# Today's Objectives

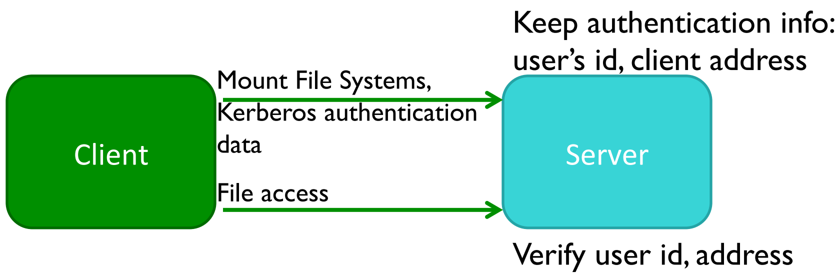
- Kerberos
- Peer To Peer
- Overlay Networks
- Final Projects



# Kerberos

- Trusted third party, runs by default on port 88
- Security objects:
  - Ticket: token, verifying sender has been authenticated by Kerberos
    - Expiry time (~several hours), session key
  - Authenticator: token constructed by client to prove identity of user
    - Only used once
    - Contains client's name and timestamp and encrypted in session key
  - Session key: secret key randomly generated
    - Issued to client for communicating with particular server
    - Used for encrypting communication with servers and authenticators
- Client must have ticket & session key for each server

## NFS with Kerberos




- Server does not maintain info at process level
- Requires only one user logged in to each client computer



## PEER TO PEER SYSTEMS

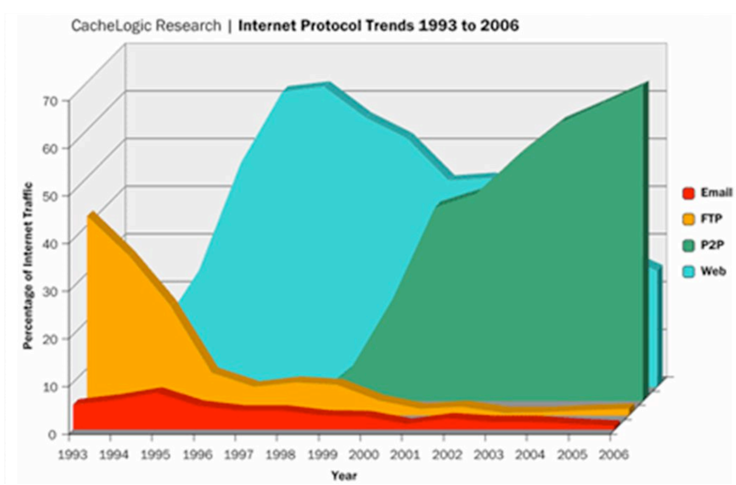
# Peer-to-Peer Network

- A distributed network architecture composed of participants that make a portion of their resources directly available to network participants without the need for **central coordination**
  - Resources: processing power, disk storage or network bandwidth
- Used largely for sharing of content files
  - audio, video, data or anything in a digital format
- There are many p2p protocols
  - Ares, Bittorrent, or eDonkey.
- Can be very large
- Can also be used for business solutions for relatively small companies that may not have resources available to implement a server solution.

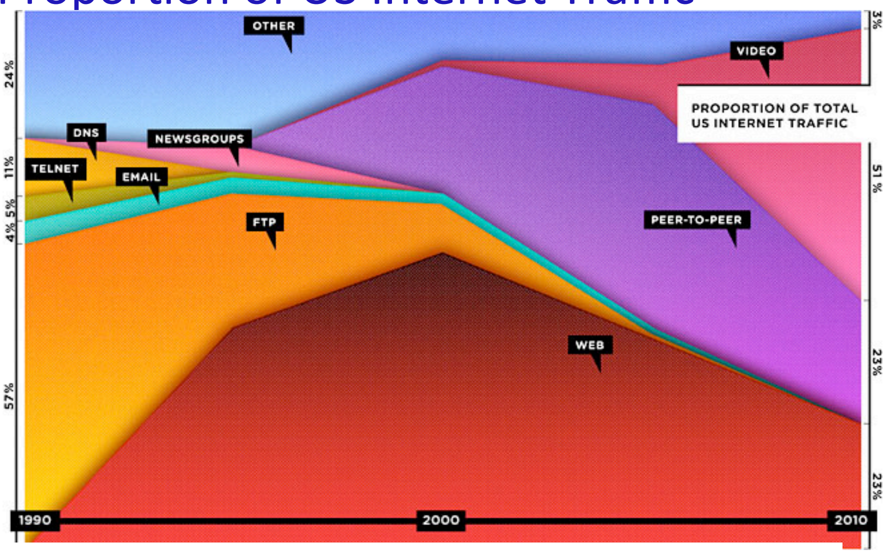
Slide content based on Clayton Sullivan



# Internet Protocol Trends, 1993-2006



## Proportion of US Internet Traffic

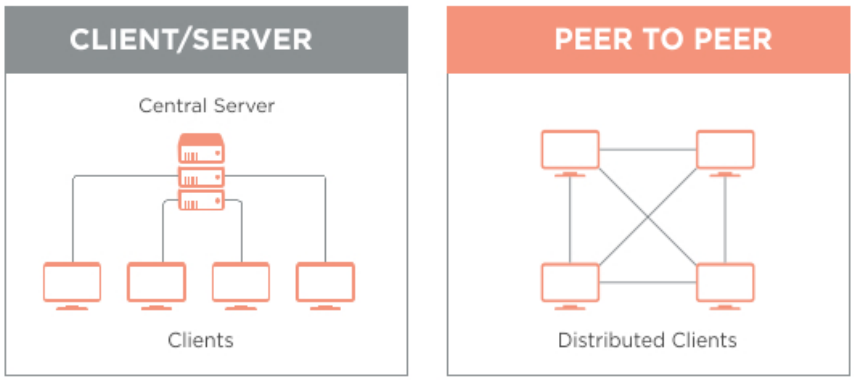



Sources: Cisco estimates based on CAIDA publications
Andrew Odlyzko https://www.wired.com/2010/08/ff_webrip/

## A Peer

- Peers are both suppliers and consumers
- In traditional client-server model, server supplies while client only consumes.

## Peer-To-Peer vs Client-Server






## Network Architecture

- Typically ad-hoc networks
  - adding and removing nodes have no significant impact on the network
- Allows peer-to-peer systems to provide enhanced scalability and service robustness
- Often, implemented as an application layer ***overlay network*** that is placed on top of native or physical network
  - Used for peer discovery and indexing

## Advantages

- The more nodes that are part of the system, demand increases and total capacity of the system also increases
  - In client-server network architectures as more clients are added to the system, the system resources decreases.
- There is no single point of failure, due to robustness of the system.
- All clients provide to the system



## Disadvantages

- Security is a major concern, not all shared files are from benign sources. Attackers may add malware to p2p files as an attempt to take control of other nodes in the network.
- Heavy bandwidth usage
- Anti-P2P companies have introduced faked chunks into shared files that rendered shared files useless upon completion of the download.
- ISP throttling of P2P traffic
- Potential legal/moral concerns

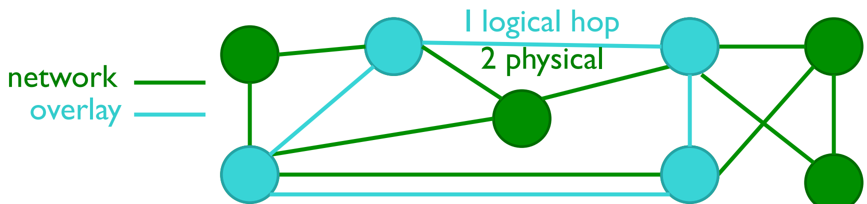
# P2P as Overlay Networking

- P2P applications need to:
  - track identities & IP addresses of peers
    - May be many and may have significant churn
  - Route messages among peers
    - If you don't keep track of all peers, this is "multi-hop"
- *Overlay network*
  - Peers doing both naming and routing
  - IP becomes "just" the low-level transport

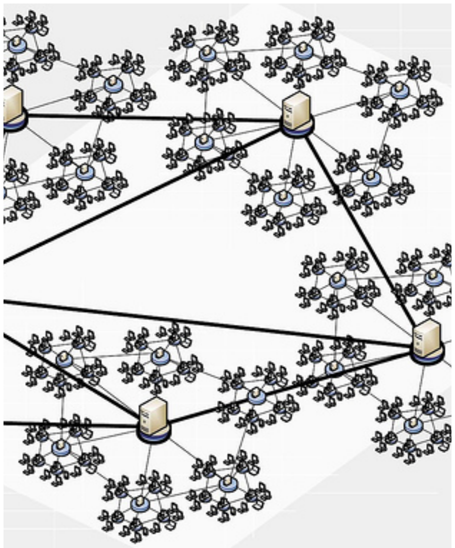


# Overlay Network

- A network built on top of one or more existing networks
  - A *virtual* network of nodes and *logical* links
- Built on top of an existing network
- Adds an additional layer of indirection/virtualization
- Changes properties in one or more areas of underlying network
- Purpose: implement a network service that is not available in the existing network



# Application Overlay Network



- P2P applications like BitTorrent create ***overlay networks*** over existing internet to perform indexing and peer collection functions
- Overlay networks have no control over physical networks or have any information on physical networks
- Weak resource coordination, as well as weak response to fairness of resource sharing



