ELEC / COMP 177 – Fall 2011

Computer Networking
→ Application Layer: Peer to Peer

Some slides from Kurose and Ross, *Computer Networking*, 5th Edition

Schedule

- Today
  - P2P Systems (Application Layer)
- Next Tuesday
  - Network socket programming in C
  - **Homework #3 due**
  - **Programming Project #1 assigned**
- Next Thursday
  - Finish socket programming discussion

Homework #3

- **Questions about project requirements?**
- **Questions about tools?**
- Status
  - **Connected to the class server from home?**
    - **How bad is the lag?**
  - **Written / compiled some code?**

Replicate Class Server

- Install Ubuntu 11.04 Desktop
  - Dual boot or inside virtual machine
- Install packages:
  - `sudo apt-get install build-essential eclipse-cdt`
- Done!

- *Talk to me if you have questions about this*

SSH Client Options

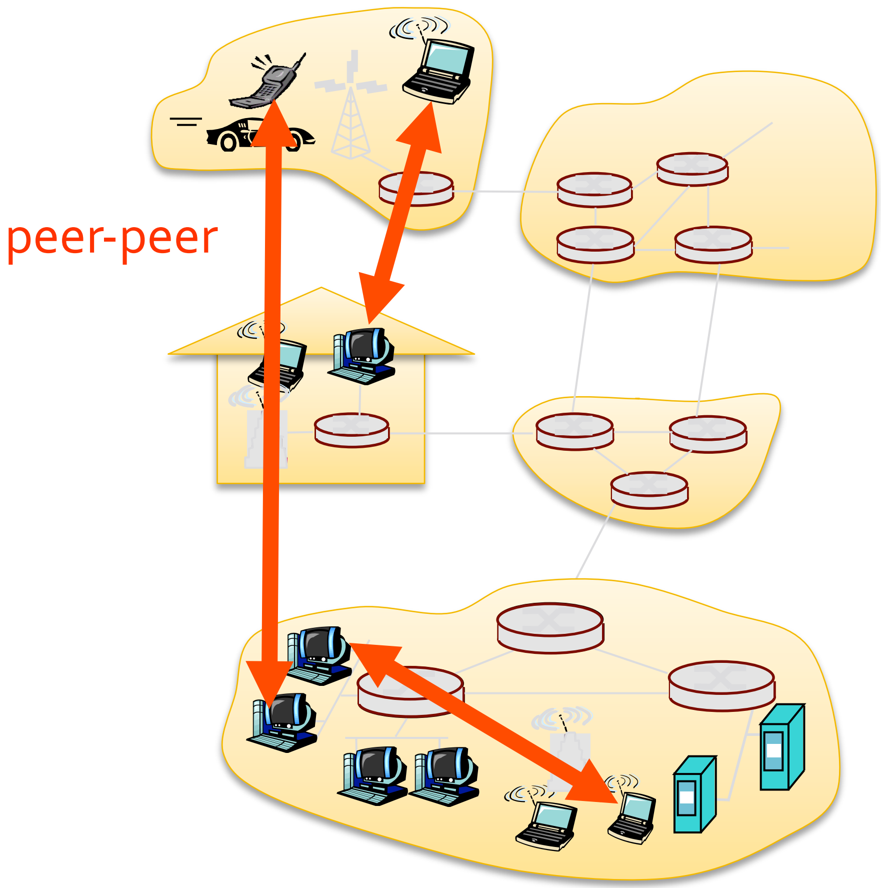
- **PuTTY**
  - Check the following boxes:
    - Connection→SSH→Enable Compression
    - Connection→SSH→X11→Enable X11 forwarding
- **Tectia SSH client**
  - Check the following boxes under Settings:
    - Tunnel→Tunnel X11 Connections
  - Save afterwards!
- **Command-line SSH**
  - `ssh ecs-network.serv.pacific.edu -l username -X -C`
  - `-l` (lowercase L) = specify login name
  - `-X` = tunnel X11
  - `-C` = enable compression

Applications: Peer to Peer (P2P)

Pure P2P architecture

- *No* always-on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change IP addresses

- **Today's Topics:**
  - File distribution (BitTorrent)
  - Telecom (Skype)
  - Searching for information (DHT)



Motivations for P2P

GREATER RESOURCES

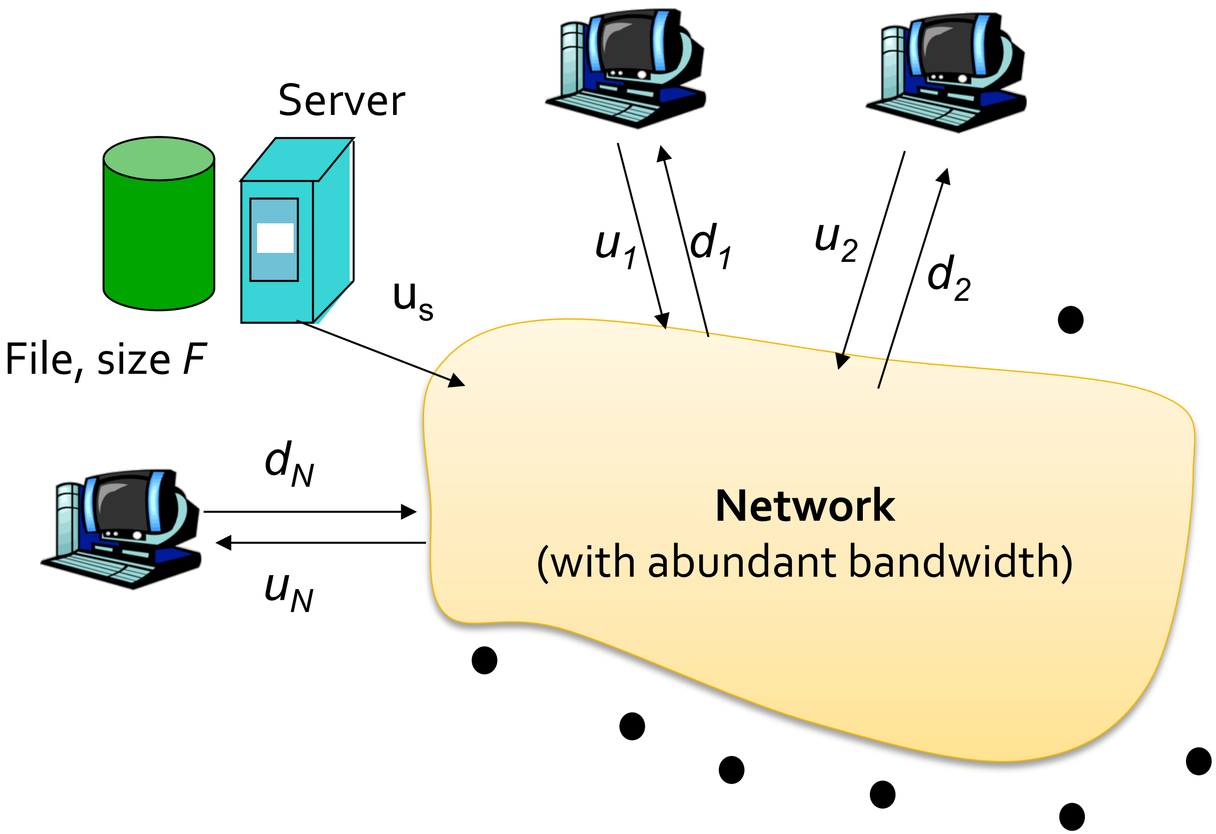
- Typically client-server relationship has many clients and 1 server
  - Server can be overwhelmed!
  - As more clients join, each gets fewer resources
- Idea: Use the client's network bandwidth / CPU / disk to assist
  - As more clients join, more resources are available

