

FS18

DISTRIBUTED HASH TABLES AND MECHANISMS

Detailed Concepts and Algorithms

T. Bocek

thomas.bocek@hsr.ch

Agenda

- **Distributed Systems in the News**
- **Key Concepts - DHT Implementations**
- **Bloom Filters**
- **Searching in DHTs**
- **Replication**

Distributed Systems - In the News

■ 18.04.2018: Another Scam ICO? Savedroid Founder Exits with \$50M to Chill on a Beach

- “There do seem to be some suspicious pointers, firstly the lack of following and engagement on the ICO’s subreddit, followed by posts such as this from the savedroid_support account:”



■ 19.04.2018: Savedroid ICO’s exit scam was actually a very dumb PR stunt

- PR for ICO advisory

URL: b.socrative.com/student/

Room: HSR

TomP2P

A P2P-based high performance key-value pair storage library

■ TomP2P is a P2P framework/library

- Implements DHT (structured), broadcasts ([un]structured), direct messages (can implement super-peers)
- NAT handling: UPNP, NATPMP, new addition: relays, hole punching (work in progress)
- Direct / indirect (tracker / mesh) storage
- Direct / indirect replication (churn prediction and ~rsync)
- Modes: key,value / multi-key (versioned) value
- Java 6, Maven, [Github](#), Netty, TCP/UDP, MapDB, Android

■ TomP2P extends DHT

- Distributed hash table concept → put(key,value) / get(key)
- Extended DHT operations →
 - put(key1,key2,value)
 - put prepare / put confirm
 - add(value)
 - digest(key) / bloomfilters / versions
 - get(key) + bloomfilters

UPnP

- Networking protocols for data sharing, communications, and entertainment
- Internet Gateway Device ([IGD](#)) Standardized Device Control Protocol: UDP mDNS broadcast to 239.255.255.250:1900
- Returns data about [UPnP](#)
 - friendlyName, manufacturer, modelDescription, presentationURL, localAddress, modelNumber, modelName
 - Responses sent over UDP, with [HTTPU](#)
- HTTP connection to router with SOAP / XML message
 - No authentication, no HTTPS
 - If not configured properly, access from Internet

NATPMP

- Introduced in 2005 by Apple as an alternative to IGD – [RFC 6886](#)
- UDP, port 5351, simple binary protocol
 - Simplicity!
- No authentication
- How to find router? ([Java](#) vs. [C#](#))

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| Vers = 0      | OP = x      | Reserved                          |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| Internal Port          | Suggested External Port              |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| Requested Port Mapping Lifetime in Seconds                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Opcodes supported:

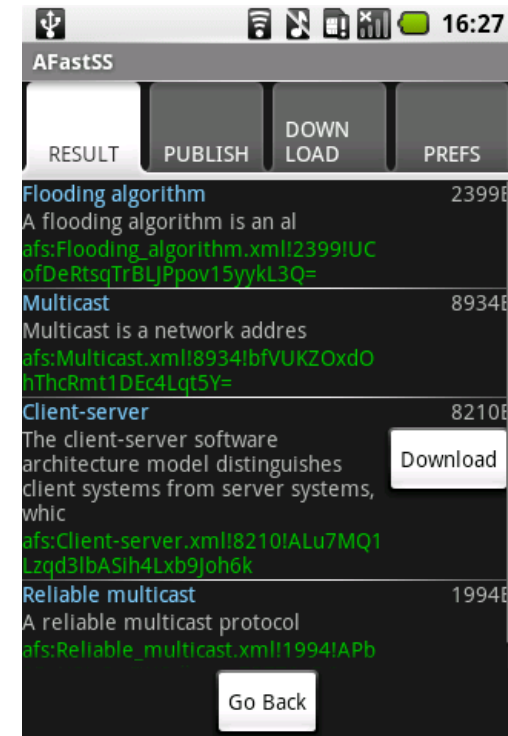
- 1 - Map UDP
- 2 - Map TCP

TomP2P

A P2P-based high performance key-value pair storage library

■ TomP2P history

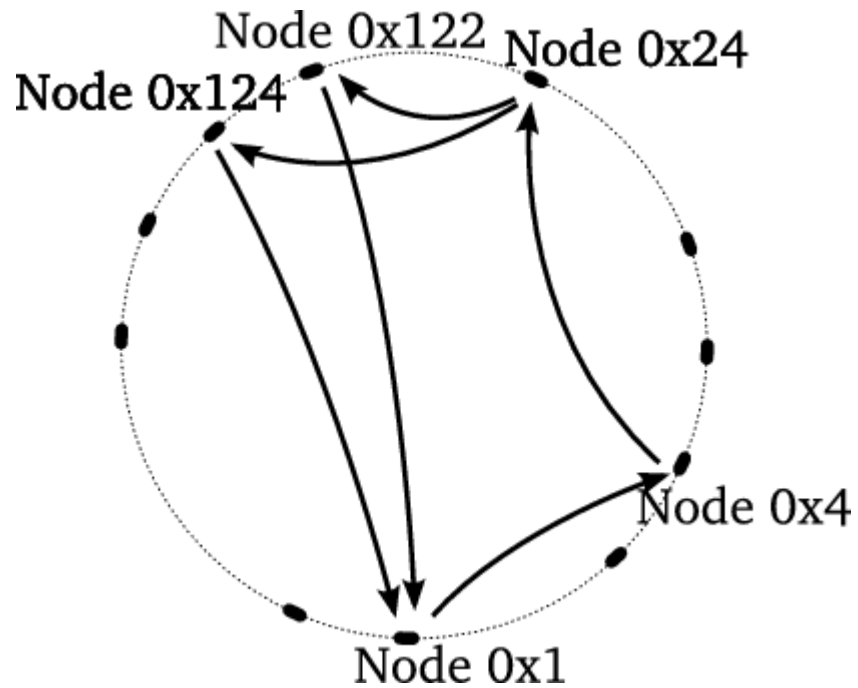
- TomP2P v1: Created in 2004 and used for a distributed DNS project
 - This version used blocking IO operations (1 thread / socket)
- TomP2P v2: Apache MINA (java.nio framework) / 6K LoC
 - Not well designed for non-blocking operations (event-driven)
- TomP2P v3: Redesigned for non-blocking operations
 - Switched to Netty / 14K LoC, 6K LoC JUnits
- TomP2P v4: API refinements, new features
 - Latest feature (work in progress) MapReduce
 - 19K LoC (core), 6K LoC JUnits (core)
- TomP2P v5 (core 18K LoC): modularization, relays, API refinements



