

Distributed Hash Table (DHT) and Peer-to-Peer

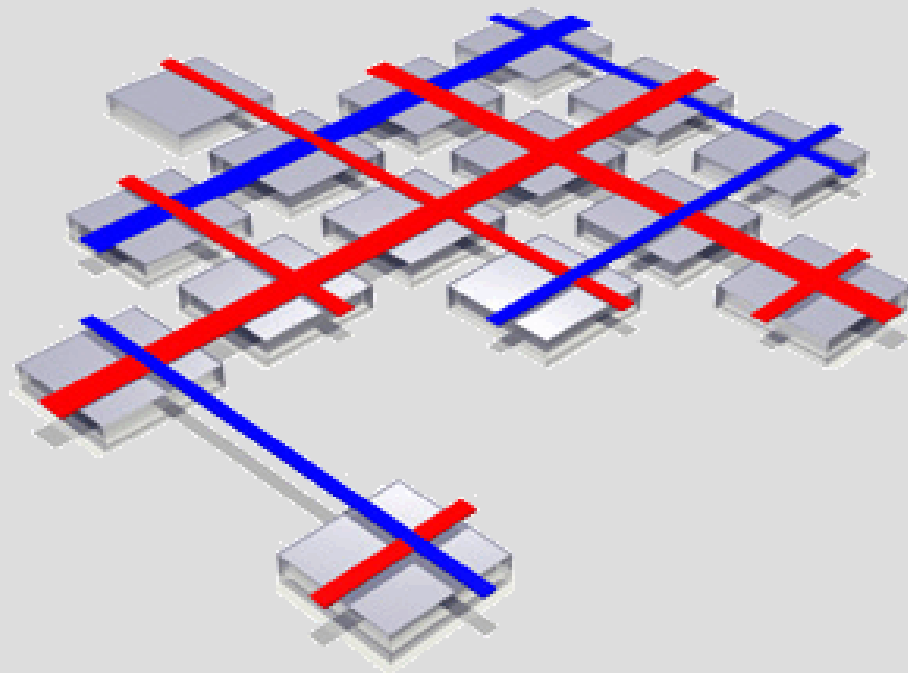
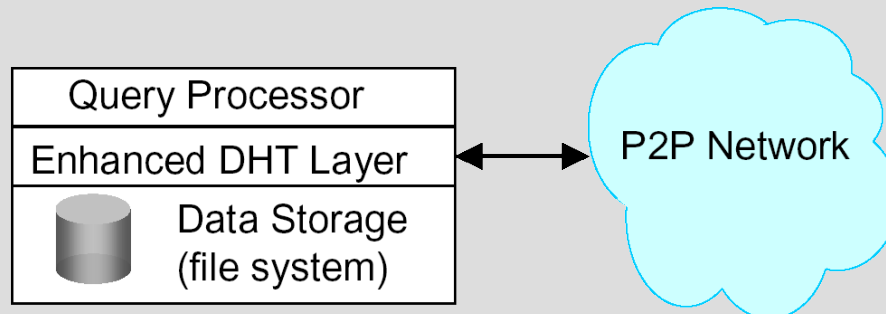


Table of Content

- What are DHTs?
- Why DHTs?
- What DHTs are there?
 - OneHop
 - Pastry, Tapestry ★
 - Chord
 - Accordion
- Simulation Results
- Problems



What are DHTs?



“DHTs are named after **hash tables** because they assign responsibility for a piece of data based on a hash function (often SHA-1); **each node** acts like a **bucket** in a hash table.

A DHT provides an **efficient lookup algorithm** (or network routing method) that allows one participating node to quickly determine which other machine is responsible for a given piece of data.” Wikipedia

What are DHTs?

- A (distributed) hash table with nodes as buckets for:
 - Finding nodes with data
 - Finding where to store data

Why DHTs?

- Lookup in $O(\log n)$ or even $O(1)$!
(n =number of nodes)
- No need for central server (weak point of many P2P networks)
- Less overhead than structureless P2P

Why DHTs?