GREATER RELIABILITY

- A single server can be a reliability problem
  - What if it crashes?
  - What if both of my servers crash?
  - What if my entire datacenter (1000's of servers) loses power?)
- Idea: Use clients all over the world to ensure resources are always accessible (somewhere)

File Distribution: Server-Client vs P2P

<u>*Question*</u> : How much time to distribute file from one server to *N* *peers*?



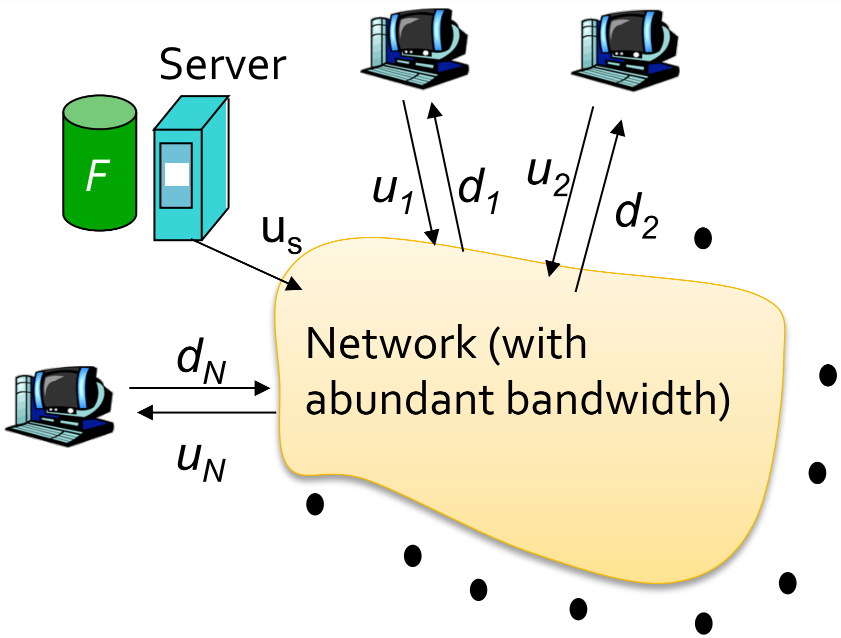
u_s: server upload bandwidth

u_i: peer i upload bandwidth

d_i: peer i download bandwidth

File Distribution Time: Client/Server

- Server sequentially sends N copies:
  - NF/u_s seconds
- Client i takes F/d_i time to download
  - Slowest client takes F/d_{min} time

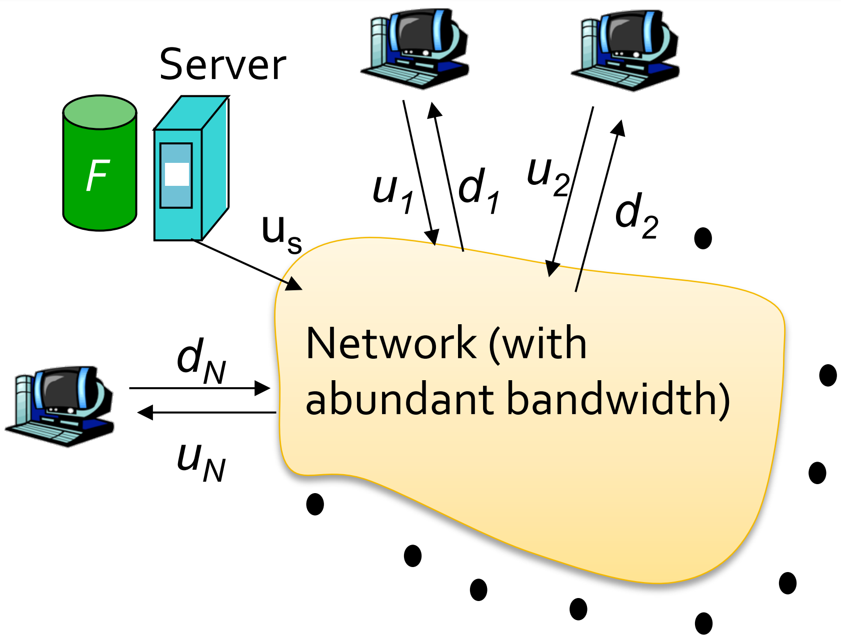


Lower bound time to distribute F to N clients using client/server approach $= d_{cs} = \max \{ NF/u_s, F/d_{min}) \}$

As N increases, the server upload time dominates! *(Server is the bottleneck…)*

File Distribution Time: P2P

- NF bits must be downloaded (aggregate)
- Server must send at least one copy: F/u_s time
- Client i takes F/d_i time to download
  - Slowest client takes F/d_{min} time
- Total aggregate upload rate: $u_s + \sum u_i$

