Encrypted Distributed Hash Tables

Archita Agarwal, Seny Kamara

ENCRYPTED SYSTEMS LAB

BROWN

[PDF] Chord: A Scalable Peer-to-peer Lookup Protocol for ... - MIT PDOS
https://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf
by I Stoica - Cited by 13655 - Related articles
presents **Chord**, a distributed lookup protocol that addresses this problem. **Chord** provides ..... tion of blocks. The **distributed hash table** uses **Chord** to identify.

Bigtable: A Distributed Storage System for Structured Data
https://dl.acm.org/citation.cfm?id=1365816
by F Chang - 2008 - Cited by 6119 - Related articles
Sep 1, 2017 - Citation Count: 432 · Downloads (cumulative): 21,596 ... **ACM Transactions** on **Computer Systems** (**TOCS**) **TOCS** Homepage archive. Volume **26** Issue **2**, June **2008**. Article No. **4**. **ACM** New York, NY, USA table of contents ...

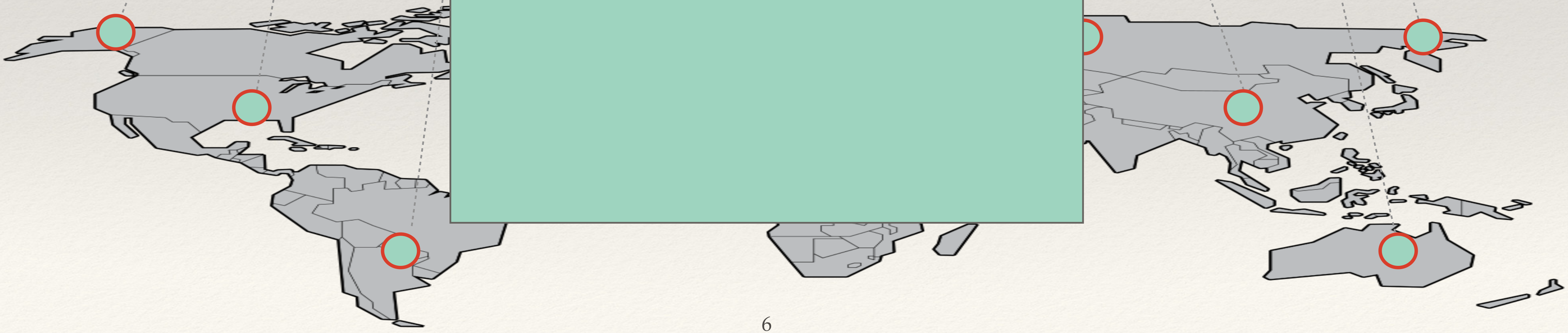
[PDF] Dynamo: Amazon's Highly Available Key-value Store
https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
by G DeCandia - 2007 - Cited by 4704 - Related articles
Oct 14, 2007 - This paper presents the design and implementation of **Dynamo**, a **highly available key-value** storage system that some of **Amazon's** core services use to provide an "always-on" experience. To achieve this level of availability, **Dynamo** sacrifices consistency under certain failure scenarios.

