

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

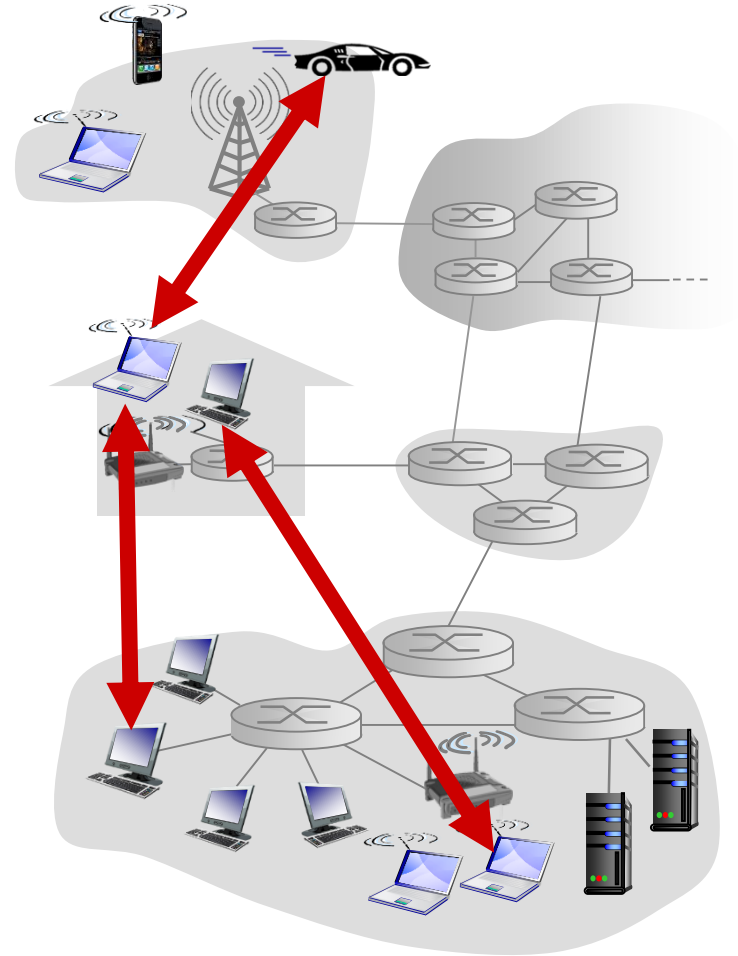
2.7 socket programming with UDP and TCP

Pure P2P architecture

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

examples:

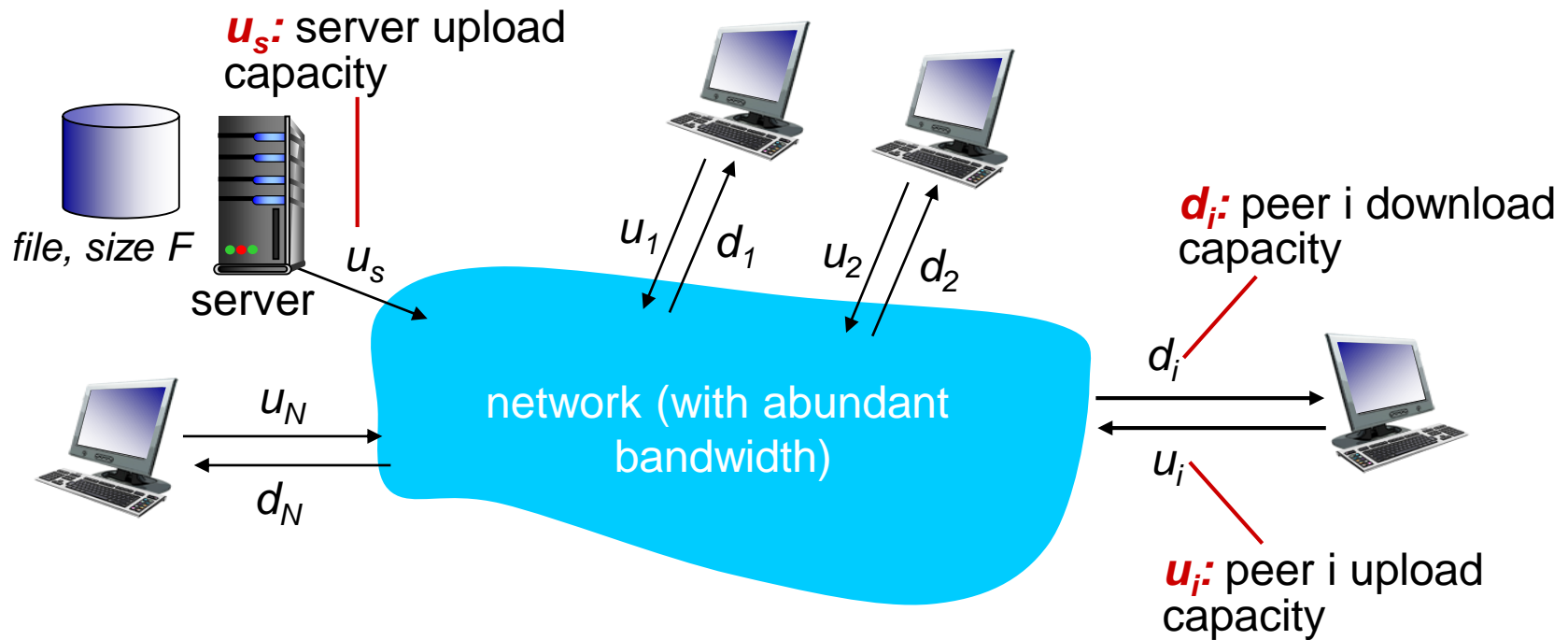
- file distribution (BitTorrent)
- Streaming (KanKan)
- VoIP (Skype)



File distribution: client-server vs P2P

Question: how much time to distribute file (size F) from one server to N peers?

- peer upload/download capacity is limited resource



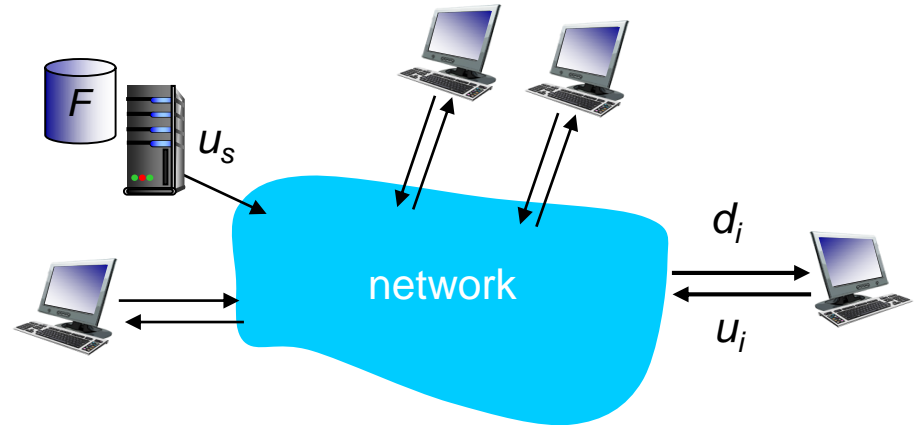
File distribution time: client-server

❖ **server transmission:** must sequentially send (upload) N file copies:

- time to send one copy: F/u_s
- time to send N copies: NF/u_s

❖ **client:** each client must download file copy

- d_{\min} = min client download rate
- min client download time: F/d_{\min}



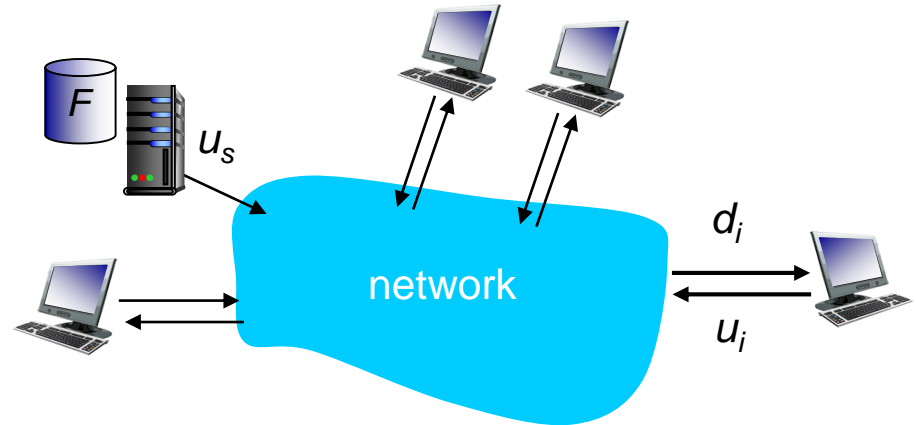
*time to distribute F
to N clients using
client-server approach*

$$D_{cs} \geq \max\{NF/u_s, F/d_{\min}\}$$

increases linearly in N

File distribution time: P2P

- ❖ **server transmission:** must upload at least one copy
 - time to send one copy: F/u_s
- ❖ **client:** each client must download file copy
 - min client download time: F/d_{\min}
- ❖ **clients:** as aggregate must download NF bits
 - max upload rate (limiting max download rate) is $u_s + \sum u_i$



*time to distribute F
to N clients using
P2P approach*

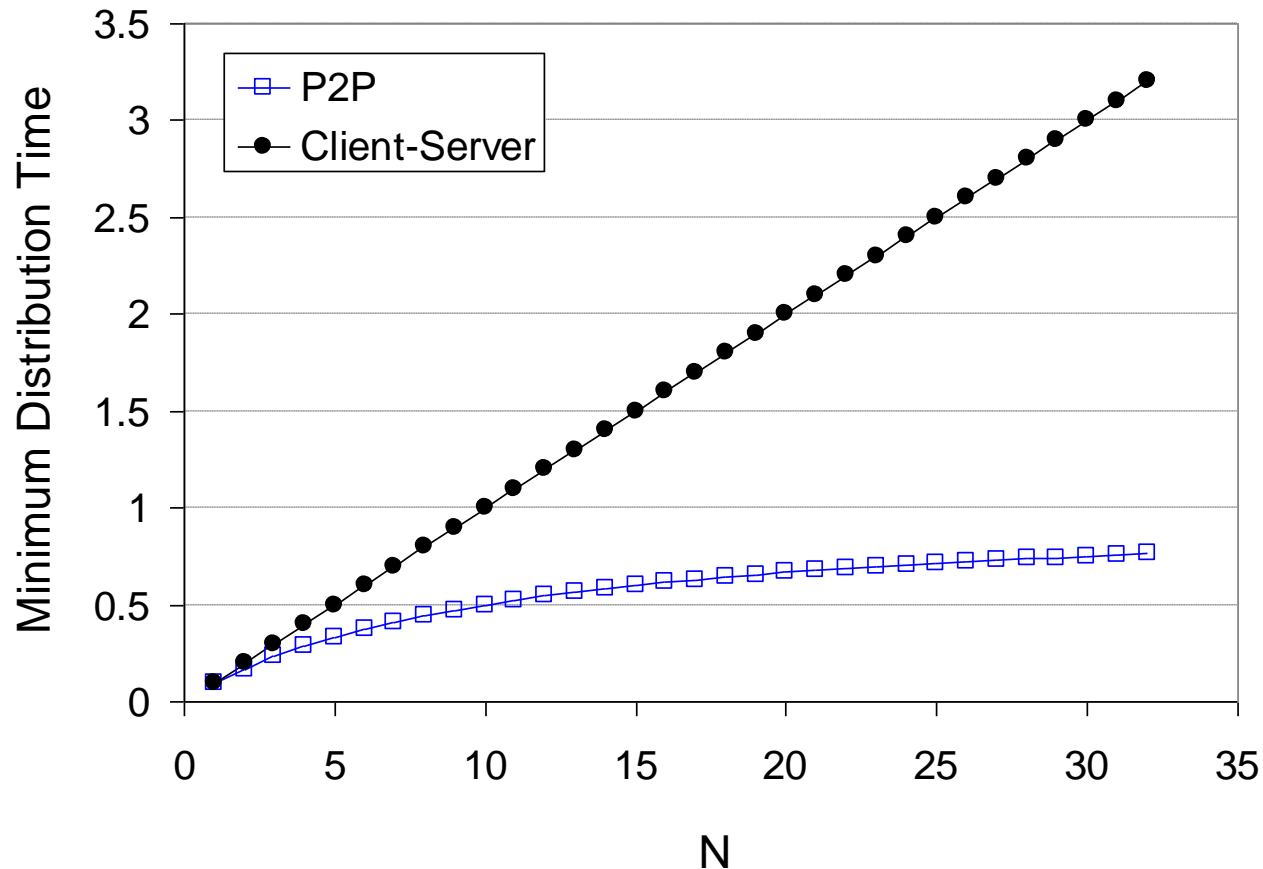
$$D_{P2P} \geq \max\{F/u_s, F/d_{\min}, NF/(u_s + \sum u_i)\}$$