# Structured vs. Unstructured

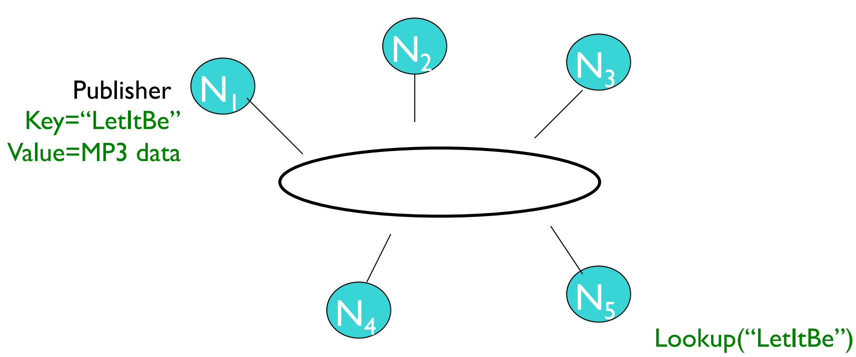
- Structured
  - Connections in the overlay are fixed
  - DHT Indexing
- Unstructured
  - No algorithm for organization or optimization
  - Connections in the overlay are created arbitrarily
  - Centralized
    - Central server is used for indexing functions
    - BitTorrent
  - Hybrid
    - Two groups of clients: client and overlay
    - eMule, Kazaa
  - Pure
    - Equipotent peers, all peers have equal amount of power
    - Gnutella, Freenet

# DISTRIBUTED HASH TABLES



## Introduction to DHTs

- Challenge: How to find data in a distributed file sharing system?




Lookup is the key problem!

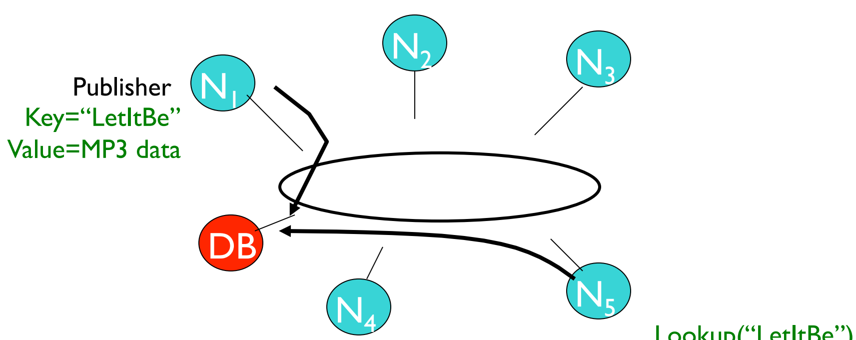

Slide content based on material from Daniel Figueiredo and Robert Morris

# Review: Possible solutions

- Centralized (example?)
- Distributed



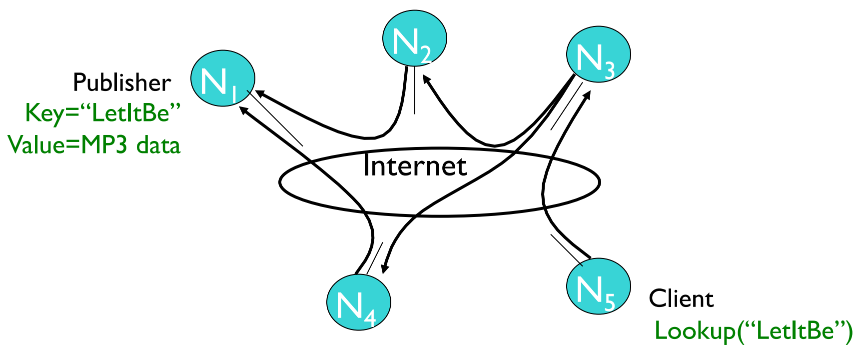
# Centralized Solution: Napster




- Requires O(N) state
- Single point of failure

# Distributed Solution: Flooding

Gnutella, Morpheus, etc.

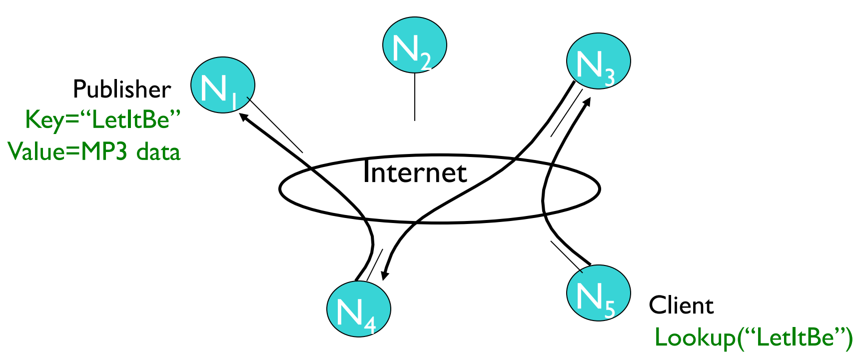



- Worst case O(N) messages per lookup



# Distributed Solution: Routed Messages

Freenet, Tapestry, Chord, CAN, etc.



