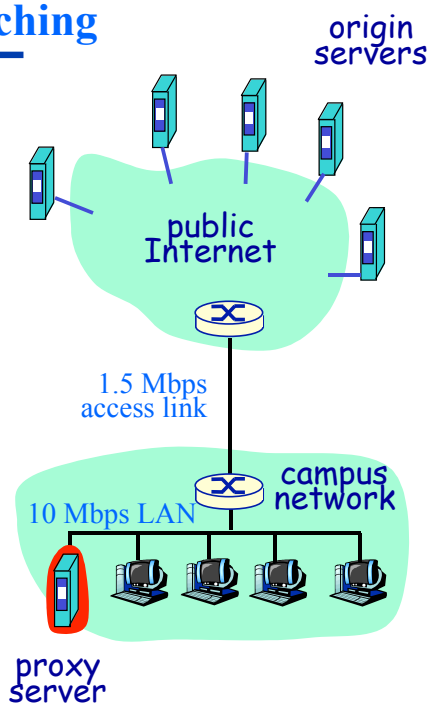


Open research question:
How does the proxy hit ratio change with the population of users sharing it?

Why do Proxy Caching?

The performance implications of caching

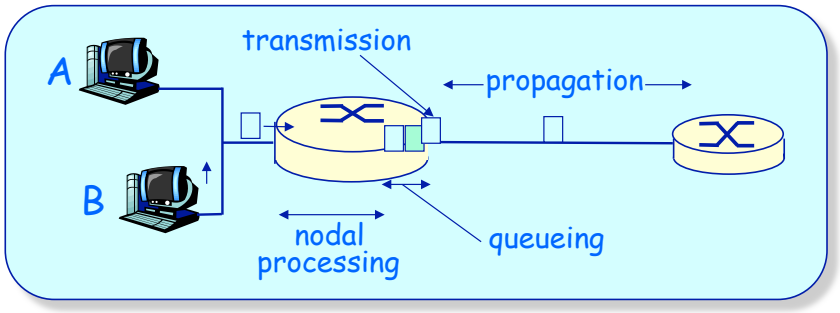
- Consider a cache that is "close" to client
  - » *E.g.*, on the same LAN
- Nearby caches mean:
  - » Smaller response times
  - » Decreased traffic on egress link to institutional ISP (often the primary bottleneck)



To improve Web response times should one buy a 10 Mbps access link or a proxy server?

Why Do Proxy Caching?

Delay in packet-switched networks (review)



- Packets experience variable delays along the path from source to destination
- Four sources of delay at each hop
  - » Queuing delay depends on the load ("traffic intensity") on the network

Why Do Proxy Caching?

Traffic Intensity, or Utilization

L = packet length (bits/packet)

R = link speed (bps)

a = average packet arrival rate (packets/second)

La = average *bit* arrival rate (bits/second)

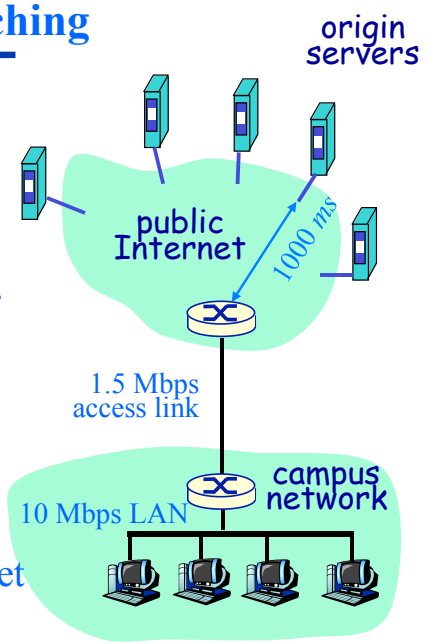
***La/R* = traffic intensity**

= number of bits arriving per second / number of bits that can be transmitted per second

What happens when La/R > 1?

Why do Proxy Caching?