Key Concepts

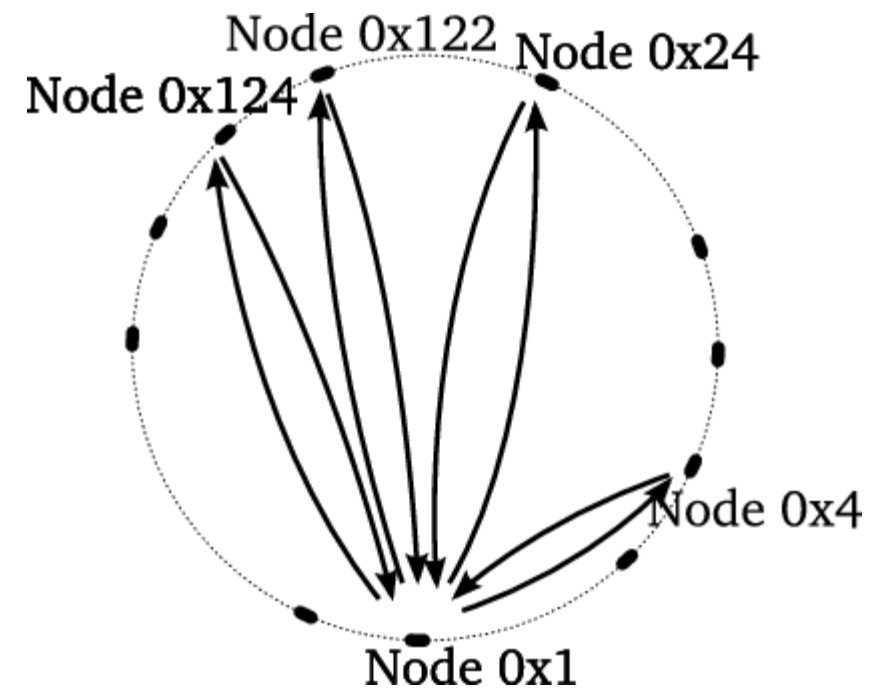
Fundamental Concepts

■ Recursive routing



- + online status update
- faulty peers cause delay

■ Iterative routing



- + control
- neighbor maintenance

5.5. Common API for Structured P2P Overlays (rec.)

■ Standardized interface between applications and overlays using recursive routing

- Implemented by Pastry, others implement it too (Chord, Tapestry, CAN)

■ **forward(RouteMessage)**

- Called just before a message is forwarded
- Message could be changed or dropped

■ **deliver(Id, Message)**

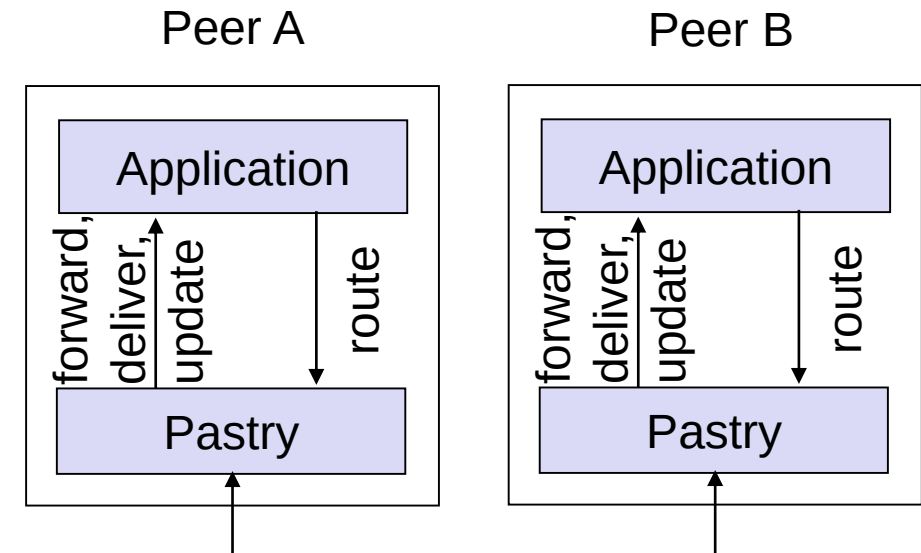
- Called when a message is received

■ **update(NodeHandle, boolean)**

- Called when a node's leafset changes (node joined or left)

■ **route(Id, Message)**

- Send a message to a node numerically closest to an Id (key)



5.5. Common API for Structured P2P Overlays (iterative)

- **Standardized interface between applications and overlays using iterative routing**

- Implemented by TomP2P, others implement it too

- **bootstrap (node ID)**

- Called when a node wants to join the network (route)

- **put (key, value)**

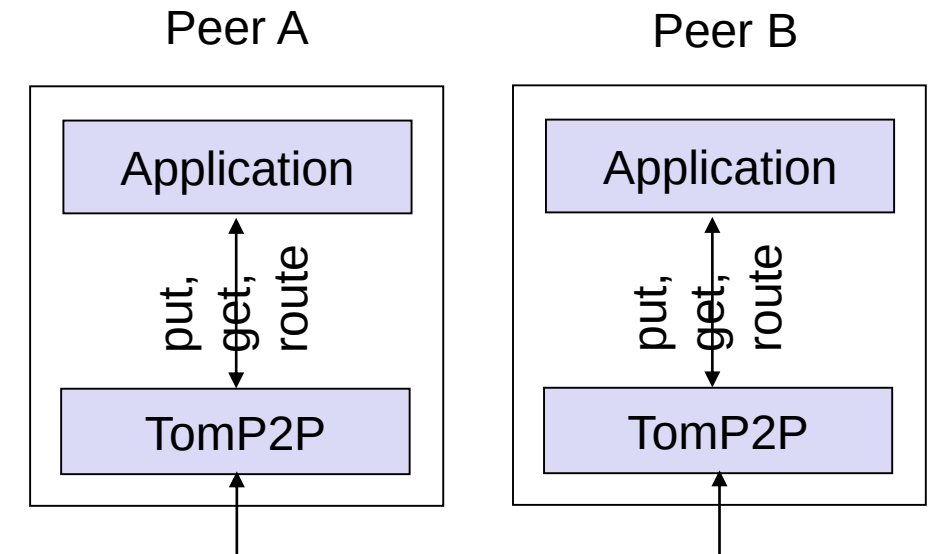
- To store (redundantly) data in the DHT

