

Chapter 2: outline

2.1 principles of network applications

- app architectures
- app requirements

2.2 Web and HTTP

2.3 FTP

2.4 electronic mail

- SMTP, POP3, IMAP

2.5 DNS

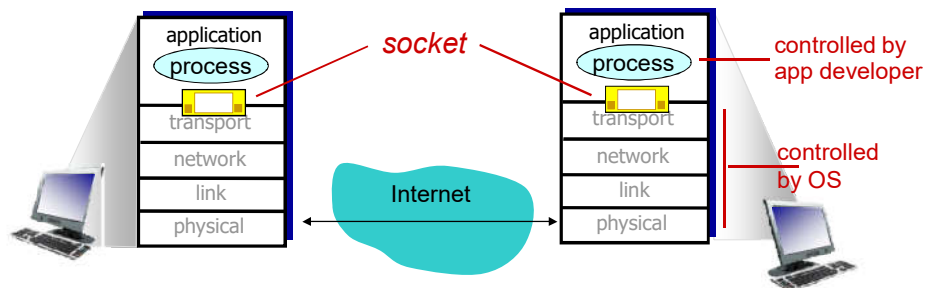
2.6 P2P applications

2.7 socket programming with UDP and TCP

Socket programming

goal: learn how to build client/server applications that communicate using sockets

socket: door between application process and end-end-transport protocol



Socket programming

Two socket types for two transport services:

- **UDP:** unreliable datagram
- **TCP:** reliable, byte stream-oriented

Application Example:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

Application Layer 2-95

Socket programming **with UDP**

UDP: no “connection” between client & server

- ❖ no handshaking before sending data
- ❖ sender explicitly attaches IP destination address and port # to each packet
- ❖ rcvr extracts sender IP address and port# from received packet

UDP: transmitted data may be lost or received out-of-order

Application viewpoint:

- ❖ UDP provides *unreliable* transfer of groups of bytes (“datagrams”) between client and server

Application Layer 2-96

Client/server socket interaction: UDP

server (running on serverIP)

create socket, port= x:
`serverSocket =
socket(AF_INET,SOCK_DGRAM)`

read datagram from
`serverSocket`

write reply to
`serverSocket`
specifying
client address,
port number

client

create socket:
`clientSocket =
socket(AF_INET,SOCK_DGRAM)`

Create datagram with server IP and
port=x; send datagram via
`clientSocket`

read datagram from
`clientSocket`

close
`clientSocket`

Application 2-97

Example app: UDP client

Python UDPClient

include Python's socket
library

from socket import *
serverName = 'hostname'
serverPort = 12000

create UDP socket for
server

clientSocket = socket(socket.AF_INET,
socket.SOCK_DGRAM)

get user keyboard
input

message = raw_input('Input lowercase sentence:')

Attach server name, port to
message; send into socket

clientSocket.sendto(message,(serverName, serverPort))

read reply characters from
socket into string

modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)

print out received string
and close socket

print modifiedMessage
clientSocket.close()

Application Layer 2-98

Example app: UDP server

Python UDPServer

```
from socket import *
serverPort = 12000

create UDP socket → serverSocket = socket(AF_INET, SOCK_DGRAM)
bind socket to local port number 12000 → serverSocket.bind(('', serverPort))
print "The server is ready to receive"

loop forever → while 1:
    Read from UDP socket into message, getting client's address (client IP and port) → message, clientAddress = serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    send upper case string back to this client → serverSocket.sendto(modifiedMessage, clientAddress)
```

Application Layer 2-99

Socket programming *with TCP*

client must contact server

- ❖ server process must first be running
- ❖ server must have created socket (door) that welcomes client's contact

client contacts server by:

- ❖ Creating TCP socket, specifying IP address, port number of server process
- ❖ *when client creates socket:* client TCP establishes connection to server TCP

- ❖ when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
 - allows server to talk with multiple clients
 - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

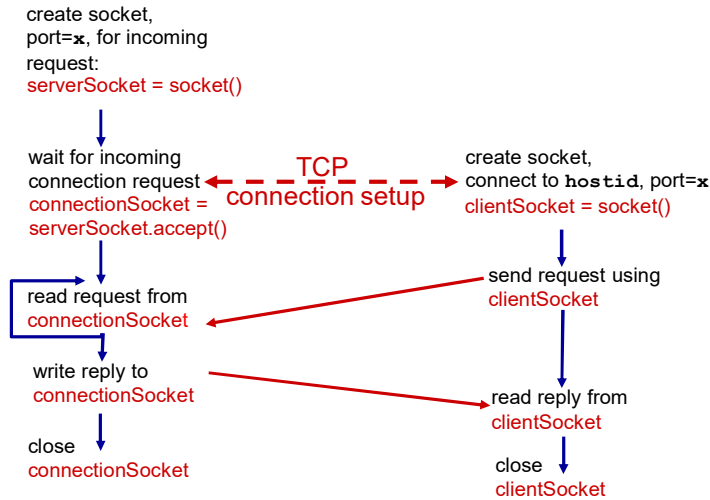
TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

Application Layer 2-100

Client/server socket interaction: TCP

server (running on `hostid`)

client



Application Layer 2-101

Example app:TCP client

Python TCPClient

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))
sentence = raw_input('Input lowercase sentence:')
clientSocket.send(sentence)
modifiedSentence = clientSocket.recv(1024)
print 'From Server:', modifiedSentence
clientSocket.close()
```

create TCP socket for server, remote port 12000 → `clientSocket = socket(AF_INET, SOCK_STREAM)`

No need to attach server name, port → `clientSocket.connect((serverName, serverPort))`

Application Layer 2-102

Example app: TCP server

Python TCP Server

create TCP welcoming socket	→	<pre>from socket import * serverPort = 12000 serverSocket = socket(AF_INET, SOCK_STREAM) serverSocket.bind(('', serverPort)) serverSocket.listen(1) print 'The server is ready to receive'</pre>
server begins listening for incoming TCP requests	→	<pre>while 1:</pre>
loop forever	→	<pre> connectionSocket, addr = serverSocket.accept()</pre>
server waits on accept() for incoming requests, new socket created on return	→	<pre> sentence = connectionSocket.recv(1024) capitalizedSentence = sentence.upper()</pre>
read bytes from socket (but not address as in UDP)	→	<pre> connectionSocket.send(capitalizedSentence) connectionSocket.close()</pre>
close connection to this client (but <i>not</i> welcoming socket)	→	