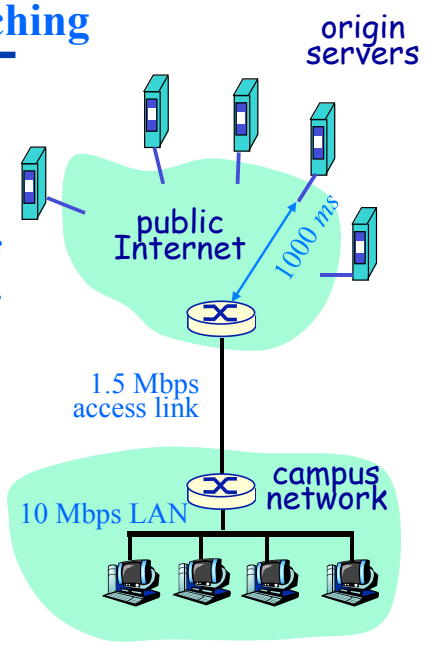
The performance implications of caching



- Web performance without caching:
  - » Mean object size (L) = 50 Kbits
  - » Mean request (object) rate = 29/*sec*
  - » Mean origin server access time = 1 *sec*
  - » Average response time = ??
- Average response time is
  - » average access time over access link
  - \+ average access time over public Internet
- Average access time over access link is...

Why do Proxy Caching?

The performance implications of caching



- Web performance without caching:
  - » Mean object size (L) = 50 Kbits
  - » Mean request (object) rate (a) = 29/*sec*
  - » Mean origin server access time = 1 *sec*
  - » Average response time = ??
- Traffic intensity on the access link:

La = (50,000 bits/obj) (29 obj/sec)
= 1,450,000 b/sec

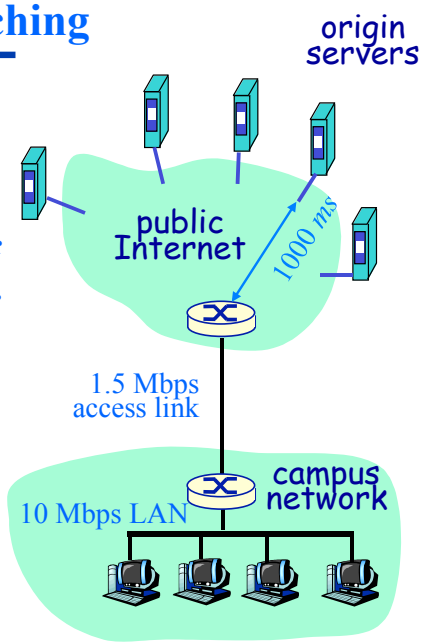
R = 1.5 Mbps

$$La/R = \frac{1{,}450{,}000 \text{ bits}}{1 \text{ sec}} \times \frac{1 \text{ sec}}{1{,}500{,}000 \text{ bits}} = 0.96$$

Why do Proxy Caching?

The performance implications of caching

- Web performance without caching:
  - » Mean object size = 50 Kbits
  - » Mean request (object) rate (a) = 29/*sec*
  - » Mean origin server access time = 1 *sec*
  - » Average response time = ??
- Average response time is
  - » average access time over access link
    + average access time over Internet
  - » *a long time* (w/traffic intensity = 0.96)
    + 1000 ms

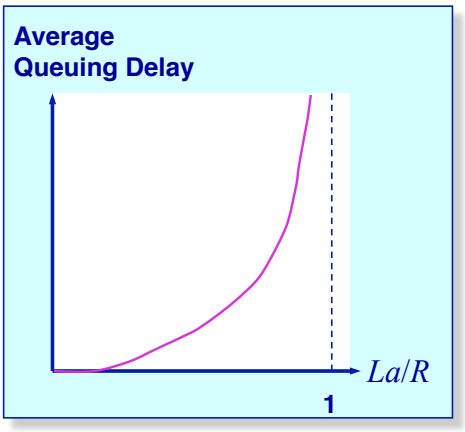


Rule of thumb: If traffic intensity < 60%, ignore queuing delay.
If traffic intensity >= 60%, assume queuing delay is large.

Why Do Proxy Caching?