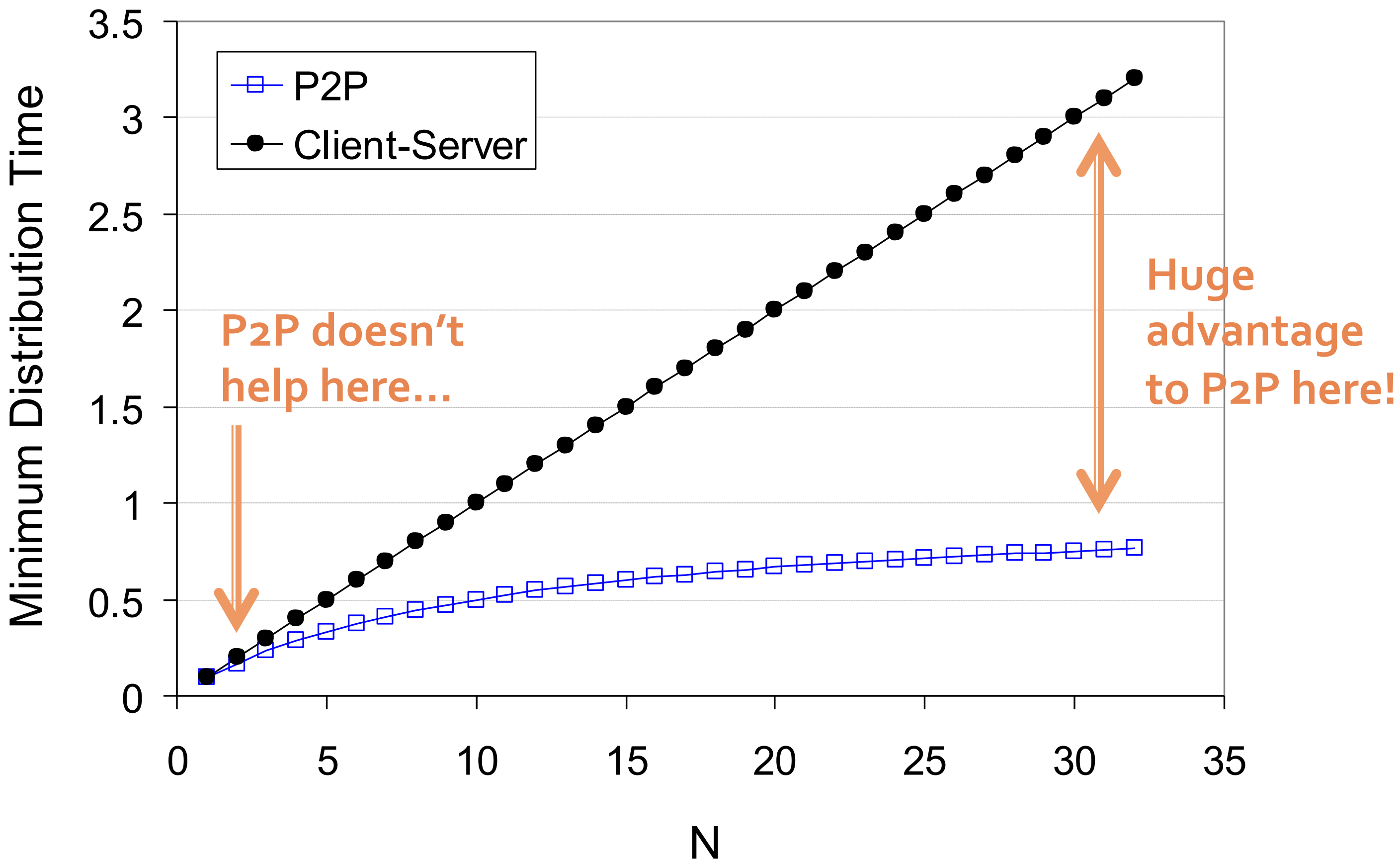


Lower bound time to distribute F to N clients using P2P approach

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

Server-Client vs. P2P: Example

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



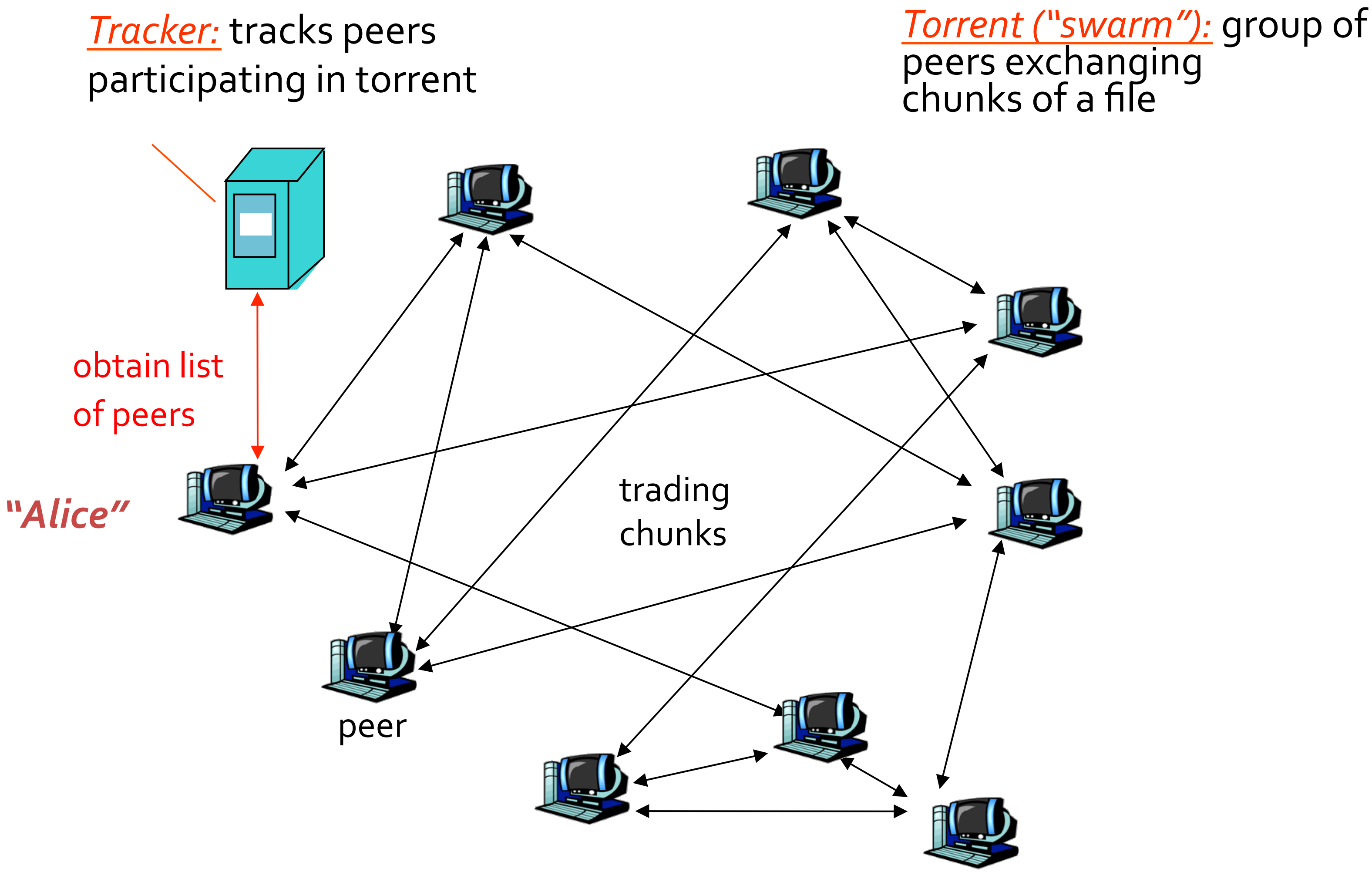
File Distribution (BitTorrent)

BitTorrent History

- Protocol (and first clients) released in 2002
- Key Motivation
  - Popularity (of a resource) exhibits temporal locality (flash crowds)
  - Examples: Slashdot effect, CNN on 9/11, new movie/game release
- Focused on Efficient **Fetching**, not **Searching**
  - Single publisher, multiple downloaders

File Distribution: BitTorrent

P2P file distribution



BitTorrent

- File divided into small **chunks** (64kB-1MB)
- Peer joining torrent:
  - Registers with tracker to get list of peers, connects to subset of peers ("neighbors")
  - Has no chunks initially, but will accumulate them over time
- While downloading, peer uploads chunks to other peers.
- Peers may come and go
  - Once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent – Downloading Chunks

