15-441 15-641 Computer Networking

Lecture 17: Delivering Content
Peer to Peer Examples
Peter Steenkiste

Fall 2014
www.cs.cmu.edu/~prs/15-441-F14

Overview

- Web
- Consistent hashing
- Peer-to-peer
  - Motivation
  - Architectures
  - TOR
  - Skype
- CDN
- Video

The Solution Space

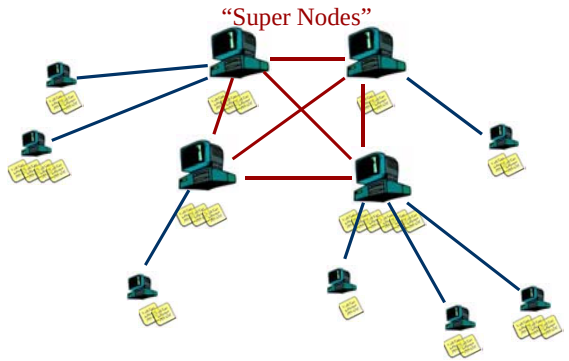
- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Structured Overlay Routing**
  - Distributed Hash Tables

KaZaA: Query Flooding

- First released in 2001 and still used today
  - Also very popular

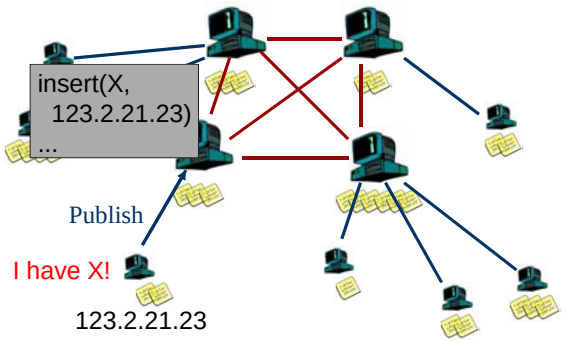
- **Join**: on startup, client contacts a "supernode" ... may at some point become one itself
- **Publish**: send list of files to supernode
- **Search**: send query to supernode, supernodes flood query amongst themselves.
- **Fetch**: get the file directly from peer(s); can fetch simultaneously from multiple peers

KaZaA: Network Design



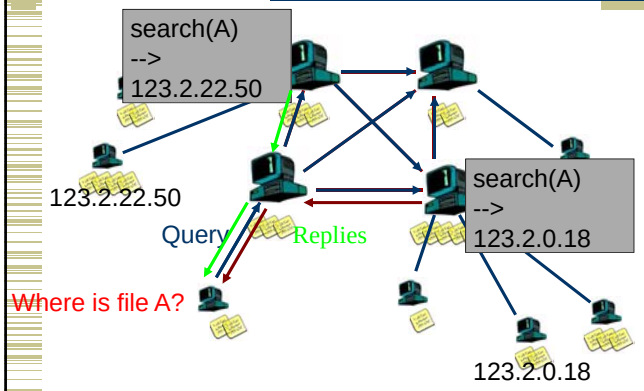
5

KaZaA: File Insert



6

KaZaA: File Search



7

KaZaA: Fetching

- More than one node may have requested file...
- How to tell?
  - Must be able to distinguish identical files
  - Not necessarily same filename
  - Same filename not necessarily same file...
- Use Hash of file
  - KaZaA uses UUHash: fast, but not secure
  - Alternatives: MD5, SHA-1
- How to fetch?
  - Get bytes [0..1000] from A, [1001...2000] from B
  - Alternative: Erasure Codes

8

KaZaA: Discussion

- Pros:
  - Tries to take into account node heterogeneity:
    - Bandwidth
    - Host Computational Resources
    - Host Availability (?)
  - Rumored to take into account network locality
- Cons:
  - Mechanisms easy to circumvent
  - Still no real guarantees on search scope or search time
- Similar behavior to gnutella, but better.

9

Stability and Superpeers

- Why superpeers?
  - Query consolidation
    - Many connected nodes may have only a few files
    - Propagating a query to a sub-node would take more b/w than answering it yourself
  - Caching effect
    - Requires network stability
- Superpeer selection is time-based
  - How long you have been on is a good predictor of how long you will be around

10

The Solution Space

- **Centralized Database**
  - Napster
- **Query Flooding**
  - Gnutella
- **Intelligent Query Flooding**
  - KaZaA
- **Swarming**
  - BitTorrent
- **Structured Overlay Routing**
  - Distributed Hash Tables

