Ossification of the Internet

• The Internet evolved as an experimental packet-switched network

• Today, many aspects appear to be “set in stone”
  - Witness difficulty in getting IP multicast deployed
  - Major obstacles to deployment of IPv6

• Yet many reasons to extend the Internet
  - E.g., BGP doesn’t even try to find optimal routes
  - Want extensions to routing—multicast, anycast, …
  - Might want greater availability than on Internet

• But can only change end nodes, not routers
Solution: Overlays

- Use Internet to form “virtual links”
- Build your own network on top of Internet
  - E.g., RON project greatly increases availability
- Use tunnels to form virtual links
  - Encapsulate your network packets in IP datagrams

- Examples:
  - IPv6-in-IPv4 packets
  - Mbone overlays
  - End-system multicast

- How to select links?
Overlay choice

- **Consider topology (a)**
  - Naive solution is iterated unicast from source (b)—suboptimal
  - Optimal requires router support (c)
  - Best overlay would be (d)

- **How to select links?**
  - Use standard routing protocol (DVMRP)
  - Want something close to underlying Internet topology
  - But must estimate link costs by measurement
Triangle inequality

- Want to use links that will improve performance
- Triangle equality holds often, but **not always**
  - I.e., Latencies \((a \rightarrow b) \not\leq (a \rightarrow c) + (c \rightarrow a)\)
Example overlay construction

- **View overlay as a mesh embedded in Internet**
  - Standard routing protocol selects routes in overlay

- **Add edges whenever a node joins**
  - Join means adding edges to one or more existing nodes

- **Add edges after failure, or to improve optimality**
  - $i$ periodically probes random node $j$
  - Add link $i \leftrightarrow j$ if sufficiently utility: $\sum_{m \neq i} \left( \frac{\Delta \text{latency w. } i \leftrightarrow j}{\text{latency without}} \right)$

- **Remove based on Cost**
  - $\max (\# \text{routes w. } i \to j \text{ as first hop}, \# \text{w. } j \to i)$
Peer-to-peer networks

- Aims to use the bandwidth and storage of the many hosts
  - Sum of access line speeds and disk space

- But to use this collection of machines effectively requires coordination on a massive scale
  - Key challenge: who has the content you are looking for?

- Moreover, the hosts are very flaky
  - Behind slow links
  - Often connected only a few minutes
  - So system must be very robust
Napster

• **Centralized search engine:**
  - All hosts with songs register them with central site
  - Users do keyword search on site to find desired song
  - Site then lists the hosts that have the song
  - User then downloads content

• **What makes this work?**
  - Central site only has to handle searches: little bandwidth
  - Vast collection of hosts can supply huge aggregate bandwidth
  - System is self-scaling: more users means more resources
What happened to Napster?

• Fastest growing Internet application ever
  - P2P traffic became, and remains, one of the biggest sources of traffic on the Internet!

• But legal issues shut site down

• Centralized system was vulnerable to legal attacks, and system couldn’t function without central site
  - Central point of failure

• What can one do without a central site?
  - That’s the hard question in peer-to-peer
Gnutella

- An example of an unstructured, decentralized P2P system

- **Context:**
  - Many hosts join a system
  - Each offers to share its own content
  - In return, each can make queries for others' content

- **Goal:**
  - Enable users to find desired content on others
  - Replaces centralized Napster DB with decentralized search
Gnutella approach

- **Step one: form an overlay network**
  - Each host, when it joins, “connects” to several Gnutella members
  - An “overlay” link is merely the fact that the nodes know each other’s IP address, and thus can send each other packets
Gnutella searches

- **Step two: search with *flood queries***
- Each query is flooded within some scope
  - Queries are typically keyword searches
  - TTL is used to limit scope of flood
  - Flooding means you don’t need any routing infrastructure beyond links
- **All responses to queries are forwarded back along path query came from**
  - Nodes remember queries they have seen
  - Avoids duplicating queries, offers some privacy
Gnutella performance

- **Tradeoff: Accuracy vs. cost of queries**
  - if TTL is small, then searches won’t find desired content
  - if TTL is large, network will get overloaded

- ** Supernode optimization:**
  - Normal nodes attach to supernodes, who search for them
  - Only flood among well-connected supernodes

- **Random-walk instead of flooding optimization:**
  - Provides correct TTL automatically

- **Proactive replication**
  - Replicate content that is frequently queried, to make it easier to find
“Unstructured Overlay”

- **Gnutella is unstructured in two senses:**
  - Links between nodes are essentially random
  - The content of each node is random (at least from the perspective of Gnutella)

- **Implications:**
  - Can’t route on Gnutella
  - Wouldn’t know where to route even if we could
Structured overlays

- Most Gnutella downloads are for widely-replicated content
  - I.e.g, Gnutella is good at finding the “hay”
  - But how would you find “needles”?

- Need *structured overlays*
  - Say you know name of object
  - And only one copy of object in the system
  - Can you index object such than anyone can find it?

- Want to lookup up name $\rightarrow$ value mapping
  - Sounds like a hash table