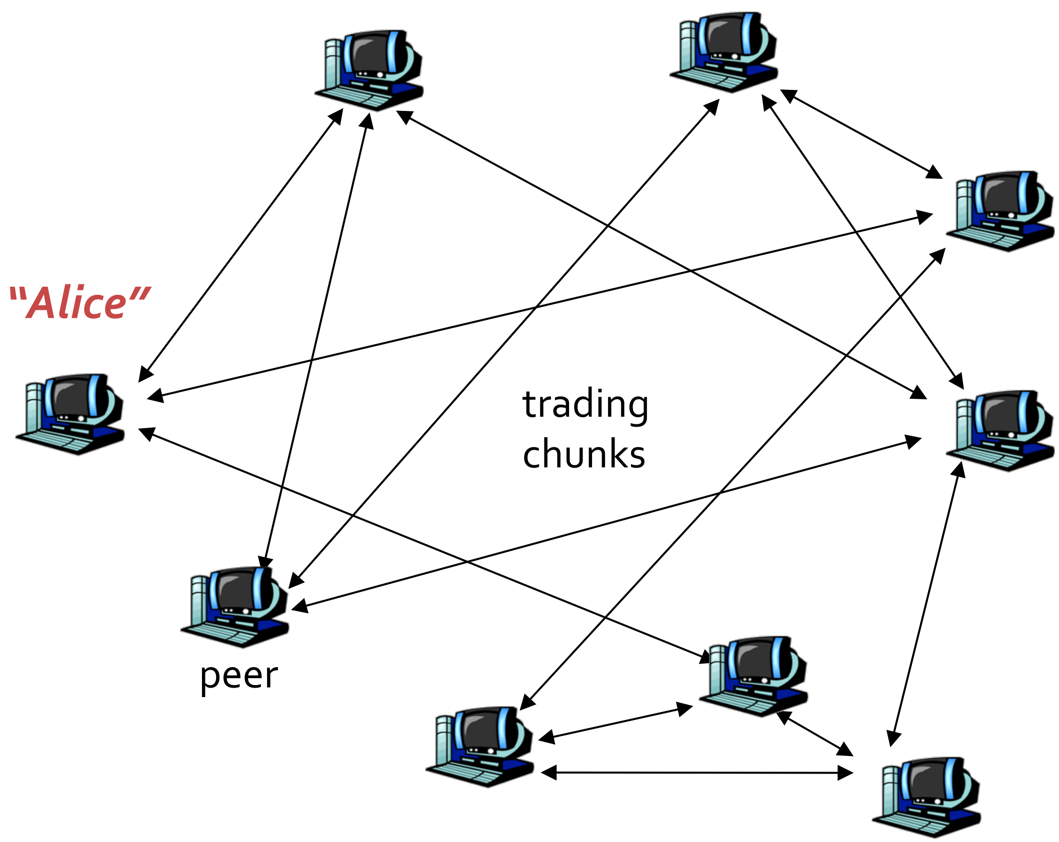
- At any given time, different peers have different subsets of file chunks
- Periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice then requests chunks that she is missing
  - Policy is "rarest first" – **Why?**

BitTorrent – Uploading Chunks

- Alice gets many requests, but has a finite upload bandwidth
  - **Which requests should she service?**
  - **How does she prevent free-riders?**
- Policy: **Tit-for-tat**
  - "I'll share with you if you share with me"
  - Alice sends chunks to 4 (or more) neighbors currently sending her chunks at the highest rate
  - Re-evaluate top 4 neighbors every 10 secs

BitTorrent - Performance

- Challenge: Alice has no idea about the performance of the overlay network
  - Are these links fiber optic, DSL, dial-up modems?
  - Are they to machines at Pacific, or across the world?
- Tit-for-tat helps!
  - Prefer neighbors with better performance

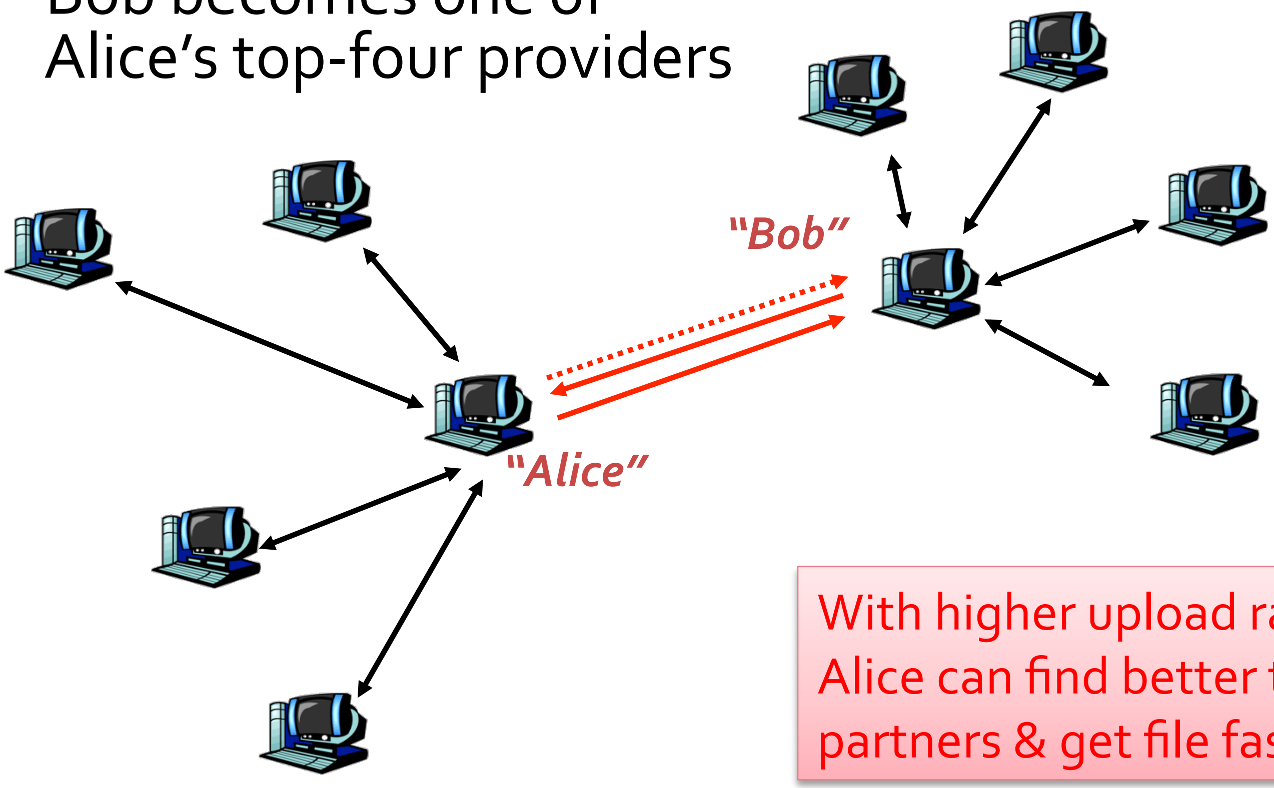


BitTorrent – Uploading Chunks

- **Will Alice ever decide to send chunks to a new neighbor that just joined and has nothing to share?**
  - **If not, what will that neighbor do?**
- Policy: **Optimistic Unchoking**
  - Every 30 seconds: Alice randomly selects another peer and starts sending the requested chunks
    - Newly chosen peer may join top 4
    - Good for Alice: maybe she will find an even better neighbor!
      - Better neighbor = faster upload speed (exchange chunks faster)
    - Good for new visitor: they finally get some data!