Can be used in many applications:

- Usenet (UsenetDHT)
- Serverless E-Mail system
(ePost, www.epostmail.org)
- Data storage (OceanStore)
- Filesharing (Trackerless BitTorrent)
- Anonymus “Backweb”
(Freenet, freenet.sourceforge.net)

What DHTs are there?

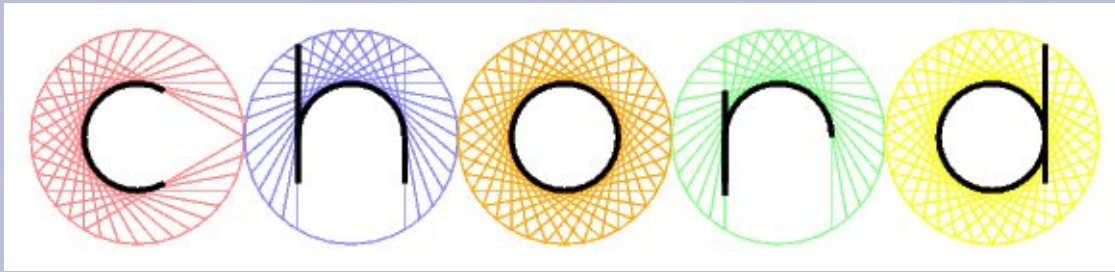
- $O(1)$ DHTs e.g. OneHop, Kelips
- $O(\log n)$ e.g. Chord, (Free)Pastry, Tapestry
- A mixture of both: Accordion

Pastry, apestry

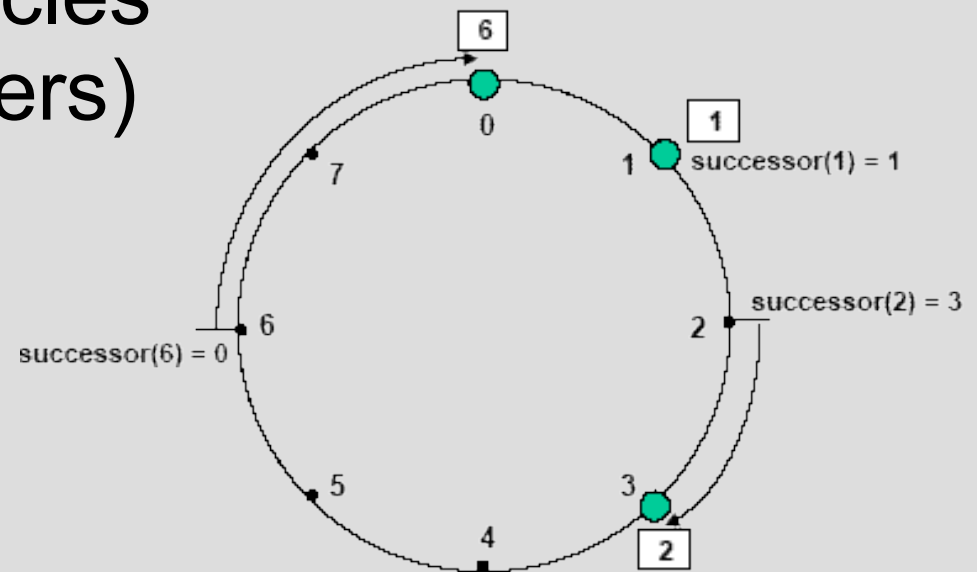
- Lookup in $O(\log n)$
- from Microsoft Research and Rice University
- Pastry: save data copy at node
- Tapestry: route to data
- Table size: $O(\log n)$

OneHop

- Lookup under normal condition $O(1)$
- Greatly increased memory and background overhead
- Big hash table with as many nodes as possible
- Better with small networks
- Table size: $O(n)$



- Used in many applications e. g. The Circle
- $O(\log n)$
- Developed at the MIT and UC Berkeley
- So far only a research implementation
- Nodes arranged in circles
- Scalable (by parameters)
- Table size: $O(\log n)$