- More on Thursday …

11

BitTorrent: History

- In 2002, B. Cohen debuted BitTorrent
- Key Motivation:
  - Popularity exhibits temporal locality (Flash Crowds)
  - E.g., Slashdot effect, CNN on 9/11, new movie/game release
- Focused on Efficient *Fetching*, not *Searching*:
  - Distribute the *same* file to all peers
  - Single publisher, multiple downloaders
- Has some “real” publishers:
  - Blizzard Entertainment using it to distribute the beta of their new game

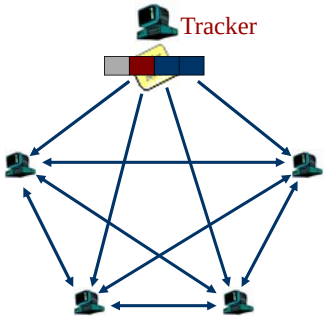
12

BitTorrent: Swarming

- Starting in 2001 to efficiently support flash crowds
  - Focus is on fetching, not searching
- **Join**: contact central “tracker” server for list of peers.
- **Publish**: Run a tracker server.
- **Search**: Find a tracker out-of-band for a file, e.g., Google
- **Fetch**: Download chunks of the file from your peers. Upload chunks you have to them.
- Big differences from Napster:
  - Chunk based downloading (sound familiar? :)
  - “few large files” focus
  - Anti-freeloading mechanisms

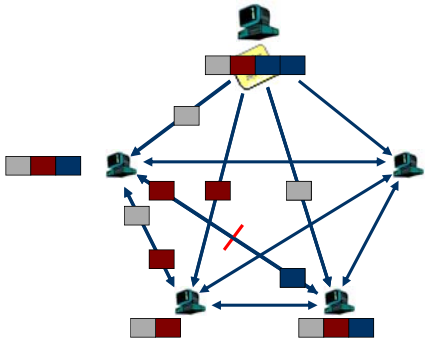
13

BitTorrent: Publish/Join



14

BitTorrent: Fetch



15

BitTorrent: Sharing Strategy

- Employ “Tit-for-tat” sharing strategy
  - A is downloading from some other people
    - A will let the fastest N of those download from him
  - Be optimistic: occasionally let freeloaders download
    - Otherwise no one would ever start!
    - Also allows you to discover better peers to download from when they reciprocate
- Goal: Pareto Efficiency
  - Game Theory: “No change can make anyone better off without making others worse off”
  - Does it work? (don’t know!)

16

BitTorrent: Summary

- Pros:
  - Works reasonably well in practice
  - Gives peers incentive to share resources; avoids freeloaders
- Cons:
  - Pareto Efficiency relative weak condition
  - Central tracker server needed to bootstrap swarm
  - (Tracker is a design choice, not a requirement, as you know from your projects. Could easily combine with other approaches.)

17

When are p2p Useful?

- Caching and "soft-state" data
  - Works well! BitTorrent, KaZaA, etc., all use peers as caches for hot data
- Finding read-only data
  - Limited flooding finds hay
  - DHTs find needles
- BUT they are not a "Google"
  - Complex intersection queries ("the" + "who"): billions of hits for each term alone
  - Sophisticated ranking: Must compare many results before returning a subset to user
  - Need massive compute power

31

Writable, Persistent p2p

- Do you trust your data to 100,000 monkeys?
  - May be ok for "free" song, but not for information critical to a company
    - E.g., how about DNS based on a DHT?
- Node availability hurts
  - Ex: Store 5 copies of data on different nodes
  - When someone goes away, you must replicate the data they held
  - Hard drives are *huge*, but cable modem upload bandwidth is tiny - perhaps 10 Gbytes/day
  - Takes many days to upload contents of 200GB hard drive. Very expensive leave/replication situation!

32

Overview

- Web
- Consistent hashing
- Peer-to-peer
  - Motivation
  - Architectures
  - TOR
  - Skype
- CDN
- Video

33

Tor Anonymity Network

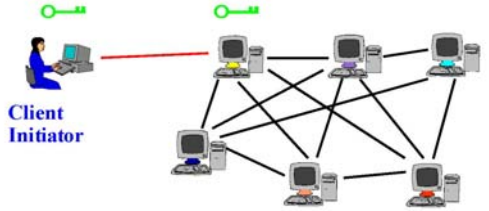
- Deployed onion routing network
  - http://torproject.org
  - Specifically designed for low-latency anonymous Internet communications
- Running since October 2003
  - Thousands of relay nodes, 100K-500K? of users
- Easy-to-use client proxy, integrated Web browser
  - Not like FreeNet – no data "in" TOR
- Really an overlay – not pure peer-to-peer

Based on slides by Vitaly Shmatikov

Tor Circuit Setup (1)

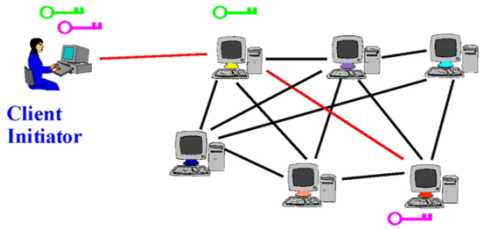
- Client proxy establish a symmetric session key and circuit with relay node #1
- All data sent over the circuit is encrypted

$A = K(B)_k$



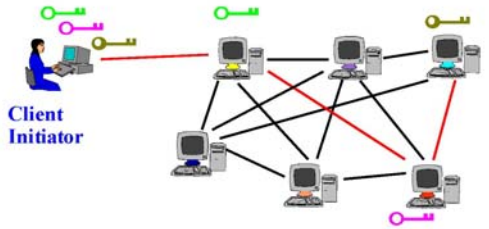
Tor Circuit Setup (2)

- Client proxy extends the circuit by establishing a symmetric session key with relay node #2
  - Tunnel through relay node #1

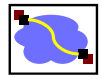


Tor Circuit Setup (3)

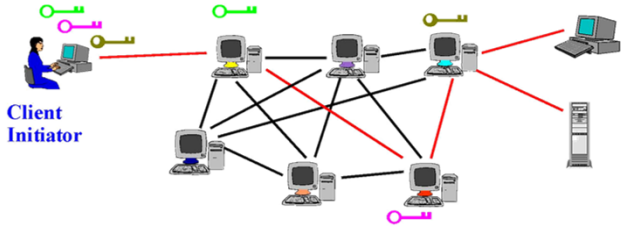
- Client proxy extends the circuit by establishing a symmetric session key with relay node #3
  - Tunnel through relay nodes #1 and #2



Using a Tor Circuit



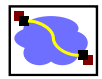
- Client applications connect and communicate over the established Tor circuit
  - Datagrams decrypted at each link
- Also want end-to-end encryption – not done by Tor



Using Tor

- Many applications can share one circuit
  - Multiple TCP streams over one anonymous connection
- Tor router doesn't need root privileges
  - Encourages people to set up their own routers
  - More participants = better anonymity for everyone
- Directory servers
  - Maintain lists of active relay nodes, their locations, current public keys, etc.
  - Control how new nodes join the network
    - "Sybil attack": attacker creates a large number of relays
  - Directory servers' keys ship with Tor code

Overview



- Web
- Consistent hashing
- Peer-to-peer
  - Motivation
  - Architectures
  - TOR
  - Skype
- CDN
- Video

What is Skype?

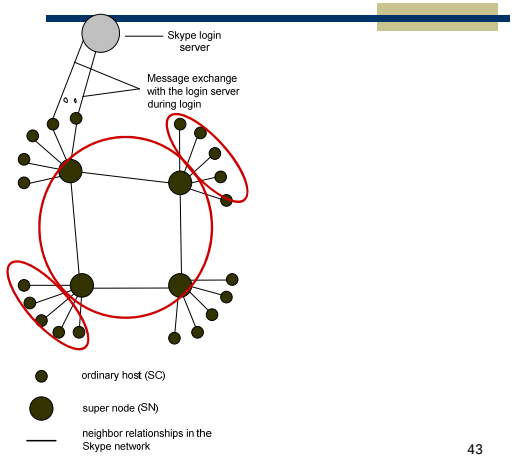
- Support pc-to-pc, pc-to-phone, phone-to-pc VoIP and IM client communication
  - Also: conference calls, video, …
- Developed by people who created KaZaa
  - Has peer-to-peer features that will look familiar
- Supported OS: Windows, Linux, MacOS, PocketPC
- A p2p illusion
  - Login server
  - Buddy-list server
  - Servers for SkypeOut and SkypeIn
  - Anonymous call minutes statistic gathering

Based on slides by Baset and Schulzrinne (Infocom 06)

What problems does it solve?

- NAT and firewall traversal
  - Nielsen September 2005 ratings
    - 61.3% of US home internet users use broadband (http://www.nielsen-netratings.com/pr/pr_050928.pdf)
    - 'Most' users have some kind of NAT
- Calls between traditional telephone and internet devicese
  - SkypeOut (pc-to-phone)
    - Terms of service: governed by the laws of Luxembourg
  - SkypeIn (phone-to-pc), voicemail
- Configuration-less connectivity
- Scalability for member data and call bandwidth

The Skype Network

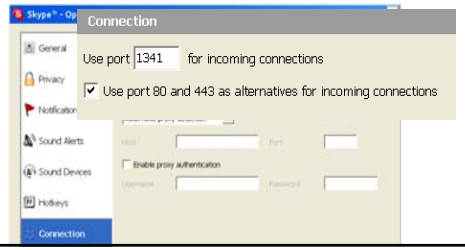


Primary Skype Components

- Ordinary host (OH)
  - A Skype client (SC)
- Super nodes (SN)
  - A Skype client (SC)
  - Has public IP address, 'sufficient' bandwidth, CPU and memory
- Login server
  - Stores Skype id's, passwords, and buddy lists
  - Used at login for authentication
  - Version 1.4.0.84: 212.72.49.141 and 195.215.8.141

Skype Components: Ports

- No default listening port
- Randomly chooses a port (P1) on installation
- Opens TCP and UDP listener sockets at P1
- Opens TCP listener sockets at port 80 (HTTP) and port 443 (HTTPS)



Skype Components: Host Cache

- IP address and port number of online Skype nodes (SNs)
- Maximum size: 200 entries
- Liang, Kumar and Ross. *Understanding KaZaA*
  - 200 entries for ordinary nodes (ON)
- Login server IP address and port number
- HC Windows location

  C:\Documents and Settings\All Users\Application Data\Skype

46

Experimental Setup Used in Study

- We have NOT reverse engineered Skype executable but it can be done (Biondi and Desclaux)
- Skype version: Linux v1.2, Windows v1.4.0.84
- Experiments performed between June-July and Nov-Dec 2005
- Tools Used
  - Ethereal (for packet capture)
  - NetPeeker (for tuning per process bandwidth)
  - NCH Tone generator (for generating tones of various frequencies)
  - APIMonitor (for monitoring the system calls)
  - LD_PRELOAD: Linux shared library and system call interception
    - Skype fails to run with **`ltrace`** and **`strace`**

47

Skype Functions

- Startup
- Login
- User search
- Call establishment
- Media transfer
- Keep-alive
- NAT and firewall traversal
- Conferencing

48

Skype Functions: LOGIN

- Public, NAT
  - Establishes a TCP connection with the SN
    - Keep connection alive by sending refresh message every 2 min.
  - Authenticates with the login server
  - Announces arrival on the network (controlled flooding)
  - Determines NAT type
- Firewall
  - Establishes a TCP connection with the SN
  - Authenticates with the login server

49

Skype Functions: USER SEARCH

- From the Skype website
  - Guaranteed to find a user it exists and logged in the last 72 hours
- Search results are cached at intermediate nodes
- Unable to trace messages beyond SN
- Cannot force a node to become a SN
  - Host cache is used for connection establishment and not for SN selection
- User does not exist. How does search terminate?
  - **Skype contacts login server for failed searches**
- SN searches for a user behind UDP-restricted firewall
- Same wildcard (sal*) search query from two different machines initiated at the same time gives different results

50

Skype Functions: CALL ESTABLISHMENT

- Call signaling always carried over TCP and goes e2e
- Calls to non buddies=search+call
- Initial exchange checks for blocked users
- Public-public call
  - Caller SC establishes a TCP connection with callee SC
- Public-NAT
  - Caller SC is behind port-restricted NAT
    - Different solutions based on the nature of the NAT
  - Caller---->Skype node (SN?) ----> Callee
  - TCP connection established between caller, callee, and more than one Skype nodes
- Firewall-firewall call
  - Same as public-NAT but no in-UDP packets

51

Skype Functions: MEDIA TRANSFER

- **No silence suppression**
- Silence packets are used to
  - play background noise at the peer
  - maintain UDP NAT binding
  - avoid drop in the TCP congestion window
- Putting a call on hold
  - 1 packet/3 seconds to call-peer or Skype node
  - same reasons as above
- Codec frequency range
  - 50-8,000 Hz (total bw of 3 kilobytes/s)
- Reasonable call quality at (4-5 kilobytes/s)

52

Skype Super Nodes

- Skype super node: A node with which a Skype establishes a TCP connection at login
- 8,153 successful login attempts over four days
- 35% hostnames had a .edu suffix
  - 102 universities
- 894 unique super nodes
- Unique SN IP distribution:
  - US 83.7%, Asia 8.9%, Europe 7.1%
- Top 20 nodes received 43.8% of the total connections
- Top 100 nodes: 70.5%

53

Summary

- Selfish application
  - Uses best CPU and bandwidth resources
  - Evades blocking
  - Change application priority to ‘High’ after call establishment
  - No application configuration to prevent machine from becoming a super node. Possible by limiting per-process bandwidth
- Code obfuscation, runtime decryption
- Login server and super nodes, not strictly peer-to-peer
- STUN and TURN equivalent functionality
- Combination of hashing and controlled flooding
- Multiple paths for ‘in-time’ switching in case of failures
- Search falls back to login server

54