- **get (key)**

- To get data from the DHT. May get more than one result

- **Additional features (add/discover/digest)**

- Differs based on implementation

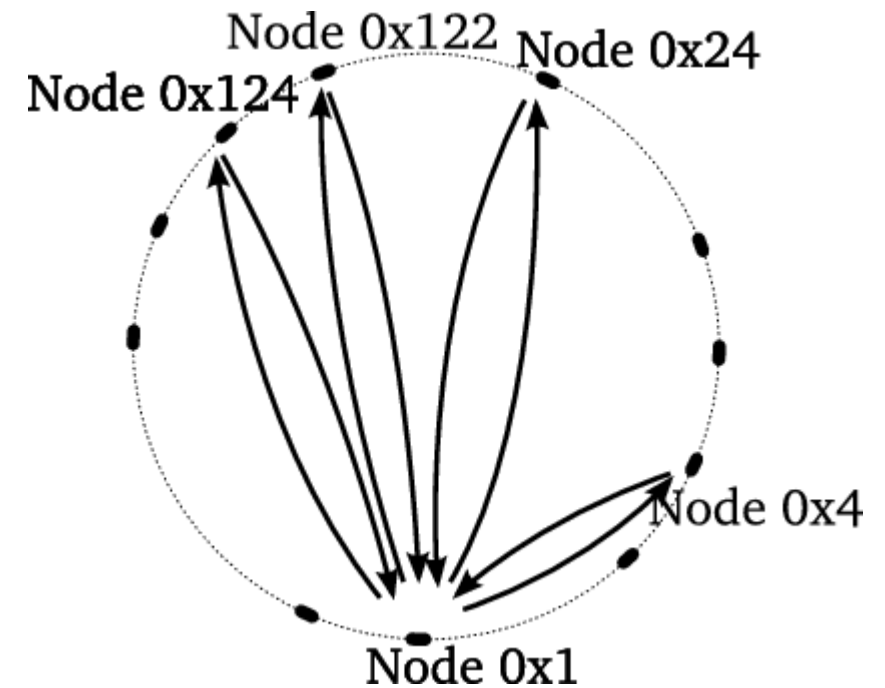


■ Iterative XOR-based routing

- Node and data item unique 160bit identifier [[SHA-1](#)]
- Keys are located on the nodes whose node ID is closest to the key
- Search for a key:
 - Lookup in neighbor table for closest peer (e.g. peers with ID: 0x1, 0x2, 0x3, 0x4)

My ID	Neighbor ID	Distance (XOR)
1	2	3
1	3	2
1	4	5

- Difference to Pastry: another metric

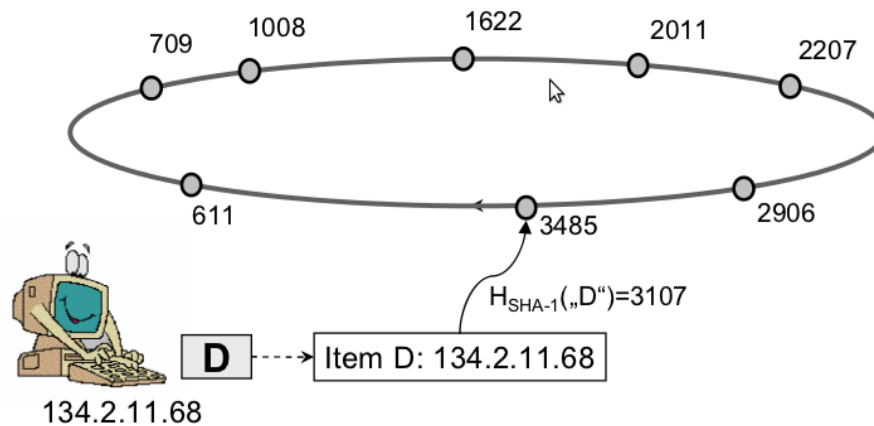


Fundamental Concepts

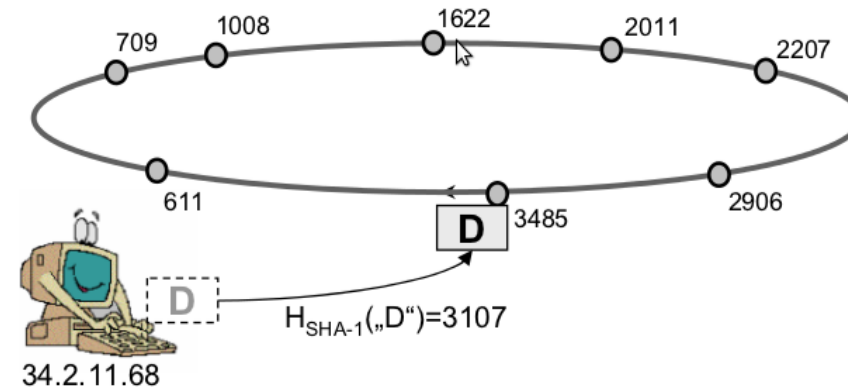
■ DHT vs. Tracker

- DHT “stored by value” – direct storage
- Tracker “stored by reference” – indirect storage

indirect (Tracker)



direct (DHT)



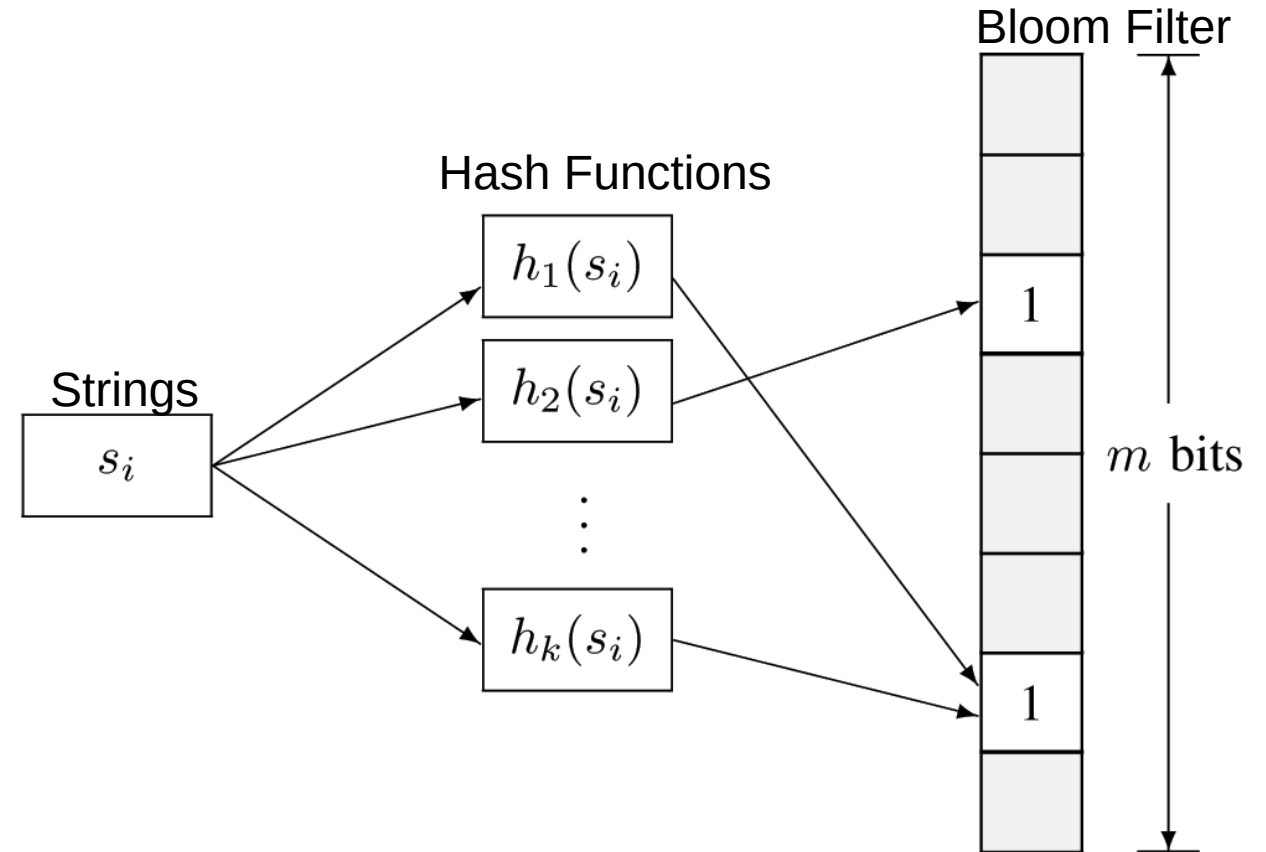
URL: b.socrative.com/student/

Room: HSR

Bloom Filter

Traditional Bloom Filter

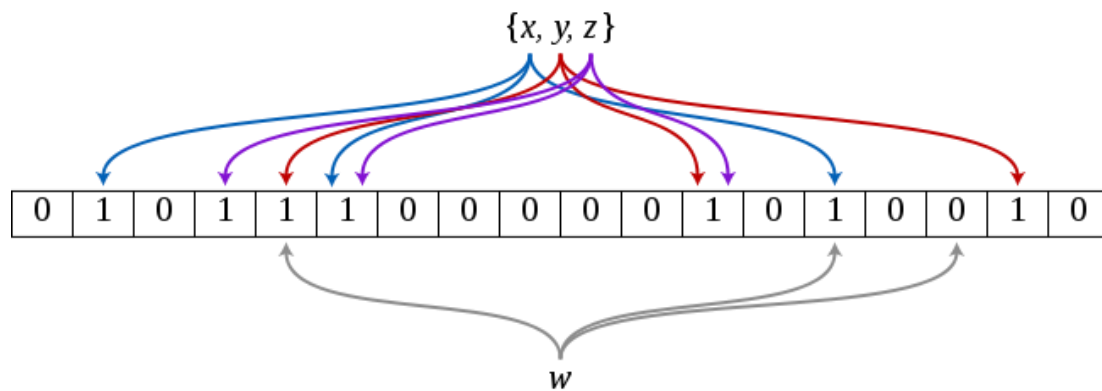
- An array of m bits, initially all bits set to 0
- A bloom filter uses k independent hash functions
 - $h_1, h_2, \dots, h_k$ with range $\{1, \dots, m\}$
- Each input is hashed with every hash function
 - Set the corresponding bits in the vector
- Operations
 - Insertion
 - The bit $A[h_i(x)]$ for $1 < i < k$ are set to 1
 - Query
 - Yes if all of the bits $A[h_i(x)]$ are 1, no otherwise
 - Deletion
 - Removing an element from this simple Bloom filter is impossible



Query of an Element, $m=18$, $k=3$

■ Insert x, y, z

■ Query w



http://en.wikipedia.org/wiki/Bloom_filter

■ Example for False-positives

■ Insertions

- Hash („color printer“) $\Rightarrow (1,4,6)$
- Hash („digital camera“) $\Rightarrow (3,4,5)$
- Bloom filter (1,3,4,5,6)

■ Query

- Hash („heat sensor“) $\Rightarrow (3,4,6)$
- Matches since bits 3,4,6 are all set to 1

■ [Online](#)

■ False-negative

■ Query

- Hash („color printer“) $\Rightarrow (1,4,6)$, matches (1,3,4,5,6) $\rightarrow$ no false-negative

Properties

■ Space Efficiency

- Any Bloom filter can represent the entire universe of elements
 - In this case, all bits are 1

■ No Space Constraints

- Add never fails
- But false positive rate increases steadily as elements are added