Accordion



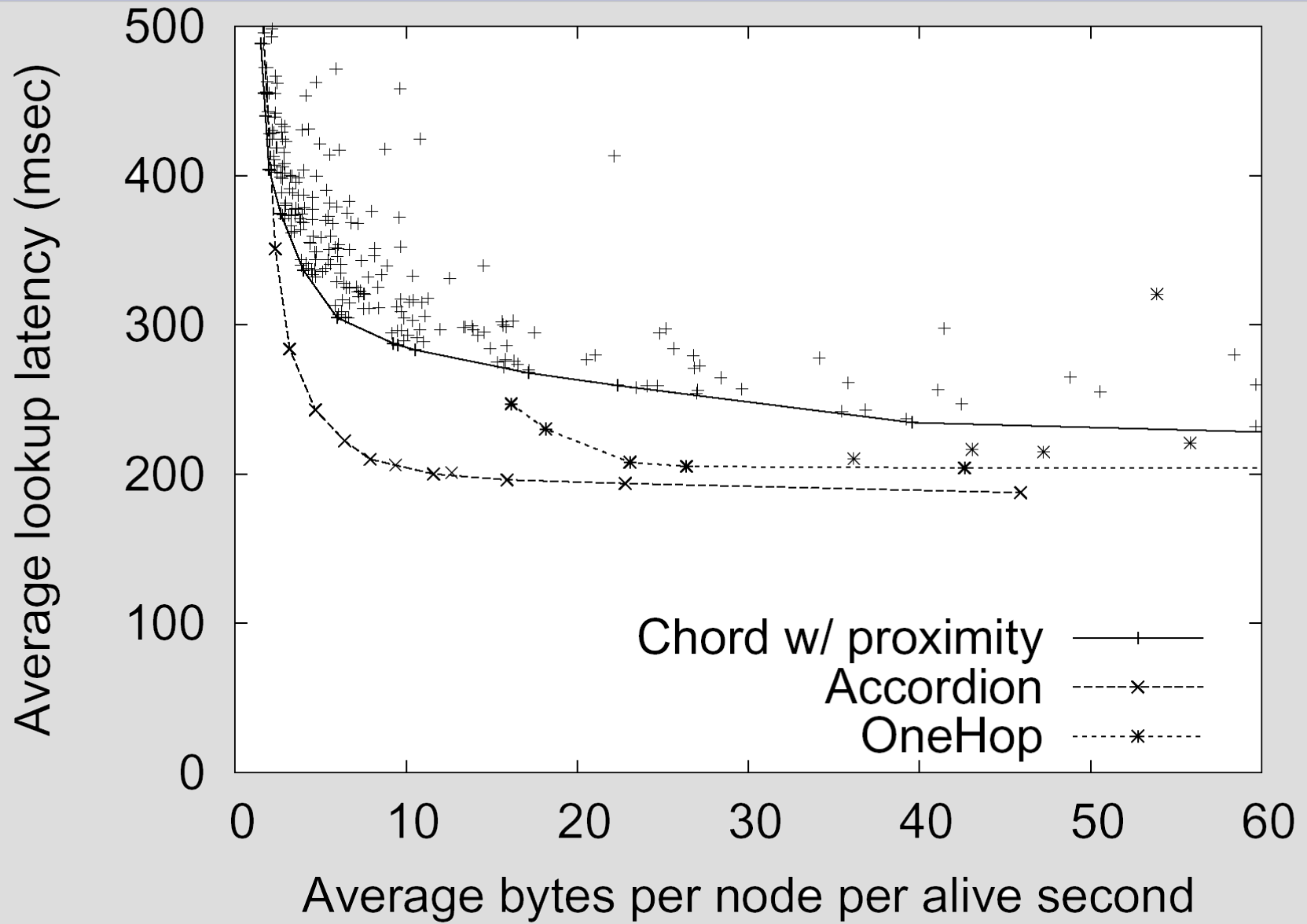
- Lookup in $O\left(\frac{\log(n)\log(\log(n))}{\log(s)}\right)$ (n=#nodes, s=table size)
- Saves “freshness” of neighbors to reduce timeouts
- Bandwidth budget (=>security)
- Learns from lookups ★
- “Explores” if budget not met
- More flexible through non-static table size
- Recursive parallel lookups ★
- Uses functions from Chord
- One parameter: Bandwidth, self tunes all other