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

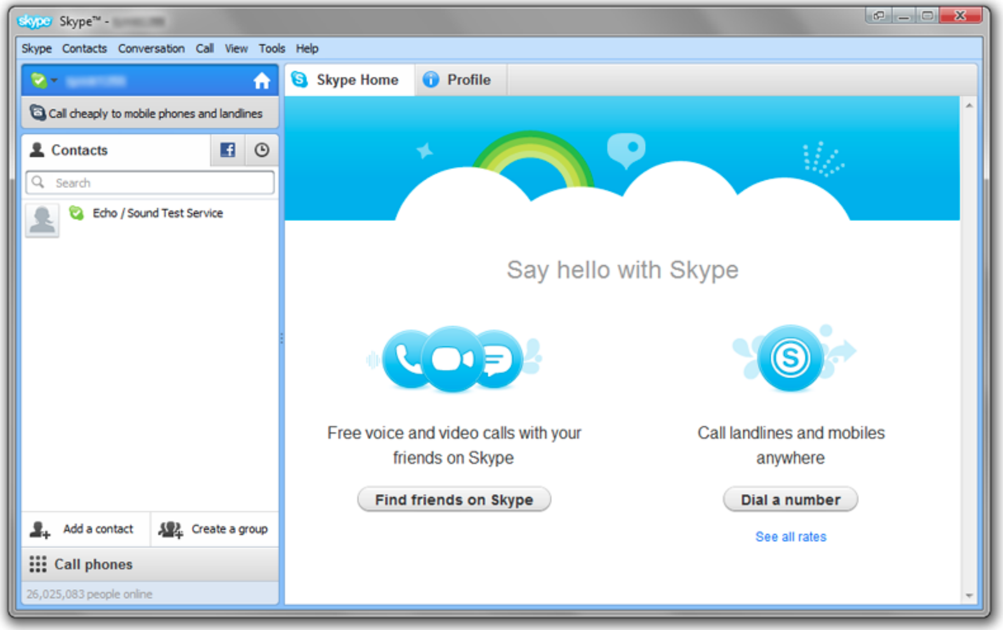


With higher upload rate, Alice can find better trading partners & get file faster!

Telecom (Skype)

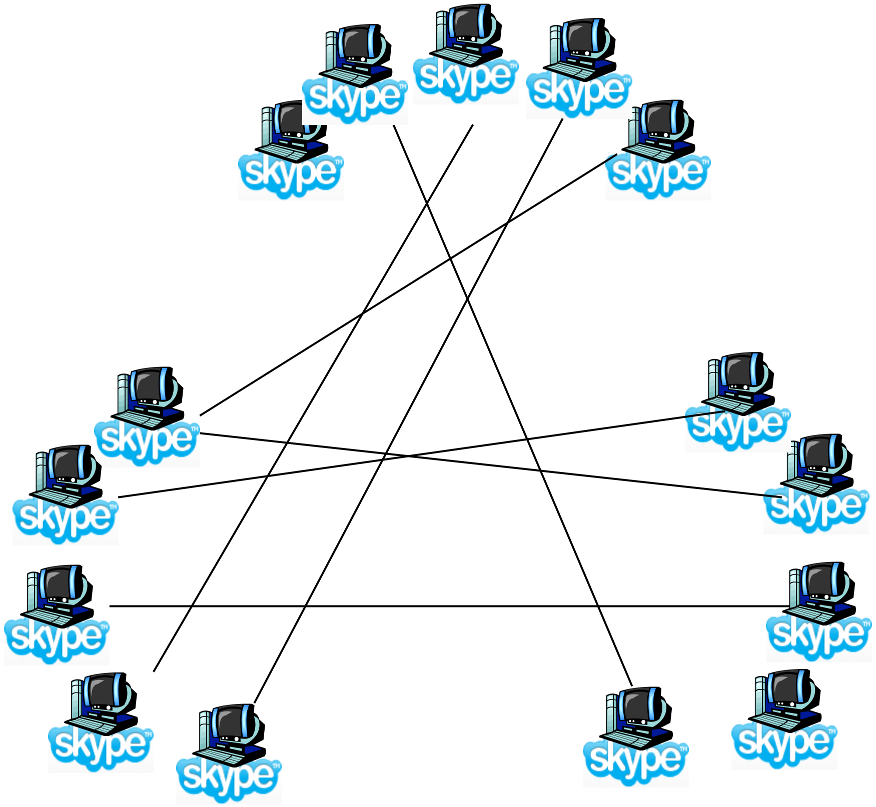
Skype

- Voice and video chat over the Internet
  - Plus file transfer, instant messaging, etc..
- Hybrid architecture
  - Client-server
  - P2P
  - Not publicly disclosed
    - Reverse engineering…



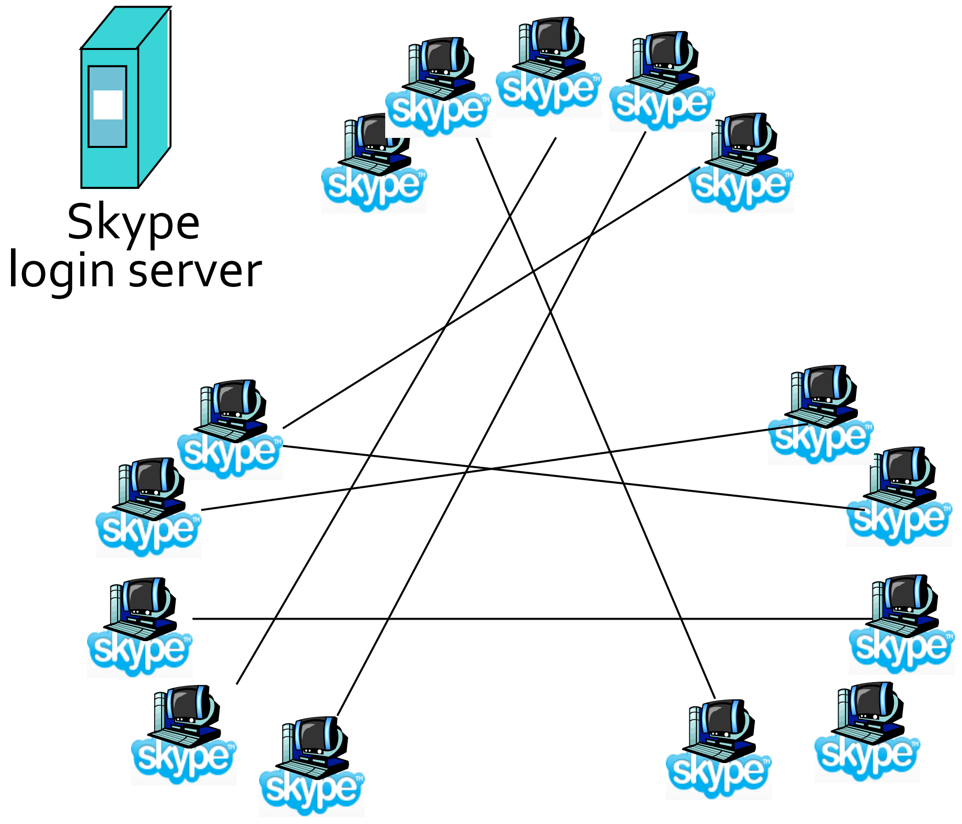
Skype – P2P

- Goal: Have pairs of users communicate with audio/video streams
  - Skype doesn't want to **pay** for the bandwidth/server costs to centralize this at their datacenter!
- Solution: P2P
  - Users directly communicate with each other
  - Data doesn't even go through a Skype computer!



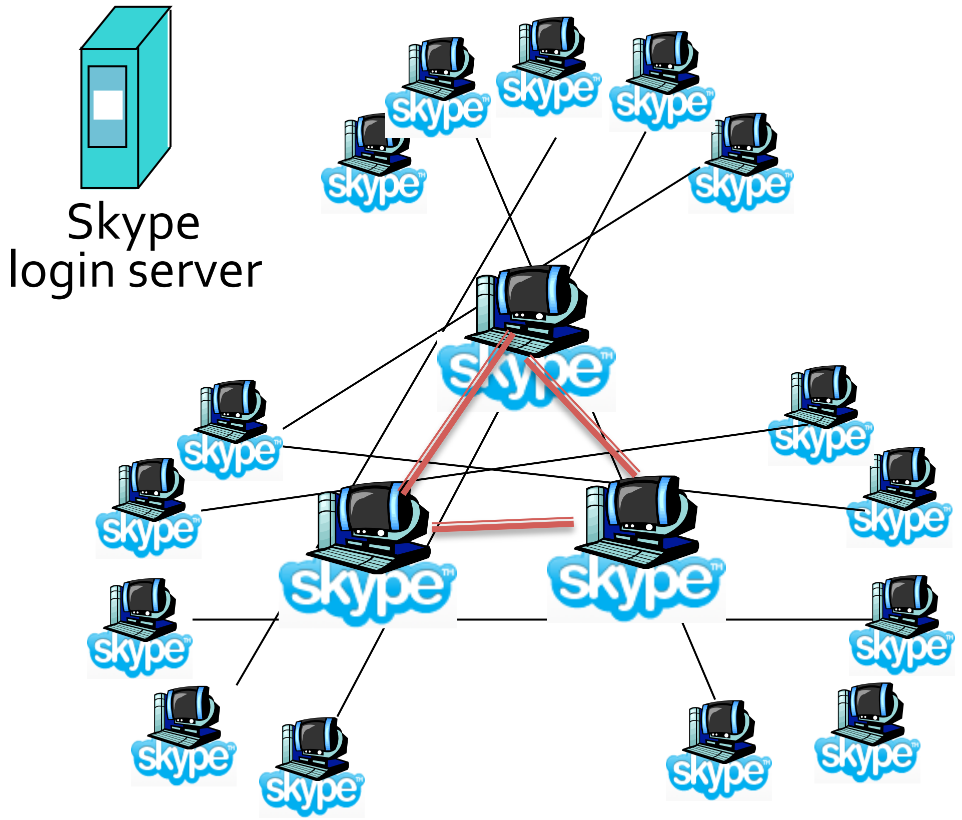
Skype – Client/Server

- **How does Skype enforce usernames/ passwords?**
  - And get $$ for premium services?
- Solution:
  - Client must contact a **Skype server** first
  - Only need a few login servers!
    - Traffic is minor compared to video/ audio streams



Skype – Client/Server

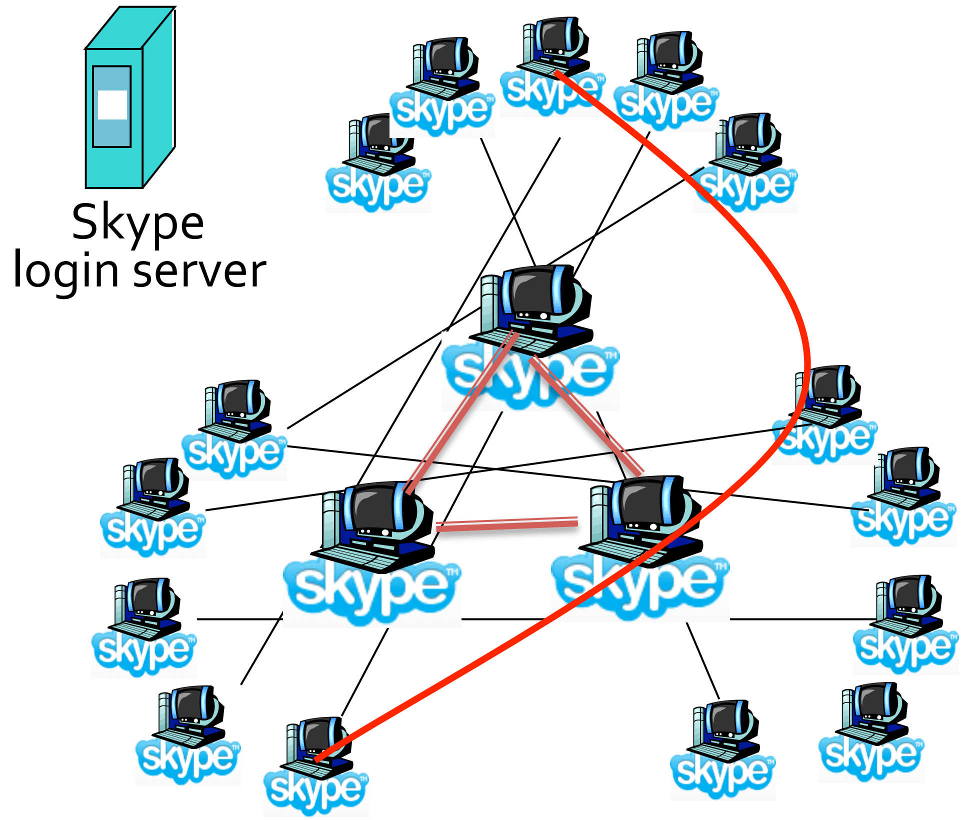
- **How does your client find the address of the peer you wish to communicate with?**
- Solution: A **distributed database** maps usernames to IP addresses
  - Stored on "**supernodes**"
  - Client gets supernode address from Skype server and queries DB
  - Implementation: DHT



Who gets to be a supernode?

Skype – P2P Relays

- **What if both parties are behind a firewall?**
  - Firewall prevents an outside peer from initiating a call to insider peer
- Solution – non-blocked peer serves as **relay!**
  - Supernodes help pick relay
  - Each peer initiates session with relay



Skype Outage – Dec 22nd 2011

- Access to Skype was intermittent (at best) for a 24-hour period – What happened?

- Problems demonstrates a weakness in Skype's design
  - Relying on customer machines to provide essential services for their network!

Skype Outage – Dec 22nd 2011

- **Trigger:** Some Skype-owned servers (managing instant messaging) became overloaded and slow
- **Result:** A bug in the Windows Skype client reacted poorly to this slowdown, and the client program crashed
  - Affected 25-30% of the supernodes
    - **What does the supernode do, again?**
    - **Where does a supernode run, again?**

http://blogs.skype.com/en/2010/12/cio_update.html

Skype Outage – Dec 22nd 2011

- This is a BIG problem!
  - Supernodes are essential to Skype functioning
  - 25-30% of them just crashed
  - The rest are overloaded by normal Skype traffic (and the added load of crashed supernodes attempting to restart and rejoin the network)
- Worse: Skype doesn't control these machines
  - They're the customer's machines!
  - They can't easily push out new software updates or re-run the client software
- **What can they do to fix this?**

Skype Recovery

- Amazon to the rescue
  - **What is cloud computing?**
  - **What is Amazon Web Services?**

Skype Recovery

- Skype pulls out their corporate credit card and rents servers in Amazons datacenter (~$1/hour/computer)
  - How many servers? ("hundreds", and after that proved insufficient, then "several thousand more")
  - Which datacenters? (Unknown, but why not all of them?)
    - Virginia, northern California, Ireland, Singapore, Tokyo
- Each Amazon EC2 server runs the Skype client and nothing else
  - Presto! Supernodes that Skype controls!
    - Called "mega-supernodes"
  - Temporarily restored Skype P2P capacity until regular customer clients could be patched/restarted

Skype Recovery

- Network restored in time for Christmas: Yay!
  - Coal in the stockings for the Skype engineers?
- Future improvements?
  - I'm sure the entire "*run our P2P network on Amazon servers*" **backup plan** has now been thoroughly tested and automated to deploy in <30 minutes, instead of 1+ day of panicked effort…

Distributed Database (DHT)

Distributed Database

- Goal: Store large amounts of data
  - on many peers
  - without a centralized server
  - in a scalable manner (millions of peers!)
  - in a reliable manner (where peers join and leave constantly)
- This is **different from DNS**
  - DNS is a distributed database
  - DNS is not ad-hoc/P2P – servers are maintained by ISP and assumed to be always running

Distributed Database

- Database has **(key, value)** pairs
  - Examples:
    - key: ss number; value: human name
    - key: username; value: IP address
  - Key might be 128 or 160 bits long
- Peers **query** DB with key
  - DB returns values that match the key
- Peers can also **insert** (key, value) peers

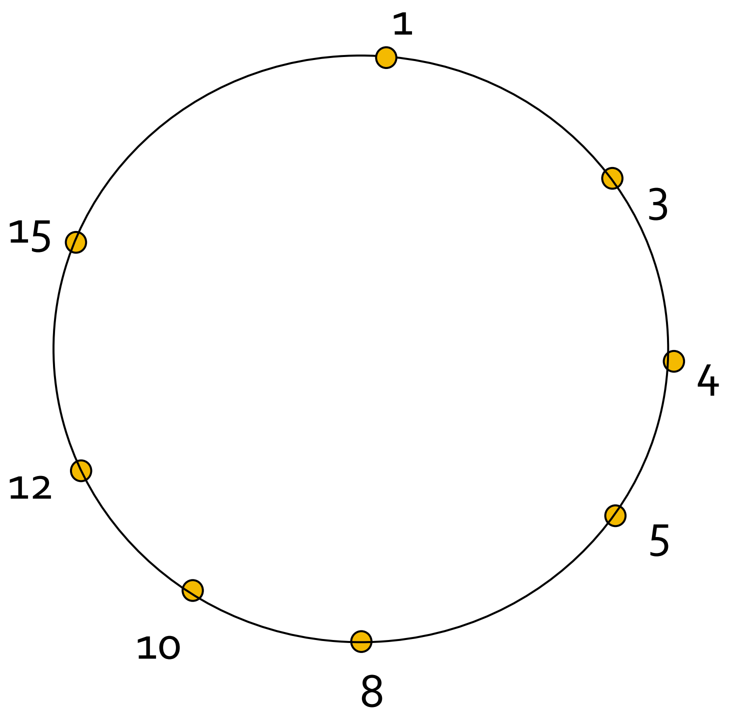
Distributed Hash Table (DHT)

- DHT = distributed P2P database
- Assign integer identifier to each peer in range $[0, 2^n - 1]$.
  - Each identifier can be represented by n bits.
- Require each key to be an integer in **same range**
- To get integer keys, hash original key.
  - eg, `key = hashFunction("Daily Show Episode 198")`
  - This is why they call it a distributed "hash" table
  - **Is a hash guaranteed to be unique?**

How to Assign Keys to Peers?

- Central issue:
  - Assigning (key, value) pairs to peers
- Rule: assign key to the peer that has the **closest** ID
- Convention in lecture: closest is the **immediate successor** of the key
- Exampe: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

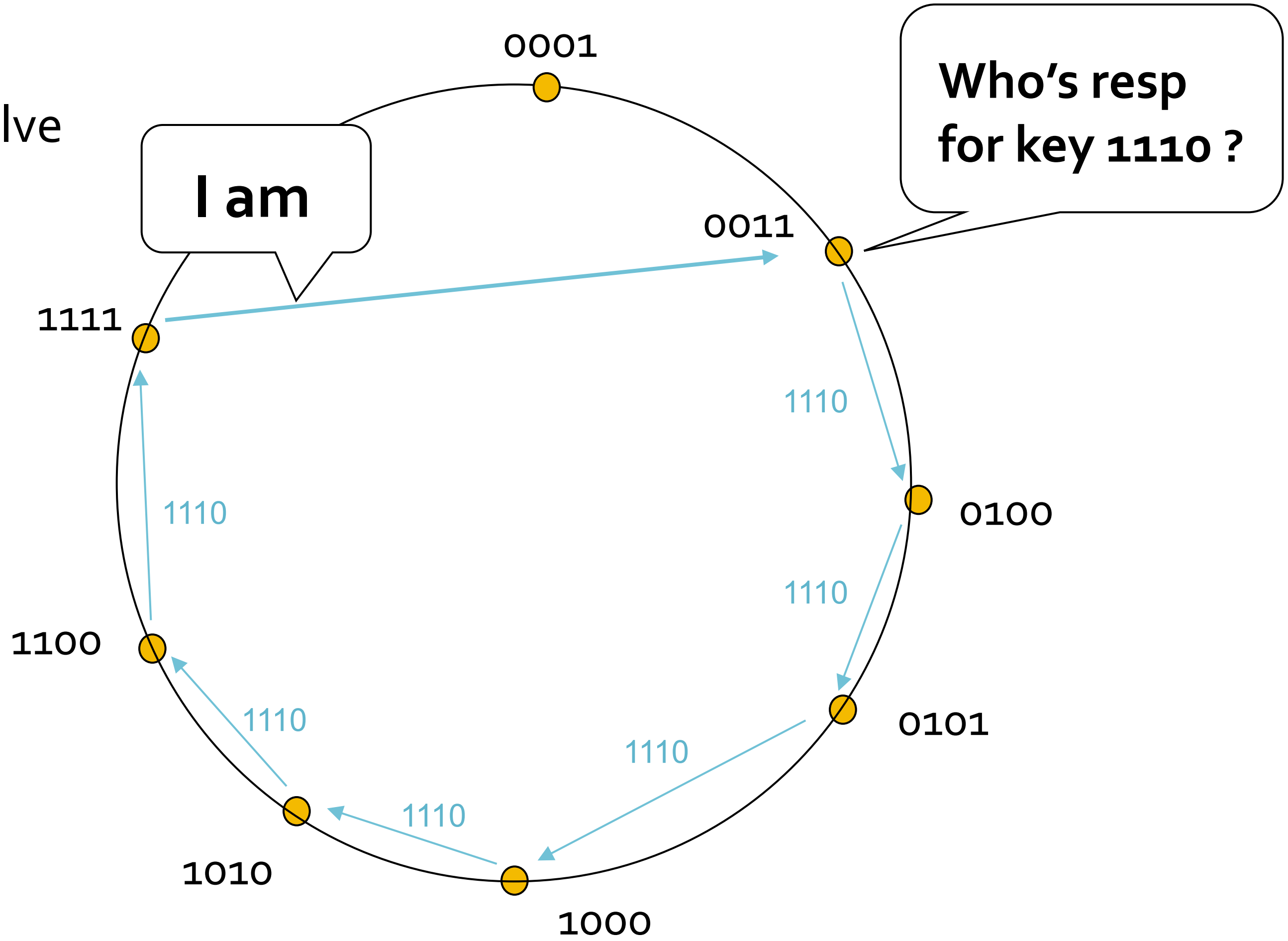
Circular DHT (1)



- Each peer *only* aware of immediate successor and predecessor.
- "Overlay network"

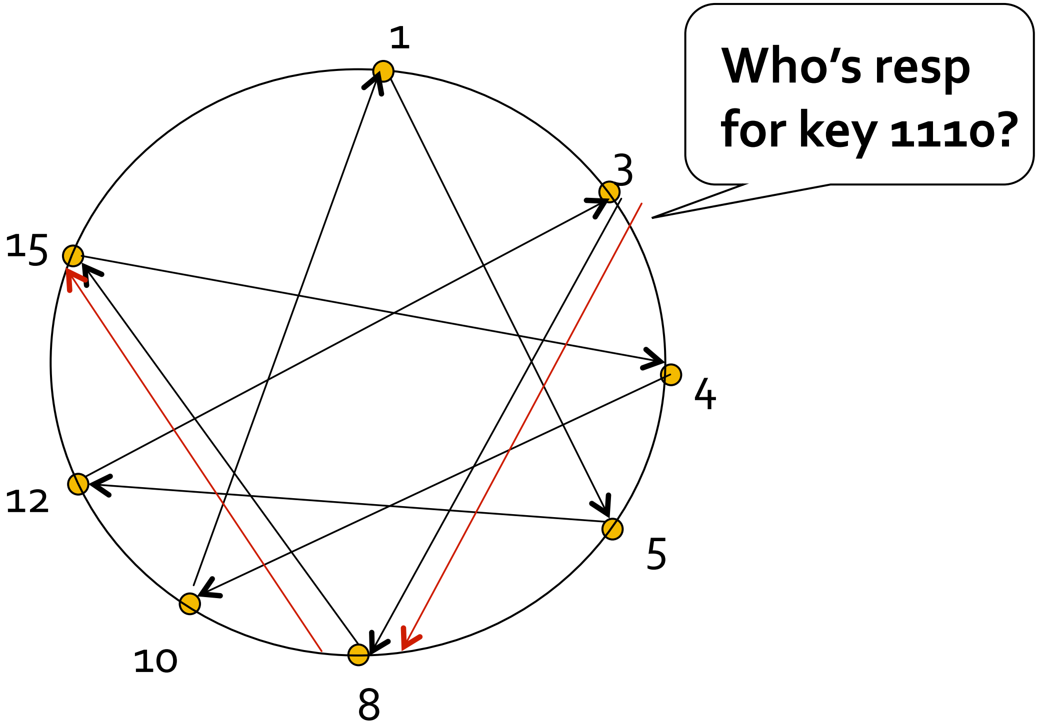
Circular DHT (2)

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



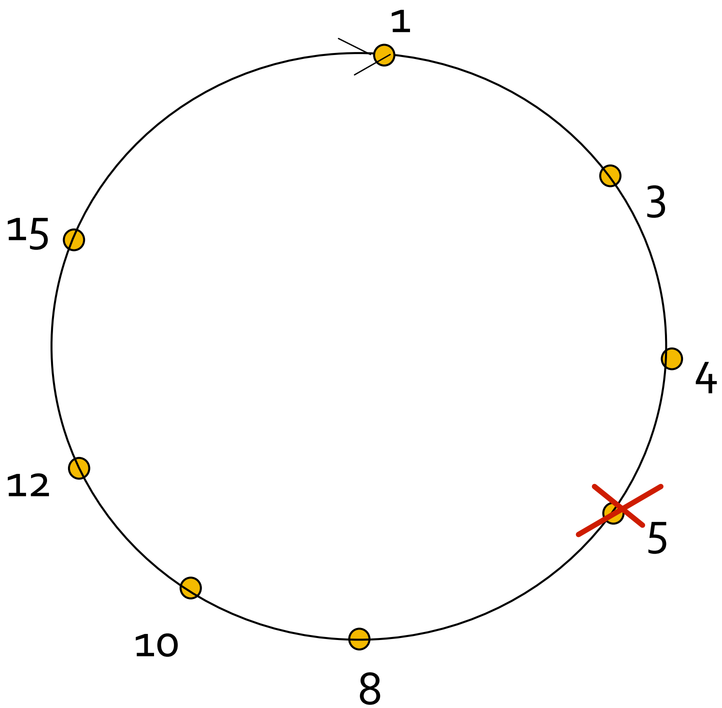
Define <u>closest</u>
as closest
successor

Circular DHT with Shortcuts



- Each peer keeps track of IP addresses of:
  - Predecessor
  - Successor
  - Short cut(s).
- Reduced from 6 to 2 messages.
- Possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn



- **Peer churn:** Peers are continually leaving and joining
  - Makes reliability harder!
- To handle peer churn, require each peer to know the IP address of its two successors
  - Each peer periodically pings its two successors to see if they are still alive

- Trigger: Peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor