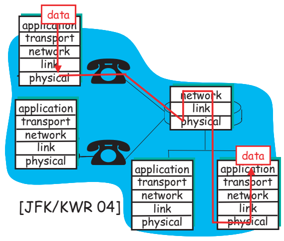
Slide 1

The Application Layer



Outline

1. Network applications: services needed versus services provided
2. Example: the WEB and the HTTP
3. Example: the Internet Domain Name System (DNS)
4. Example: peer-to-peer applications (self reading)
5. Key concepts of the Socket API: demultiplexing done by UDP and TCP

Slide 2

1. Network applications: services needed versus services provided

1. In general, each application has
   (a) data loss requirements
   (b) bandwidth (throughput) requirements
   (c) delay requirements
   (d) security requirements
2. Data loss
   - loss-sensitive applications
     - e.g., e-mail, ftp, web documents transfers
     - for the Internet: TCP provides reliable data transfer
   - loss-tolerant applications
     - e.g., some audio/video encoding schemes generate data that can tolerate a certain amount of loss (in such cases, loss degrades the

Slide 3

- playback quality)
- for the Internet: UDP provides fast, yet unreliable transfer mode (the media player handles transmission errors)

3. Bandwidth
   - bandwidth sensitive applications
     - require a minimum amount of bandwidth to be effective
     - e.g., audio: 32 Kbps to 128 Kbps (MP3 compression), video: about 1.5 Mbps (MPEG-1 compression)
   - elastic applications: make use of whatever bandwidth they get
4. Delay
   - delay sensitive applications
     - e.g., Internet telephony, interactive games, etc.
     - have tight constraints on the required end-to-end delay (a few hundred msecs)
   - delay insensitive applications

Slide 4

5. Security
   - Confidentiality: no unauthorized disclosure of information
   - Integrity: data is not altered in an unauthorized way
   - Authentication: sender and receiver are who/what they claim to be
   - Access and availability: services are accessible and available to users
6. What services does the current Internet offer?
   (a) Best effort service model; no service guarantees
   (b) Connection oriented service
     - provided by the TCP transport protocol
     - gives a loose sense of a connection
     - provides reliable, in-order, data transfer over an unreliable network
     - provides flow control : sender won't overwhelm receiver
     - provides congestion control : slow down sender when network is overloaded
     - TCP-enhanced-with-SSL provides: confidentiality, integrity, and

end-point authentication

- does not provide any bandwidth or delay guarantees

(c) Connectionless service

- provided by the UDP transport protocol
- light and fast protocol: server can respond to many clients efficiently
- provides unreliable, unordered, delivery
- no flow control (so, sender can overrun receiver's buffer)
- no congestion control (so, sender can overwhelm the network)

Slide 5

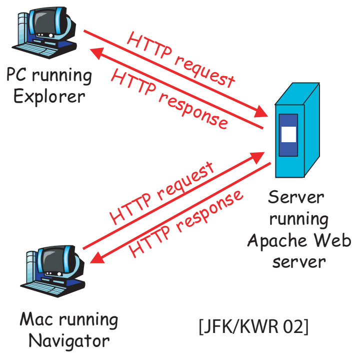
2. The WEB and the HTTP

1. Section 2.2 of the textbook
2. Aside from the included multimedia content, the WEB traffic is *loss-sensitive* (hence, carried by TCP), and *elastic*.
3. Basic Components
   - Clients (user agents: browsers)
   - Servers (e.g., the Apache server, Sun Java System Web Server, Microsoft Internet Information Server)
   - HTML pages:
     - base HTML files
     - embedded URLs of objects (other HTML files, JPEG images, Java applets, audio files, video clips, etc.)
   - The HTTP (an application layer protocol)
     - defines syntax and semantics of messages, and rules of

Slide 6

Slide 7

communications



- currently: HTTP/1.0 (RFC 1945) and HTTP/1.1 (RFC 2616).
- HTTP is stateless protocol : i.e., server maintains no information about past client requests.

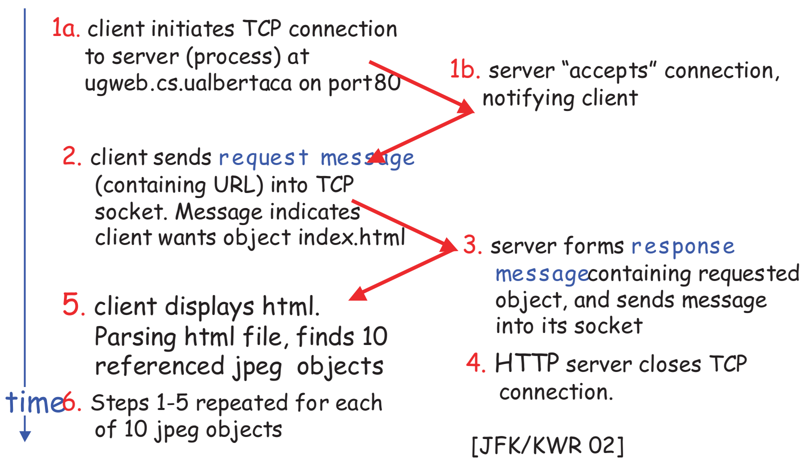
4. HTTP Connections: HTTP/1.0

- Provides non-persistent connections: i.e., at most one object is sent over a TCP connection. e.g., suppose user enters URL

Slide 8

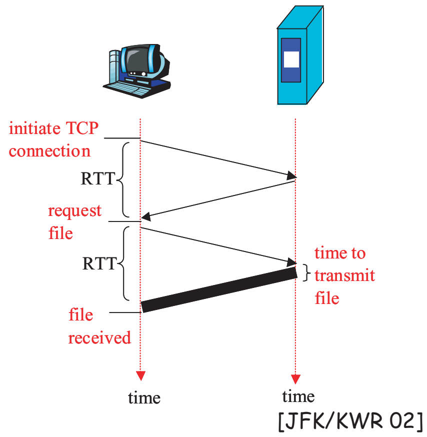
ugweb.cs.ualberta.ca/index.html,

and the requested file contains references to 10 JPEG images. Then



- Response time modelling:
  - RTT (round trip time): time for a small packet travels from client to server and back

Slide 9



- total file transmission time= $2RTT +$ transmission time

- [−ve] as WEB pages became more loaded with objects, browsers resorted to opening parallel TCP connections (more server overhead)

5. HTTP Connections: HTTP/1.1

- provides persistent connections: server leaves connection open after

Slide 10

sending response, subsequent HTTP messages are sent over the connection, server closes connection after timeout.

- persistent with pipelining (default mode), and persistent without pipelining

6. HTTP Messages

- Two types: request and response
- HTTP/1.0 methods:
  - GET : client requests an object
  - POST : client uploads data to server (possibly user data after filling a form), e.g.,

    `www.somesite.com/citysearch?Canada&Edmonton`
  - HEAD : client requests a Web page's header without the rest of its content (e.g., for indexing, or debugging, purposes)
- HTTP/1.1 additional methods:
  - PUT : client requests uploading files in the entity body to a path

specified in the URL field

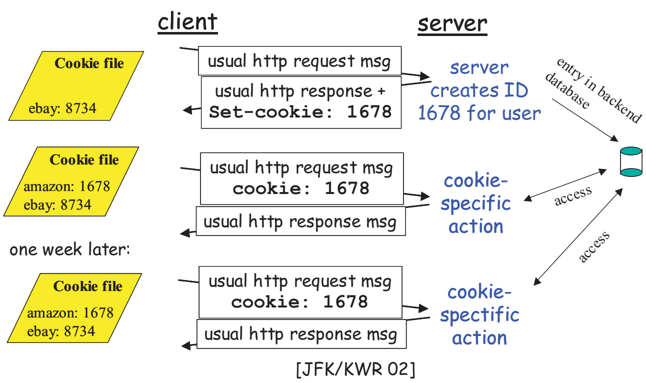
- DELETE : client requests files to be removed from the server (e.g., for WEB publishing)
- . . .

Slide 11

7. Keeping state using cookies

- *cookies* enable tracking of user activities (e.g., for maintaining "shopping carts")
- Components:
  (a) cookie header line in the HTTP request/response messages
  (b) cookie file kept on user's host (managed by user's browser)
  (c) back-end data base at the server's Web site

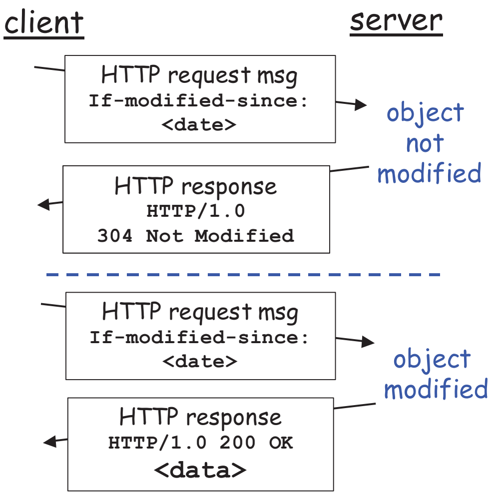
Slide 12



8. Client-side caching: conditional GET

- Goal: server does not send object, if client has an up-to-date cached version

Slide 13

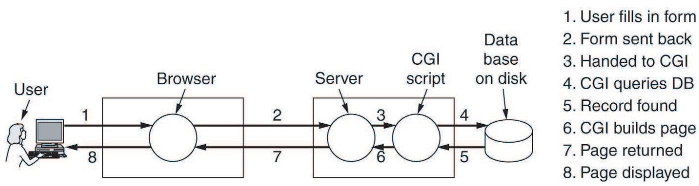


[JFK/KWR 02]

9. Other Issues

- Server-side dynamic Web page generation (e.g., CGI scripts, PHP, JavaServer Pages, ASP, etc.)

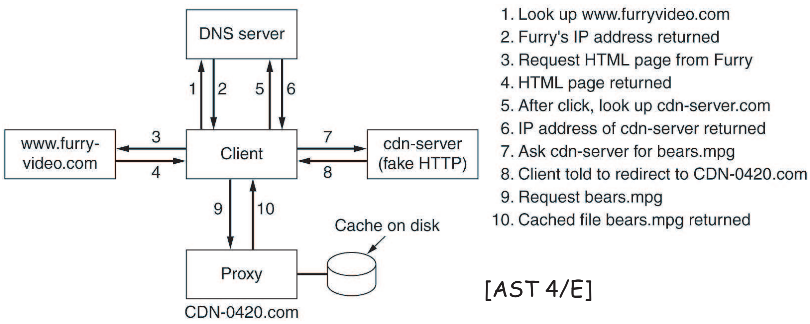
Slide 14



[TAS 4/E]

- Client-side dynamic Web page generation (e.g., JavaScript)
- Hierarchical caching (proxy servers)
- Server replication
- Content delivery networks

Slide 15



Slide 16

3. Example: the Internet Domain Name System (DNS)

1. Section 2.5 in the textbook
2. Currently, Internet hosts and routers are identified by
   - 32-bit (IPv4) addresses (e.g. 129.128.4.241), or
   - 128-bit (IPv6) addresses
   - Human readable names: e.g., *www.ibm.com*
3. The DNS is a global distributed database system for resolving such hostname-IP address mappings. It also provides:
   - host aliasing: e.g., *www.ibm.com* is really *servereast.backup2.ibm.com*
   - mail server aliasing: same as above (e.g., *hotmail.com* may be an alias for *relay1.west-coast.hotmial.com*)
   - load distribution: e.g., *cnn.com* may be replicated at 3 hosts, the DNS can rotate the host names after each reply, causing even access to the replication servers.

Slide 17

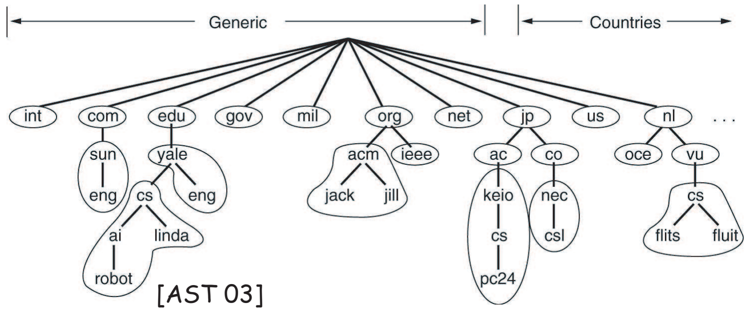
4. Note: although DNS is a core Internet service, it is implemented as a typical client-server application using the UDP transport protocol.
5. DNS is implemented using various types of name servers :
   - Local name servers (in close proximity of hosts)
   - A hierarchy of name servers:
     - Root name servers
     - Top-level domain (TLD) name servers
     - Authoritative name servers
6. Functionality:
   - Local name servers
     - organizations run one (or more) name servers (e.g., using BIND, a public-domain name server for UNIX machines)
     - application queries local servers first (e.g., an application may call `'gethostbyname()'`)
   - Authoritative name servers

Slide 18

- By definition, a name server is authoritative for a host if it always have a DNS record of it.
- Each host is registered with at least two authoritative name servers (one is in the host's local ISP).

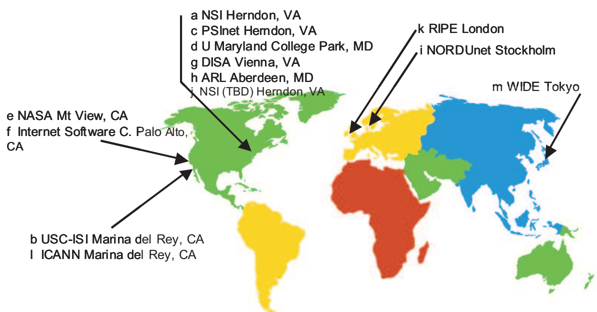
- Top-level domain (TLD) servers
  - Responsible for TLDs: e.g. .com, .org, .edu, .ca, .uk, .fr, etc.
  - Example companies involved at this layer:
    - Network Solutions: servers for **.com** TLD
    - Educause: servers for **.edu** TLD
  - For each domain (e.g., "ualberta.ca"), a TLD server stores either
    - authoritative servers for the domain, or
    - intermediate servers that know about authoritative servers for the domain

Slide 19



- **Root name servers**
  - contacted by TLD servers (and other name servers) that can not resolve a name
  - about a dozen root name servers exist worldwide

Slide 20



[JFK/KWR 04]

13 root name servers worldwide

7. DNS Resource Records (RR)

   Typical RR Fields: (name, time_to_live, class, type, value) , e.g.,

Slide 21

flits.cs.vu.nl.	86400	IN	HINFO	Sun Unix
flits.cs.vu.nl.	86400	IN	A	130.37.16.112
flits.cs.vu.nl.	86400	IN	A	192.31.231.165
flits.cs.vu.nl.	86400	IN	MX	1 flits.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	2 zephyr.cs.vu.nl.
flits.cs.vu.nl.	86400	IN	MX	3 top.cs.vu.nl.
www.cs.vu.nl.	86400	IN	CNAME	star.cs.vu.nl
ftp.cs.vu.nl.	86400	IN	CNAME	zephyr.cs.vu.nl

where

- name: is a hostname (simple such as 'flits', or fully qualified), or a domain name (e.g., `'foo.com'`)
- time_to_live: indicator of how stable the record is (highly volatile records are assigned small values)
- class: always `'IN'` for Internet information
- type and value (some examples):
  - if (type= HINFO) value is CPU and OS information

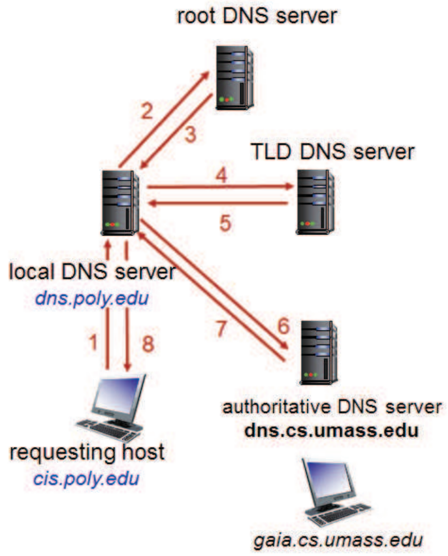
Slide 22

  - if (type= A) value is the IP address of the hostname
  - if (type= NS) value is IP address of authoritative name server for the domain
  - if (type= CNAME) value is a real name for the alias name in the first field
  - if (type= MX) value is name of a mailserver associated with the hostname (first field) plus a preference number (starting with the smallest value)

8. Example: iterated DNS query

- in iterated query, server replies with name of yet another server to contact

Slide 23

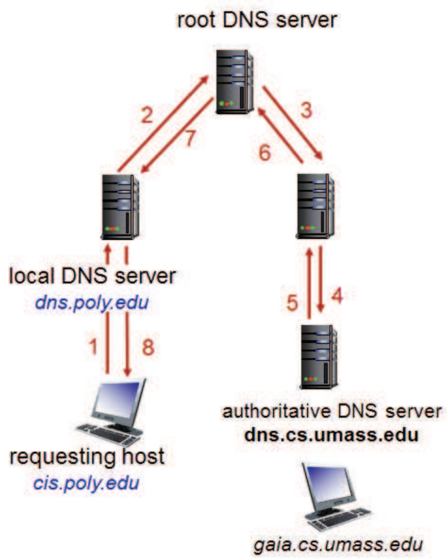


[JFK/KWR 04]

9. Example: recursive DNS query

Slide 24

- recursive query puts burden on contacted name server



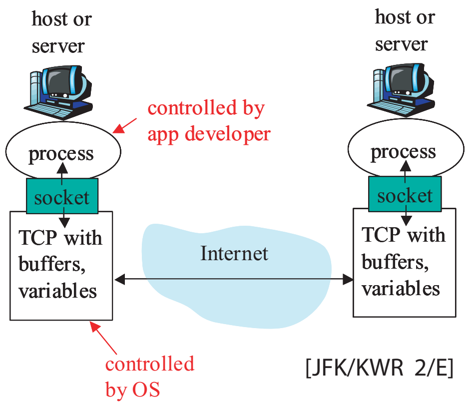
[JFK/KWR 04]

Slide 25

4. Key Concepts of the Socket API

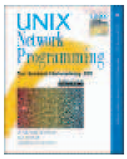
1. Background

- First introduced in Unix BSD (4.2) to support the ARPA network (nowadays, an industry standard).
- Sockets as gateways to the transport layer



Slide 26

- Unix Network Programming [Stevens, Fenner, and Rudoff 2004]:



[SFR 04]

  - structured around 'short' programs, each program illustrates some interesting concept
  - code is available free on-line
  - code uses a library that simplifies matters, without hiding details, e.g.

```
#define    SA       struct sockaddr

void
Connect(int fd, const struct sockaddr *sa, socklen_t salen)
{
        if (connect(fd, sa, salen) < 0)
                err_sys("connect error");
}
```

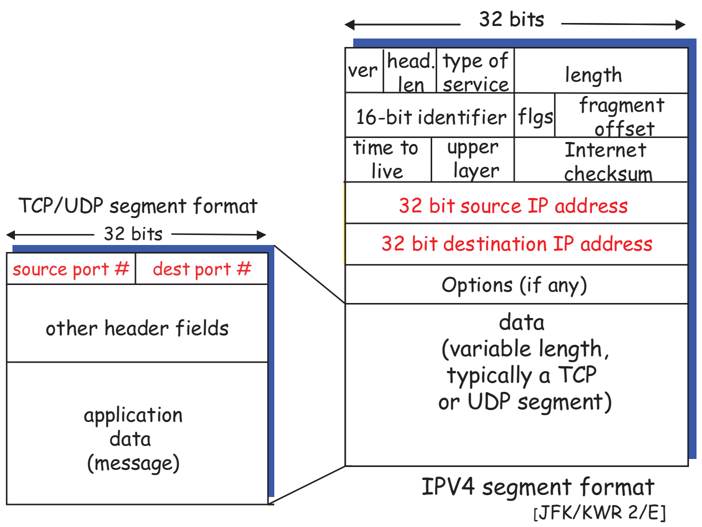
Slide 27

2. Demultiplexing: a transport layer service

- Among other services, the transport layer provides a logical communication channel between processes (process-to-process service)
- Using sockets, how does the transport layer find the right process to deliver a TCP or UDP datagram to?

1. To start, let's look at parts of the TCP/UDP and IPv4 segment structures:

Slide 28



2. The client (or server) programs opens a socket using a sequence like

```
int     sockfd;
sockfd= socket (AF_INET, SOCK_STREAM, 0);  // for TCP socket
sockfd= socket (AF_INET, SOCK_DGRAM, 0);   // for UDP socket
sockfd= socket (AF_INET, SOCK_RAW, 0);     // ...
```

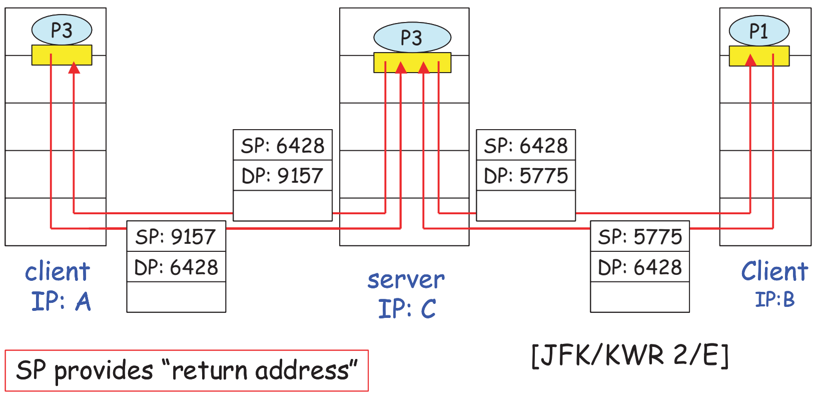
Slide 29

and then binds the socket to a port number.

3. port numbers assigned by the Internet Assigned Numbers Authority (IANA):
   - well-known ports : 0 through 1023
   - registered ports : 1024 through 49151 (not controlled by IANA, but kept track of by the IANA as a convenience to the community)
   - dynamic or private ports : 49152 to 65535 (called ephemeral ports in [SFR 04])
4. So, demultiplexing at rcv host means delivering received segments to correct socket
5. Demultiplexing done by UDP (connectionless) :
   - each UDP socket is identified by two-tuple:

     (dest IP address, dest port number)

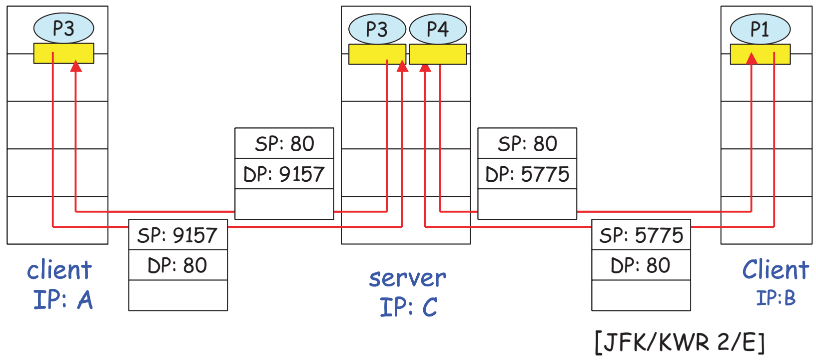
Slide 30



6. Demultiplexing done by TCP (connection-oriented) :
   - each TCP socket is identified by 4-tuple:

     (source IP address, source port number, dest IP address, dest port number)

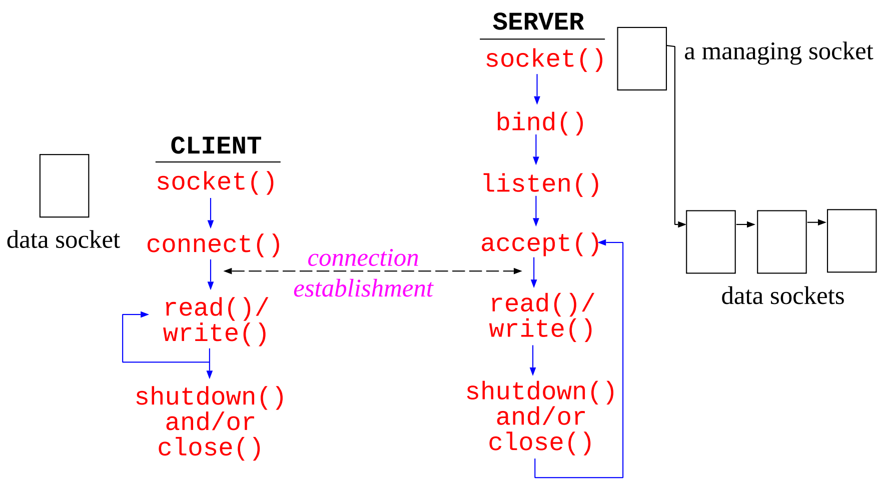
Slide 31



3. Connection Setup for TCP Sockets

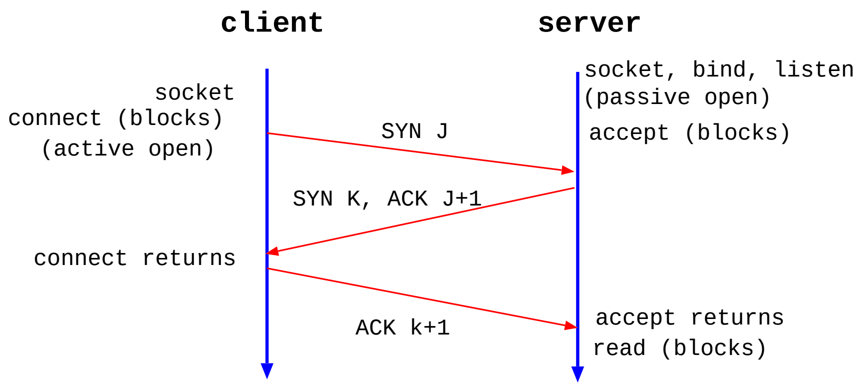
- The general sequence of calls required to implement such reliable stream delivery is *asymmetric*, as shown below:

Slide 32



- In more detail:

Slide 33


client
server
socket, bind, listen
(passive open)
socket
connect (blocks)
(active open)
SYN J
accept (blocks)
SYN K, ACK J+1
connect returns
accept returns
ACK k+1
read (blocks)