T-110.5140 Network Application Frameworks and XML

Distributed Hash Tables (DHTs)

1.2.2010

Tancred Lindholm, Sasu Tarkoma

Contents

- Motivation
- Terminology
- Distributed Hash Tables (DHTs)
- Overlays
  - Clusters and wide-area
- Non-filesharing applications for DHTs
  - Internet Indirection Infrastructure (i3)
  - Delegation Oriented Architecture (DOA)

DHT Motivation

- Challenges in the wide-area
  - Scalability
  - Increasing number of users, requests, files, traffic
  - Resilience: more components -> more failures
  - Management: intermittent resource availability -> complex management
- DHTs promise to address these

DHT Motivation (cont.)

- Directories are needed
  - Name resolution & lookup
  - Mobility support with fast updates
- Required properties
  - Fast updates
  - Scalability
  - Reliability
- DNS has limitations
  - Update latency
  - Administration
- Alternative: DynDNS
  - Still centralized management
  - Inflexible, reverse DNS?

Terminology

- Peer-to-peer (P2P)
  - Different from client-server model
  - Each peer has both client/server features
- Distributed Hash Tables (DHT)
  - An algorithm for creating efficient distributed hash tables (lookup structures)
  - Used to implement overlay networks
- Overlay networks
  - Routing systems that run on top of another network, such as the Internet
- Typical features of P2P / overlays
  - Scalability, resilience, high availability, and they tolerate frequent peer connections and disconnections

Peer-to-peer in more detail

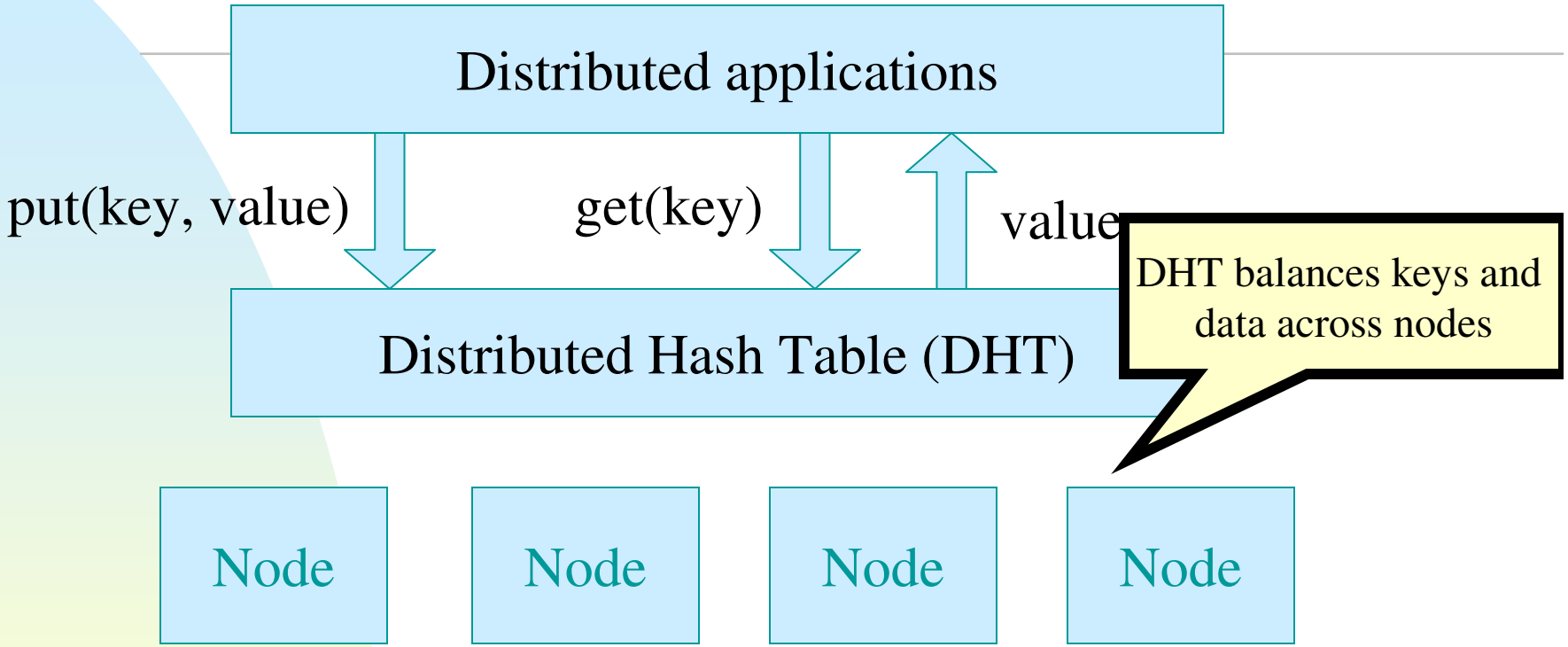
- A P2P system is distributed
  - No centralized control
  - Nodes are symmetric in functionality
- Comparatively large faction of nodes are unreliable
  - Nodes come and go
- P2P enabled by evolution in data communications and algorithms (DHT)
- P2P driven by mostly illegal file sharing?
- Current challenges:
  - Security (zombie networks,trojans), IPR issues
- P2P systems are decentralized overlays

Evolution of P2P systems

- Started from centralized servers
  - Napster
    - Centralized directory, transfer P2P
    - Central directory single point of failure
- Second generation used flooding
  - Gnutella
    - Local directory for each peer
    - Search by flooding network with queries
    - High cost, worst-case O(N) messages
- Research systems use DHTs
  - Chord, Tapestry, CAN, ..
  - Decentralization, scalability
- Data lookup by fixed key in file sharing apps
  - e.g. BitTorrent DHT extension
  - not suited for free-form queries

DHT interfaces

- DHTs offer typically two functions
  - put(key, value)
  - get(key) --> value
  - optionally delete(key)
- Supports wide range of applications
  - Similar interface to UDP/IP
    - Send(IP address, data)
    - Receive(IP address) --> data
- No restrictions are imposed on the semantics of values and keys
- Commonly key of value = hash(value)
- Key/value pairs are persistent and global


Distributed applications
put(key, value)
get(key)
value
DHT balances keys and data across nodes
Distributed Hash Table (DHT)
Node
Node
Node
Node

Some DHT applications

- Storage-related
  - File sharing
  - Web caching
  - Censor-resistant data storage
  - Backup storage
  - Web archive
- Event notification
- Naming systems
- Query and indexing
- Communication primitives

Examples in this lecture

DHT Example 1: Chord

- Chord is an overlay algorithm from MIT
  - Stoica et. al., SIGCOMM 2001
- Chord is a lookup structure (a directory)
  - Idea resembles binary search
- Support for rapid joins and leaves
  - Churn
  - Maintains routing tables

Chord

- Uses consistent hashing to map keys to nodes
  - Consistent hashing means any node computes same node for a given value (compare to using distributed protocols to obtain an id)
  - Keys are hashed to m-bit identifiers
  - Nodes have m-bit identifiers
    - IP-address is hashed
  - SHA-1 is used as the baseline algorithm

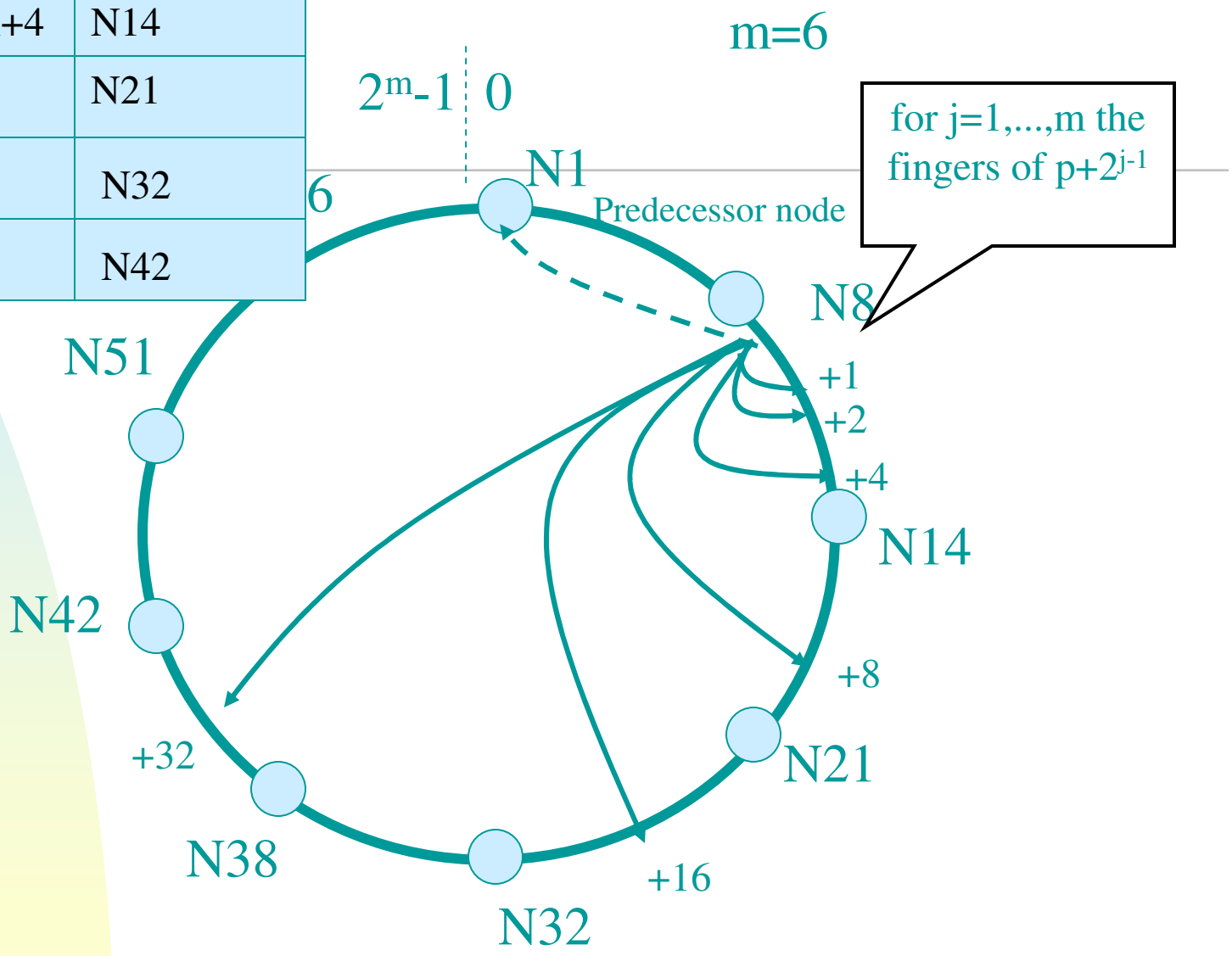
Chord routing I

- A node has a well determined place within the ring
- Identifiers are ordered on an identifier circle modulo 2^m ("clock math", e.g. 5+7 mod 2^3 = 4)
  - The Chord ring with m-bit identifiers
- A node has a predecessor and a successor
- A node stores the keys between its predecessor (inclusive) and itself (exclusive)
  - The (key, value) is stored on the successor node of key
- A routing table (finger table) keeps track of other nodes
- NOTE: Slightly different descriptions in the literature (range endpoint behavior), the idea is the same though

Finger Table

- Each node maintains a routing table with at most m entries
- Each node also knows its predecessor
- The i:th entry of the table at node n contains the identity of the first node, s, that succeeds n by at least 2^{i-1} on the identifier circle
- s = successor(n + 2^{i-1})
  - The i:th finger of node n

Finger	Maps to	Real node
1,2,3	x+1,x+2,x+4	N14
4	x+8	N21
5	x+16	N32
6	x+32	N42

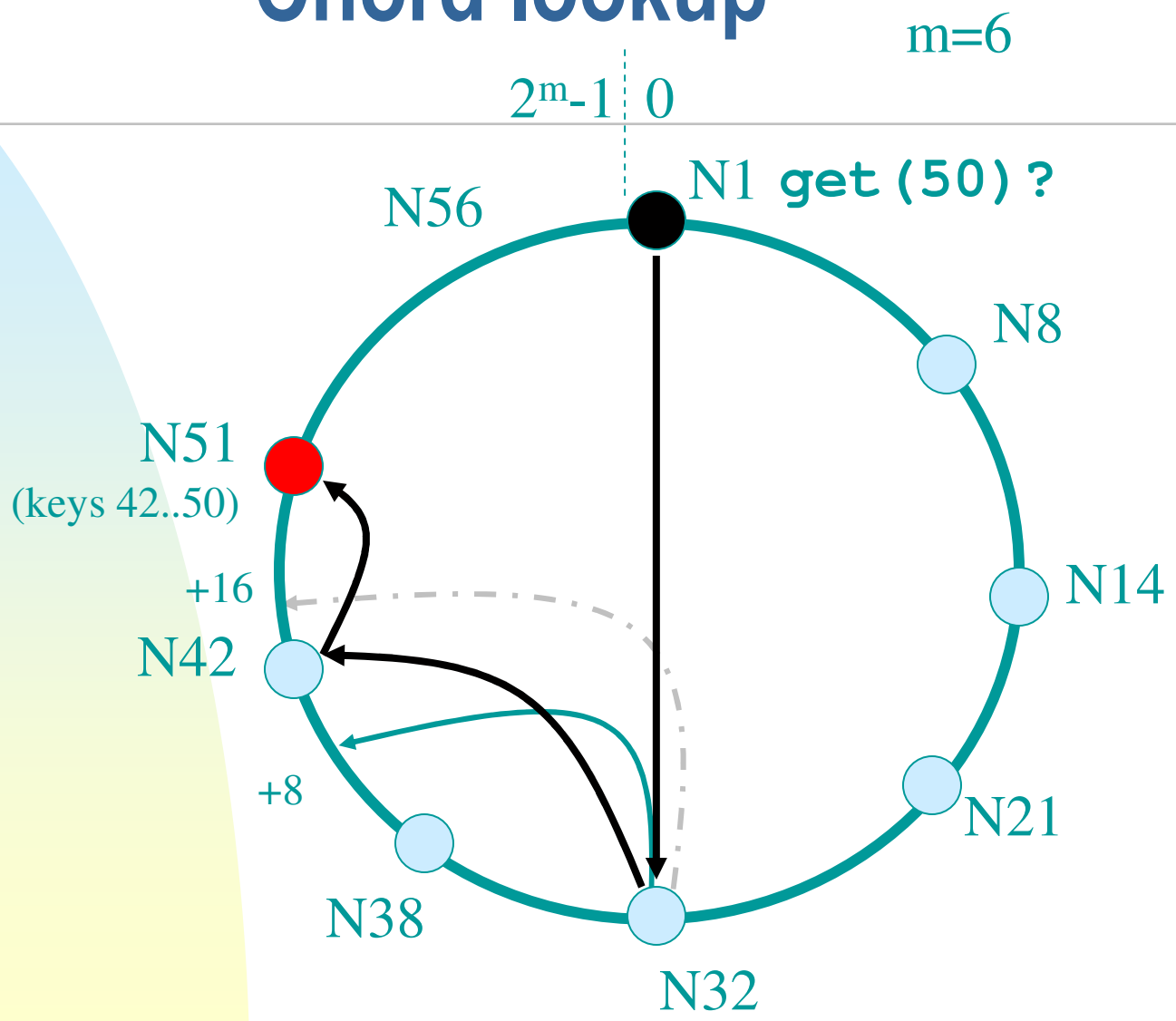


E.g. N32 responsible for keys 21 to 31

Chord routing II

- Routing steps
  - check whether the key k is found between n and the successor of n → successor(n)
  - if not, forward the request to the closest finger preceding k
  - Each knows a lot about nearby nodes and less about nodes farther away
- The target node will be eventually found
  - Find right halfcircle
  - Then right quadrant
  - Then next 1/8th
  - etc.

Chord lookup



Invariants

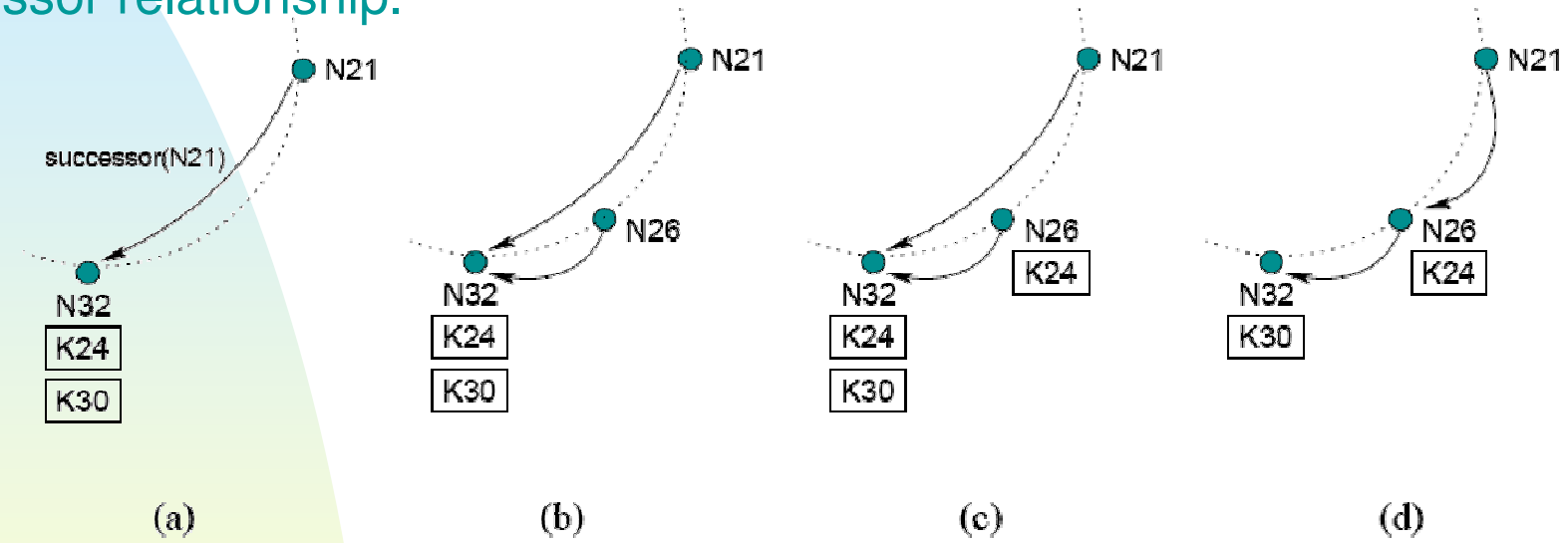
- Two invariants:
  - Each node's successor is correctly maintained.
  - For every key k, node successor(k) is responsible for k.
- A node stores the keys between its predecessor and itself
  - The (key, value) is stored on the successor node of key

Join

- A new node *n* joins
- Needs to know an existing node *n'*
- Three steps
  - 1. Initialize the predecessor and fingers of node
  - 2. Update the fingers and predecessors of existing nodes to reflect the addition of *n*
  - 3. Notify the higher layer
- Leave uses steps 2. (update removal) and 3. (relocate keys)

Chord Join Overview

Node 26 joins the system between nodes 21 and 32. The arcs represent the successor relationship.



(a) Initial state: node 21 points to node 32;
(b) node 26 finds its successor (i.e., node 32) and points to it;
(c) node 26 copies all keys less than 26 from node 32;
(d) the stabilize procedure updates the successor of node 21 to node 26.

Source: Stoica et al., *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*

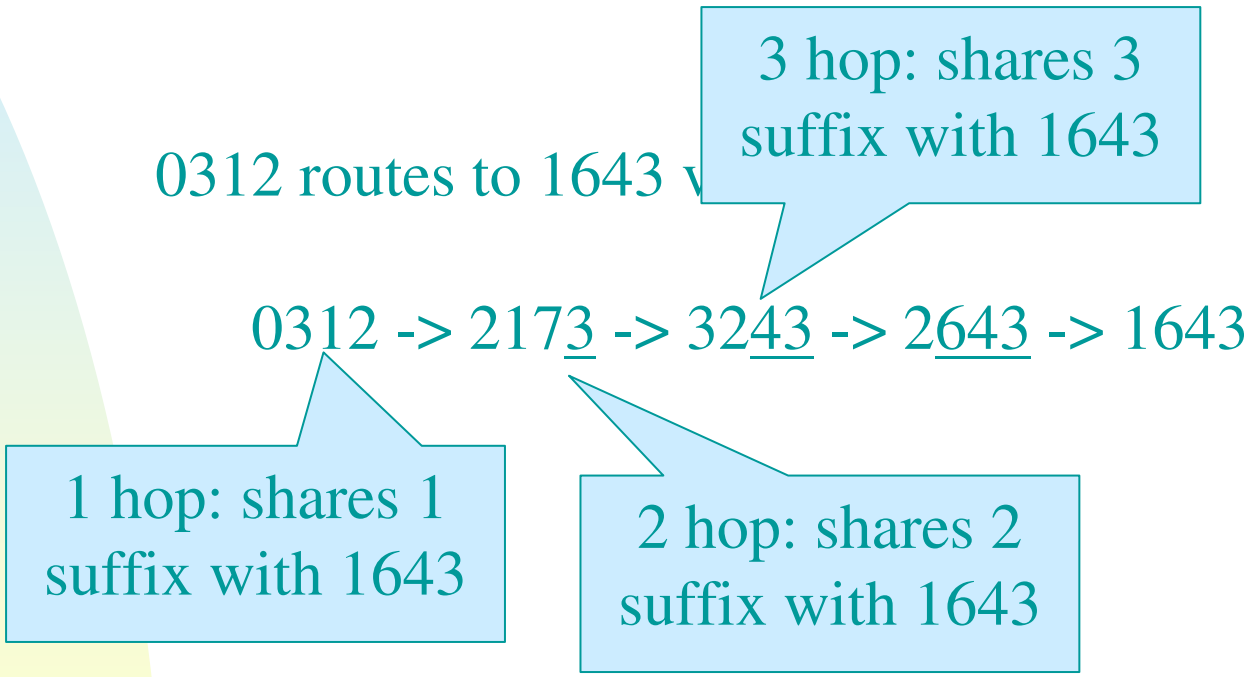
Chord Properties

- Each node is on average responsible for K/N keys (K is the number of keys, N is the number of nodes)
- When a node joins or leaves the network only O(K/N) keys will be relocated
- Lookups take O(log N) messages
- To re-establish routing invariants after join/leave $O(\log^2 N)$ messages are needed

DHT Example 2: Tapestry

- DHT developed at UCB
  - Zhao et. al., UC Berkeley TR 2001
- Used in OceanStore
  - Secure, wide-area storage service
- Tree-like geometry
- Suffix - based hypercube
  - 160 bits identifiers
- Suffix routing from A to B
  - hop(h) shares suffix with B of length digits
- Tapestry Core API:
  - publishObject(ObjectID,[serverID])
  - routeMsgToObject(ObjectID)
  - routeMsgToNode(NodeID)

Suffix routing



Rounting table with b*logb(N) entries

Entry(i,j) – pointer to the neighbour ...ji (i = suffix)

Pastry I

- A DHT based on a circular flat identifier space like Chord
- Prefix-routing
  - Message is sent towards a node which is numerically closest to the target node
  - Procedure is repeated until the node is found
  - Prefix match: number of identical digits before the first differing digit
  - Prefix match increases or numerical distance decreases by every hop
  - Customizable with different distance metrcis
- Similar performance to Chord

Pastry II

- Routing a message
  - If the list of closeby nodes ("leaf set") has the key --> send to node directly
  - else send to the identifier in the routing table with the longest common prefix (longer then the current node)
  - else query leaf set for a numerically closer node with the same prefix match as the current node
  - Used in event notification (Scribe)

Security Considerations

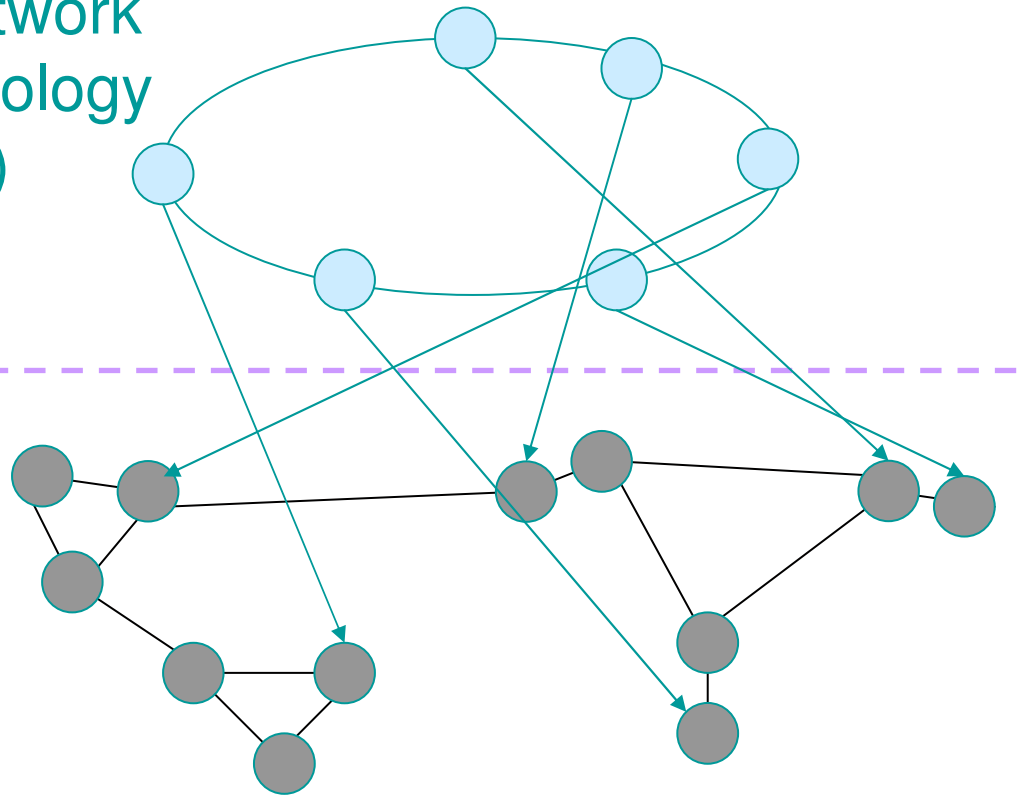
- Malicious nodes
  - Attacker floods DHT with data
  - Attacker returns incorrect data
    - self-authenticating data (hash of value is key)
  - Attacker denies data exists or supplies incorrect routing info
- Basic solution: using redundancy
  - k-redundant networks
- What if attackers have quorum, i.e. nodes strategically placed to defeat redundancy?
  - Need a way to control creation of node Ids
  - Solution: secure node identifiers
    - Use public keys

Overlay Networks

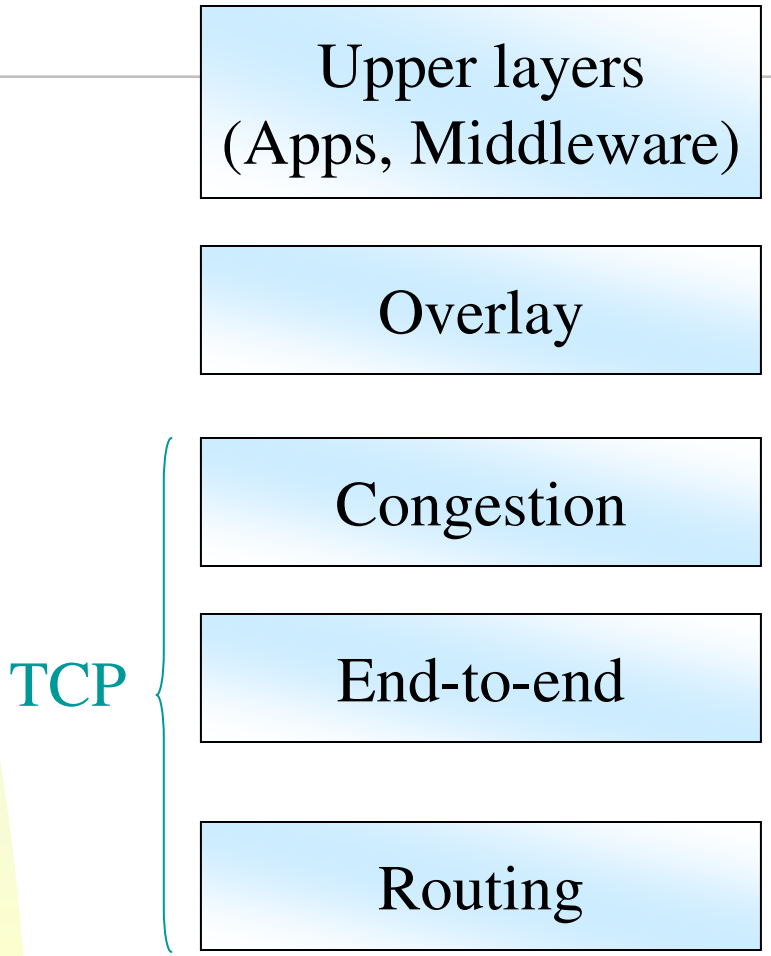
- Origin in Peer-to-Peer (P2P)
- Builds upon Distributed Hash Tables (DHTs)
- Easy to deploy
  - No changes to routers or TCP/IP stack
  - Typically on application layer
- Overlay properties
  - Resilience
  - Fault-tolerance
  - Scalability

Overlay Networks (cont.)

Overlay Network
Logical Topology
(e.g. Chord)



Node "real" topology in IP network


Upper layers
(Apps, Middleware)
Overlay
Congestion
TCP
End-to-end
Routing

Distributed Data Structures (DSS)

- DHTs are an example of DSS
- DHT algorithms are available for clusters and wide-area environments
  - They are different!
- Cluster-based solutions
  - Ninja
  - LH* and variants
- Wide-area solutions
  - Chord, Tapestry, ..
  - Flat DHTs, peers are equal
  - Maintain a subset of peers in a routing table

Cluster vs. Wide-area

- Clusters are
  - single, secure, controlled, administrative domains
  - engineered to avoid network partitions
  - low-latency, high-throughput SANs
  - predictable behaviour, controlled environment
- Wide-area
  - **Unstable configuration: nodes join & leave**
  - Heterogeneous networks
  - Unpredictable delays and packet drops
  - Multiple administrative domains
  - Network partitions possible

Wide-area requirements

- Easy deployment
- Scalability to millions of nodes and billions of data items
- Availability
  - Copes with routine faults
- Self-configuring, adaptive to network changes
- Takes locality into account

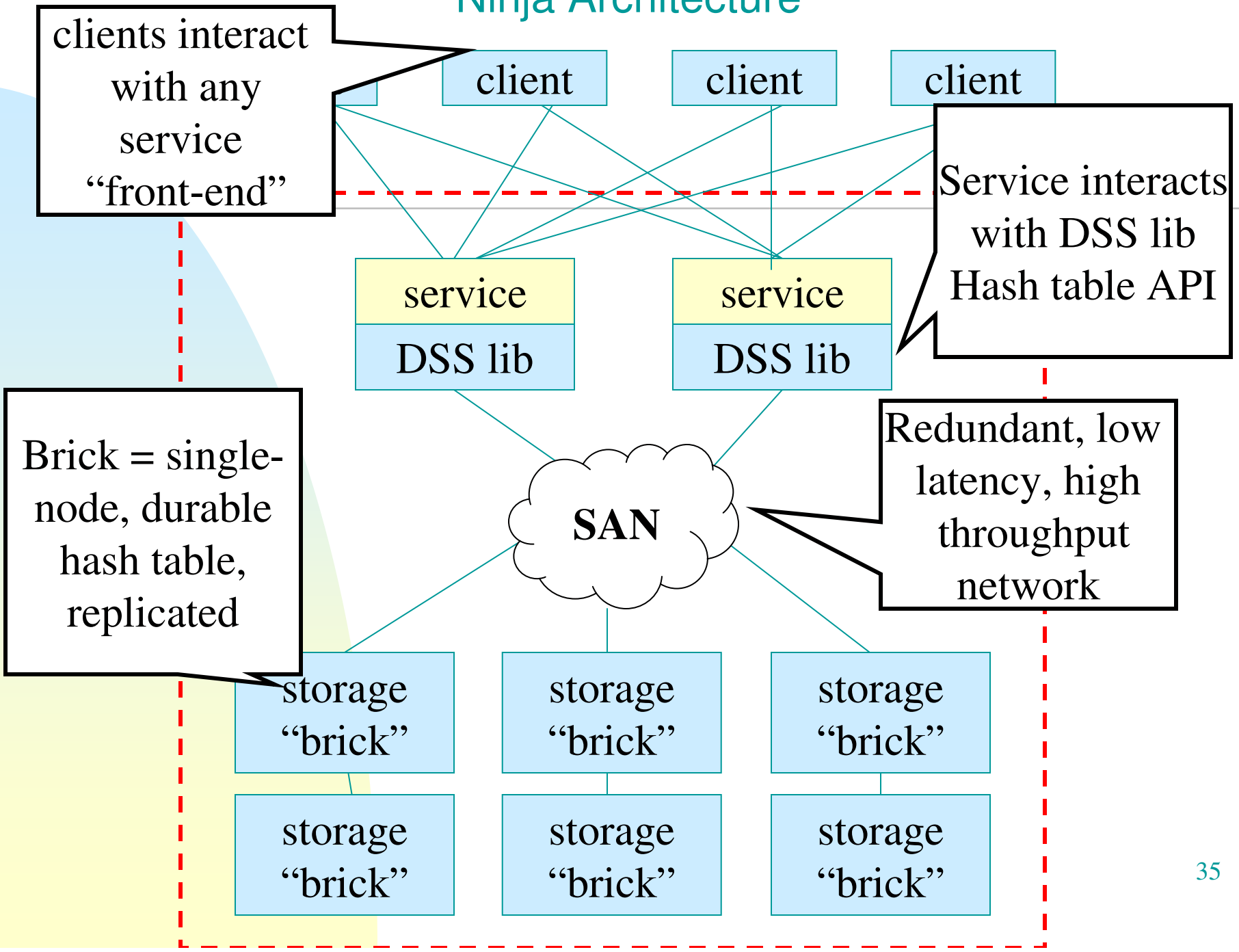
Distributed Data Structures (DSS) for Clusters

- Ninja project (UCB)
  - New storage layer for cluster services
  - Partition conventional data structure across nodes in a cluster
  - Replicate partitions with replica groups in cluster
    - Availability
  - Sync replicas to disk (durability)
- Other DSS for data / clusters
  - Amazon Dynamo Storage
    - Dynamo: Amazon's Highly Available Key-Value Store (SOSP 2007)
  - LH* Linear Hashing for Distributed Files
  - Redundant versions for high-availability

Cluster-based Distributed Hash Tables (DHT) in Ninja

- Directory for non-hierarchical data
- Several different ways to implement
- A distributed hash table
  - Consists of “bricks" which each maintains a partial map
  - Adding bricks increases capacity/ performance
- Resilience through parallel, unrelated mappings

Ninja Architecture



Applications for DHTs

- DHTs are used as a basic building block for an application-level infrastructure
  - Internet Indirection Infrastructure (i3)
    - New forwarding infrastructure based on Chord
  - DOA (Delegation Oriented Architecture)
    - New naming and addressing infrastructure based on overlays

Internet Indirection Infrastructure (i3)

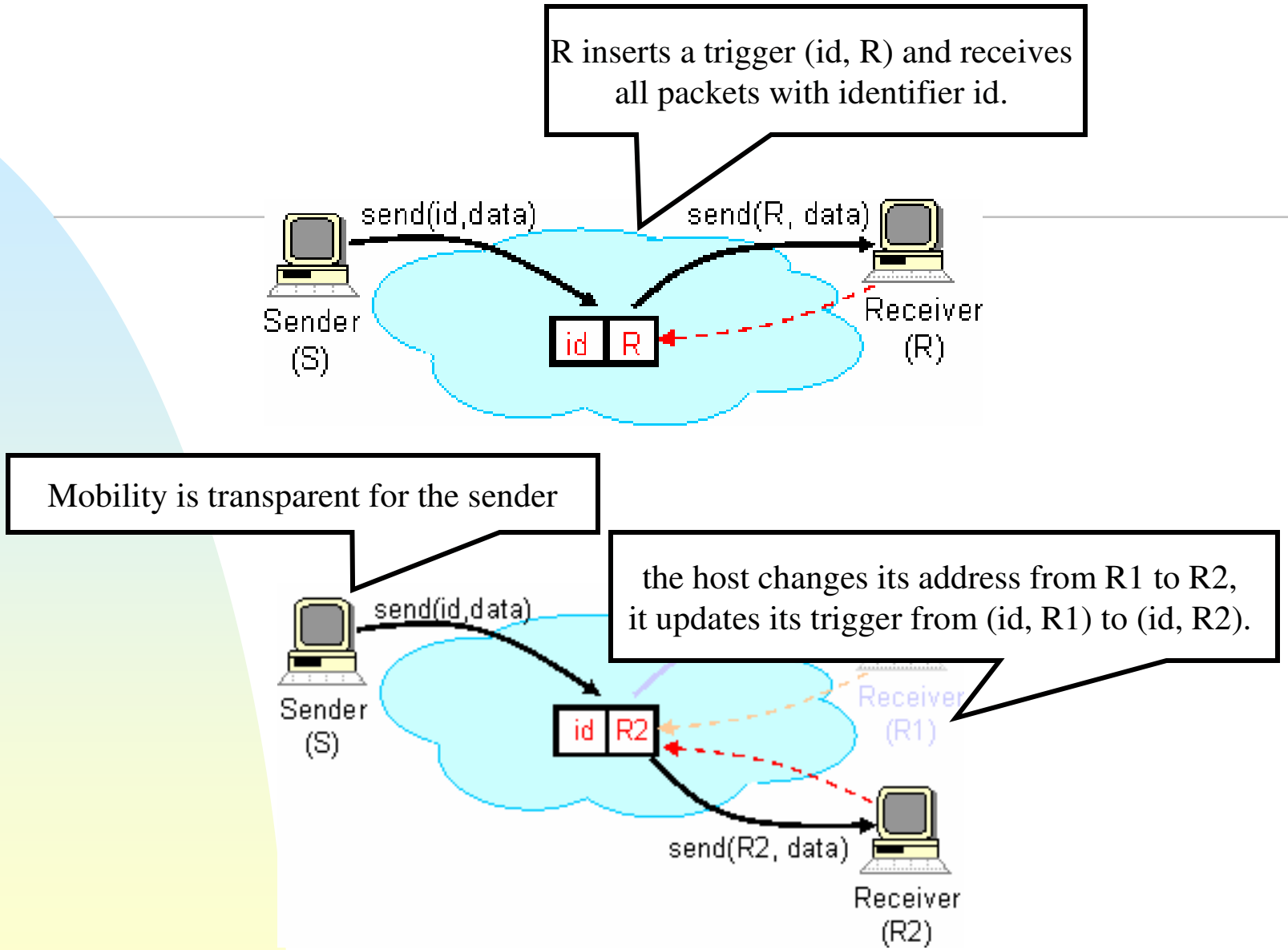
- A DHT - based overlay network
  - Based on Chord
- Aims to provide more flexible communication model than current IP addressing
- Decouples sender from receiver by introducing indirection point
- One proposal to fix some fundamental problems in the Internet

i3 II

- i3 packets are sent to identifiers
- each identifier is routed to the i3 node responsible for that identifier
- the node maintains triggers that are installed by receivers
- when a matching trigger is found the packet is forwarded to the receiver

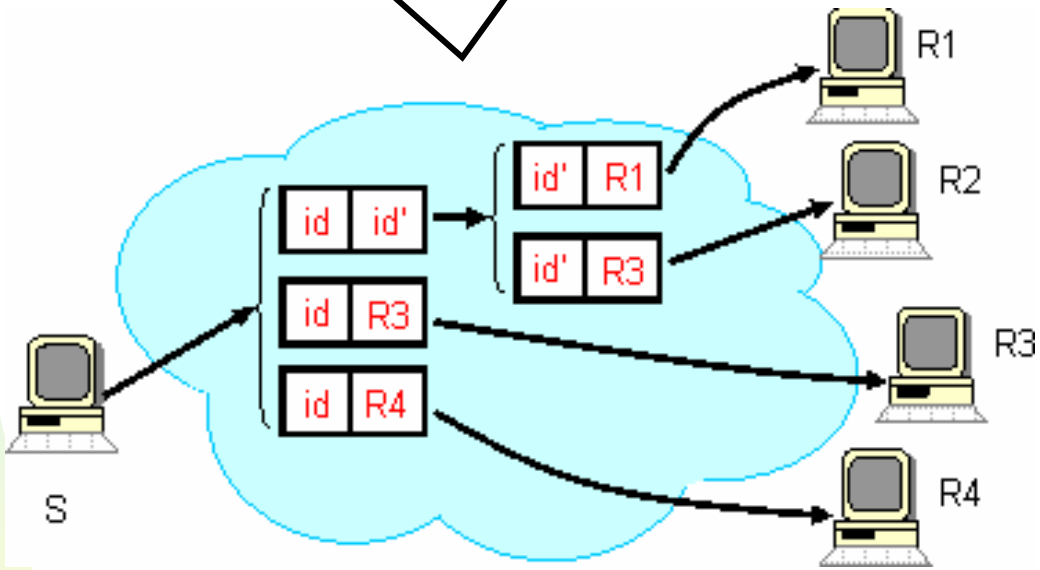
i3 III

- An i3 identifier may be bound to a host, object, or a session
- i3 has been extended to
  - Allows end hosts to control the placement of rendezvous-points (indirection points) for efficient routing and handovers
  - This is Robust Overlay Architecture for Mobility (ROAM)
- Legacy application support
  - user level proxy for encapsulating IP packets to i3 packets

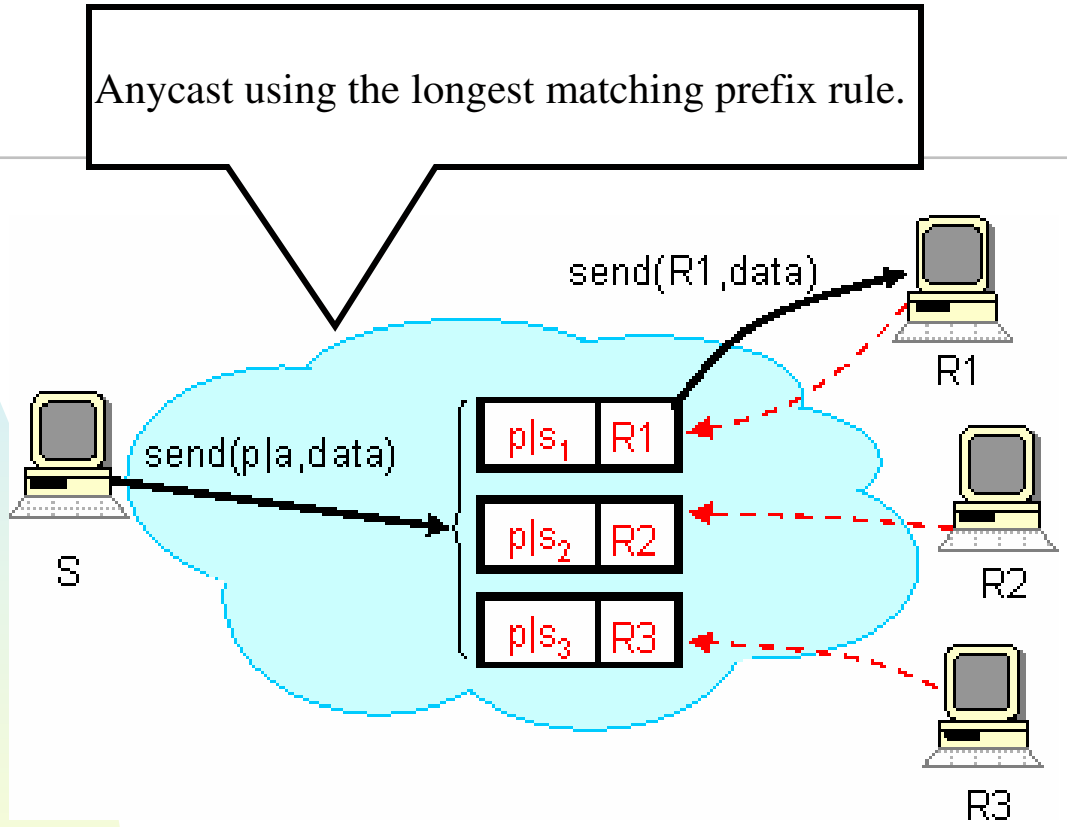


Source: http://i3.cs.berkeley.edu/

A multicast tree using a hierarchy of triggers



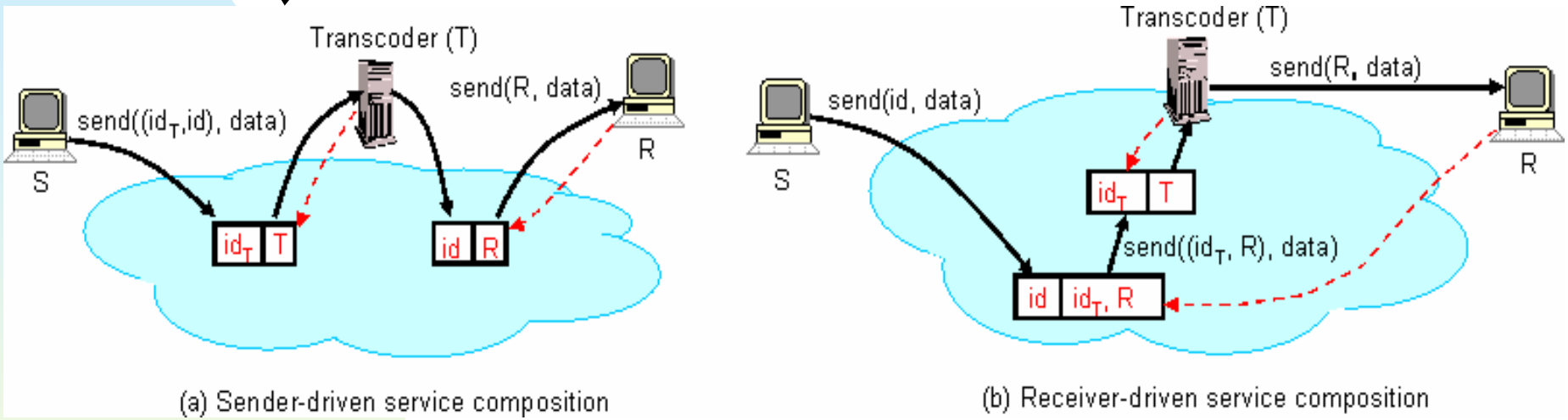
Source: http://i3.cs.berkeley.edu/



Suffixes indirectly give preference to receivers in group

Source: http://i3.cs.berkeley.edu/

Sender-driven service composition using a stack of identifiers



Receiver-driven service composition using a stack of identifiers

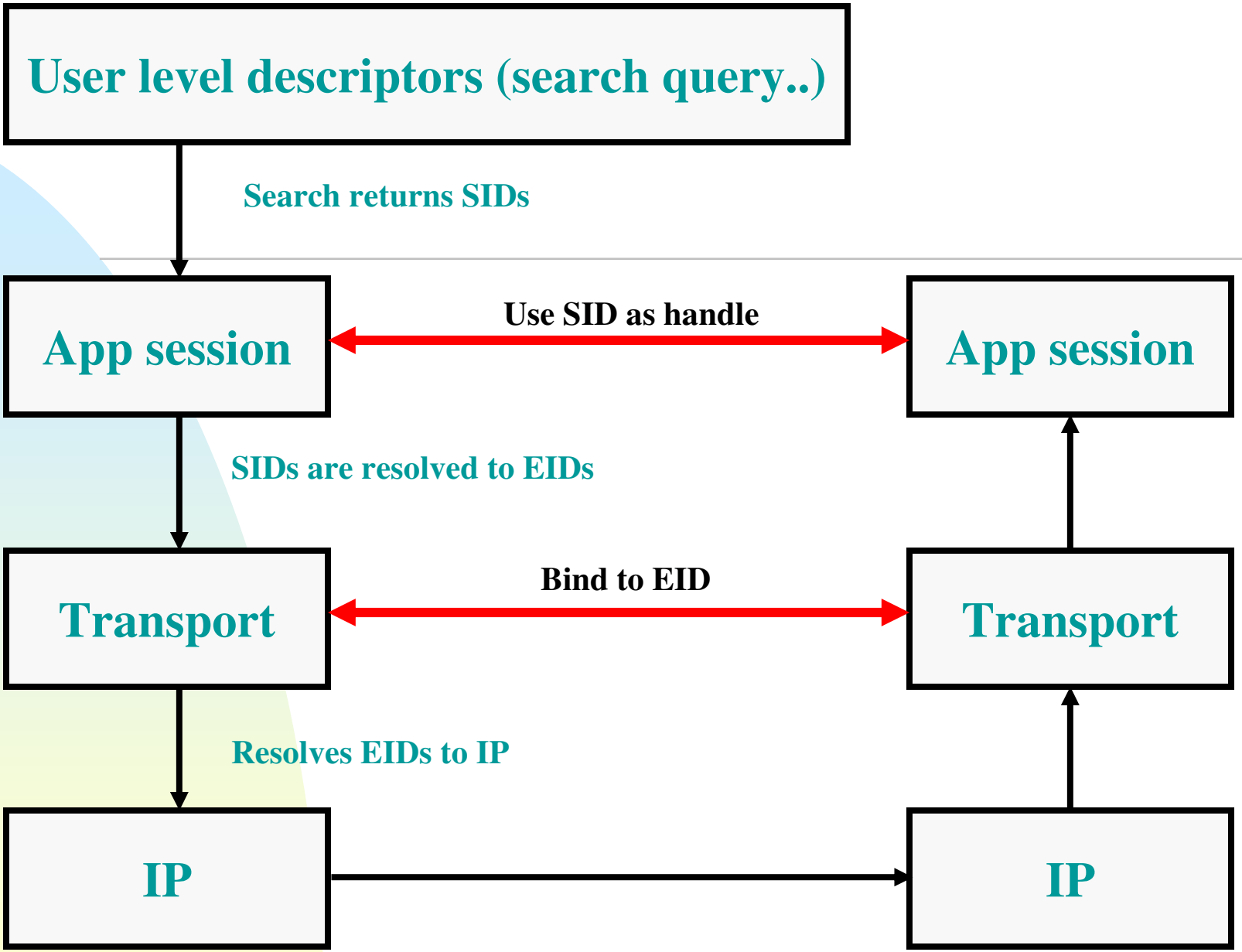
Source: http://i3.cs.berkeley.edu/

Layered Naming Architecture

- Presented in paper:
  - *A Layered Naming Architecture for the Internet, Balakrishnan et al. SIGCOMM 2004*
- What is wrong with http://images.org/tux.jpg ?
  1. OK: names an image
  2. BAD: tells us HOW and WHERE to reach it
  3. BAD: WHERE (DNS->IP) is bound to Inet toplogy
- Service Identifiers (SIDs) are host-independent data or serice names (fix point 2)
- End-point Identifiers (EIDs) are location-independent host names (fix point 3)
- E.g.
  1. Resolve URL to SID = persistent name of data
  2. Resolve SID to transport and EID = how and from whom to get data
  3. Resolve EID to IP = address to data

Layered Naming Architecture

- Protocols bind to names and resolve them
  - Applications use SIDs as handles
- SIDs and EIDs should be flat
  - Stable-bame principle: A stable name should not impose restrictions on the entity it names
- Inspiration: HIP + i3 + Semantic Free Referencing
- Prototype: Delegation Oriented Architecture (DOA)


User level descriptors (search query..)
Search returns SIDs
App session
Use SID as handle
App session
SIDs are resolved to EIDs
Transport
Bind to EID
Transport
Resolves EIDs to IP
IP
IP

DOA cont.

- DOA header is located between IP and TCP headers and carries source and destination EIDs
- The mapping service maps EIDs to IP addresses **or 1 or more EIDS,** which are intermediaries
- This allows the introduction of various middleboxes to the routing of packets
  - Service chain, end-point chain
- Outsourcing of intermediaries
  - Ideally clients may select the most useful network elements to use

OpenDHT

- A publicly accessible distributed hash table (DHT) service
- Discontinued in 2009
- OpenDHT ran on a collection of 200 - 300 nodes on PlanetLab.
- Client do not need to participate as DHT nodes.
- Used bamboo DHT
- www.opendht.org
- OpenDHT: A Public DHT Service and Its Uses. Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Proceedings of ACM SIGCOMM 2005, August 2005.

Summary

- Mobility and multi-homing require directories
  - Scalability, low-latency updates
- Overlay networks have been proposed
  - Searching, storing, routing, notification,..
  - Lookup (Chord, Tapestry, Pastry), coordination primitives (i3), middlebox support (DOA)
  - Logarithmic scalability, decentralised,…
- Many applications for overlays
  - Lookup, rendezvous, data distribution and dissemination, coordination, service composition, general indirection support

Discussion Points

- Why do most popular P2P programs have simple algorithms?
- How secure should P2P / overlays be?
- How much will DHTs be deployed in commercial software?
- Will DHTs be part of future core Internet?
- Will there be more name layers (outside single apps) or are such fundamental changes unrealistic?
  - the joke is that DOA = Dead on Arrival

T-110.5140 Network Application Frameworks and XML

Additional material: Chord Joins and Leaves

Invariants

- Two invariants:
  - Each node's successor is correctly maintained.
  - For every key k, node successor(k) is responsible for k.
- A node stores the keys between its predecessor and itself
  - The (key, value) is stored on the successor node of key

Join

- A new node *n* joins
- Needs to know an existing node *n'*
- Three steps
  - 1. Initialize the predecessor and fingers of node
  - 2. Update the fingers and predecessors of existing nodes to reflect the addition of *n*
  - 3. Notify the higher layer software and transfer keys
- Leave uses steps 2. (update removal) and 3. (relocate keys)

1. Initialize routing information

- Initialize the predecessor and fingers of the new node *n*
- *n* asks *n'* to look predecessor and fingers
  - One predecessor and *m* fingers
- Look up predecessor
  - Requires log (N) time, one lookup
- Look up each finger (at most *m* fingers)
  - log (N), we have Log N * Log N
  - $O(Log^2 N)$ time

Steps 2. And 3.

- 2. Updating fingers of existing nodes
  - Existing nodes must be updated to reflect the new node (any any keys that are transferred to the new node)
  - Performed counter clock-wise on the circle
    - Algorithm takes i:th finger of *n* and walks in the counter-clock-wise direction until it encounters a node whose i:th finger precedes *n*
    - Node *n* will become the i:th finger of this node
  - $O(Log^2 N)$ time
- 3. Transfer keys
  - Keys are transferred only from the node immediately following *n*

References

- http://www.pdos.lcs.mit.edu/chord/