Delay in packet-switched networks (review)

- Understand queuing delay in terms of traffic intensity La/R
  - » R = link transmission speed (bps)
  - » L = packet length (bits/packet)
  - » a = average packet arrival rate (packets/second)

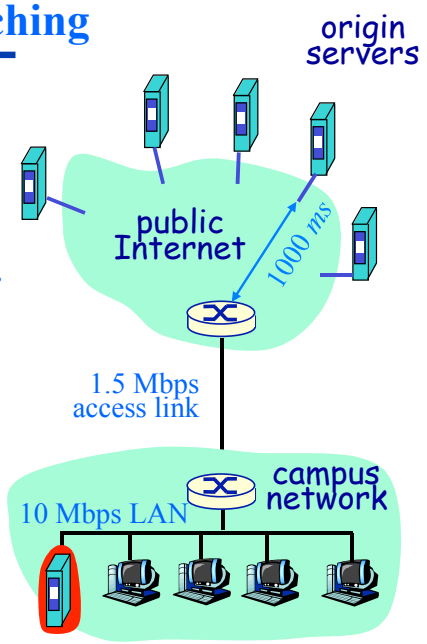


- If $La/R \sim 0$: Average queuing delay small
- As $La/R \Rightarrow 1$: Delays become large
- If $La/R > 1$: Work arrives faster than it can be serviced
  - » Average delay goes to infinity!

Why Do Proxy Caching?

The performance implications of caching

- Web performance without caching:
  - » Mean object size = 50 Kbits
  - » Mean request (object) rate = 29/*sec*
  - » Mean origin server access time = 1 *sec*
  - » Average response time = ??
- What is traffic intensity on LAN?

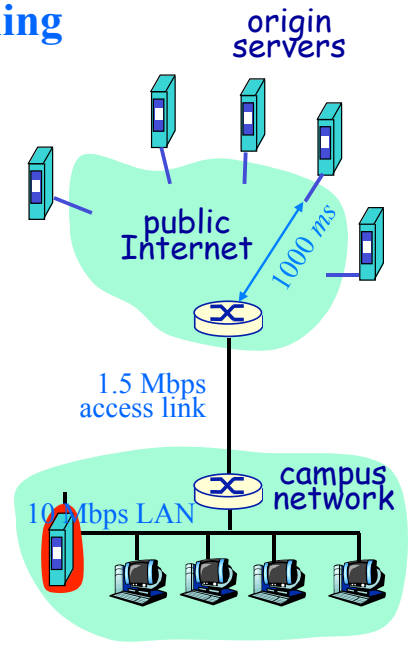


Why Do Proxy Caching?

The performance implications of caching

- Upgrade the access link to 10 Mbps
  - » Response time = ??
  - » Queuing is negligible hence response time ~ 1 *s*
- Leave access link at 1.5 Mbps but add a proxy cache with 40% hit ratio and 10 *ms* access time
  - » *Cache access time* includes time to send request to proxy, time to search cache, and time to send response back to client
  - » Response time = ??
  - » Traffic intensity on access link = 0.6 x 0.97 = 0.58
  - » Response time =

    0.4 x 10 *ms* + 0.6 x 1,010 ms = 610 *ms*



A proxy cache lowers response time, lowers access link utilization, and saves money!

Cache Performance for HTTP Requests

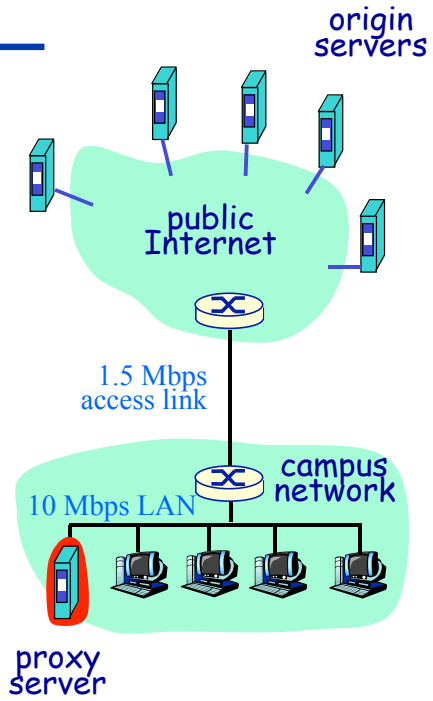
What determines the hit ratio?