Distributed Hash Tables

DHT

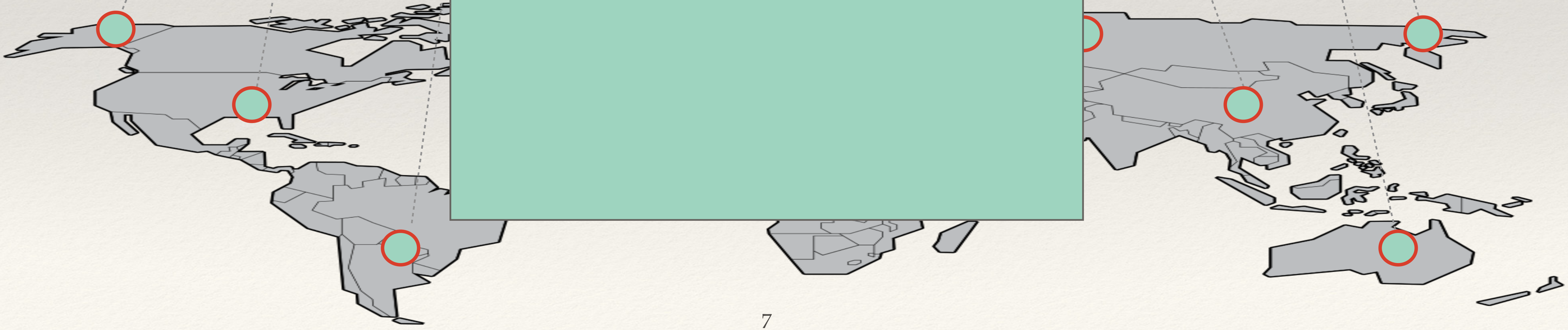
DHT

- Decentralised Systems
- Distribute (ℓ, v) pairs to nodes



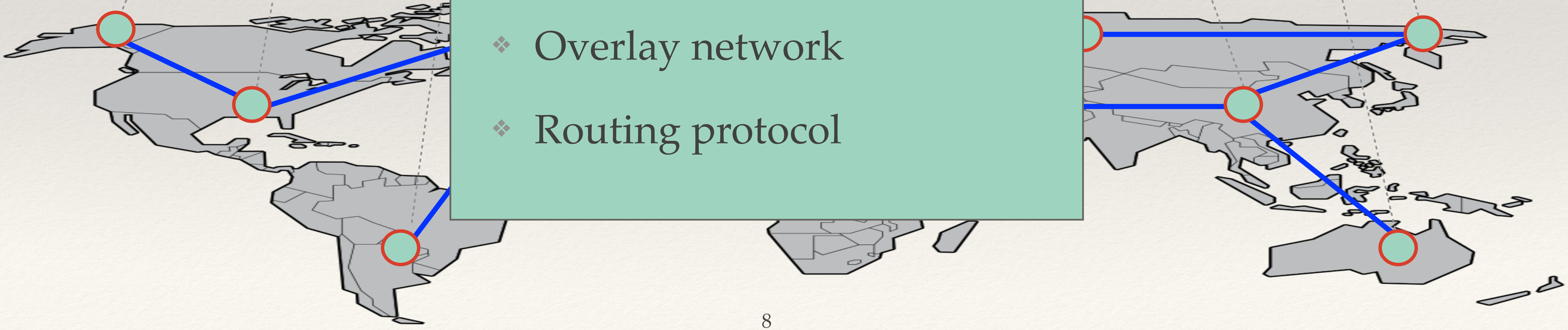
DHT

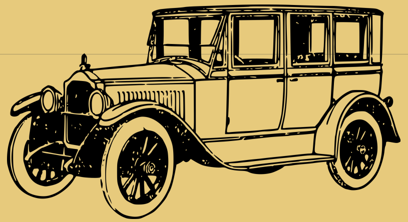
- Decentralised Systems
- Distribute (ℓ, v) pairs to nodes
- Supports Get(ℓ), Put(ℓ, v) operations



DHT

- Decentralised Systems
- Distribute (ℓ, v) pairs to nodes
- Supports Get(ℓ), Put(ℓ, v) operations
- Overlay network
- Routing protocol





Classic Applications of DHTs

Content Delivery Networks

Democratizing content publication with Cor

Michael J. Freedman, Eric Freudenthal, David Mazières

Faster Content Access in KAD

Moritz Steiner, Damiano Carra, and Ernst W. Biersa

Eurécom, Sophia–Antipolis, France

A DHT-based Infrastructure for Content-based Publish/Subscribe Services *

Xiaoyu Yang and Yiming Hu

Department of Electrical and Computer Engineering

University of Cincinnati, Cincinnati, OH 45221, USA

{yangxu,yhu}@ececs.uc.edu

Squirrel: A decentralized peer-to-peer web cache

Sitaram Iyer*
Rice University
6100 Main Street, MS-132
Houston, TX 77005, USA
ssiyer@cs.rice.edu

Antony Rowstron
Microsoft Research
7 J J Thomson Close
Cambridge, CB3 0FB, UK
antr@microsoft.com

Peter Druschel
Rice University

ABSTRACT

This paper presents a decentralized, peer-to-peer web cache called Squirrel. The key idea is to enable web browsers on desktop machines to share their local caches, to form an efficient and scalable web cache, without the need for dedicated hardware and the associated administrative cost. We propose and evaluate decentralized web caching algorithms for Squirrel, and discover that it exhibits performance comparable to a centralized web cache in terms of hit ratio, bandwidth usage and latency. It also achieves the benefits of decentralization, such as being scalable, self-organizing and resilient to

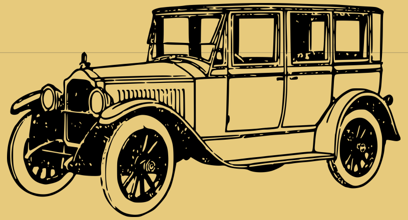
There is substantial literature in web caching [3, 6, 9, 20, 23, 24] and acterization [4]. This paper demons desirable and efficient to adopt a web caching in a corporate LAN ty in a single geographical region. tion, it shows how most of the funct of a traditional web cache can be self-organizing system that needs ministration, and is fault-resilient. elaborate on these ideas.

SCAN: A Dynamic, Scalable, and Efficient Content Distribution Network

Yan Chen, Randy H. Katz and John D. Kubiatowicz

Computer Science Division, University of California at Berkeley

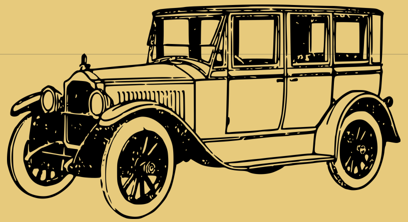
Abstract. We present *SCAN*, the Scalable Content Access Network.



