6.033 Lecture 14 -- Peer to peer networks 3/29/2010

Recap -- where are we:

Before spring break:

3 layer network model

E2E
Network
Link

Looked at Internet. On Internet, Net layer is IP. Provides unreliable packet oriented service.

E2E Layer (TCP on Internet). Provides reliable delivery through acks and retries, performance through windowing, uses congestion control to avoid congestion collapse.

Then saw how we can use this basic structure to build "overlays" that provide additional functionality -- in particular, looked at DNS, content delivery networks, and mobile IP.

Today going to study a 4th kind of overlay -- peer to peer networks.

What is a peer to peer network? Rather than client server, in P2P nodes are both clients and servers. Typically indicates a network with little centralized infrastructure, and with many, many participants.

Most common examples are file sharing networks (e.g., users exchanging media with each other) and chat applications, though P2P is also used in online gaming and to build big caching networks (like the CDNs we studied last time.)

What is wrong with centralized infrastructure?

- Centralized point of failure
- High management costs if one org has to host millions of files, conversations, etc.

What we are going to do today is to look at some ways to build peer to peer networks.

Napster - first widely used peer to peer network was napster. File sharing systems. Centralized server, with a list of all users and all files. To find a file, sent request to server. Server sent back names of other users with that file. Then you contacted those users directly and downloaded files.

Diagram:

Worked pretty well. Required a centralized server, but it just listed files, didn't store them so managed to offload some of the work.

Problem was people started using it to share copyright materials, and centralized point of failure was an easy way to take it down.

Gnutella - Designed to eliminate this centralized point of failure. Gnutella is quite popular -- Limewire and many other P2P programs use it.

Initial idea: flood queries throughout network -- diagram:

Show TTL.

How to connect to network initially? Use a list of server that comes with binary, or contact some tracker website, or have the user input it (find tracker via Google, eg.)

Problem: everyone (within some range) sees all queries -- really bad for scalability.

Fix: add "super peers" and "query routing tables". Idea is that some nodes are promoted to super peers. These are nodes with lots of bandwidth that aren't behind NATs. Each node registers with one or more super-peers, and then queries are routed between super peers.

To prevent all queries going to all nodes, when a node connects it sends a "routing table" that indicates keywords it has files about. Queries are only sent to peers that might have a match based on the keywords in their routing tables.

Diagram:

Problems:

scalability

no incentive to participate

Will talk about BitTorrent in recitations, which is peer-to-peer file sharing network that adds a "fairness" requirement. (People have tried to add this to Gnutella)

Skype -- Voice chat program. Works in a similar way to Gnutella -- each peer connects to a super peer. Super peers can be any node that is not behind a firewall without good bandwidth. Super peers exchange lists of users connected to them. Adds a centralized login server which must authenticate you before super peers will route calls for you.

Diagram:

When you want to establish a call, you connect directly to the user you are chatting with, unless you or the person you are talking to is behind a NAT, in which case both of you connect through a super-peer. In some cases this means that all voice packets are actually being relayed through the super-peer. No way to opt out of this.

<u>Chord -- designed to be a highly scalable peer to peer protocol.</u>

Key idea -- rather than having to "search" for nodes with a given object, you store all objects with a given name at a well-known place in the network.

Objects might be files, or pointers to nodes with files. Names could be actual file names (e.g., song1.mp3) or keywords (e.g., "song").

Use hashing to map a name to an integer key. H(name) -> 1…N. (N e.g., 2^32)
H("song") -> 1244

Hash functions have the property that they randomly assign an integer to an input string; same string gets same integer, but a 1 bit difference in string may result in a completely different integer.

In Chord, peers are responsible for a contiguous subset of the keys. Typically visualized as a circle, with keys from 0 to N. Nodes assigned key space from themselves to the next node. Each node keeps track of "successor" -- next node in circle. Nodes also track predecessor (used in joining.)

Diagram:

(Show 1244)

<u>To perform a lookup:</u>

1) hash name of object
2) walk around ring following successor pointers until I find node that stores objects

Note that there can is only one node responsible for a given name

If M nodes, M/2 hops on average. Can we do better? Yes! Store K "finger pointers" of successor nodes that are 2, 4, … 2^K hops away, and the key range they are responsible for.

Example:

Using this, can route in log M steps -- because with each step can halve the distance to the goal (as long as at least 32 entries in finger table.) Example:

Joining the chord network -- how does a new node join? Just like with Gnutella, you discover one other node participating somehow. Can then join the network at some new location on ring. Take over part of key space from predecessor.

Diagram (See Lec 15 slides)

Chord recap:

O(log M) hops for queries -- usually much less -- vs linear in number of nodes in Gnutella.

Lots of stuff built on top of this, including file systems, file sharing protocols, etc.