■ Simple Operations

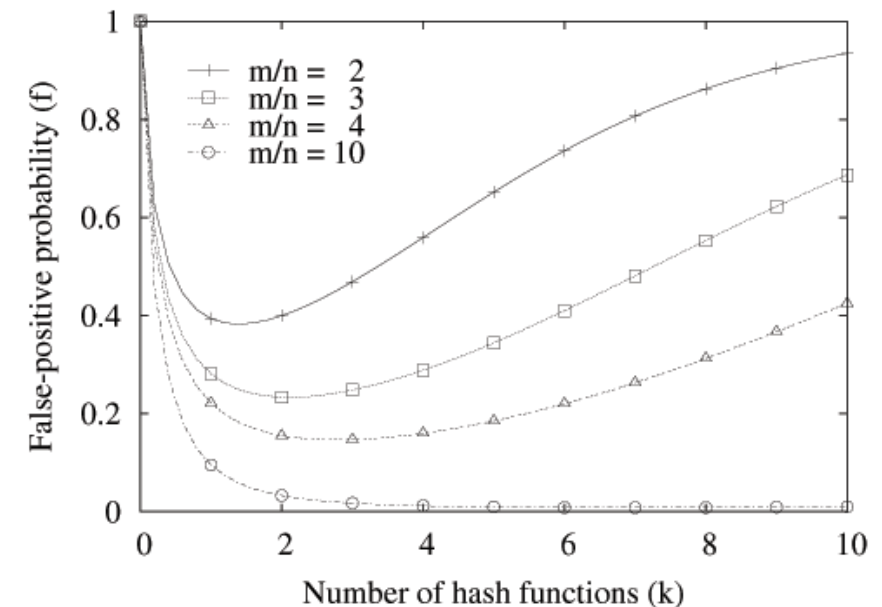
- Union of Bloom filters: bitwise OR
- Intersection of Bloom filters: bitwise AND

■ No false negative, but false positive

■ False-positive probability:

- n number of strings; k hash functions; m -bit vector

$$f = \left(1 - e^{-\frac{nk}{m}}\right)^k$$



=> Given m/n , there is an optimal number of hash functions (opt. $k = m/n \ln 2$) (when 50% of the bits are set)

Bloom Filter Variants (1)

■ Compressed Bloom Filters

- When the filter is intended to be passed as a message
- False-positive rate is optimized for the compressed bloom filter (uncompressed bit vector m will be larger but sparser)
- However, compression/decompression, more memory

■ Generalized Bloom Filter

- Two type of hash functions g_i (reset bits to 0) and h_j (set bits to 1)
- Start with an arbitrary vector (bits can be either 0 or 1)
- In case of collisions between g_i and h_j , bit is reset to 0
- Store more info with low false positive
- Produces either false positives or false

negatives

■ Counting Bloom Filters

- Entry in the filter not be a single bit but a counter
- Delete operation possible (decrementing counter)
- [Variable-Increment Counting Bloom Filter](#)

■ Scalable Bloom Filter

- Adapt dynamically to number of elements, consist of regular Bloom filters
- “A SBF is made up of a series of one or more (plain) Bloom Filters; when filters get full due to the limit on the fill ratio, a new one is added; querying is made by testing for the presence in each filter”

URL: b.socrative.com/student/

Room: HSR

And / or searching Range queries

Mechanisms based on Hashing in DHTs

■ Search in DHT

- `DHT.get(h(«Institut für Software»))`
- In order to find it: `DHT.put(h(«Institut für Software»), value)`

■ Keywords

- `DHT.get(h(«Institut»))`
- Find it: `DHT.put(h(«Institut»), value), DHT.put(h(«für»), value), DHT.put(h(«Software»), value)`
- value points to `h(«Institut für Software»)`

■ Keywords drawbacks

- Find good keywords → “the”, “a” are not good keywords
- Exact matches only

Mechanisms based on Hashing in DHTs

■ Find “Communication” - OR Systems

- `DHT.get(h(«Institut»))` and `DHT.get(h(«Software»))`, combine results

■ Find “Communication” - AND Systems

1. `DHT.get(h(«Institut»))` and `DHT.get(h(«Software»))`, intersect results
2. `DHT.get(h(«Institut») xor h(«Software»))`

- In order to find it:

`DHT.put(h(«Institut») xor h(«Software»), value),`

`DHT.put(h(«Institut») xor h(«für»), value)`

`DHT.put(h(«für») xor h(«Software»), value)`

- Combination needs to be known in advance

3. Use Bloom Filters

- `bf = DHT.getBF(h(«Institut»))` and `DHT.get(h(«Software»), bf)`
- Sequential (less network, slower) vs. parallel (more network, faster)

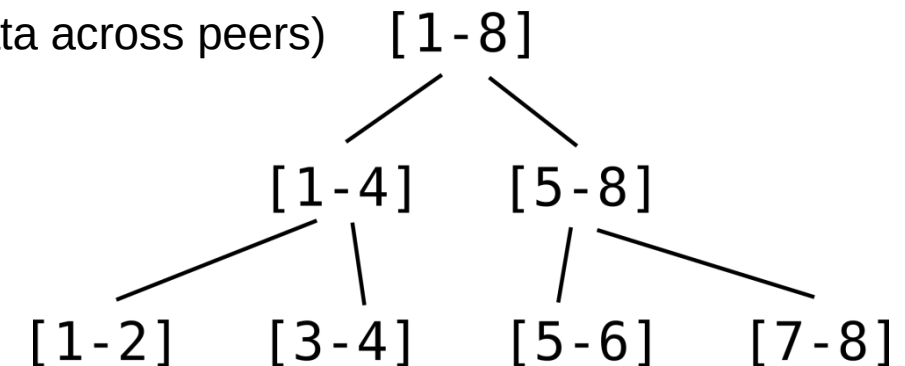
Mechanisms based on Hashing in DHTs

■ Range Queries

- Problem: random insert vs. sequence insert
- Sequence $\rightarrow [0..n-1] [n..2n-1] [2n..3n-1] [\dots] \rightarrow$ peer responsible for range, hash it, store it, done.
 - Insert 10 items: $N = 5 \rightarrow [0, 1, 2, 3, 4], [5, 6, 7, 8, 9]$ – sequential, 2 peers
 - Insert 10 items: $N = 5 \rightarrow [0], [5], [10], [15], [20], [25], [30], [35], [40], [45]$ – random, 10 peers
 - But random: worst case: 1 peers has 1 data item, range query for range $[0..x]$ contacts x/n peers.

■ Over-DHT

- **PHT**: trie (prefix tree); **DST**: segment $\rightarrow$ tree on top of DHT
- Main idea: hash of tree-node (resp. for range) $\rightarrow$ DHT
- PHT: Peer stores n data items, if n reached, splits data (moves data across peers)
- DST: stores data on each level (redundancy) up to a threshold
 - No data splitting



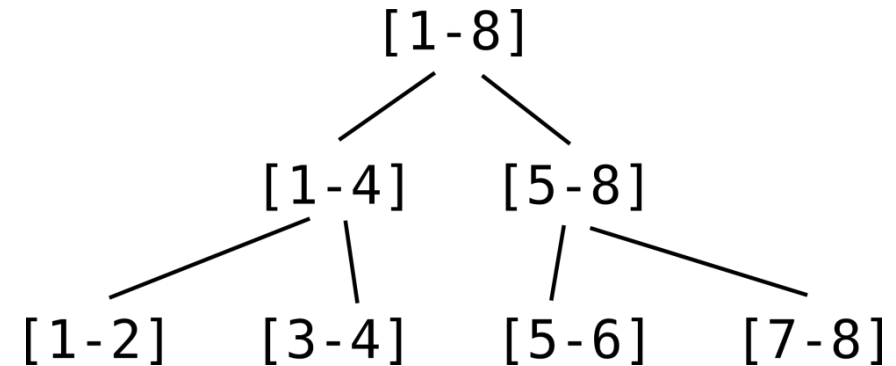
Mechanisms based on Hashing in DHTs

■ Example:

- Set $n = 2$, $m=8$
- 1, "test"; 2, "hallo";
3, "world"; 5, "sys"; 6, "hsr";
7, "ifs"

■ Tree: store value

- Translate `putDST(1, "test")` to
 - `put(hash([1-8]), "test")`
→ may be stored (only if threshold not reached)
 - `put(hash([1-4]), "test")` → may be stored
 - `put(hash([1-2]), "test")` → will be stored
 - Store `put(2, "hallo")`, `put(3, "world")`, `put(5, "sys")`, ...



■ Query `getDST(1..5)` translates to

- `get(hash[1-8])` → returns "1,test; 2,hallo"
- `get(hash[1-4])` → returns "1,test; 2,hallo"
- `get(hash[1-2])` → returns "1,test; 2,hallo"
- `get(hash[3-4])` → returns "3,world"
- `get(hash[5-8])` → returns "5,sys; 6,hsr"
- `get(hash[5-6])` → returns "5,sys; 6,hsr"

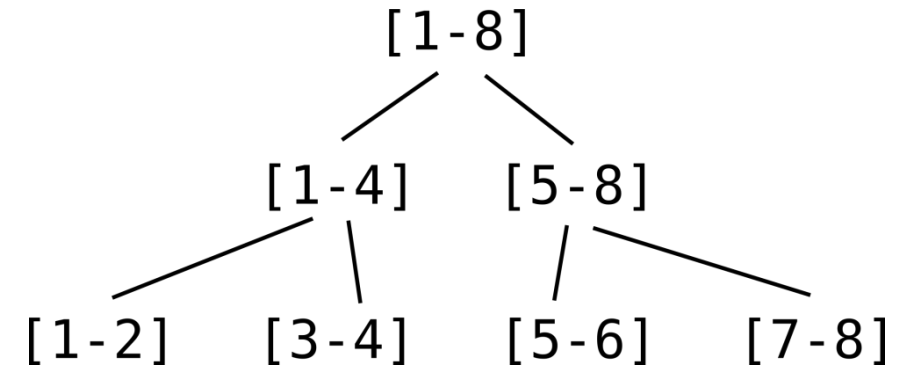
Mechanisms based on Hashing in DHTs

■ Example:

- Set $n = 2$, $m=8$
- 1, "test"; 7, "ifs"

■ Tree: store value

- Translate `putDST(1, "test")` to
 - `put(hash([1-8]),"test")`
→ may be stored (only if threshold not reached)
 - `put(hash([1-4]),"test")` → may be stored
 - `put(hash([1-2]),"test")` → will be stored
- Store `put(7, "ifs")`



■ Query `getDST(1..5)` translates to

- `get(hash[1-8])` → returns "1,test; 7,ifs"
- `get(hash[1-4])` → returns "1,test;"
- `get(hash[5-8])` → returns "7,ifs"

■ Range query as series of `put()` and `get()`

Similarity Search

■ Similarity Search in DHT

- <http://fastss.csg.uzh.ch>

■ Project that brings similarity search to HT / DHT

- Problem: Search for “netwrk” fails for DHTs

■ Similarity: Edit distance / Levenshtein distance

- Min operations to transform one string into another, operations: insert, delete, replace
- Calculated in matrix size $O(m \times n)$



$$\begin{aligned}d[i, 0] &= i, \quad d[0, j] = j, \\d[i, j] &= \min (d[i-1, j] + 1, \quad d[i, j-1] + 1, \\&\quad d[i-1, j-1] + (\text{if } s1[i] = s2[j] \text{ then } 0 \text{ else } 1))\end{aligned}$$

Mechanisms based on Hashing in DHTs

- Example $d(\text{test}, \text{east}) = 2$ (remove a, insert t)
- Expensive operation if all words need testing
- Main idea: pre-calculate errors
 - All possible errors? Neighbors for test with ed 2: test, testa, testaa, testab, ... , tea, teb, tec, ..., teaa, teab, ... → 23883 more of those!

		T	E	S	T
	0	1	2	3	4
E	1	1	1	2	3
A	2	2	2	2	3
S	3	3	3	2	3
T	4	3	4	3	2

$$\begin{aligned}d[i, 0] &= i, d[0, j] = j, \\d[i, j] &= \min(d[i-1, j] + 1, d[i, j-1] + 1, \\&\quad d[i-1, j-1] + (\text{if } s1[i] = s2[j] \text{ then } 0 \text{ else } 1))\end{aligned}$$