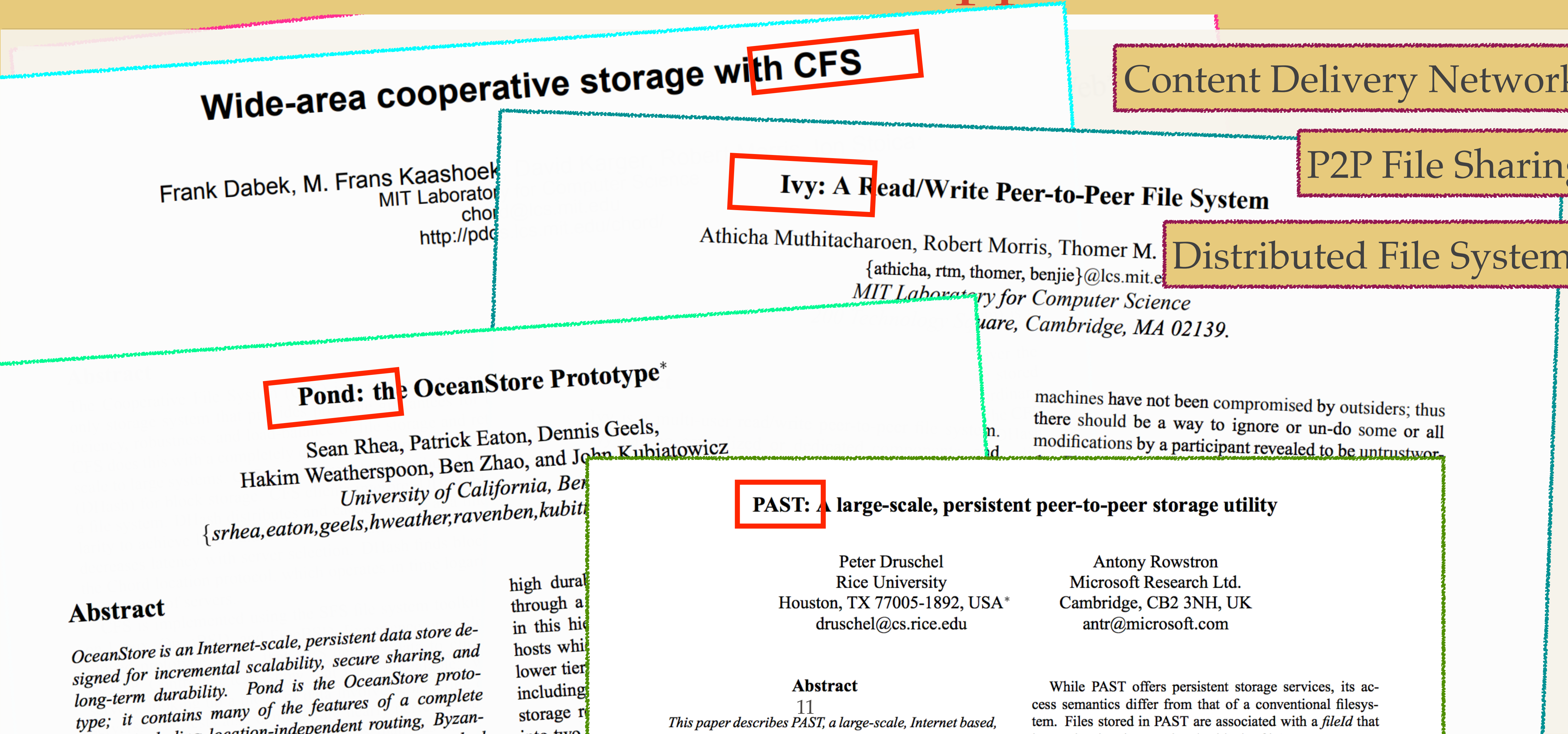
Classic Applications of DHTs

Content Delivery Networks

P2P File Sharing



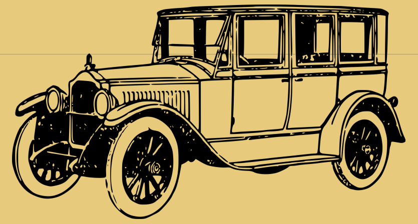
Classic Applications of DHTs



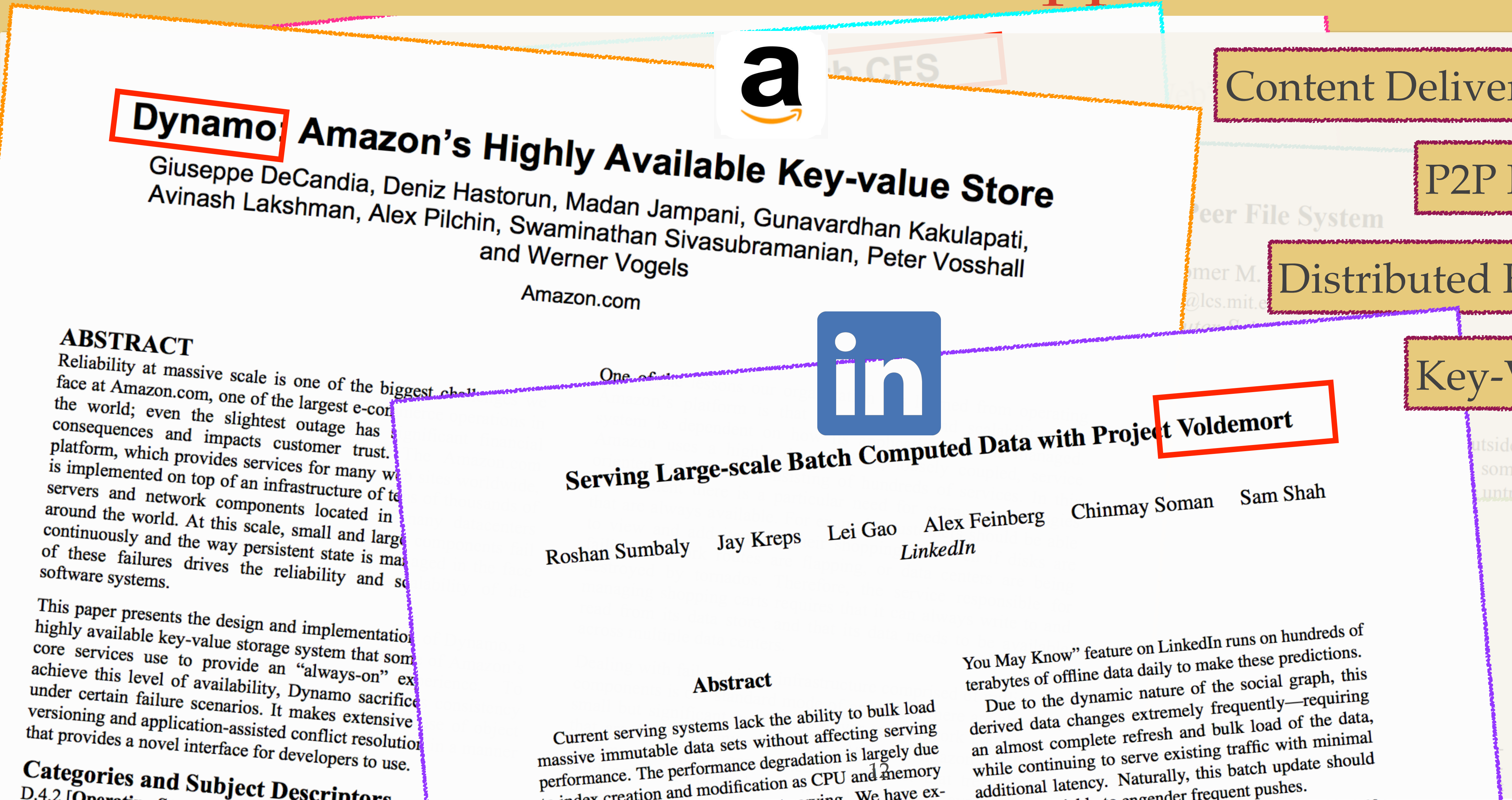
Content Delivery Networks

P2P File Sharing

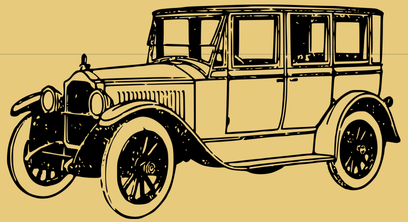
Distributed File Systems



Classic Applications of DHTs



- Content Delivery Networks
- P2P File Sharing
- Distributed File Systems
- Key-Value Stores



Classic Applications of DHTs

Cassandra - A Decentralized Structured Storage System

Avinash Lakshman
Facebook

Prashant Malik
Facebook

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.c
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Fi-

achieved scalability and h
provides a different interfac
does not support a full rel
provides clients with a sir
dynamic control over dat
lows clients to reason abo

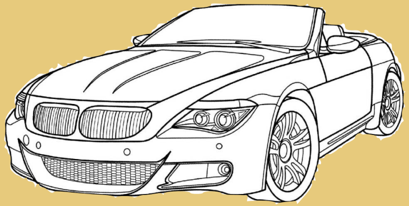
Amazon DynamoDB: A Seamlessly Scalable Non-relational Datastore

Swami Sivasubramanian
Amazon.com
Seattle, WA, USA
swami@amazon.com

ABSTRACT

- Content Delivery Networks
- P2P File Sharing
- Distributed File Systems
- Key-Value Stores
- NoSQL Databases