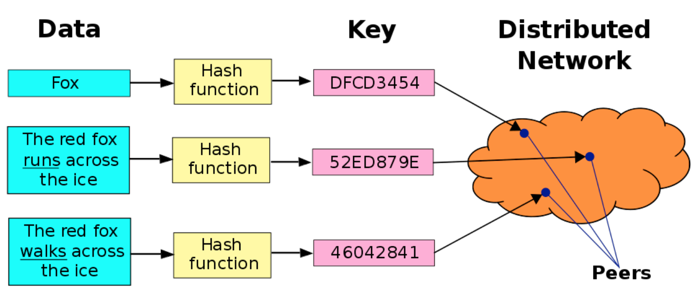
# Routing Challenges

- Define a useful key nearness metric
- Keep the hop count small
- Keep the routing tables "right size"
- Stay robust despite rapid changes in membership



# Structured DHT

- Employ globally consistent protocol to ensure that any node can efficiently route a search to some peer that has a desired file.
  - Guarantee → more structured pattern of overlay links
- DHT is a lookup service
  - allows any participating node to efficiently retrieve the value associated with a given key whether the file is new or older/rarer.
- Maintaining the mappings from keys to values is handled by nodes that any change in the number of participants causes minimal amount of disruption
- Allows for continual node arrival and departure, fault tolerant



## Chord Discussion

- Chord: emphasizes efficiency and simplicity
- Provides peer-to-peer hash lookup service:
  - Lookup(key) → IP address
  - Note: Chord does not store the data
- How does Chord locate a node?
- How does Chord maintain routing tables?



## Chord Properties

- Efficient: O(log(N)) messages per lookup
  - N is the total number of servers/peers
- Scalable: O(log(N)) state per node
- Robust: survives massive failures
- Proofs are in 2001 paper
  - Assume no malicious participants

# Chord IDs

- *m* bit identifier space for both keys and nodes
- Key identifier = SHA-1(key)

  Key="LetItBe" →(SHA-1) ID=60

- Node identifier = SHA-1(IP address)

  IP="137.165.10.100" →(SHA-1) ID=123

- Both are uniformly distributed and exist in same ID space

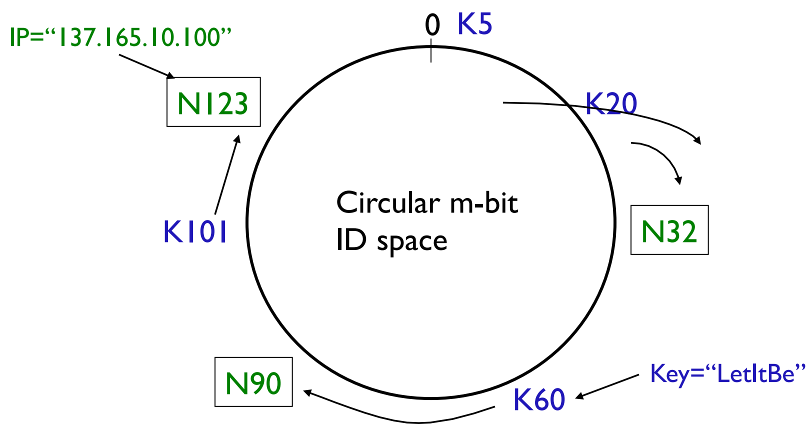
How map key IDs to node IDs?



# Consistent Hashing [Karger97]

- Given a set of *n* nodes, a consistent hash function will map keys (e.g., filenames) uniformly across the nodes
- Feature of consistent hashing for node addition:
  - Only *1/n* keys must be reassigned to new nodes
    - Only to new node

# Consistent Hashing

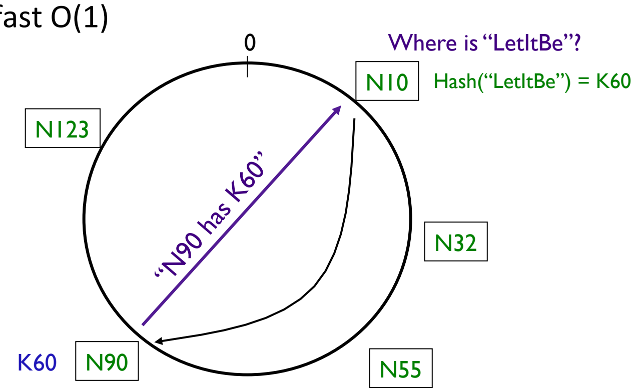



- Key is stored at its *successor*: node with next higher ID



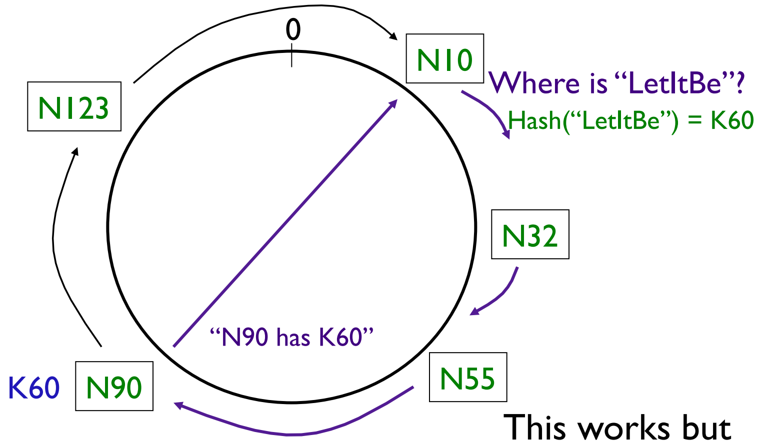
# Consistent Hashing

- Every node must know about every other node
  - requires global information!
- Routing tables are large O(N)
- But…lookups are fast O(1)



# Chord: Basic Lookup

Every node knows its successor in the ring

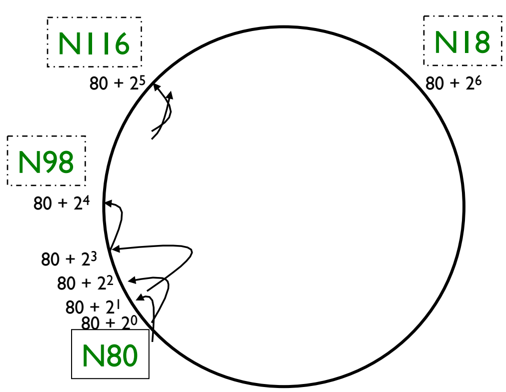



This works but requires O(N) time



# "Finger Tables"

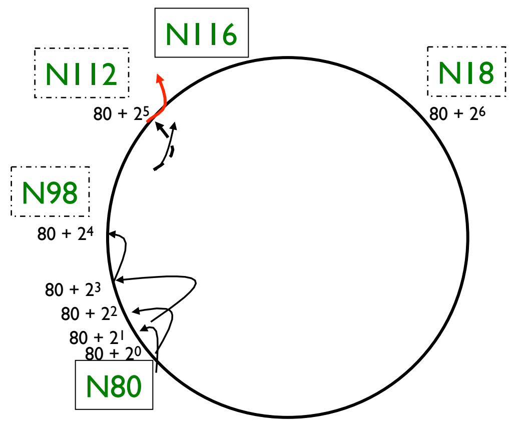
- Every node knows up to m other nodes in the ring
- Increase distance exponentially
- m=7 in this example



## "Finger Tables"

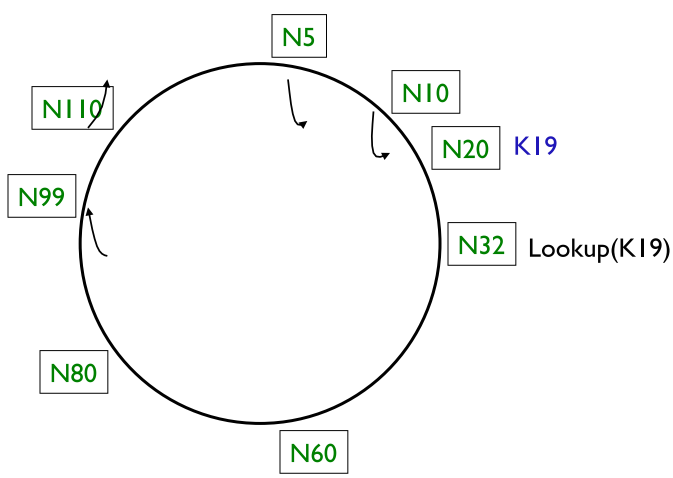
- Finger i points to *successor* of $n+2^i$
  - $i^{th}$ entry in n's finger table has ID > $(n+2^i) \% 2^m$

| | |
|---|---|
| N80+1 | N98 |
| N80+2 | N98 |
| N80+4 | N98 |
| N80+8 | N98 |
| N80+16 | N98 |
| N80+32 | N116 |
| N80+64 | N18 |






## Lookups are Faster




Lookups take O(log N) hops

## Chord Discussion

- How does Chord cope with changes in membership?

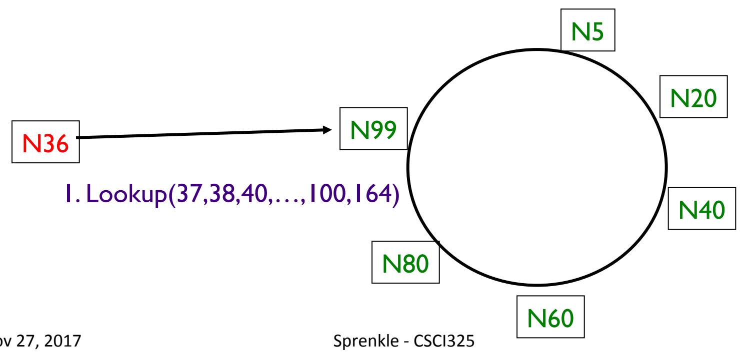


## Joining the Ring

- Three step process:
  1. Initialize all fingers of new node
  2. Update fingers of existing nodes
  3. Transfer keys from successor to new node
- Less aggressive mechanism (lazy finger update):
  1. Initialize only finger to successor node
  2. Periodically verify immediate successor, predecessor
  3. Periodically refresh finger table entries

# Joining the Ring - Step 1

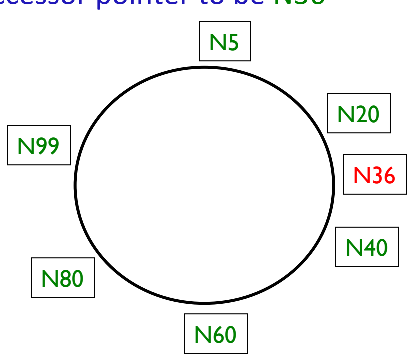
- Initialize new node finger table
  - Locate any node *p* in the ring
  - Ask node p to lookup fingers of new node N36
  - Return results to new node





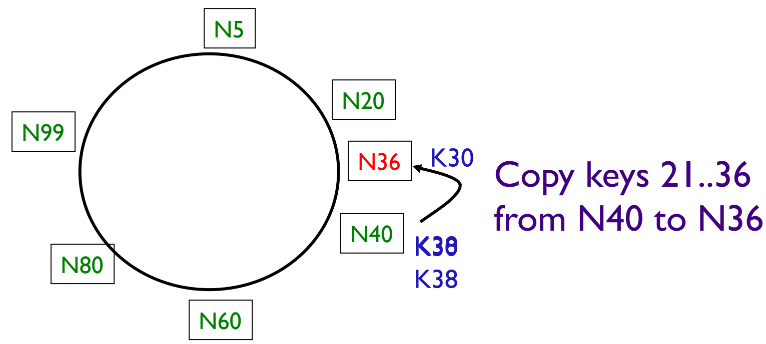

# Joining the Ring - Step 2

- Update fingers of existing nodes
  - New node calls *update* function on existing nodes
  - Existing nodes can recursively update fingers of other nodes
  - N36 sets successor pointer to be N40
  - N20 sets successor pointer to be N36



## Joining the Ring - Step 3

- Transfer keys from successor node to new node
  - Only keys in the range are transferred

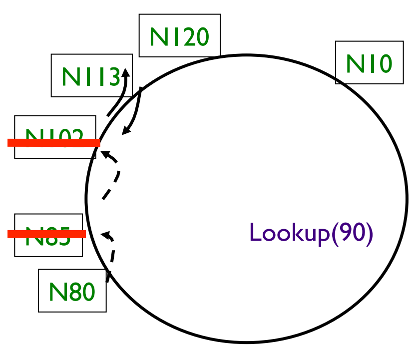



When a node leaves ring, all keys are copied to successor



## Handing Failures

Failure of nodes might cause incorrect lookup




N80 doesn't know correct successor, so lookup fails

## Chord Discussion

- How does Chord handle failures?



## Handing Failures

- Use successor list
  - Each node knows *r* immediate successors
  - After failure, will know first *live* successor
  - Correct successors guarantee correct lookups
- Guarantee is with some probability
  - Can choose *r* to make probability of lookup failure arbitrarily small

## Chord Discussion

- How did the authors evaluate Chord?
- What were the major results?



## Evaluation Overview

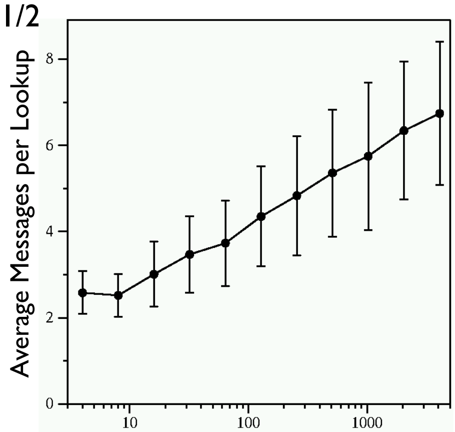
- Quick lookup in large systems
- Low variation in lookup costs
- Robust despite massive failure

- Experiments confirm theoretical results

# Cost of Lookup

Cost is O(log N), as predicted by theory

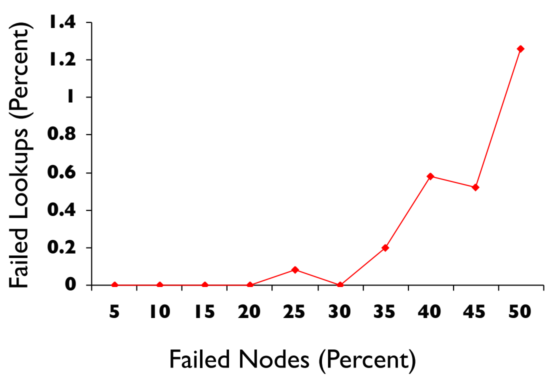
- Constant is 1/2





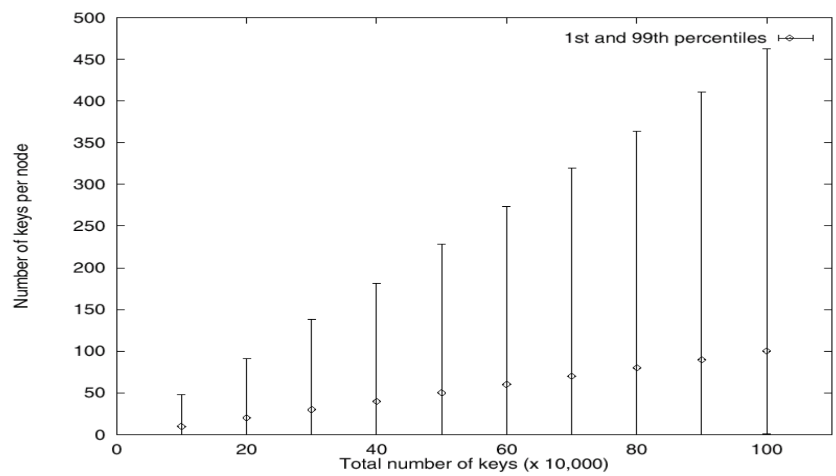

# Robustness

- Start with 1000 peers
- Insert 1000 key/value pairs (and replicate each 5 times)
- Stop X% of peers
- Perform 1000 lookups



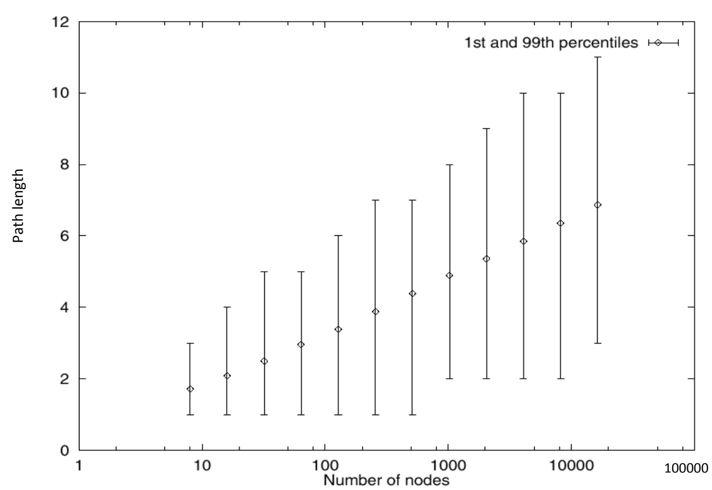

Massive failures have little impact!

# Effectiveness of Load Balancing



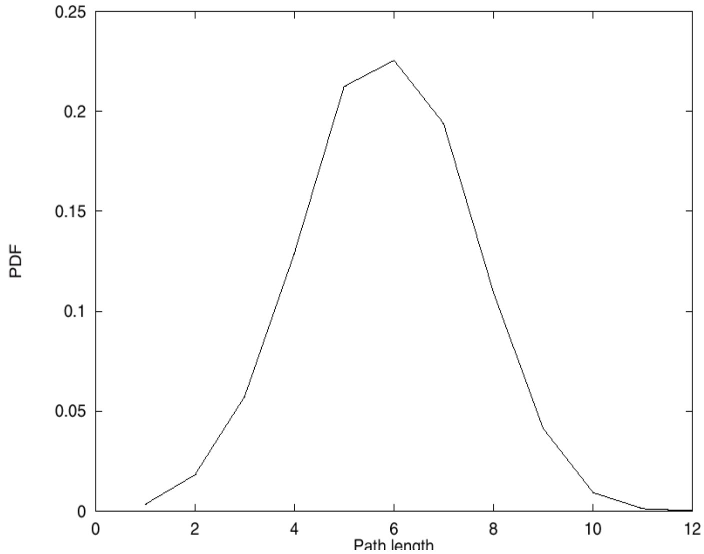

Nov 27, 2017 Sprenkle - CSCI325 47

# Path Length of Lookup




Nov 27, 2017 Sprenkle - CSCI325 48

## Distribution of Path Length (4096 nodes)






## Discussion

- Any of your questions?
  - What are typical issues we need to think about?
  - How do they fit into Chord?
- Locality with respect to the underlying network?
  - From SD, first lookup goes to Australia, second to Europe, third to Asia
- Even *O(log n)* steps too many for routing in large networks?
- Single popular key mapping to a single node?
- What about search?
- How does replication fit into the picture?

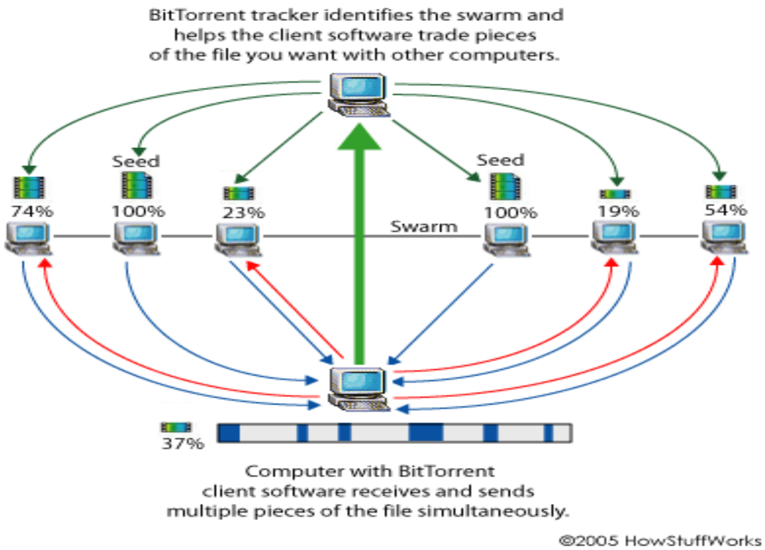
## Unstructured

- Overlay links are established arbitrarily
- When a peer wants to find the file, the request must be flooded through network to find as many peers as possible that share the data.
- This flooding creates a large amount of signal traffic.
- No guarantee that file will be found especially when the file is older or rare
- Very poor search efficiency
- Most popular p2p networks are unstructured networks



## BitTorrent

- One of many forms of p2p protocols for file-sharing.
- Created in 2001
- Estimated to account for 43% of a
- Many clients that work on bittorre
  - Utorrent, Vuze, BitTorrent
- Most are of the Unstructured p2p architecture
  - Centralized
    - tracker
  - Most clients have started to implement DHT functions






# BitTorrent

- Creates an application overlay network over existing internet infrastructure
- Peers when trying to download file, make request to the network and attempt to get the most possible peers connected to download file
  - Resources are not optimized and fairness is a concern
- Clients have started to implement DHT as a better way to connect to peers in order to download files more efficiently.
- When new files are added to the, small data requests are carried out over TCP connections to different machines in order to share the load of initial file sharer.
- Trackers assist in the communication between peers
- DHT would remove need for trackers

# OVERLAY NETWORKS

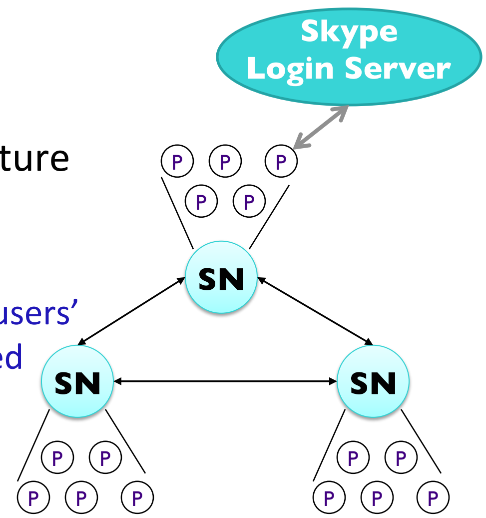


## Overlay Network: Example

- The Internet
  - Goal: connect local area networks
  - Built on local area networks (e.g., Ethernet), phone lines
  - Add an Internet Protocol header to all packets

# Another Example: Skype

- From Kazaa
- Voice over IP service
- Peer-to-peer infrastructure
  - Hosts: ordinary users' machines
  - Super nodes: ordinary users' machines with enhanced roles






# Skype

- No IP address or port is required to establish call
- Users authenticated by well-known login server
- Then, connect to super node
- Global index of users is distributed across super nodes
  - Needs to be searched for other users
  - Super node initiates search on ~8 super nodes
  - Takes between ~3-4 seconds
- Establishes connection using TCP
- UDP or TCP for streaming
- Encoding and decoding → excellent call quality

# Final Project

- Proposal due today
- GitHub repository