Middleware and Distributed Systems

Peer-to-Peer Systems

Martin v. Löwis

Peer-to-Peer Systems (P2P)

- Concept of a decentralized large-scale distributed system
  - Large number of networked computers (peers)
  - Each peer has equivalent capabilities and responsibilities, merging the roles of client and server
  - Data distribution over participants, no central authority
- Avoids limitations of pure client/server in terms of scalability
- Increased interest with file-sharing applications (1999)
- First peer-based systems long before the Internet
  - USENET (1979), FidoNet BBS message exchange system (1984)

Usenet

- Started 1979, after introduction of UUCP in UNIX v7
- Tom Truscott and Jim Ellis, Grad students at Two Duke University
- Reading and posting of articles in distributed newsgroups
    - User subscription to hierarchically ordered newsgroups
    - Synchronization of client application with local news server, local server synchronizes with other news-feeds
    - Time of slow networks, batch transfer of messages twice a day
    - Flooding to all servers which did no see the new message so far
    - Message disappears from group after some time (meanwhile web archives)
- Also binary transport (uuencode, Base64 / MIME encoding) - *alt.binary*
- NNTP for TCP transport (1985), message format similar to eMail (RFC 850, 1983)

Characteristics Of P2P

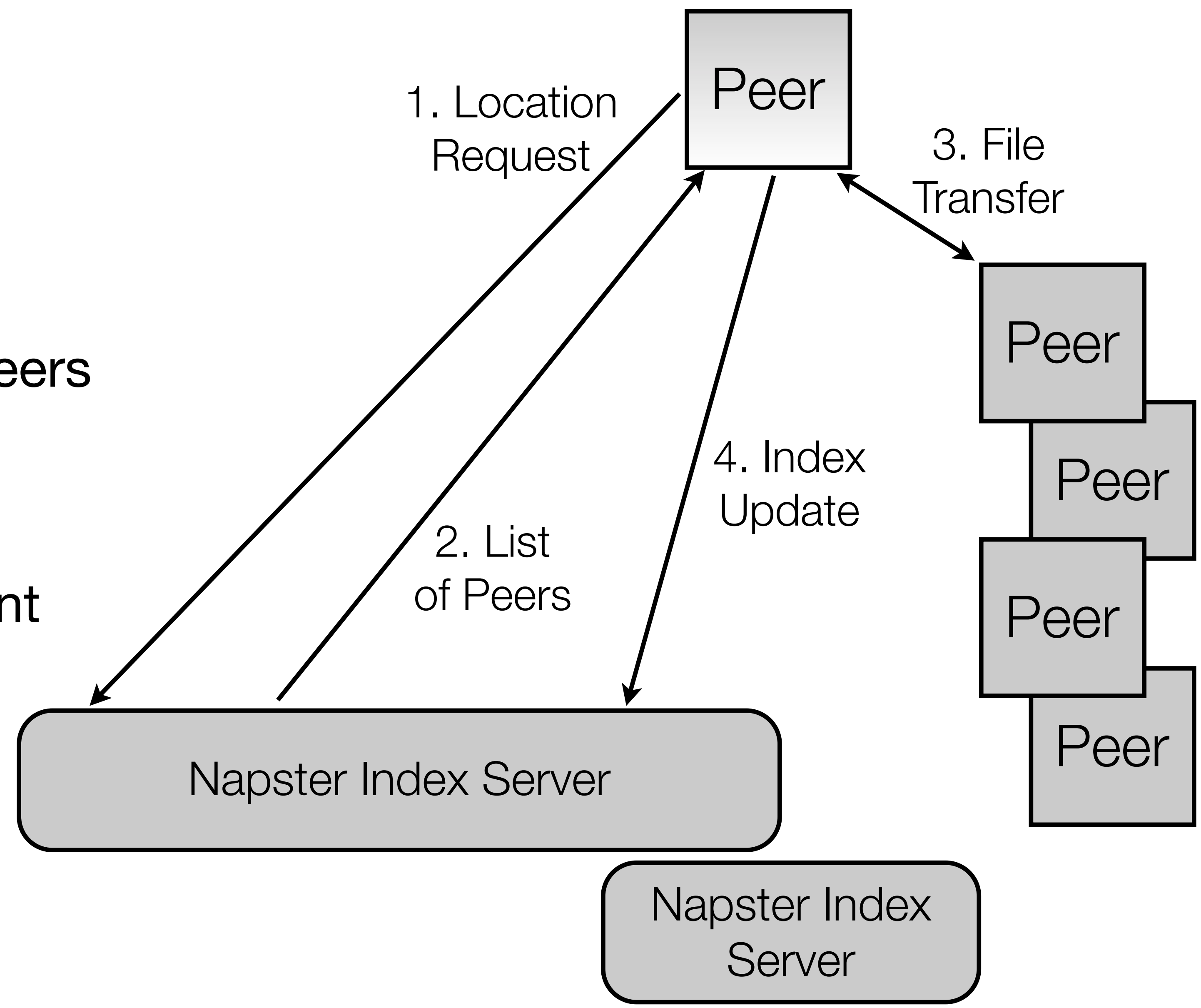
- Placement of data objects across many hosts
  - Balancing of access load, techniques for search and retrieval of data
- Each participating machines contributes resources
  - Volatile and non-exclusive availability of nodes
  - Nodes usually disappear, cheat, or fail
- Better scalability for large number of objects, due to distributed storage
- Routes and object references can be replicated, tolerating failures of nodes
- Complexity and runtime behavior of modern large-scale P2P systems still under research (P2P crawlers)