Recent Application of DHTs



Off-Chain Storage in Blockchains

- ethereum
- enigma
- STORJ.IO
- Filecoin

Content Delivery Networks

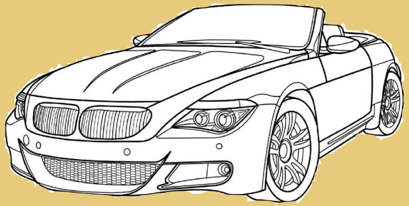
P2P File Sharing

Distributed File Systems

Key-Value Stores

NoSQL Databases

Recent Application of DHTs



Off-Chain Storage in Blockchains

ethereum

enigma

STORJ.IO

Filecoin

Encryption

Content Delivery Networks

P2P File Sharing

Distributed File Systems

Key-Value Stores

NoSQL Databases

Simple Standard Scheme

Put(ℓ, v)

- Apply PRF on label
- Encrypt value
- Store in DHT

Get(ℓ)

- Apply PRF on label
- Fetch value from DHT using pseudorandom label

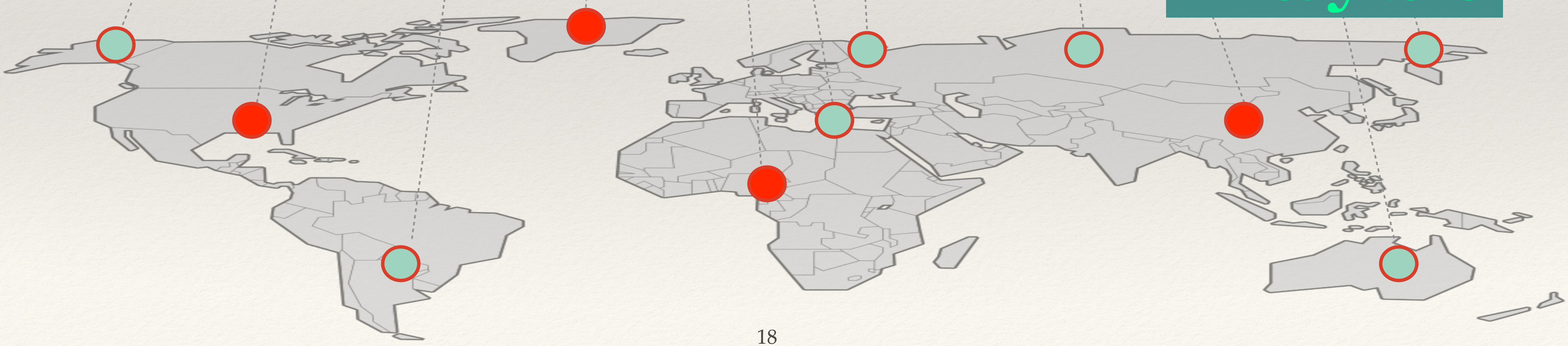
Q: What is the security of this standard scheme?

Leakage Preview

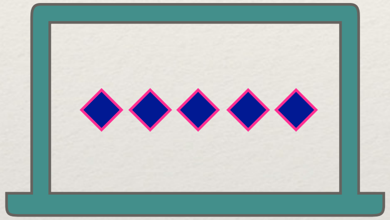
Q1: What information is learnt by Adversary about these pairs?

Q2: Does it only learn information about the pairs it stores?

Maybe



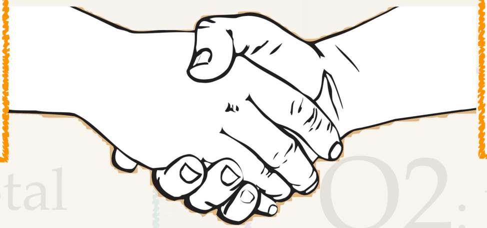
Relation to Structured Encryption



Learns about all the pairs

Leakage Preview

System architecture



Security

Infer a good approximation of total number of pairs!

Q2: Does it only learn information about the pairs it stores?

Maybe

Analyzing leakage in Distributed Systems is tricky!

- if DHTs are load balanced

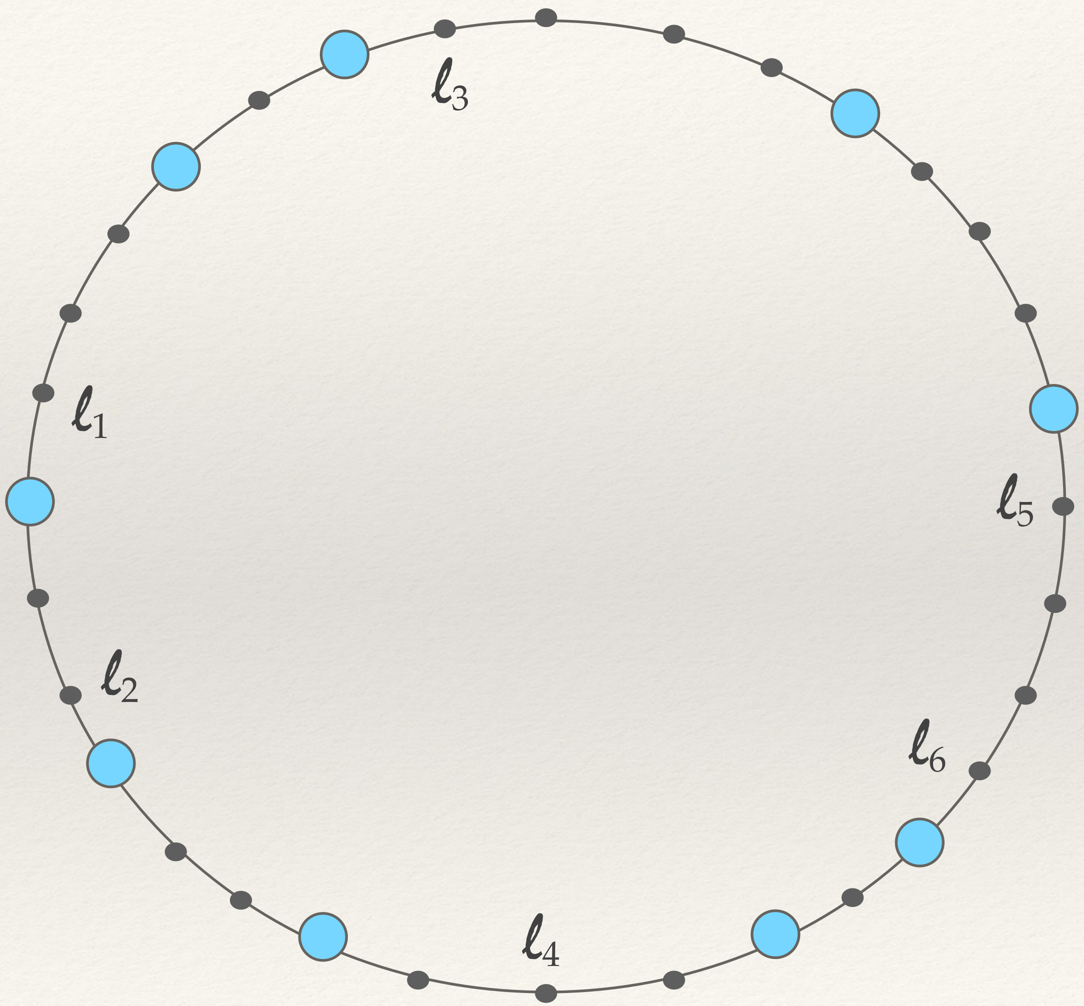
NO

Formalize the use of end-to-end encryption in DHTs

- [x] Chord DHT
- [x] Formalize DHTs
- [x] Formalize EDHTs
  - [x] Syntax
  - [x] Security
- [x] Analyze Standard Scheme
- [x] Extend to Transient Setting
- [x] Takeaways & Open Questions