Simulation Results

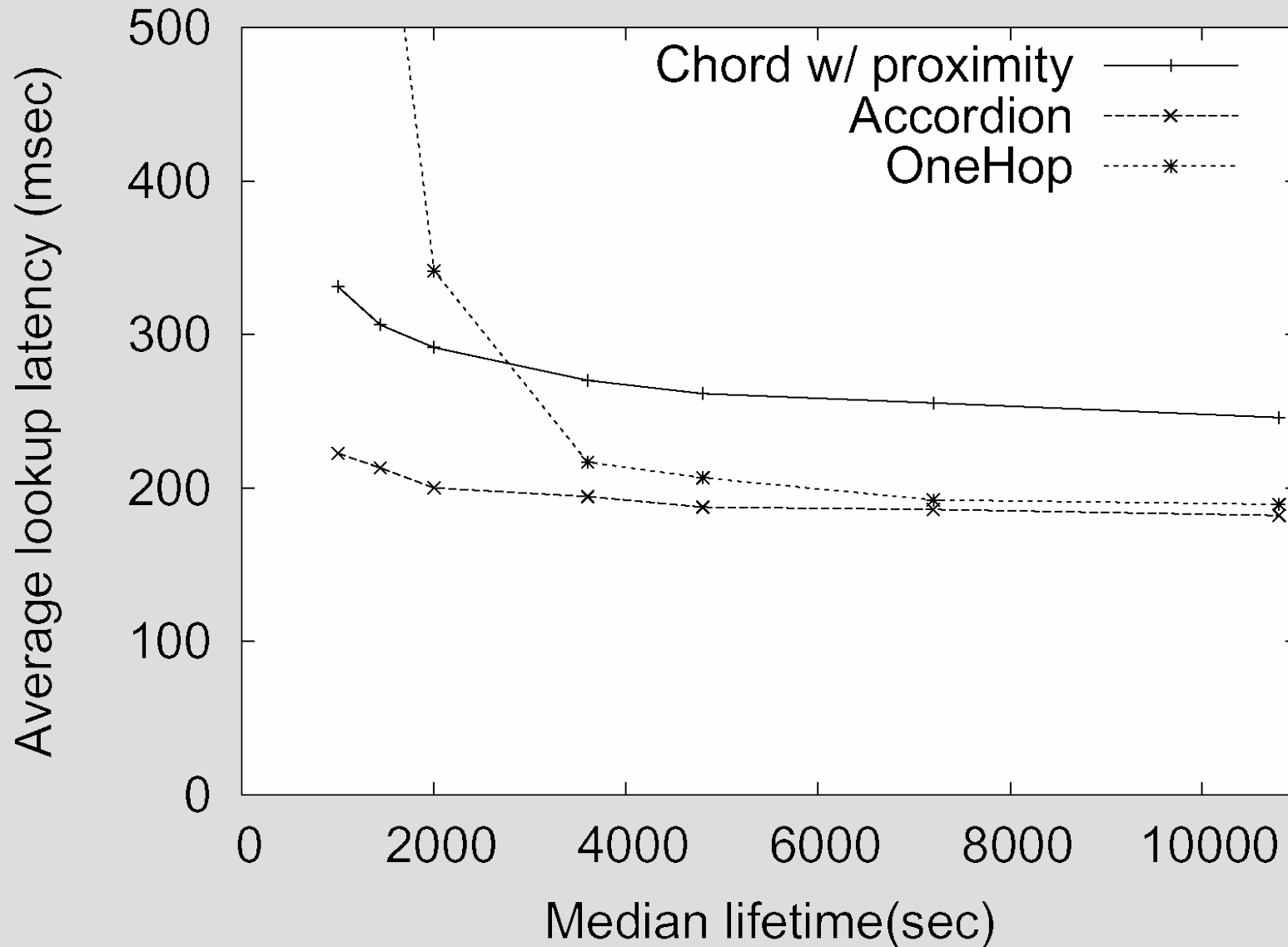


- Made with a simulator for peer-to-peer protocols (<http://pdos.csail.mit.edu/p2psim>)
- 3000 nodes simulated
- Only key lookups – no data transferred
- 4 hours simulated
- Averaged over 5 simulation runs

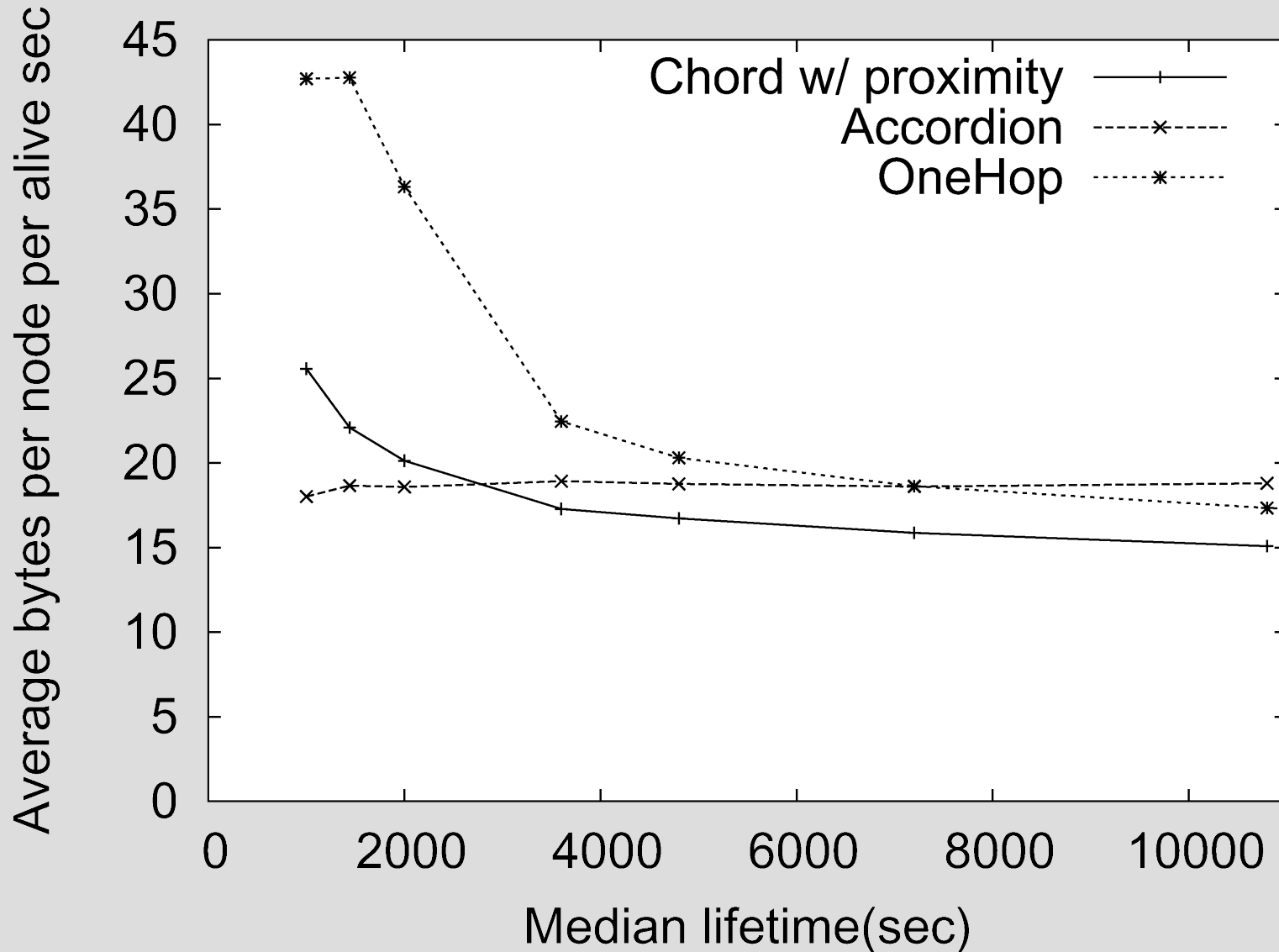
Simulation Results - Latency



Simulation Results - Churn



Simulation Results - Churn



Problems

Main problems:

- Searching
- Security

Searching

Problem:

Looking up a string in a hash table is easy

Looking up a substring is not.

Hashing algorithms need the **exact** search term (e. g. filename / title etc).

Similar search terms are not necessary in “nearby” buckets / nodes

Searching

Possible solution:

n-grams

Strings are split up into *n*-length substrings

e.g.: “Beethovens 9th”->

Bee, eet, eth,tho, hov, ove, ven, ens, ns%, s%9, %9t, 9th

these substrings are inserted into the DHT

a search for “thoven” is also split into *n*-grams (e.g., tho, hov,ove, ven) and each *n*-gram is looked up.

Searching

Advantages:

- Substring queries possible
- Searching for similar strings possible

Disadvantages:

- More entries in DHT
- False-positives
- More CPU power needed
- More search overhead

Security

- No mention of security in the Papers
- Paper even suggests that a high-bandwidth node is allowed to send more data to a node than that node's budget.

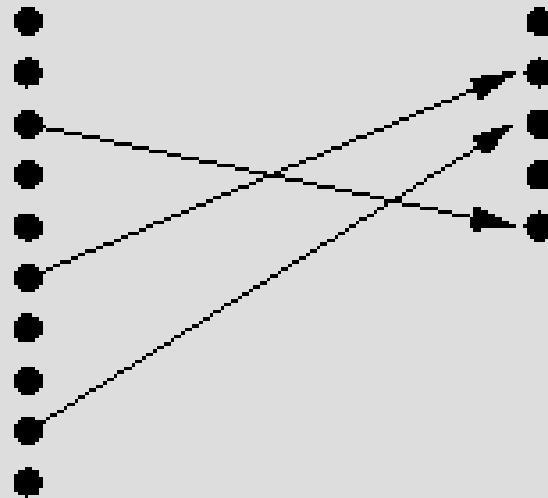
Distributed Hash Table (DHT) and peer-to-peer

END

Hashing



- Multiple values mapped to fewer “buckets”



mögliche Elemente

N Adressen

- $\text{key} = \text{hash}(\text{data}) := \{\text{return } (\text{data} \% \text{prime})\}$




Secure Hash Algorithm (SHA)

The original specification of the algorithm was published in 1993 as the Secure Hash Standard, FIPS PUB 180, by **US government standards agency** NIST (National Institute of Standards and Technology).


This version is now often referred to as "**SHA-0**". It was withdrawn by NSA shortly after publication and was superseded by the **revised version**, published in 1995 in FIPS PUB 180-1 and commonly referred to as "**SHA-1**".

This was done, according to NSA, to correct a flaw in the original algorithm which reduced its cryptographic security. However, NSA **did not provide** any further explanation or identify **what flaw was corrected**. Weaknesses have subsequently been reported in both SHA-0 and SHA-1. SHA-1 appears to provide greater resistance to attacks, supporting NSA's assertion that the change increased the security.

Freshness and Learning from Lookups

- With high churn, nodes often disappear
- Timeouts are expensive (mult. RTT)
- DHT saves “freshness” for each node
- Each lookup from other nodes used to update freshness 
- Node not fresh? => ping Node
- => only node that are “probably dead” are pinged
- => Less overhead

Parallel lookups

- Parallel lookups also in other implementations, but:
- Accordion uses recursive lookups
- No direct reply to original node (until search found) 

Names

- Pastry
 - Tapestry
 - Chord
 - Accordion
- 
- Die Pastete
 - Der Wandteppich
 - Der Akkord
 - Das Akkordeon