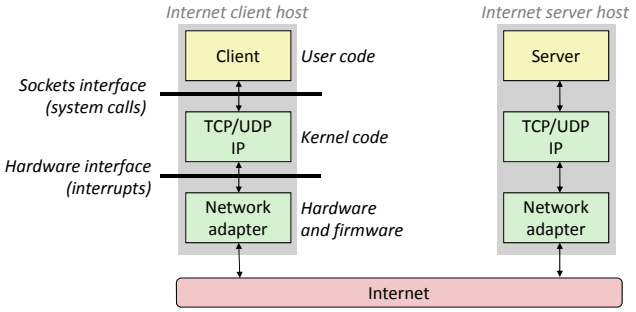
Network Programming

Kai Shen

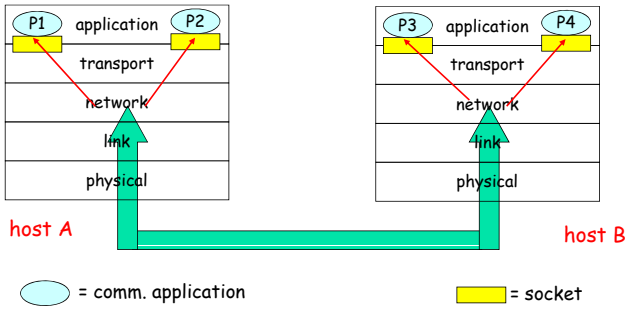
1

Hardware and Software Organization of an Internet Application



2

Multiplexing/demultiplexing



P1 on host A is communicating with P3 on host B; while at the same time, P2 on host A is communicating with P4 on host B.

3

How multiplexing/demultiplexing works?

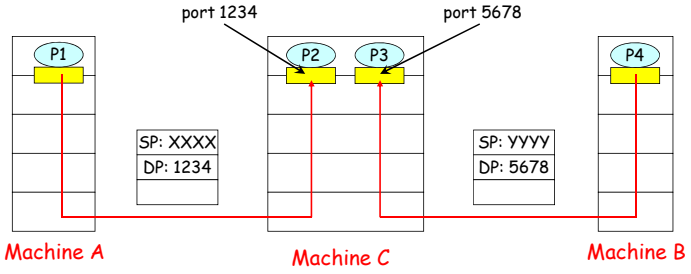
IP packet format

- Using port numbers
  - each IP packet has source IP address, destination IP address
  - each IP packet carries a transport-layer segment
  - each segment has source, destination port number
- Dest. IP address for routing to the host; IP addresses and port numbers for going to appropriate socket in the dest. host.

4

UDP Demultiplexing: An Example

UDP socket identified by dest. IP address and port number



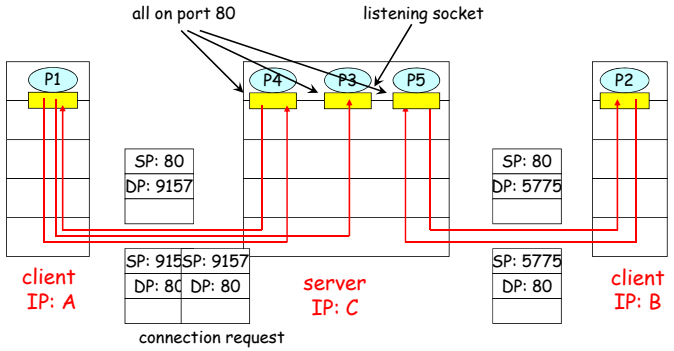
5

TCP Demultiplexing (Connection-oriented)

- Multiple TCP connections may simultaneously share a single port:
  - Example: web server (on default port 80) has multiple connections, one for each client
- TCP socket identified by 4-tuple:
  - (source IP address, source port number, dest IP address, dest port number)
  - recv host uses all four values to direct segment to appropriate socket

6

TCP Demultiplexing: An Example



7

Port Number Specifics

- A *port* is a 16-bit integer (0-65535) that helps identify a socket:
  - ***Ephemeral port*:** Assigned automatically when client makes a connection request (applications don't need to know)
  - ***Server port:*** Associated with some service provided by a server that clients need to know when connecting
- Port numbers ranging from 0 to 1023 are restricted for system-level services, e.g.,
  - Port 22: Secure shell server
  - Port 25: Mail server
  - Port 80: Web server
- Your server (without root privilege) can use ports at or above 1024.
- Ports are shared across the whole machine
  - If your buddy takes your favorite port number, just change your favorite.

8

TCP/UDP Sockets

Connection-oriented byte streams (TCP):

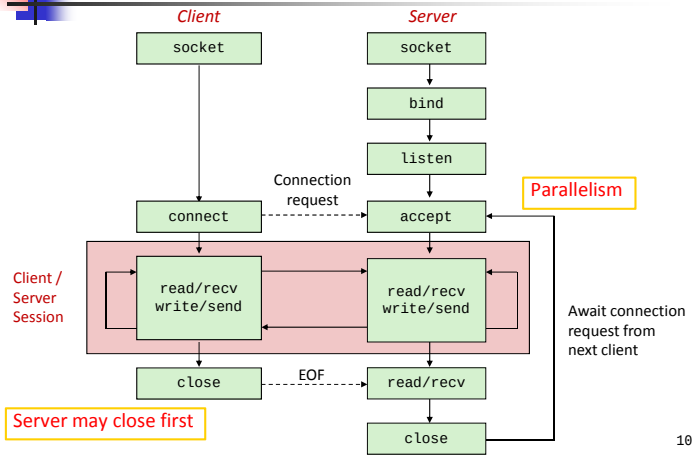
- Client/server need to establish connection before communications
- Communications flow in streams of ordered bytes
- But no clear sense of messages

Connectionless datagram (UDP):

- No connection is needed before communications;
- Clear boundaries between groups of bytes (messages)
- But no ordering between messages.

9

Overview of TCP Sockets



10

Socket Programming Notes

- **`gethostbyname()`**
  - Query name server for IP to name translation.
- Note
  - "**gdb** tells me **`gethostbyname`** fails with segmentation fault"!!!

11

Socket Programming Notes

- **`connfd = accept(listenfd, …)`**
  - Server waiting for a connection request.
- Notes
  - Returns when the connection with a new client is established and ready for data transfers
  - Data communications are done through the connection socket, not the listening socket
  - One listening socket for the whole server, one connection socket per active connection with a client

12

Socket Programming Notes

- **`read/recv, write/send`**
  - Data transfer calls.
- Note
  - TCP communications are stream-oriented, with no clear sense of message boundaries
  - when one sends 2000 bytes, the other side may receive twice (with 1460 bytes the first time, and 540 bytes the next time)
- How do you know when you are done with receiving?
  - Your protocol semantics should tell, three typical ways:
    - Fixed-size messages defined by protocol
    - Fixed-size header which contains a field that indicates the size of body
    - Always ends with a special pattern (e.g. an empty line "\n\n")

13

Socket Programming Notes

- How to write your receiver code?
- Fixed-size messages defined by protocol
  - **`read/recv`** for a certain size?
  - Must retry in case you don't get all the first time
  - Likely fine with one try if the size is small
- Fixed-size header which contains a field that indicates the size of body
  - Similar to above
- Always ends with a special pattern (e.g. an empty line "\n\n")
  - Be careful that the ending pattern may straddle over two messages

14

Socket Programming Notes

- Why I can't **`bind`** while restarting my server?
  - Your server port is kept by the OS for a certain amount of time in case some lingering packets belonging to the previous socket connection suddenly appears
  - Can be disabled by setting certain socket option

15

Socket Programming Notes

- Important!
  - Check the return code of every socket call (in fact, every system call)
  - Do error processing, warning messages for all cases
  - See UNIX manual pages for error codes and possible meaning

16

Testing Servers Using `telnet`

- The **`telnet`** program is invaluable for testing servers that transmit ASCII strings over Internet connections
  - Your simple FOOBAR server
  - Web servers
  - Mail servers
- Usage:
  - **`unix> telnet <host> <portnumber>`**
  - Creates a connection with a server running on **`<host>`** and listening on port **`<portnumber>`**

17

Testing the Echo Server With `telnet`

```
linux> telnet www.google.com 80
Trying 173.194.73.99...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.0 200 OK
Date: Thu, 12 Apr 2012 17:28:59 GMT
… … … …
```

```
linux> telnet <gmail mail server> 25
220 gmail.com
HELO cs.rochester.edu
250 Hello cs.rochester.edu, pleased to meet you
MAIL FROM: <bob@cs.rochester.edu>
250 bob@cs.rochester.edu... Sender ok
RCPT TO: <alice@gmail.com>
250 alice@gmail.com... Recipient ok
… … … …
```

18

Learn Network Programming

- Learn to use UNIX manual pages
- Complete versions of an echo client and server in the text
  - Updated versions linked to course website
- Find information on the web
- Reference book
  - Network programming in C: Stevens, "Unix Network Programming, Volume I", 2nd edition, 1998.
- C versus Java

19

Disclaimer

Parts of the lecture slides contain original work of James Kurose, Larry Peterson, and Keith Ross. The slides are intended for the sole purpose of instruction of computer organization course at the University of Rochester. All copyrighted materials belong to their original owner(s).

20