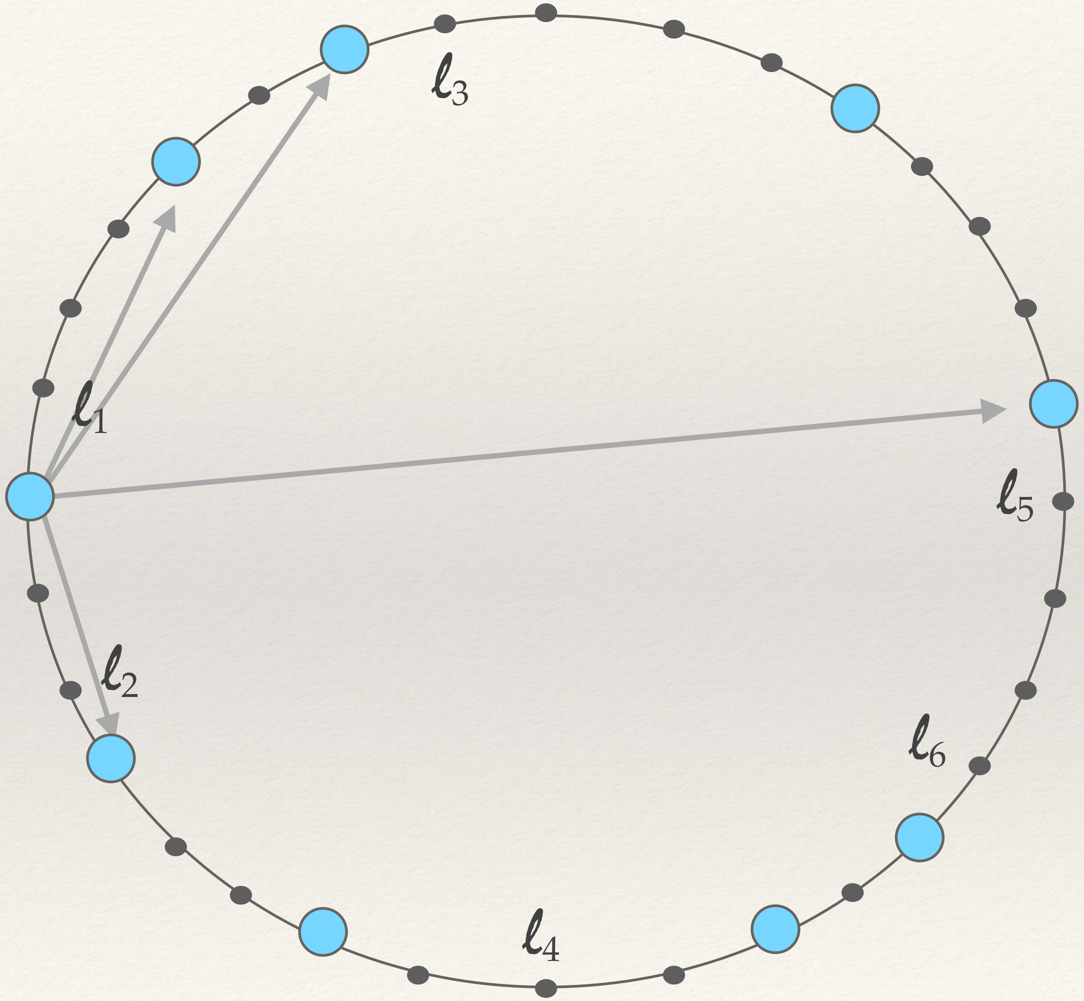
Chord DHT

- Address Space : **A**
- Two hash functions:
  - H_1 : hashes node ids to addresses
  - H_2: hashes labels to addresses
- server(ℓ): successor($H_2(\ell)$)



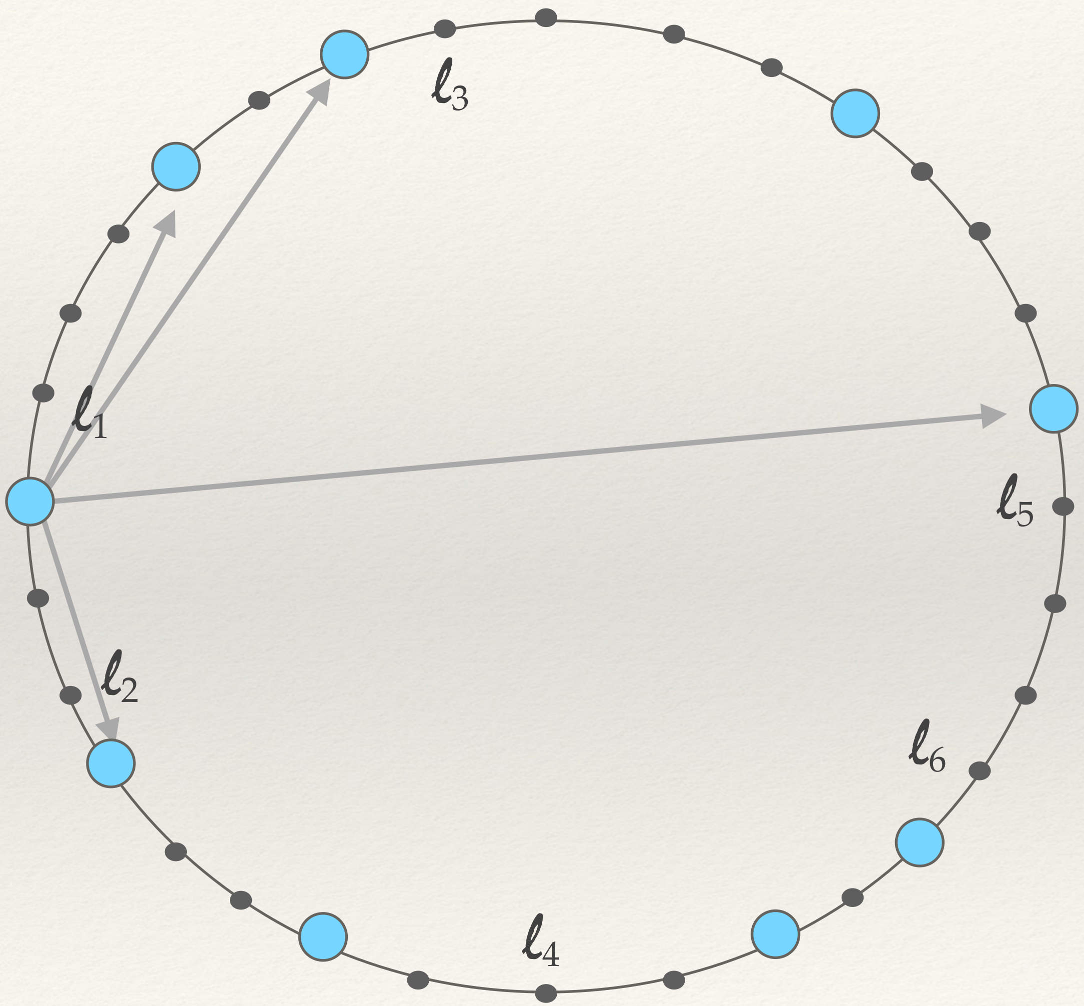
Chord DHT

- Address Space : **A**
- Two hash functions:
  - H_1 : hashes node ids to addresses
  - H_2: hashes labels to addresses
- server(ℓ): successor($H_2(\ell)$)
- route(a_1, a_2):
  - logarithmic sized routing tables
  - logarithmic sized paths