increases linearly in N ...

... but so does this, as each peer brings service capacity

Client-server vs. P2P: example

client upload rate = u , $F/u = 1$ hour, $u_s = 10u$, $d_{min} \geq u_s$

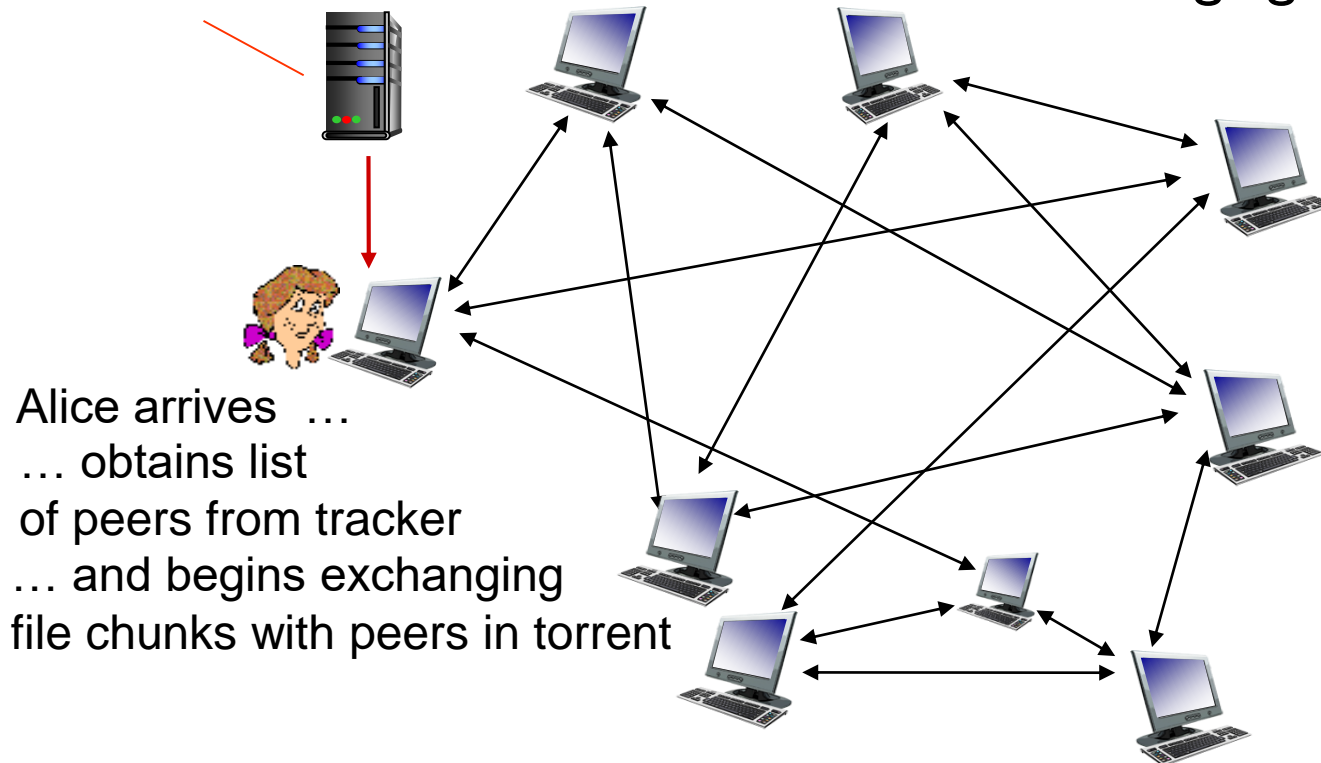


P2P file distribution: BitTorrent

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

tracker: tracks peers participating in torrent

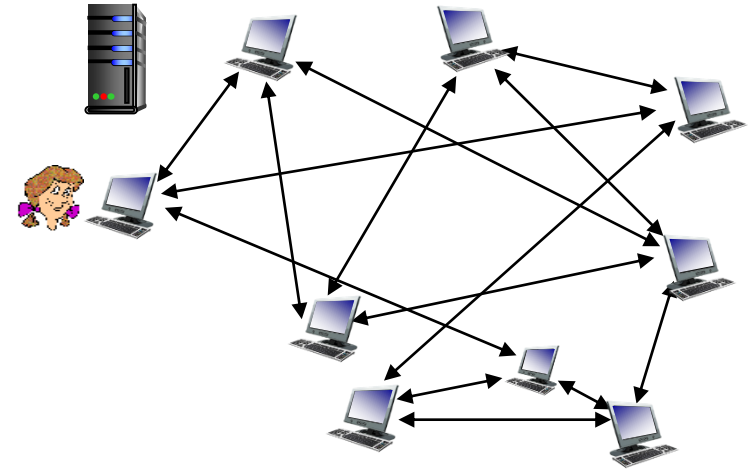
torrent: group of peers exchanging chunks of a file



Alice arrives ...
... obtains list
of peers from tracker
... and begins exchanging
file chunks with peers in torrent

P2P file distribution: BitTorrent

- ❖ peer joining torrent:
 - has no chunks, but will accumulate them over time from other peers
 - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- ❖ while downloading, peer uploads chunks to other peers
- ❖ peer may change peers with whom it exchanges chunks
- ❖ **churn**: peers may come and go
- ❖ once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent



BitTorrent: requesting, sending file chunks

requesting chunks:

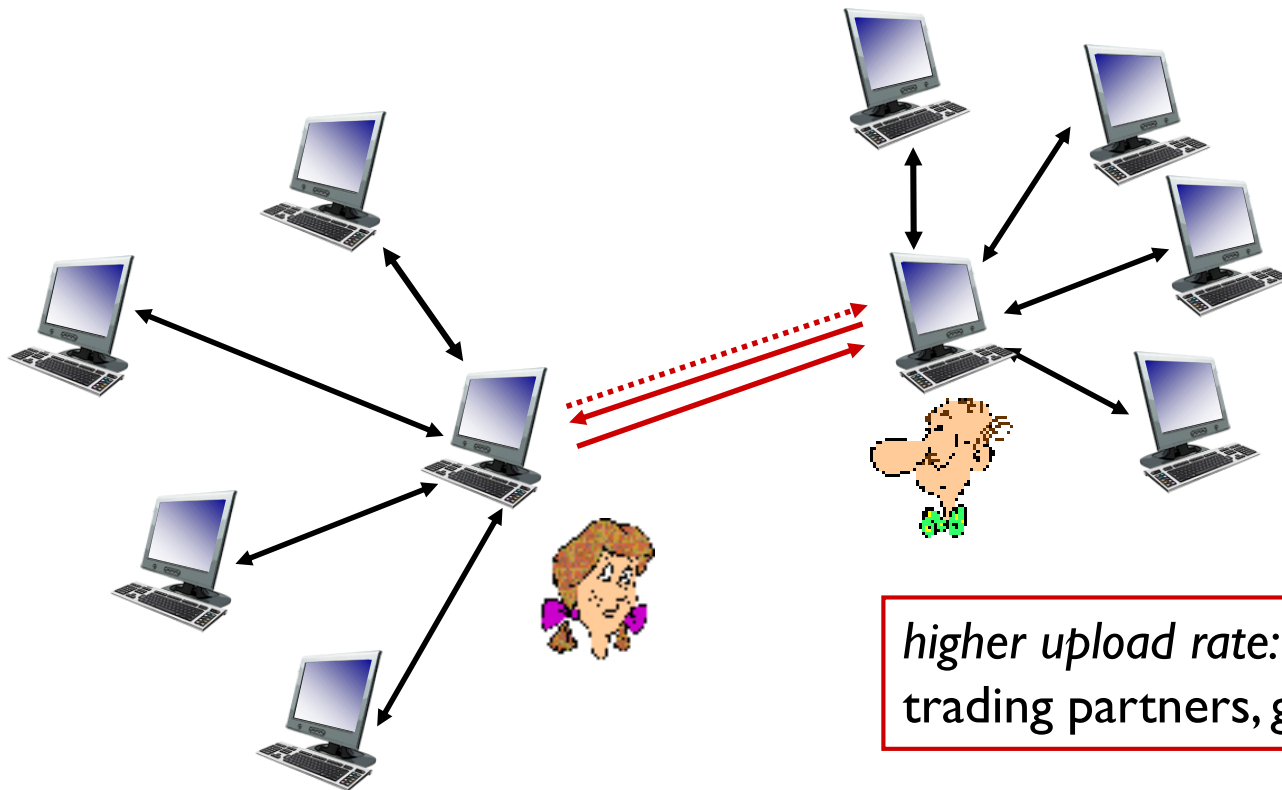
- ❖ at any given time, different peers have different subsets of file chunks
- ❖ periodically, Alice asks each peer for list of chunks that they have
- ❖ Alice requests missing chunks from peers, rarest first

sending chunks: tit-for-tat

- ❖ Alice sends chunks to those four peers currently sending her chunks *at highest rate*
 - other peers are choked by Alice (do not receive chunks from her)
 - re-evaluate top 4 every 10 secs
- ❖ every 30 secs: randomly select another peer, starts sending chunks
 - “optimistically unchoke” this peer
 - newly chosen peer may join top 4

BitTorrent: tit-for-tat

- (1) Alice "optimistically unchokes" Bob
- (2) Alice becomes one of Bob's top-four providers; Bob reciprocates
- (3) Bob becomes one of Alice's top-four providers



Distributed Hash Table (DHT)

- ❖ Hash table
- ❖ DHT paradigm
- ❖ Circular DHT and overlay networks
- ❖ Peer churn

Simple Database

Simple database with (key, value) pairs:

- key: human name; value: social security #

Key	Value
John Washington	132-54-3570
Diana Louise Jones	761-55-3791
Xiaoming Liu	385-41-0902
Rakesh Gopal	441-89-1956
Linda Cohen	217-66-5609
.....
Lisa Kobayashi	177-23-0199

- key: movie title; value: IP address

Hash Table

- More convenient to store and search on numerical representation of key
- $\text{key} = \text{hash}(\text{original key})$

Original Key	Key	Value
John Washington	8962458	132-54-3570
Diana Louise Jones	7800356	761-55-3791
Xiaoming Liu	1567109	385-41-0902
Rakesh Gopal	2360012	441-89-1956
Linda Cohen	5430938	217-66-5609
.....	
Lisa Kobayashi	9290124	177-23-0199

Distributed Hash Table (DHT)

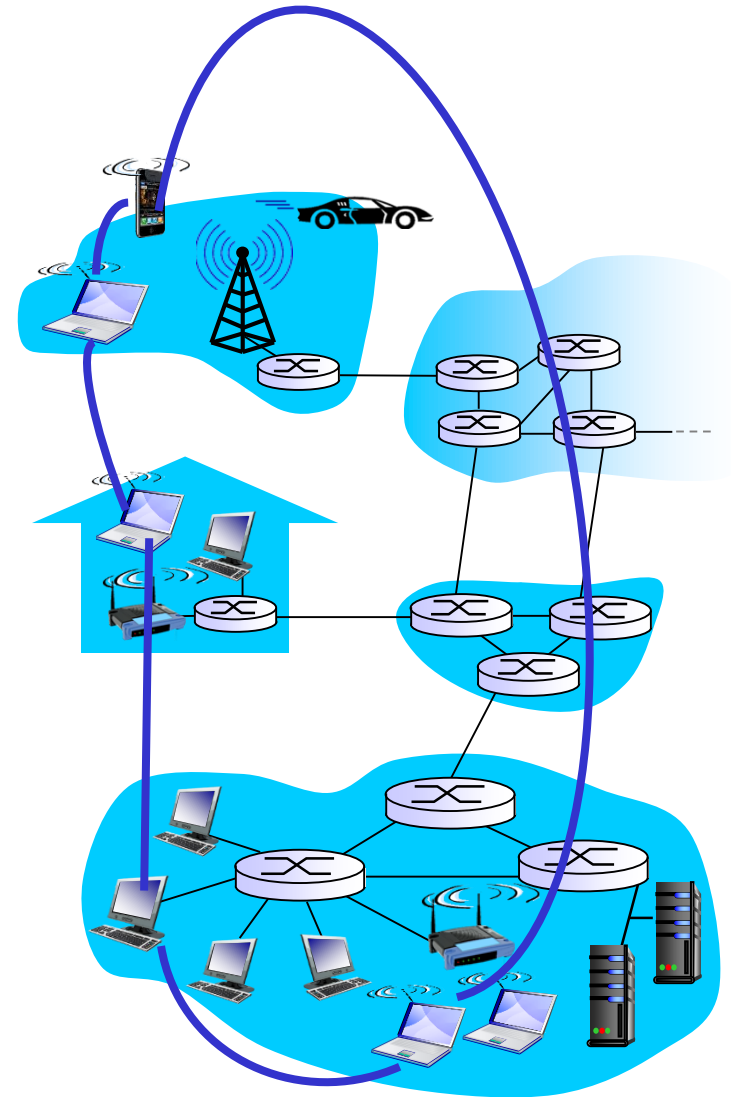
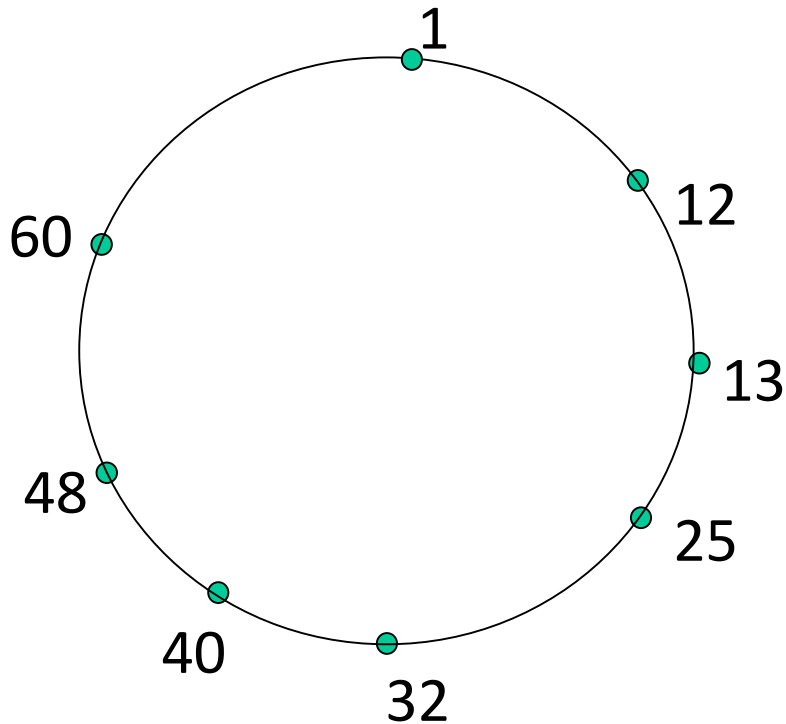
- ❖ Distribute (key, value) pairs over millions of peers
 - pairs are evenly distributed over peers
- ❖ Any peer can **query** database with a key
 - database returns value for the key
 - To resolve query, small number of messages exchanged among peers
- ❖ Each peer only knows about a small number of other peers
- ❖ Robust to peers coming and going (churn)

Assign key-value pairs to peers

- ❖ rule: assign key-value pair to the peer that has the *closest* ID.
- ❖ convention: closest is the *immediate successor* of the key.
- ❖ e.g., ID space $\{0, 1, 2, 3, \dots, 63\}$
- ❖ suppose 8 peers: 1, 12, 13, 25, 32, 40, 48, 60
 - If key = 51, then assigned to peer 60
 - If key = 60, then assigned to peer 60
 - If key = 61, then assigned to peer 1

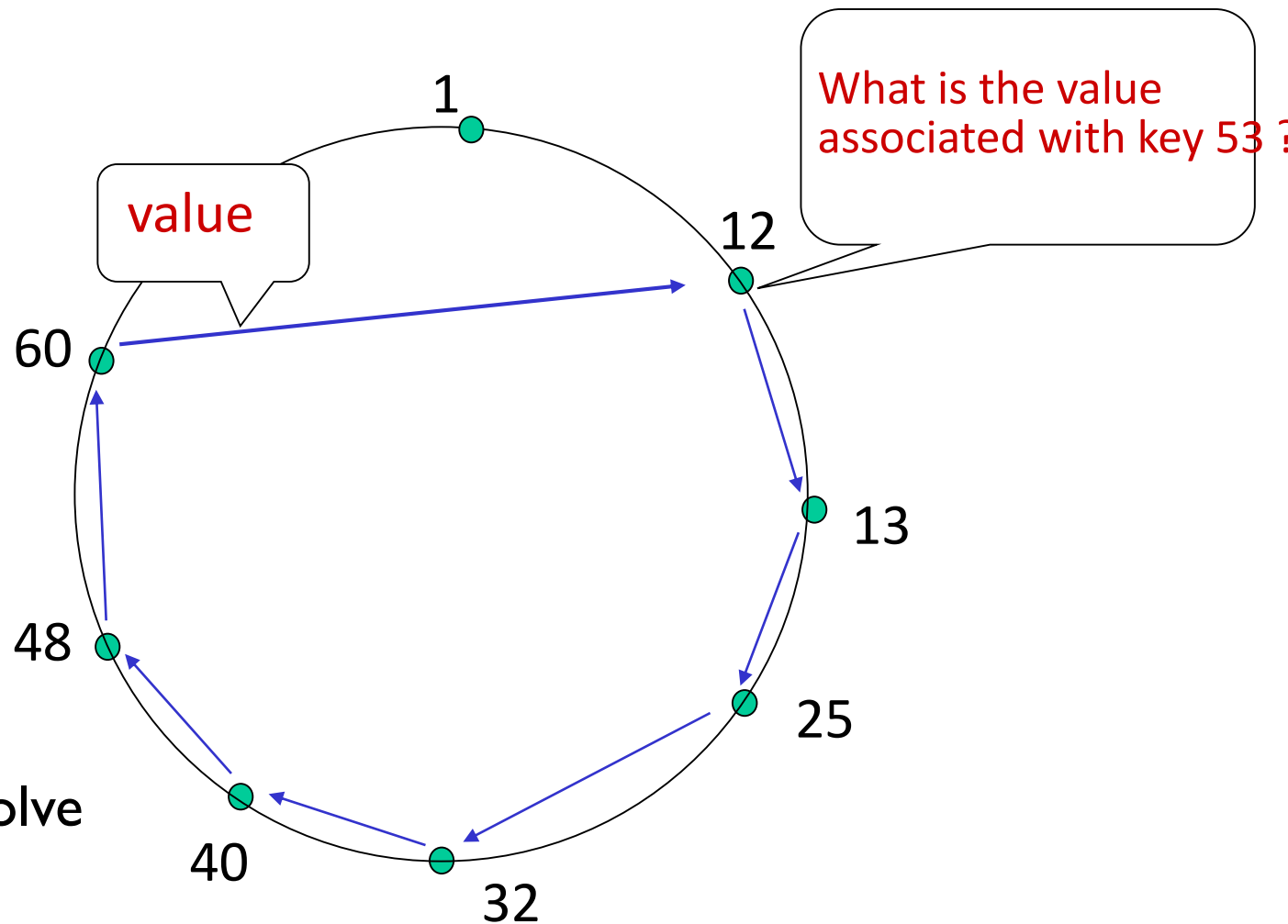
Circular DHT

- each peer *only* aware of immediate successor and predecessor.



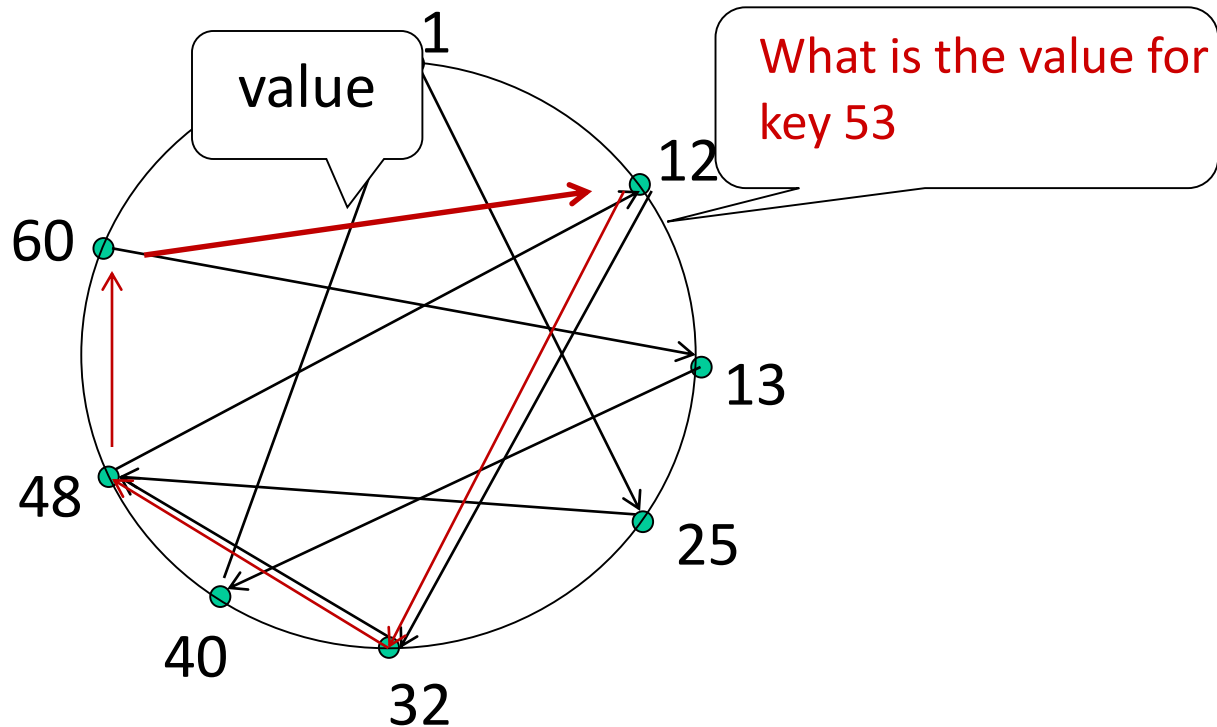
"overlay network"

Resolving a query



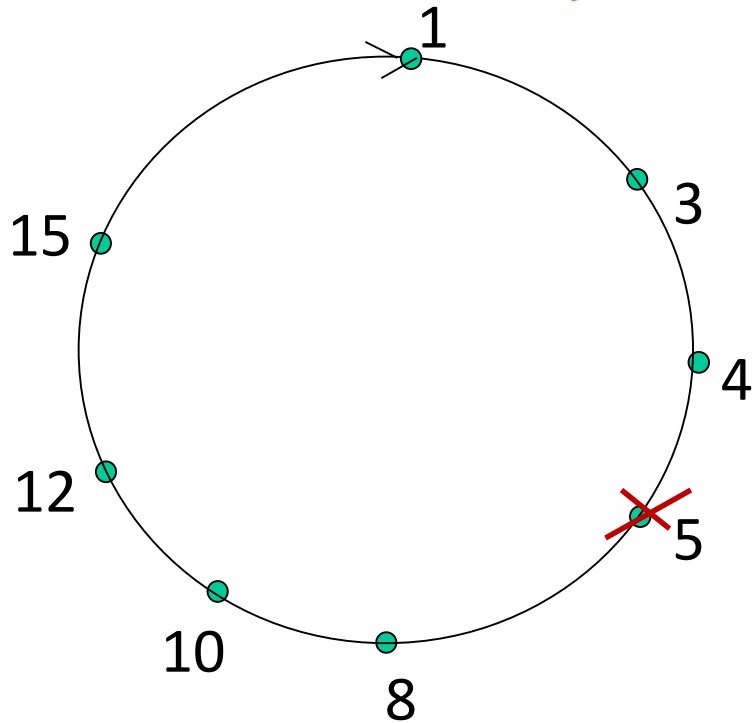
$O(N)$ messages
on average to resolve
query, when there
are N peers

Circular DHT with shortcuts



- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 3 messages.
- possible to design shortcuts with $O(\log N)$ neighbors, $O(\log N)$ messages in query

Peer churn

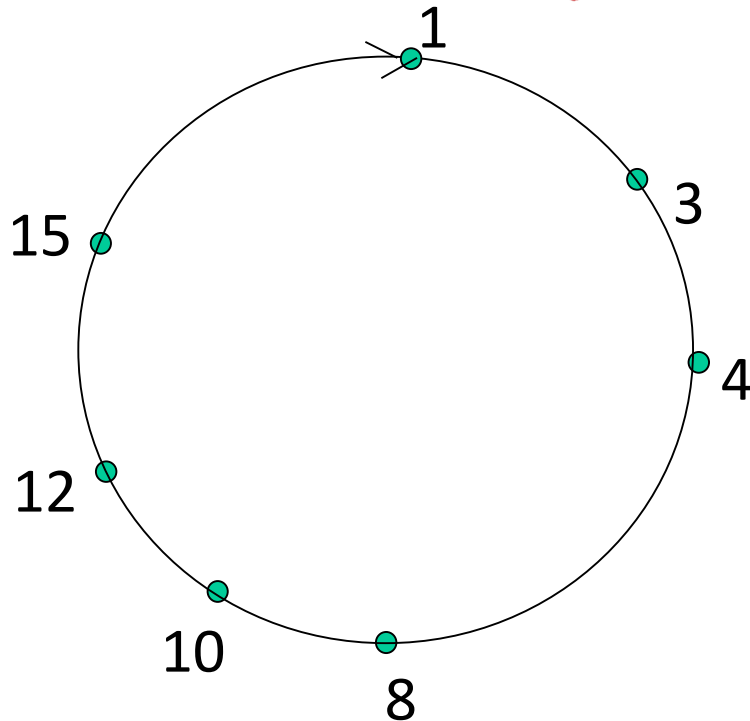


example: peer 5 abruptly leaves

handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

Peer churn



handling peer churn:

- ❖ peers may come and go (churn)
- ❖ each peer knows address of its two successors
- ❖ each peer periodically pings its two successors to check aliveness
- ❖ if immediate successor leaves, choose next successor as new immediate successor

example: peer 5 abruptly leaves

- ❖ peer 4 detects peer 5's departure; makes 8 its immediate successor
- ❖ 4 asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

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

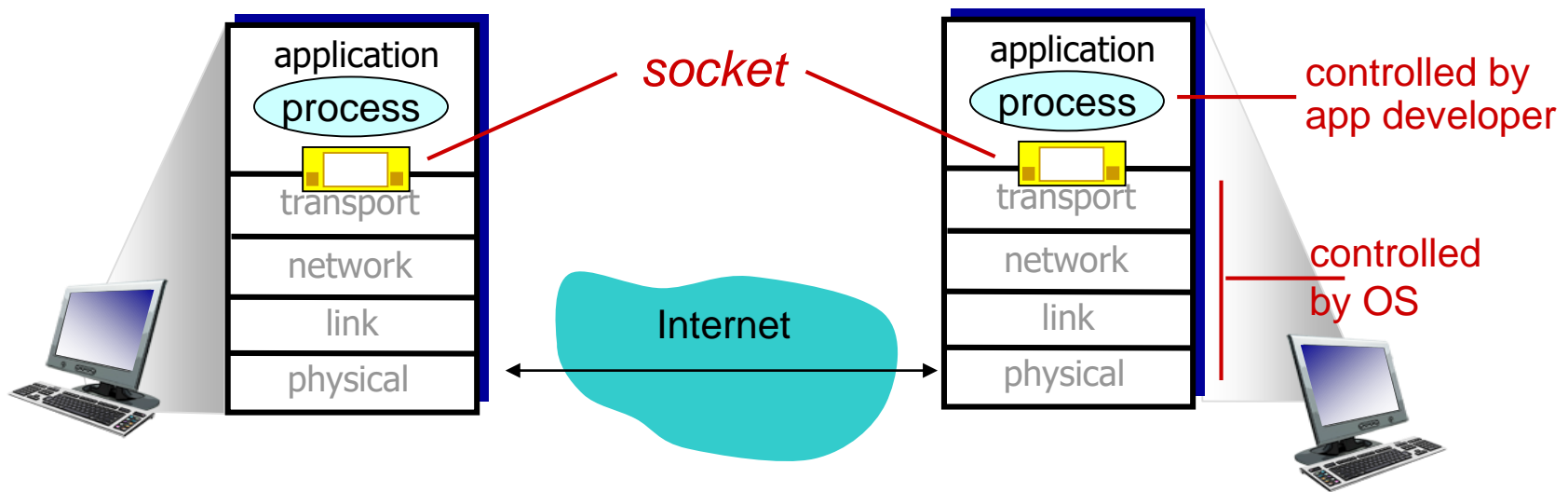
2.6 P2P applications

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.

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

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`

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()
```

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)`

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)

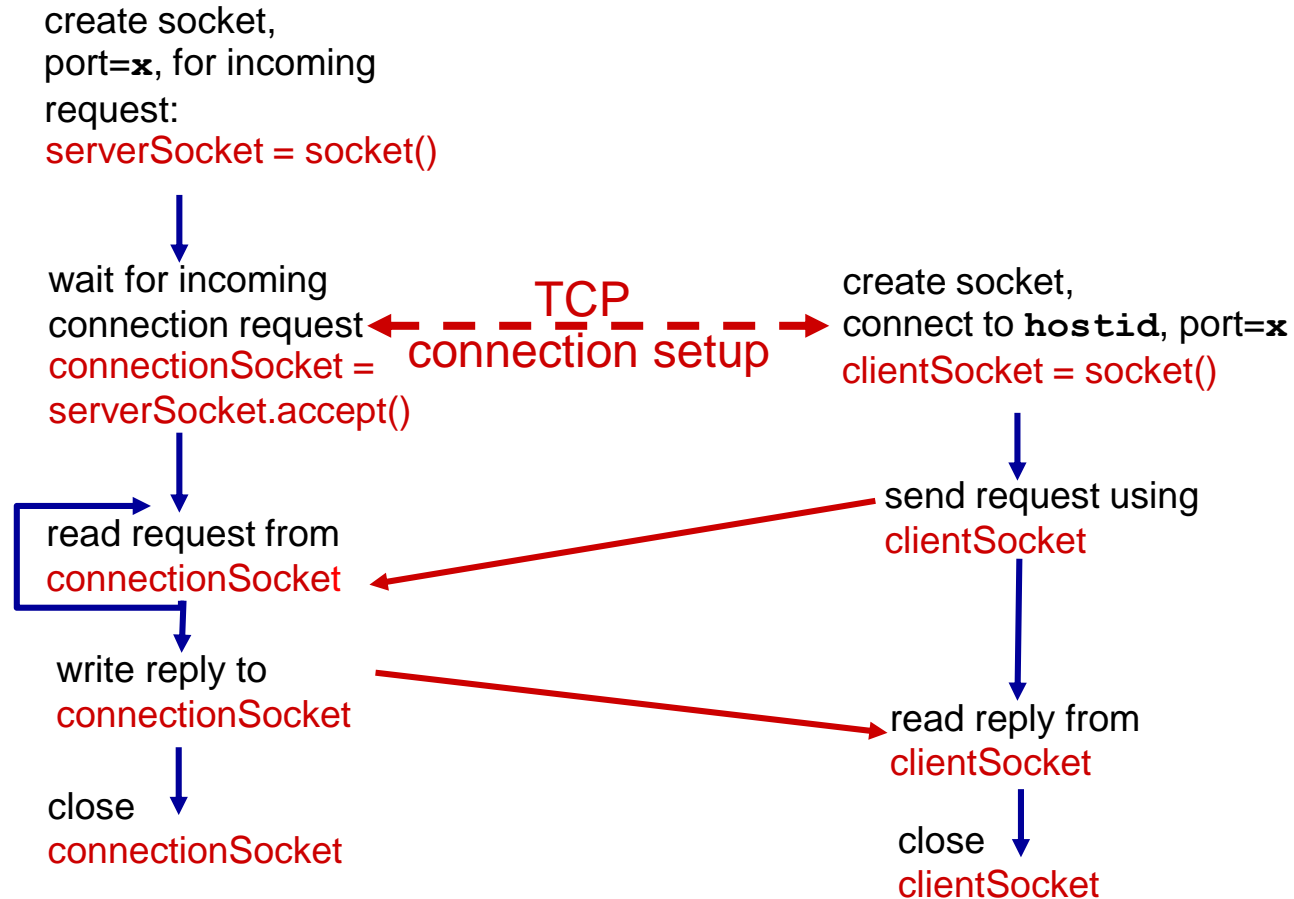
application viewpoint:

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

Client/server socket interaction: TCP

server (running on `hostid`)

client



Example app:TCP client

Python TCPClient

```
from socket import *
```

```
serverName = 'servername'
```

```
serverPort = 12000
```

create TCP socket for
server, remote port 12000

```
→ clientSocket = socket(AF_INET, SOCK_STREAM)
```

```
clientSocket.connect((serverName,serverPort))
```

```
sentence = raw_input('Input lowercase sentence:')
```

No need to attach server
name, port

```
→ clientSocket.send(sentence)
```

```
modifiedSentence = clientSocket.recv(1024)
```

```
print 'From Server:', modifiedSentence
```

```
clientSocket.close()
```

Example app: TCP server

Python TCPServer

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

Chapter 2: summary (part I)

our study of network apps now complete!

- ❖ application architectures
 - client-server
 - P2P
- ❖ application service requirements:
 - reliability, bandwidth, delay
- ❖ Internet transport service model
 - connection-oriented, reliable: TCP
 - unreliable, datagrams: UDP
- ❖ specific protocols:
 - HTTP
 - FTP
 - SMTP, POP, IMAP
 - DNS
 - P2P: BitTorrent, DHT
- ❖ socket programming: TCP, UDP sockets

Chapter 2: summary (part 2)

most importantly: learned about protocols!

- ❖ typical request/reply message exchange:
 - client requests info or service
 - server responds with data, status code
- ❖ message formats:
 - headers: fields giving info about data
 - data: info being communicated

important themes:

- ❖ control vs. data msgs
 - in-band, out-of-band
- ❖ centralized vs. decentralized
- ❖ stateless vs. stateful
- ❖ reliable vs. unreliable msg transfer
- ❖ "complexity at network edge"