Routing Overlays

- Routing overlay: Own network over another set of networks
  - Addresses its own nodes on-top-of existing network nodes
  - Overlay network provides full-meshed connectivity graph to application
- **Unstructured P2P Overlay**
  - Peers build random graph starting from *boot peer*
    - Flooding or random graph walk, supports content-based lookup
    - Two-tier approach: Unstructured super-peers, with connected leaf peers
  - Examples: Gnutella, eDonkey, FastTrack, Kazaa, Skype(?)
- Structured P2P Overlay: Assign keys to data items and build graph that maps each key to a particular node

File Sharing With Unstructured P2P Overlays

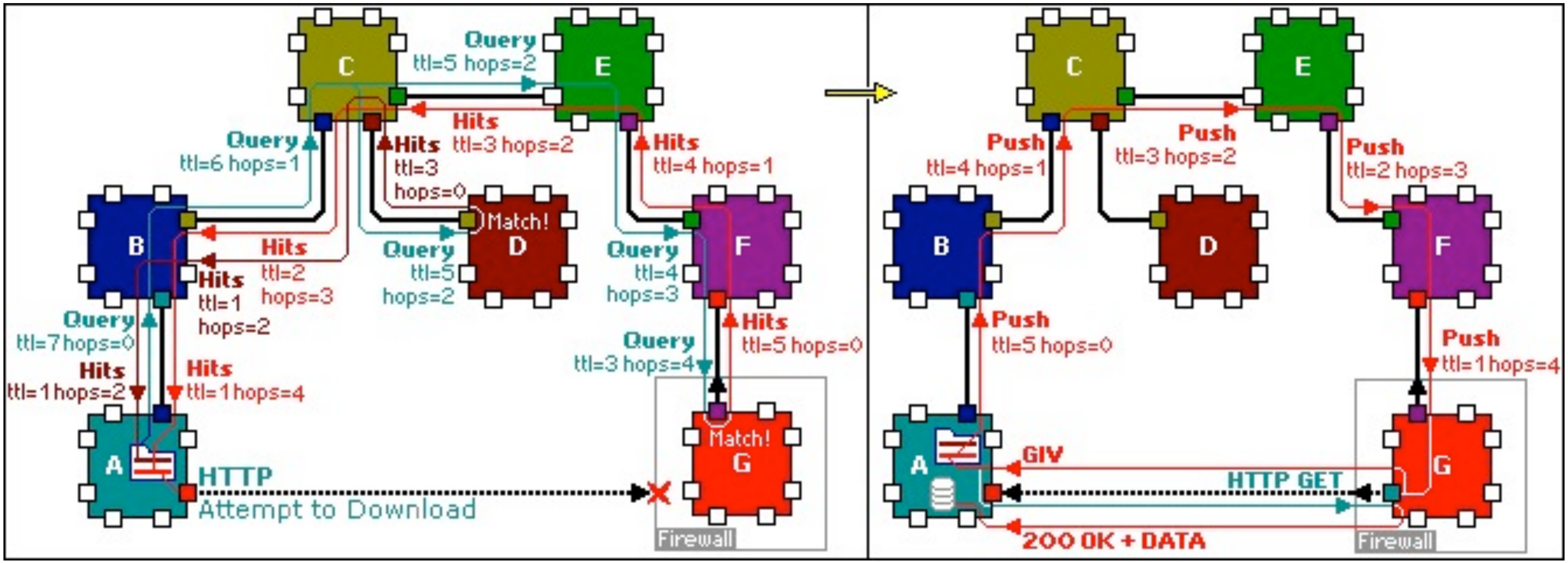
- First Generation File Sharing
  - Napster
  - Central index, distributed data
  - Consideration of hops between peers
- Second Generation File Sharing
  - Freenet, Gnutella, Kazaa, BitTorrent
  - No central entity
  - Improved anonymity
  - Super-peer concepts



Gnutella

- Justin Frankel and Tom Pepper, 2000
  - Simple spreading of search queries over all peers
- Initial neighbor from external source (built-in, IRC, gWebCache, ...)
  - First request for working addresses from other peers
- Discovery of new peers by TTL-restricted multi-hop ping messages
  - Pong message contains IP and port number for further connections
  - Travels original overlay path back (by cached message id on intermediaries)
- Each message typically sent to all known neighbor peers
  - Descriptor ID (to avoid cycles), TTL, Hops field ($TTL_i + Hops_i = TTL_0$), payload
- Periodic one-hop ping messages to all connected peers, support for “bye” message

Gnutella Network



- Based on UDP, typically long search duration and high network load
- Remote peers might only have open Gnutella port -> push request message
- Super peers make up the overlay, usually have permanent internet connection
- Leaf peers have intermittent connectivity, using super peers as proxies

Gnutella

- Discovery can be enhanced by Pong caching
- Queries similar to discovery Pings, meanwhile direct response sending
  - Upon receiving, peer looks up local content if query matches
  - Data transfer outside of overlay protocol
- Lower and upper limit on amount of peer connections
  - Peer is in *connecting* state, *connected* state or *full* state
- Dynamic querying
  - Only gather enough results to satisfy the user (50-200), by starting with low TTL queries
  - Rare matches: Many approximately visited peers, low result count

BitTorrent Protocol

- Bram Cohen, 2001
- Protocol for distributing files
  - Content identified by announce URL, defined in metadata (*.torrent*) file
  - Torrent files available from *Indexer* web sites
  - Downloaders (peers) upload to each other, distribution starts with first downloader that has the complete file
  - *Tracker*: HTTP/HTTPS server providing list of peers for announce URL
    - Subject for closing in recent Copyright law suites
- Metainfo files (*torrents*)
- No focus on content localization, but on efficient content delivery instead

BitTorrent Tracker Protocol

- Torrent file
  - Announce URL(s) for tracker(s)
  - Suggested file name, file length; piece size (typically 512kB) and piece count
  - SHA1 hash values of all pieces
- Tracker HTTP GET request parameters
  - Hash of torrent file information
  - Own (randomly chosen) peer id, includes tag for type of client software ; IP and port (6881 - 6889) the downloader listens on, optional client key
  - Uploaded / downloaded / left bytes for the file(s)
  - Number of demanded peers for download (default 50)
  - Event: Started, completed, stopped

BitTorrent Tracker Protocol

- Tracker response:
  - Human-readable error, or list of peer ID's and IP addresses
  - Timer how long client should wait between subsequent requests
  - Number of peers with completed file (seeders)
  - Number of peers with incomplete file (leechers)
- Number of peers is relevant to protocol overhead, since notification of downloaded pieces is sent to all peers (-> typically not more than 25 peers)
- Peers report status to tracker every 30 minutes, or on status change
  - If peer set size falls below limit (~20), tracker is contacted again
- DHT extension - peer acts as tracker, based on Kademlia DHT (UDP)

BitTorrent Peer Protocol

- Clients maintains state information for each peer
  - choked - client requests will not be answered until unchoke notification
  - interested - remote peer notified interest for blocks, and will start requesting after unchoke
  - Clients needs also to maintain its own interest in peer packets, and if it has choked the remote peer
- Clients start for each peer with „choked“ and „not interested“
  - Download of piece from peer: client claims interest and is „not choked“
  - Upload of piece: peer is „interested“, and client is not choking him
- Client should always notify peers about interest, even in choked state

Peer Wire Protocol

- TCP connection, starts with handshake message from both sides
  - Human-readable protocol header, hash of torrent file, peer ID
  - Handshake for non-served torrent results in connection dropping
  - Trackers send out handshake messages without peerID for NAT-checking
- Protocol messages
  - <length prefix><message id><payload>
  - *keep-alive message*: typically connection drop after 2 minutes
  - *choke, unchoke, interested, not interested messages*
  - *have message*: 4-byte index for downloaded and verified piece
    - Suppression of HAVE messages for pieces the peer already has

Peer Wire Protocol

- *bitfield message*: Optional after handshake, bitmask for available pieces
- *request (piece index, begin, length) message*: Request block of data from specified piece
  - Close connection on big data requests (discussions)
  - Typically 16kB - 32 kB requests, latency vs. slow lines
- *piece message*: Requested payload, with index, begin, and length
- *cancel message*: cancels request for data block
- *port message*: Port number of the peers DHT tracker, to include in own routing table

Choking Algorithm

- Avoid problems with TCP congestion control in case of many connections
- Cap number of simultaneous transfers, while reciprocating peers that allow downloading
  - Un-choke three of the interested peers by best download rate
  - Non-interested peers with better rate are un-choked, in case preferred
  - If client has complete file, use upload rate instead to decide
- Find out if unused peers might behave better
  - Optimistic un-choking: Pick one peer regardless of download rate
- Avoid fibrillation with minimum delay between choke and un-choke (10s)
- *Free riders* are penalized

Rarest First Algorithm

- Downloaders should receive pieces from peers in random order, to avoid partitioning of file content (*random first algorithm*)
  - Might lead to unbalanced distribution of pieces
- *Rarest first algorithm*: Each peer maintains list of number of copies for each piece in available peer set
  - Peer selects next piece to download from rarest pieces
  - Not used in the beginning, to ensure faster initial download (offer needed)
  - Always prioritize requests for blocks of the same piece
- *End Game Mode*: Last blocks usually come in very slowly
  - Last requests are sent to all peers in the set

Other P2P File Sharing Issues

- Anti-Snubbing - avoid to be choked by nearly all peers
  - After 1 minute, upload to according peer is stopped (except optimistic unchoke)
  - Results in more than one optimistic unchoke with limited peer list
- Encryption features in client applications
  - Avoid traffic shaping by ISPs for P2P traffic
  - Meanwhile 10% - 80% of Internet traffic through P2P file sharing (depends on information source)
- Anti-leech strategies
  - Credit point system in eDonkey
  - Special trackers for BitTorrent with minimal upload rate

Structured P2P Overlay

- Provides subject-based lookup, instead of content-based lookup
- Map peer and data identifiers to the same logical ID space
  -> peers get responsibility for their related data
- Key-based routing of client requests to an object through a sequence of nodes
- Knowledge about replica location and ‘nearest’ valid object [Plaxton97]
- Hash value as typical opaque object identifier
- High-level APIs: Distributed Hash Table (DHT) and Distributed Object Location and Routing (DOLR)
- Examples: Pastry, Chord, CAN
- Applications: Digital library, object location in MMOG, spam filtering

Distributed Hash Table (DHT)

- Node makes new data (object) available, together with objectID
  - Overlay must replicate and store data, to be reachable by all clients
  - Replicas stored at all nodes responsible for this objectID
- Client submits request for particular objectID
  - Overlay routes the request to the nearest replica
- Client requests removal of data identified by objectID
  - Overlay must remove associated data from responsible nodes
- Nodes may join or leave
  - Overlay must re-arrange responsibilities for data replicas
- Example: Pastry communication library

Distributed Object Location and Routing (DOLR)

- Objects can be stored anywhere, DOLR layer must maintain mapping between objectID and replica node addresses
  - Replication location decision outside of the routing protocol
- Node makes new objectID available
  - Overlay must recognize this node as responsible for data-derived objectID
- Nodes wants to send request to n objects identified by objectID
  - Overlay forwards request to responsible node(s)
- Example: Tapestry communication framework
- Overlay behavior can be implemented with DHT approach

Programming Interfaces

- Distributed Hash Table Overlay
  - `put(objectID, data)`
  - `remove(objectID)`
  - `value=get(objectID)`
- Distributed Object Location And Routing Overlay
  - `publish(objectID)`
  - `unpublish(objectID)`
  - `sendToObject(msg, objectID, n)`

Pastry

- Since 2001, base framework for several P2P applications
  (Antony Rowstron - Microsoft Research, Peter Druschel - Rice University)
- Each node gets nodeID from strong hash function, based on join time and physical identifier (e.g. IP address or public key)
- Assumes large distance of adjacent nodes for fault tolerance (avalanche effect)
- Subject-based routing: Route message to peer with *nodeId* that is numerically closest to the given subject (==destination id) of the message
  - Final peer is responsible to handle the message content
  - Frameworks differ in proximity metric for message subject and nodeId
- Prefix routing with 128bit IDs in ring overlay
  - Routing of message in O(log N) steps, routing table creation in O(log N)
- Routing scheme typically implemented on UDP without acknowledge

Pastry Application Interface

- Pastry exports:
  - *nodeId = pastryInit(Credentials)* : Local node joins Pastry network
  - *route(msg, key)* : Route given message to *nodeId* which is numerically closest to *key*
  - *send(msg, IP address)* : Send message to specified node through Pastry
- Application exports:
  - *deliver(msg, key)* : Message received for local node (by *route* or *send*)
  - *forward(msg, key, nextId)* : Called before forwarding to next node, application can terminate message or change next node
  - *newLeafs(leafSet)* : Called whenever leaf set changes

Pastry Routing Information Example

- 16bit nodeIds, b=2, L=8
- Entry syntax:
  common prefix with 10233102
  - next digit - rest of nodeId
- Shaded cell shows corresponding digit of present node nodeId‘s
- Rows are managed when nodes join or leave
- Circular ID space: lower neighbor of ID 0 is ID 2^{16}-1

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

(C) Rowstron & Druschel

Pastry Routing Information

- Each node maintains *routing table*, *neighborhood set* and *leaf set*
  - IDs as hexadecimal values, one row per prefix length
  - Entry keys in row match prefix length digits, but not the next one
  - Entry contains one of the possible IP addresses matching the according prefix length, under consideration of network proximity (might be empty)
  - Length of row (2^b-1) depends on configuration parameter b, trade-off between routing table size and maximum number of hops
- Neighborhood set contains nodeIds and IP addresses of closest nodes
  - Normally not used, good for locality properties
- Leaf node set contains L/2 numerically closest smaller and L/2 larger nodeIDs

Routing Algorithm in Pastry

- Incoming message for node
  - Check if destination key falls in the range of the leaf set, then forward directly to destination node
  - Forward message to a node that shares a common prefix with the key by at least one more digit
    - If entry is empty or node not reachable, forward to node which shares same prefix length as current node, and is numerically closer to destination key
    - Best-possible destination is reached if leaf set has no better candidate
- Routing always converges, since each step takes message to a node with longer prefix share, or smaller numerical distance

Pastry Node Arrival

- New node X knows nearby Pastry node A by some mechanism (e.g. multicast)
- Node asks A to route special join message with ID of X as destination
  - Routed to node Z, which is numerically closest to X
  - All nodes on the path send their state tables back to X
  - Neighborhood of A is initial neighborhood of X, due to proximity promise
  - Leaf set of Z is initial leaf set of X
  - Row zero in routing table is independent of own ID -> take from A
  - B has valuable row for prefix length 1, C for length 2, ...
- Resulting information forwarded to leaf set, routing entries and neighborhood
- Data exchange with timestamps, to detect in-between changes

Pastry Node Departure

- Neighbor detects failed node in the leaf set
  - Asks live node with largest index on side of the failed node for its leaf set, which partially overlaps with present node‘s leaf set
  - From new ones, alive node is added to present nodes leaf set
- Each node repairs it‘s leaf set lazily, until L/2 nodes failed simultaneously
  - Unlikely event due to demanded diversity of nodes with adjacent numbers
- Failed node in the routing table does not stop routing, but entry must be replaced
  - Ask other nodes in same row (or in other rows) for entry with according prefix
- Periodic check of neighborhood, in case ask other neighbors for their values and add the one with the shortest distance

PAST

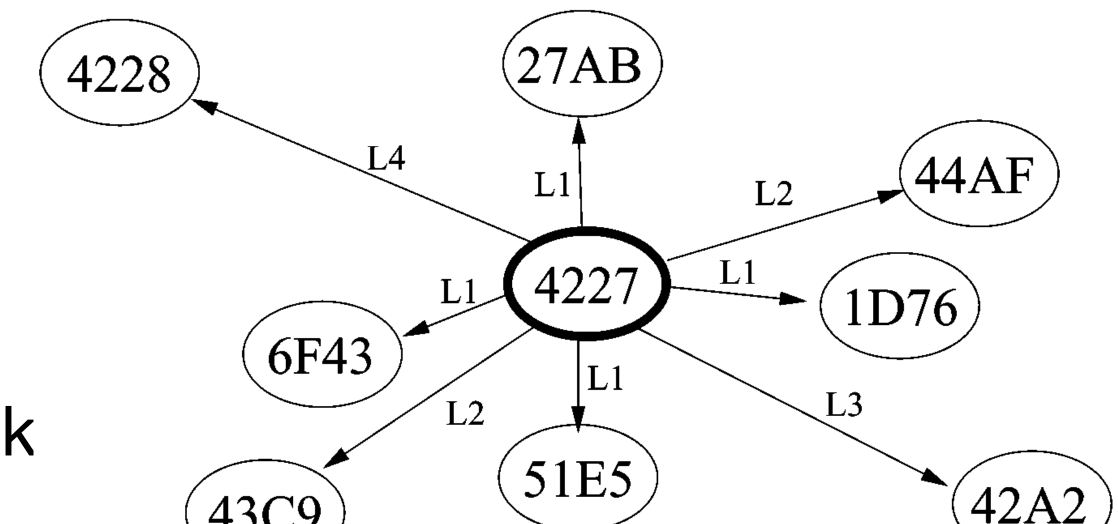
- PAST: Distributed replicating file system, based on Pastry
  - *fileId* as hash of file name, client certificate and random salt
  - File certificate: *fileId*, file content hash, creation date
  - File and certificate routed via Pastry, with *fileId* as destination
  - Closest node accepts responsibility after certificate checking
    - Forwards insert request to other closest nodes
- Lookup finds nearest replica due to proximity consideration of Pastry
- Replica diversion: Balance remaining free space in leaf set - allow to choose other members than the nearest ones in the leaf set
- File diversion: Balancing storage space in nodeId space - vary salt in error case

Tapestry Overlay Network

- 2001, Zhao et. al.
- Nodes and application endpoints with ID‘s
  - 160bit values, evenly distributed, e.g. by using same hash algorithm
- Every message contains application-specific identifier (similar to port number)
  - One large Tapestry network is encouraged, since efficiency increases
- DOLR approach, routing of messages to endpoints by opaque identifiers
  - PublishObject (objectID, application ID) - best effort, no confirmation
  - UnpublishObject (objectID, application ID) - best effort
  - RouteToObject(objectID, application ID) - route message to object
  - RouteToNode(Node, application ID, exact destination match)

Routing and Object Location

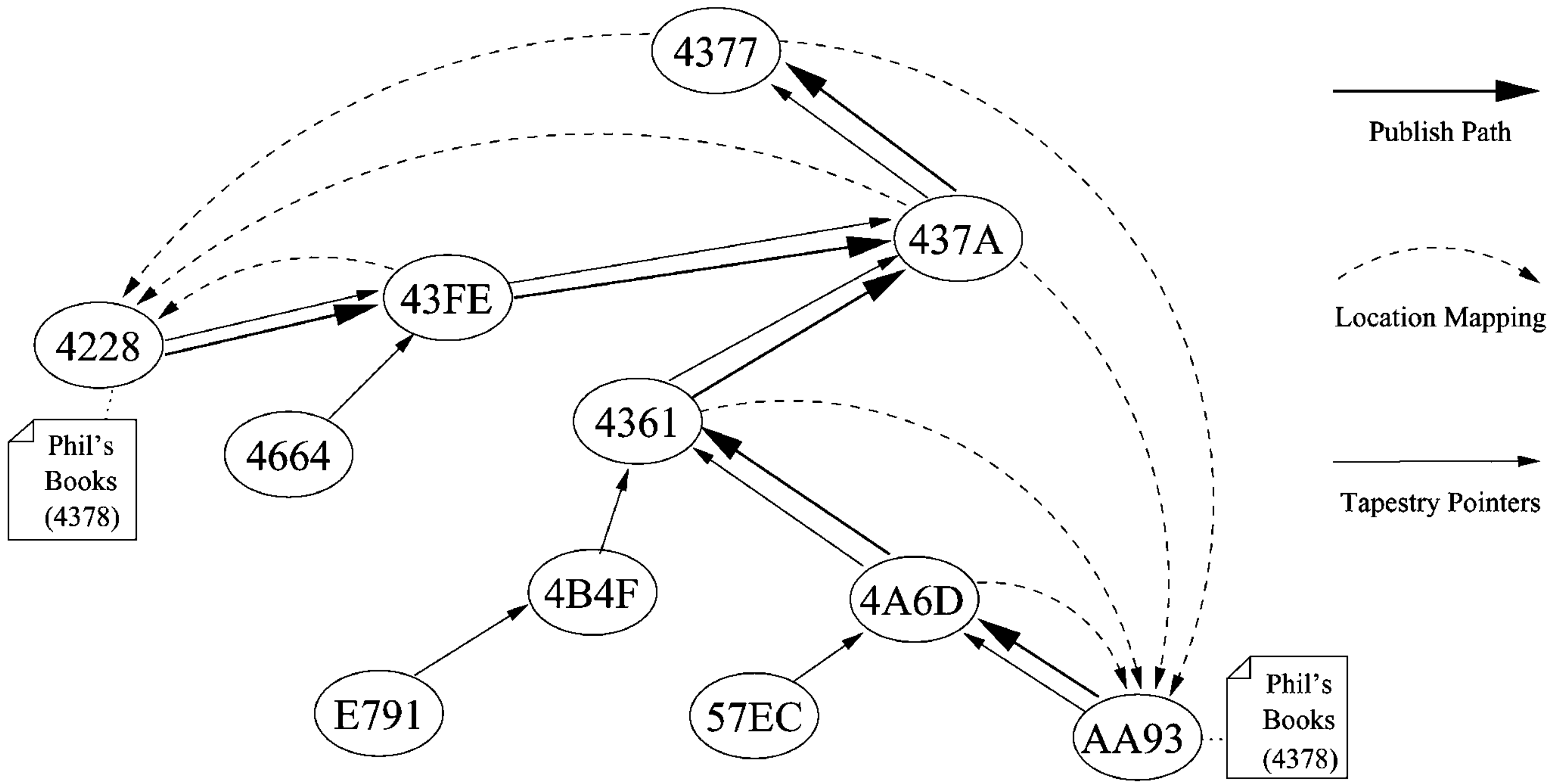
- Each identifier is mapped to a live node (identifiers root)
    - If node ID is the same as the identifier, this one becomes the root
- Each nodes maintains table of outgoing neighbor links
    - Common matching prefix, higher levels match more digits, increasing prefix size from hop to hop
    - Again similar to classless inter-domain routing (CIDR) for IP addresses
- Non-existent IDs are mapped to some live node (‚close' digit)
- Backup links with same prefix as neighbor link



Tapestry Object Publication

- Each identifier has a root node
  - Participants publish objects by periodically routing ‚publish‘ messages towards the root node
  - Each node along the path stores object key and publisher IP
  - Each replica is announced in the same way - nodes store ordered replica list based on own network latency to publisher
- Objects are located by routing a message towards the root node
  - Each node along the path checks mapping and redirects accordingly
  - Convergence of nearby paths heading to the same direction
- Client locates object by routing a message to the route node
  - Each node on path checks for cached pointer and re-directs to server

Object Announcement



Overlay Management in Tapestry

- Node insertion
  - Multicast message to all nodes sharing the same prefix
  - May take over to be root node for some objects
- Node departure
  - Transmits replacement node for each level
  - Node failure is handled by backup links on other nodes