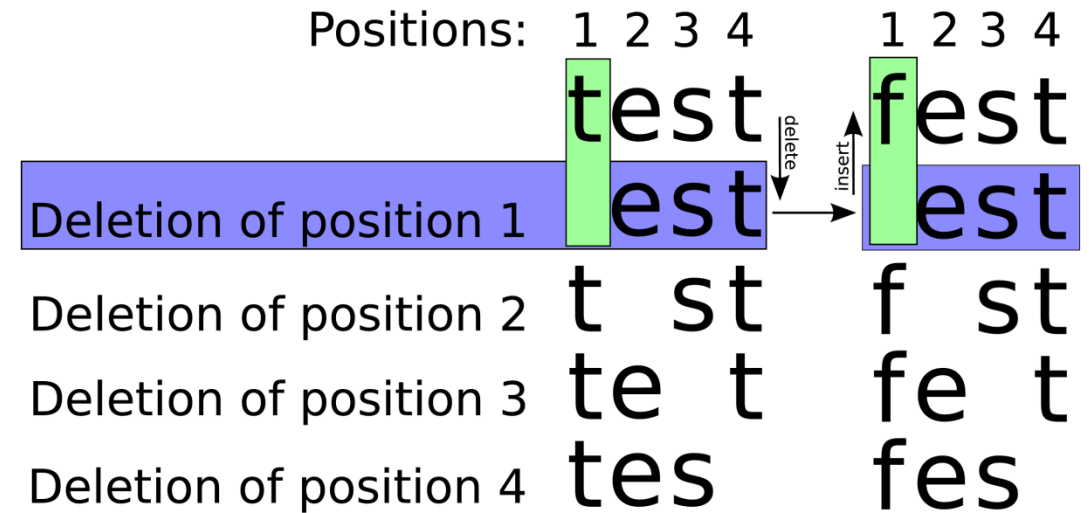
Mechanisms based on Hashing in DHTs

■ FastSS pre-calculates with deletions only

- Neighbors for *test* with ed 2: test, est, st, et, es, tst, tt, ts, tet, te, tes
- Pre-calculation on query **and** index
- 11 neighbors → 11 more queries, indexed enlarged by 11 entries

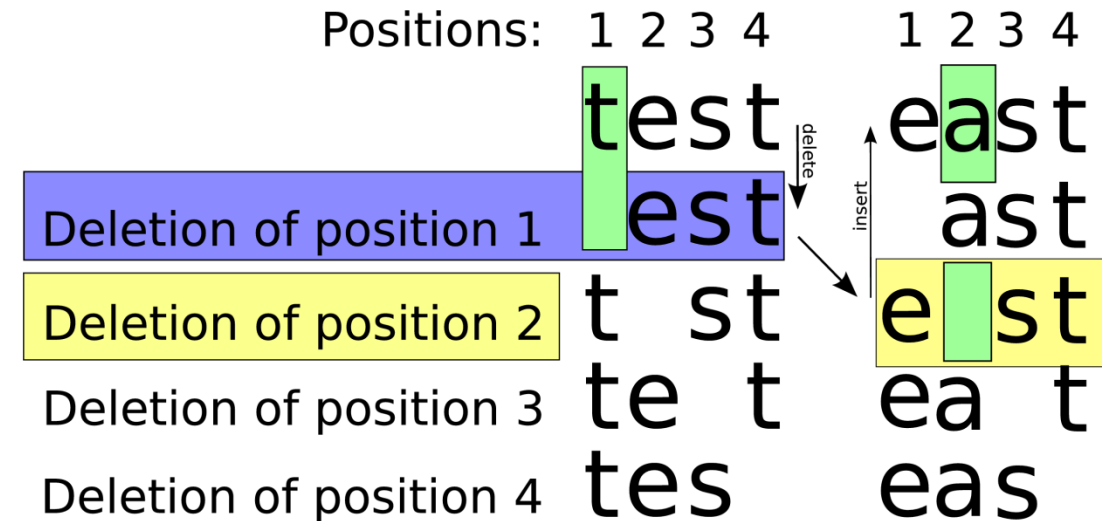
■ Example $d(\text{test}, \text{fest})=1$

- test: indexed
- fest: query



Mechanisms based on Hashing in DHTs

- **Example $d(\text{test}, \text{east})=2$**
 - test: indexed
 - east: query
- **P2PFastSS implemented on top of TomP2P (early version) – tests with indexing Wikipedia abstracts**



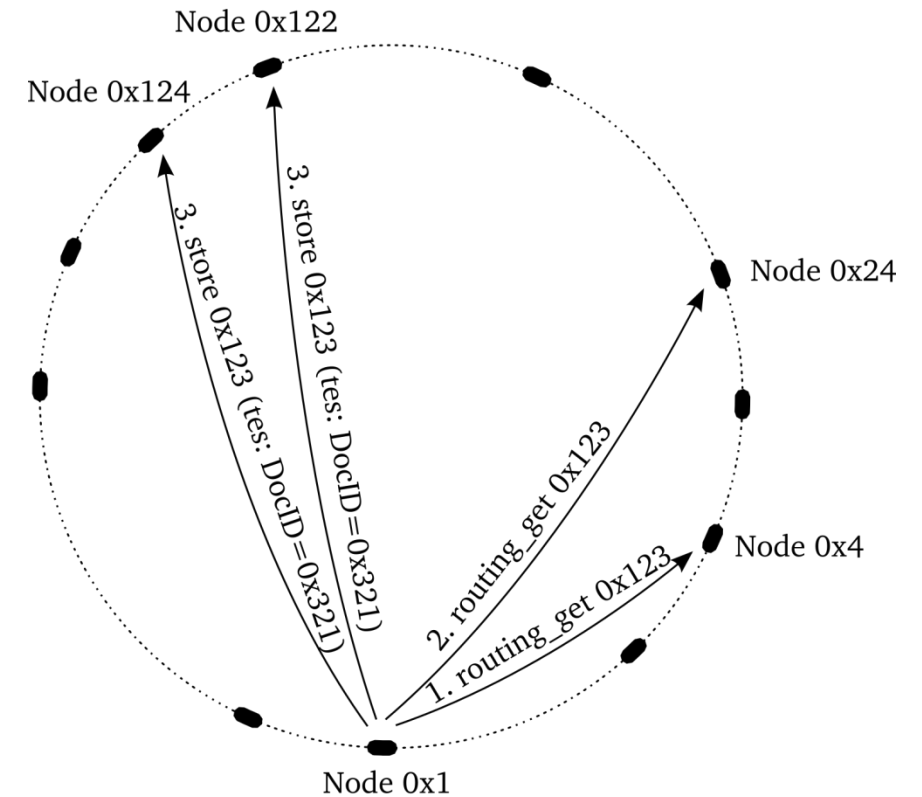
Mechanisms based on Hashing in DHTs

- Index documents using `put(hash(document), document)`

- Document (0x321) contains word test

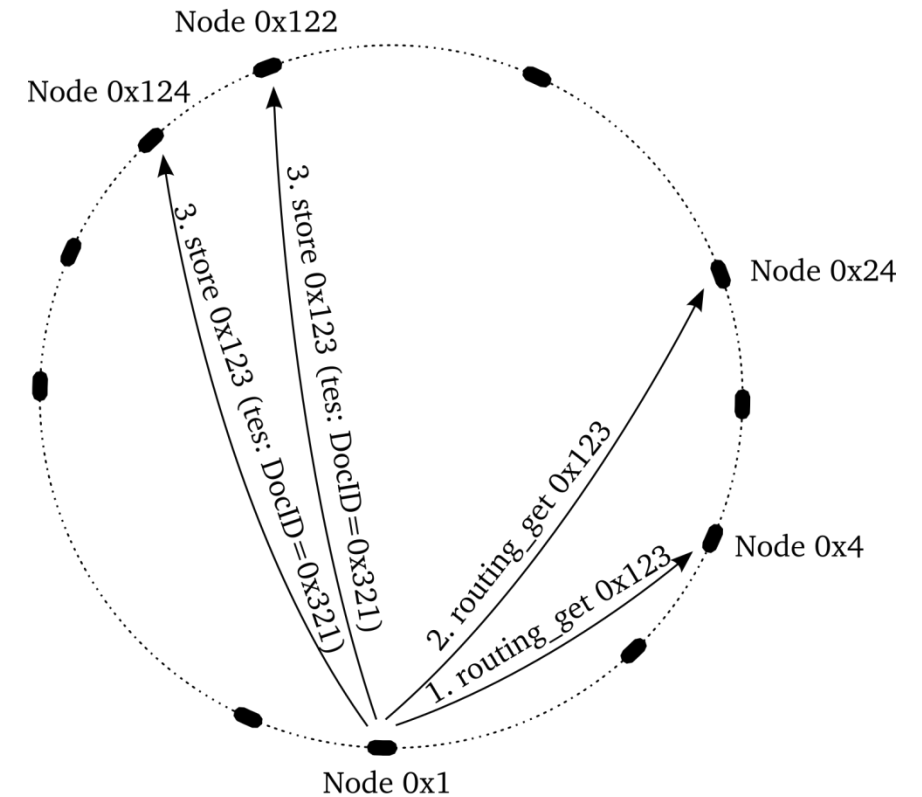
- Index all neighbors (test, tes, tst, tet, est) using `put(hash(neighbor), point to document)`

- `hash("tes") = 0x123`



Mechanisms based on Hashing in DHTs

- User searches for “tesx”
- Neighbors are generated (tesx, esx, tsx, tex, **tes**)
 - $\text{get}(\text{hash}(\text{neighbor})) \rightarrow 0x123$
 - Find pointer to document (0x321)
 - $\text{document} = \text{get}(0x321)$
- Tests with edit distance 1, partially 2, ignoring delete pos.
 - Overhead (n choose k) for query and index
- Similarity search as series of `put()` and `get()`

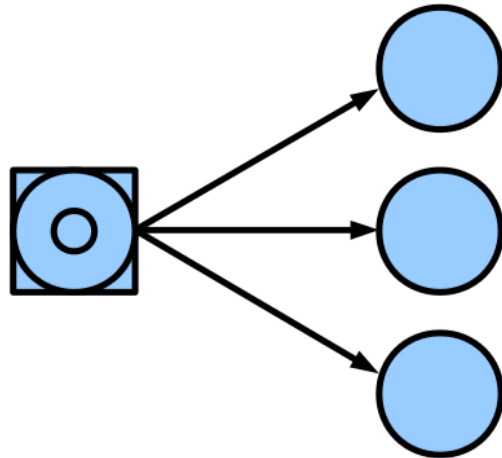


Replication

Redundancy in DHTs

■ Replication

- Enough replicas
- Direct replication
 - Originator peer is responsible
 - Periodically refresh replicas
 - Example: tracker that announces its data



Responsible for X



Originator of X



Close peers to X

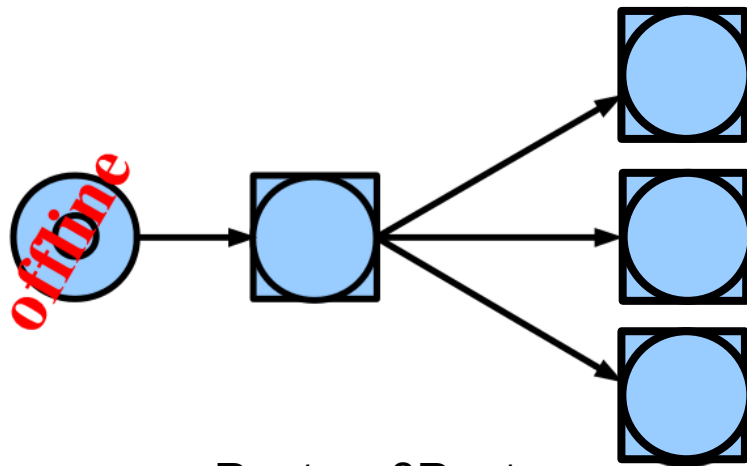
■ Problem

- Originator offline → replicas disappear.
Content has TTL

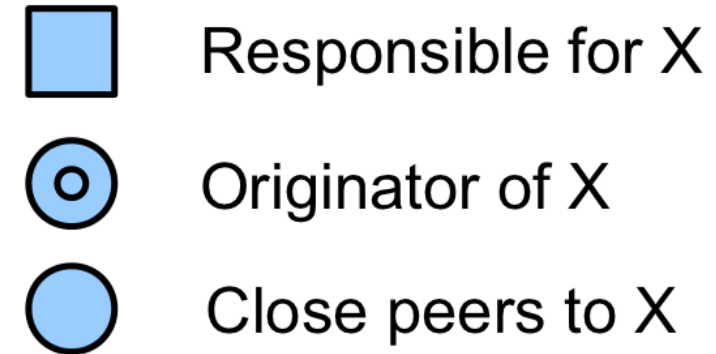
Redundancy in DHTs

■ Indirect Replication

- The closest peer is responsible, originator may go offline (0Root)
 - Periodically checks if enough replicas exist
 - Detects if responsibility changes



nRoot vs 0Root



■ Problem

- Requires cooperation between responsible peer and originator
- Multiple peers may think they are responsible for different versions → eventually solved

Replication and Consistency

■ DHTs have weak consistency

- Peer A put X.1
- Peer B gets X.1
- Peer B modifies it puts B.2

■ Same time (time in distributed systems):

- Peer C gets X.1
- Peer C modifies it puts C.2

■ Which one is stored

- B.2 of B
- C.2 of C

■ Replication makes it worse

- Consistency: generic issue in distributed systems, requires typically coordinator

URL: b.socrative.com/student/

Room: HSR

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch