Overlay Networks: Motivations

- Changes in the network happen very slowly

- Why?
  - Internet network is a shared infrastructure; need to achieve consensus (IETF)
  - Many of proposals require to change a large number of routers (e.g., IP Multicast, QoS); otherwise end-users won’t benefit

- Proposed changes that haven’t happened yet on large scale:
  - More Addresses (IPv6 ‘91)
  - Security (IPSEC ‘93); Multicast (IP multicast ‘90)
Motivations (cont’d)

- One size does not fit all

- Applications need different levels of
  - Reliability
  - Performance (latency)
  - Security
  - Access control (e.g., who is allowed to join a multicast group)
  - …

Goals

- Make it easy to deploy new functionalities in the network → accelerate the pace of innovation

- Allow users to customize their service
Solution

- Deploy processing in the network
- Have packets processed as they traverse the network

Overview

- Resilient Overlay Network (RON)
  - Overlay Multicast
  - Peer-to-peer systems
Resilient Overlay Network (RON)

- Premise: by building application overlay network, can increase performance and reliability of routing

- Install N computers at different Internet locations

- Each computer acts as an overlay network router
  - Between each overlay router is an IP tunnel (logical link)
  - Logical overlay topology is all-to-all (N^2)

- Computers actively measure each logical link in real time for
  - Packet loss rate, latency, throughput, etc

- Route overlay network traffic based on measured characteristics

Example

Berkeley

Default IP path determined by BGP & OSPF

MIT

Reroute traffic using red alternative overlay network path, avoid congestion point

UCLA

Acts as overlay router
Overview

- Resilient Overlay Network (RON)
  - Overlay multicast
- Peer-to-peer systems

IP Multicast Problems

- Seventeen years of research, but still not widely deployed
- Poor scalability
  - Routers need to maintain per-group or even per-group and per-sender state!
  - Multicast addresses cannot be aggregated
- Supporting higher level functionality is difficult
  - IP Multicast: best-effort multi-point delivery service
  - Reliability and congestion control for IP Multicast complicated
- No support for access control
  - Nor restriction on who can send → easy to mount Denial of Service (Dos) attacks!
Overlay Approach

- Provide IP multicast functionality above the IP layer → application level multicast
- Challenge: do this efficiently
- Projects:
  - Narada
  - Overcast
  - Scattercast
  - Yoid
  - ...

Narada [Yang-hua et al, 2000]

- Source Specific Trees
- Involves only end hosts
- Small group sizes <= hundreds of nodes
- Typical application: chat
Narada: End System Multicast

Overlay Tree

Properties

- Easier to deploy than IP Multicast
  - Don't have to modify every router on path
- Easier to implement reliability than IP Multicast
  - Use hop-by-hop retransmissions
- But
  - Consume more bandwidth than IP Multicast
  - Typically has higher latency than IP Multicast
  - Harder to scale
- Optimization: use IP Multicast where available
Overview

- Resilient Overlay Network (RON)
- Overlay multicast
  - Peer-to-peer systems

How Did it Start?

- A killer application: Napster
  - Free music over the Internet
- Key idea: share the storage and bandwidth of individual (home) users
Model

- Each user stores a subset of files
- Each user has access (can download) files from all users in the system

Main Challenge

- Find where a particular file is stored
  - Note: problem similar to finding a particular page in web caching (see last lecture – what are the differences?)
Other Challenges

- Scale: up to hundred of thousands or millions of machines
- Dynamicity: machines can come and go any time

Napster

- Assume a centralized index system that maps files (songs) to machines that are alive
- How to find a file (song)
  - Query the index system → return a machine that stores the required file
    - Ideally this is the closest/least-loaded machine
  - ftp the file
Napster: Example

Advantages:
- Simplicity, easy to implement sophisticated search engines on top of the index system

Disadvantages:
- Robustness, scalability (?)
Gnutella

- Distribute file location
- Idea: flood the request
- How to find a file:
  - Send request to all neighbors
  - Neighbors recursively multicast the request
  - Eventually a machine that has the file receives the request, and it sends back the answer

Gnutella: Example

- Assume: m1’s neighbors are m2 and m3; m3’s neighbors are m4 and m5;…
Gnutella: Discussion

- Advantages:
  - Totally decentralized, highly robust

- Disadvantages:
  - Not scalable; the entire network can be swamped with request (to alleviate this problem, each request has a TTL)

Other Solutions to the Location Problem

- Use a distributed rather than a centralized directory (like in the case of Napster)

- Distributed hash-table data (DHT) abstraction
  - insert(id, item);
  - item = query(id);
  - Note: item can be anything: a data object, document, file, pointer to a file…

- Proposals
  - CAN, Chord, Kademlia, Pastry, Tapestry, etc
DHT Design Goals

- Make sure that an item (file) identified is always found
- Scales to hundreds of thousands of nodes
- Handles rapid arrival and failure of nodes

Structured Networks

- Distributed Hash Tables (DHTs)
- Hash table interface: `put(key, item), get(key)`
- \( O(\log n) \) hops
- Guarantees on recall
Content Addressable Network (CAN)

- Associate to each node and item a unique id in an $d$-dimensional Cartesian space on a $d$-torus
- Properties
  - Routing table size $O(d)$
  - Guarantees that a file is found in at most $d \cdot n^{1/d}$ steps, where $n$ is the total number of nodes

CAN Example: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
  - Node n1:(1, 2) first node that joins cover the entire space
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2
CAN Example: Two Dimensional Space

- Nodes n4:(5, 5) and n5:(6,6) join

CAN Example: Two Dimensional Space

- Nodes: n1:(1, 2); n2:(4,2); n3:(3, 5); n4:(5,5); n5:(6,6)
- Items: f1:(2,3); f2:(5,1); f3:(2,1); f4:(7,5);
CAN Example: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space

![Diagram of CAN Example]

CAN: Query Example

- Each node knows its neighbors in the $d$-space
- Forward query to the neighbor that is closest to the query $id$
- Example: assume $n1$ queries $f4$
- Can route around some failures

![Diagram of CAN Query Example]
Chord

- Associate to each node and item a unique \( id \) in an \( uni \)-dimensional space \( 0..2^m-1 \)

- Key design decision
  - Decouple correctness from efficiency

- Properties
  - Routing table size \( O(\log(N)) \), where \( N \) is the total number of nodes
  - Guarantees that a file is found in \( O(\log(N)) \) steps

Identifier to Node Mapping Example

- Node 8 maps [5,8]
- Node 15 maps [9,15]
- Node 20 maps [16, 20]
- ...
- Node 4 maps [59, 4]

- Each node maintains a pointer to its successor
Lookup

- Each node maintains its successor
- Route packet (ID, data) to the node responsible for ID using successor pointers

Joining Operation

- Each node A periodically sends a stabilize() message to its successor B

- Upon receiving a stabilize() message, node B
  - returns its predecessor B'=pred(B) to A by sending a notify(B') message

- Upon receiving notify(B') from B,
  - if B' is between A and B, A updates its successor to B'
  - A doesn't do anything, otherwise
Joining Operation

- Node with id=50 joins the ring
- Node 50 needs to know at least one node already in the system
  - Assume known node is 15
  - Node 50: send join(50) to node 15
  - Node 44: returns node 58
  - Node 50 updates its successor to 58
Joining Operation

- Node 50: send stabilize() to node 58
- Node 58:
  - update predecessor to 50
  - send notify() back

Joining Operation (cont’d)

- Node 44 sends a stabilize message to its successor, node 58
- Node 58 reply with a notify message
- Node 44 updates its successor to 50
Joining Operation (cont’d)

- Node 44 sends a stabilize message to its new successor, node 50.
- Node 50 sets its predecessor to node 44.

This completes the joining operation!
Achieving Efficiency: finger tables

\[ (80 + 2^i) \mod 2^m = 16 \]

\[ \text{Say } m=7 \]

\[ i \text{th entry at peer with id } n \text{ is first peer with id } \geq n + 2^i \mod 2^m \]

Achieving Robustness

- To improve robustness each node maintains the \( k (> 1) \) immediate successors instead of only one successor

- In the notify() message, node A can send its \( k-1 \) successors to its predecessor B

- Upon receiving notify() message, B can update its successor list by concatenating the successor list received from A with A itself
Discussion

- Query can be implemented
  - Iteratively
  - Recursively

- Performance: routing in the overlay network can be more expensive than in the underlying network
  - Because usually there is no correlation between node ids and their locality; a query can repeatedly jump from Europe to North America, though both the initiator and the node that store the item are in Europe!
  - Solutions: Tapestry takes care of this implicitly; CAN and Chord maintain multiple copies for each entry in their routing tables and choose the closest in terms of network distance

Conclusions

- The key challenge of building wide area P2P systems is a scalable and robust directory service

- Solutions covered in this lecture
  - Naptser: centralized location service
  - Gnutella: broadcast-based decentralized location service
  - CAN, Chord, Tapestry, Pastry: intelligent-routing decentralized solution
    - Guarantee correctness
    - Tapestry, Pastry provide efficient routing, but more complex