- Cache size
- *Locality* of references
  - » How often the same web object is requested
- How long objects remain "fresh" (unchanged)
- Object references that can't be cached at all
  - » Dynamically generated content
  - » Protected content
  - » Content purchased for each use
  - » Content that must always be up-to-date
  - » Advertisements ("pay-per-click" issues)

Why Do Proxy Caching?

The case for proxy caching

- Lower latency for user's web requests
- Reduced traffic at all network levels
- Reduced load on servers
- Some level of fault tolerance (network, servers)
- Reduced costs to ISPs, content providers, *etc*., as web usage continues to grow exponentially
- More rapid distribution of content



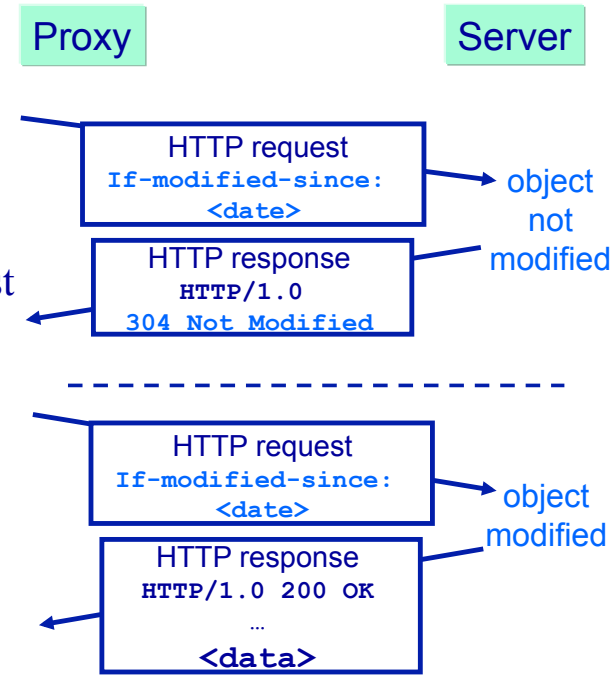
HTTP User-Server Interaction

The conditional GET

- Goal: don't send object if cache has up-to-date cached version
  - » no object transmission delay
  - » lower link utilization
- Proxy specifies the date of cached copy in HTTP request

  `If-modified-since:<date>`
- Server's response contains the object only if it has been changed since the cached date
- Otherwise server returns:

  `HTTP/1.0 304 Not Modified`



HTTP Message Format

Telnet example

Connect to HTTP server port →

```
% telnet www.cs.odu.edu 80
```

Telnet output

```
Trying 128.82.4.2...
Connected to xenon.cs.odu.edu.
Escape character is '^]'.
```

Type GET command **plus blank line**

```
GET /~mweigle/files/foo.txt HTTP/1.0
```

HTTP response status line →

```
HTTP/1.1 200 OK
```

HTTP response headers plus blank line

```
Date: Wed, 30 Jan 2013 01:44:23 GMT
Server: Apache/2.2.17 (Unix) PHP/5.3.5 ...
Last-Modified: Thu, 19 May 2011 19:23:43 GMT
ETag: "5c-4a3a5f178cdd0"
Accept-Ranges: bytes
Content-Length: 92
Connection: close
Content-Type: text/plain
```

Object content

```
This test file is stored in the UNIX file
system at
/home/mweigle/public_html/files/foo.txt
```

Telnet output →

```
Connection closed by foreign host.
```

The Web & HTTP
Outline

- Terminology (KR 2.2.1)
- HTTP protocol
  - » message format (KR 2.23)
  - » non-persistent and persistent connections (KR 2.2.2)
  - » pipelining
- Authentication
- Cookies (KR 2.2.4)
- Web caches (KR 2.2.5-2.2.6)
- Security (KR 8.1-8.3)

Security
HTTPS

- HTTP over Secure Socket Layer
  - » Secure version of HTTP
  - » Encrypts the session data
    - ❖ Using either the SSL (Secure Socket Layer) protocol or the TLS (Transport Layer Security) protocol
  - » SSL and TLS work above TCP but below application protocols (HTTP, SMTP, etc.)
- Transferred using HTTP, encrypted
  - » with default TCP/IP port 443
- For Web pages, the URL begins with https://
- Provides server authentication and encrypted communication

Security
Authentication vs. Encryption

- Authentication
  - » to confirm the sender is who they say they are
  - » using a digital certificate issued by a trusted third party

- Encryption
  - » to prevent others from reading the message
  - » using a cipher (encryption-decryption algorithm)
  - » with symmetric or asymmetric (related public and private) keys

Security
Encryption and Decryption

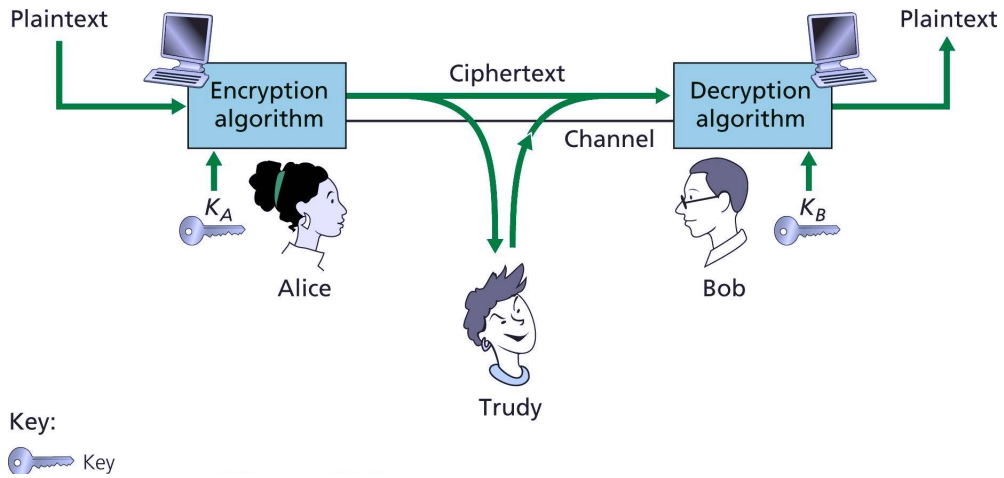
- Symmetric cryptography
  - » Same key for encryption and decryption
    - ❖ would be efficient as long as the key is pre-agreed and secure
  - » Not suitable for the Web

- Asymmetric-key cryptography (also called Public-key cryptography)
  - » Using a pair of related public and private keys. Encryption and decryption are asymmetric.
  - » Used in HTTPS

Principles of Cryptography

The language of cryptography

"crypto" - secret
"graphy" - writing



symmetric key crypto: sender, receiver keys *identical*

public-key crypto: encryption key *public*, decryption key *secret*

Principles of Cryptography

Symmetric key cryptography

Substitution cipher: substituting one thing for another

» Caesar cipher: substitute one letter for another by shifting alphabet *k* letters

```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  lmnopqrstuvwxyzabcdefghijk
```

E.g.:

```
Plaintext: bob. i love you. alice
ciphertext: mzm. t wzgp jzf. lwtnp
```

Principles of Cryptography

Symmetric key cryptography

Substitution cipher: substituting one thing for another

» monoalphabetic cipher: substitute one letter for another

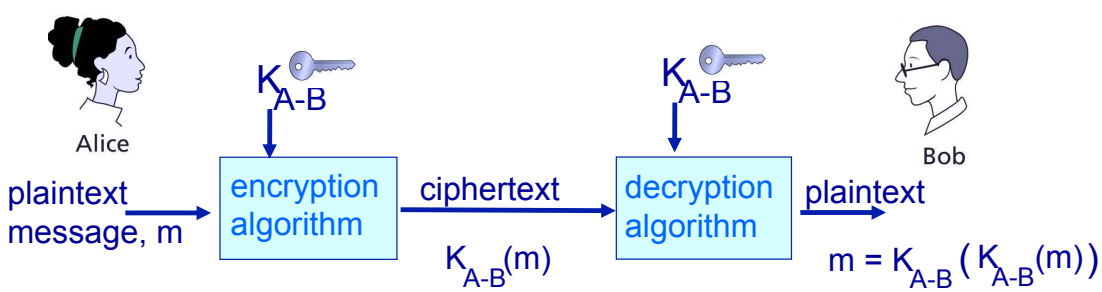
```
plaintext:   abcdefghijklmnopqrstuvwxyz

ciphertext:  mnbvcxzasdfghjklpoiuytrewq
```

E.g.: **Plaintext: bob. i love you. alice**
ciphertext: nkn. s gktc wky. mgsbc

Principles of Cryptography

Symmetric key cryptography



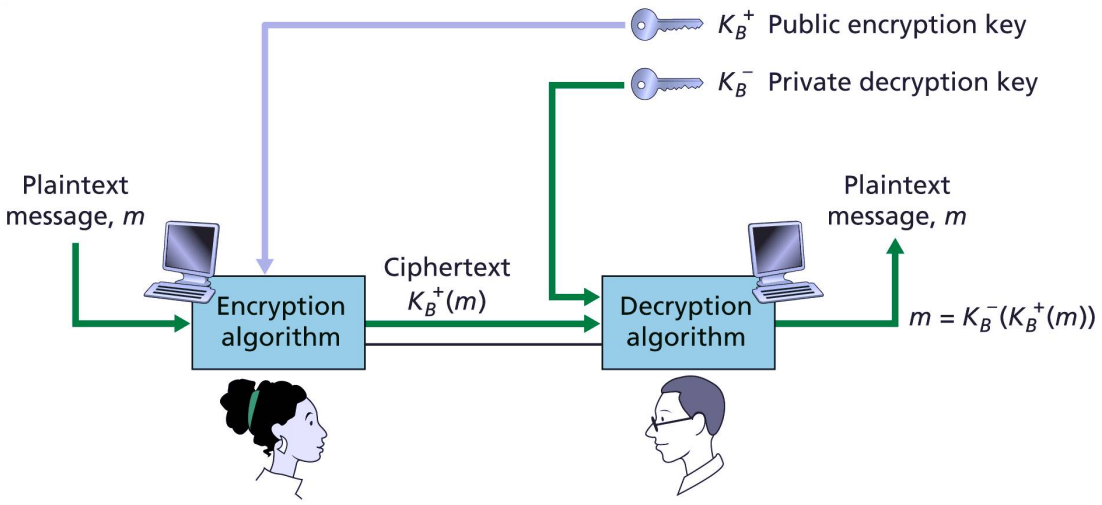
Symmetric key crypto: Bob and Alice know same key, K_{A-B}

E.g.: Key is knowing substitution pattern in monoalphabetic substitution cipher

Principles of Cryptography
Public Key Cryptography

- Symmetric key cryptography
  - » requires sender and receiver to know shared secret key
- Public key cryptography
  - » sender and receiver do not share secret key
  - » public encryption key known to all
  - » private decryption key known only to receiver

Principles of Cryptography
Public Key Cryptography



Public Key Cryptography Algorithms

Requirements

Need $K^+_B(m)$ and $K^-_B(m)$ such that

1. $K^-_B(K^+_B(m)) = m$
2. Given public key K^+_B, it should be impossible to compute private key K^-_B

RSA: **R**ivest, **S**hamir, **A**delson algorithm

RSA

Another Important Property

The following property will be *very* useful later:

$$\underbrace{K^-_B(K^+_B(m))}_{\text{use public key first, followed by private key}} = m = \underbrace{K^+_B(K^-_B(m))}_{\text{use private key first, followed by public key}}$$

Result is the same!

Public Key Cryptography

How To Bind a Public Key to Its User?

- Public-key infrastructure (PKI)
  - » Provide a trusted third party (Certifying Authority) to use identity *certificates* to bind public keys to users
  - » PKI may refer to the software that manages certificates in a large-scale setting
- A certificate may be revoked
  - » check the *certificate revocation list (CRL)*

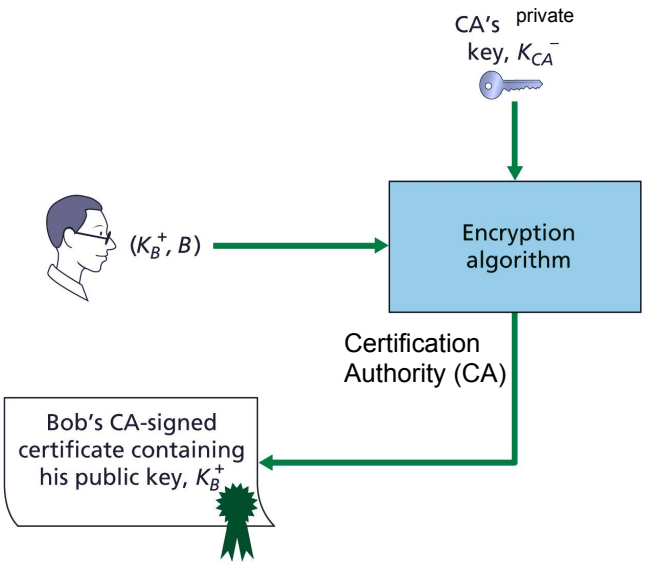
Public Key Cryptography

How to Generate a Certificate?

- Generate a public-private key pair from a large random number
- Keep the private key, send the public key and identifying information to a Certificate Authority (CA)
- Pay a fee to the CA
- The CA verifies the identity
- The CA creates a certificate (including all ID information and the URL of the web site)
- The CA *signs* the certificate (encoded with its own private key) and sends the signed certificate to you

Key Distribution and Certification
Certification Authorities

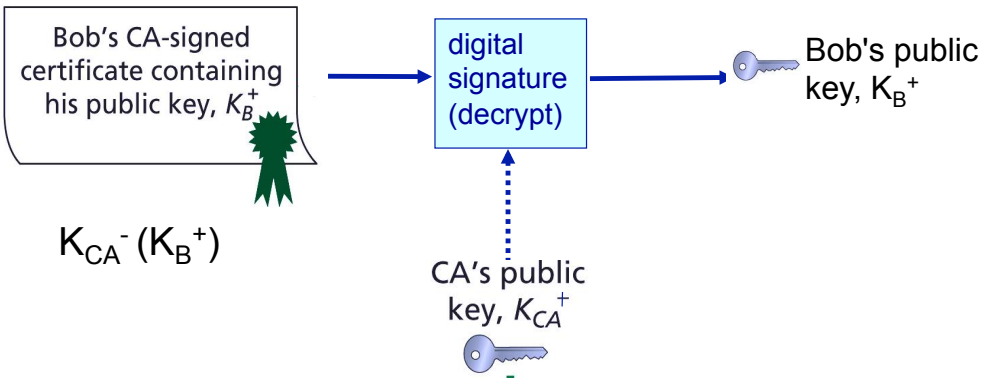
- Binds public key to particular entity, *E*.
- *E* registers its public key with CA.
  - » *E* provides "proof of identity" to CA.
  - » CA creates certificate binding E to its public key.
  - » Certificate containing *E*'s public key *digitally signed* by CA – CA says "this is *E*'s public key"



Key Distribution and Certification
Certification Authorities

When Alice wants Bob's public key:

- » get Bob's certificate (Bob or elsewhere).
- » apply CA's public key to Bob's certificate, get Bob's public key
  - ❖ web browsers and other apps are pre-loaded with public keys for various CAs



HTTPS

Putting Everything Together

- Server has obtained a certificate validating that it is who it says it is.
  - » certificate contains: CA^- (S^+, ID information)
- Server sends: m, $S^-(m)$, CA^- (S^+, ID information)
- Client uses CA's public key to open certificate, then gets server's public key.
- Client uses server's public key S^+ to open encrypted message $S^-(m)$
- Client compares message to unencrypted message m
- Once authenticated, client can encrypt its messages with the server's public key, S^+.

The Web & HTTP

Outline

- Terminology (KR 2.2.1)
- HTTP protocol
  - » message format (KR 2.23)
  - » non-persistent and persistent connections (KR 2.2.2)
  - » pipelining
- Authentication
- Cookies (KR 2.2.4)
- Web caches (KR 2.2.5-2.2.6)
- Security (KR 8.1-8.3)

Application-Layer Protocols

Outline

- The architecture of distributed systems
  - » Client/Server computing
  - » P2P computing
  - » Hybrid (Client/Server and P2P) systems
- The programming model used in constructing distributed systems
  - » Socket programming
- Example client/server systems and their application-layer protocols
  - » The World-Wide Web (HTTP)
  - » Reliable file transfer (FTP)
  - » E-mail (SMTP & POP)
  - » Internet Domain Name System (DNS)