Chord DHT

- Address Space : **A**
- Two hash functions:
  - H_1 : hashes node ids to addresses
  - H_2: hashes labels to addresses
- server(ℓ): successor($H_2(\ell)$)
- route(a_1, a_2):
  - logarithmic sized routing tables
  - logarithmic sized paths



Chord DHT

- Address Space : **A**

  Overlay param

- ... ns:

  Allocation param

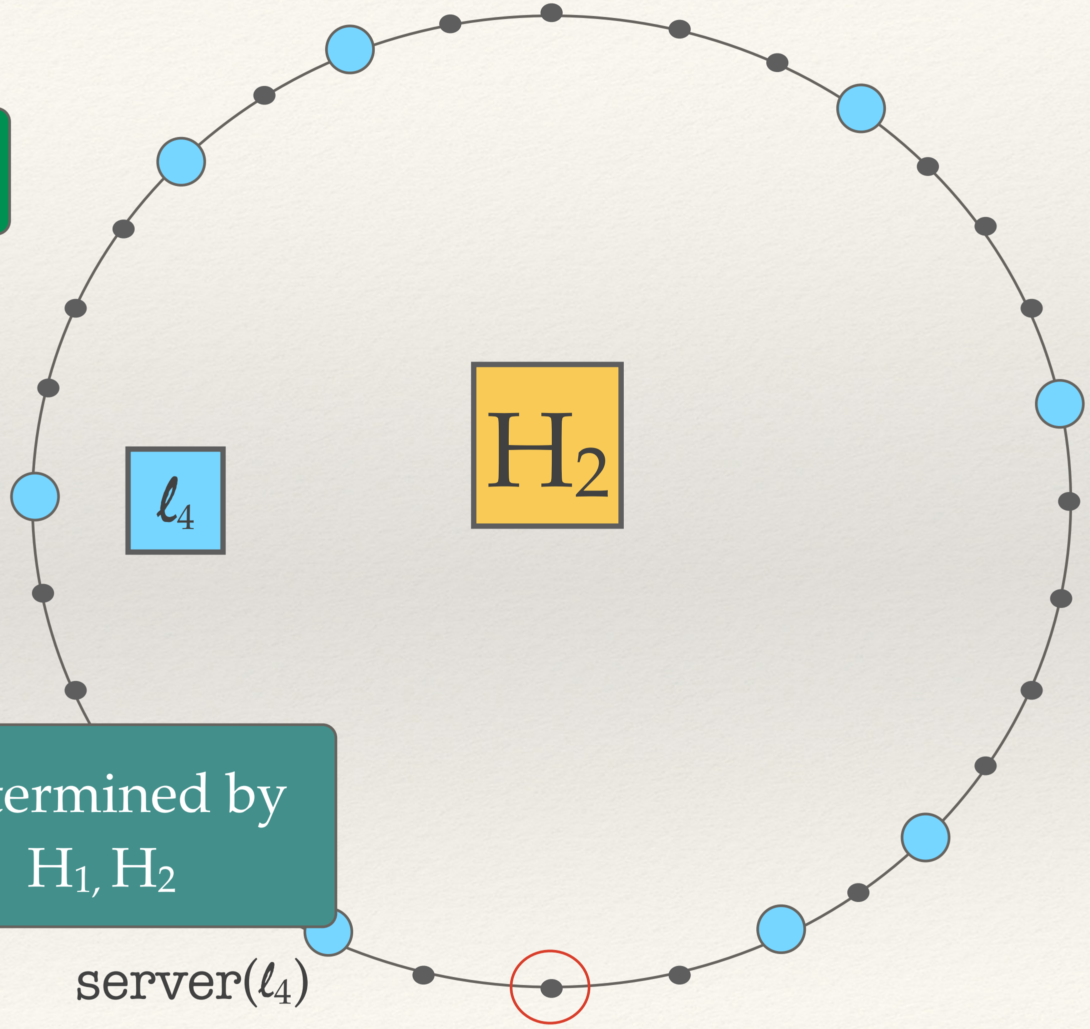
  - H_1 : has... node ids to addresses
  - H_2: hashes labels to addresses
- server(ℓ): successor(H_2(ℓ))

  Determined by H_1, H_2

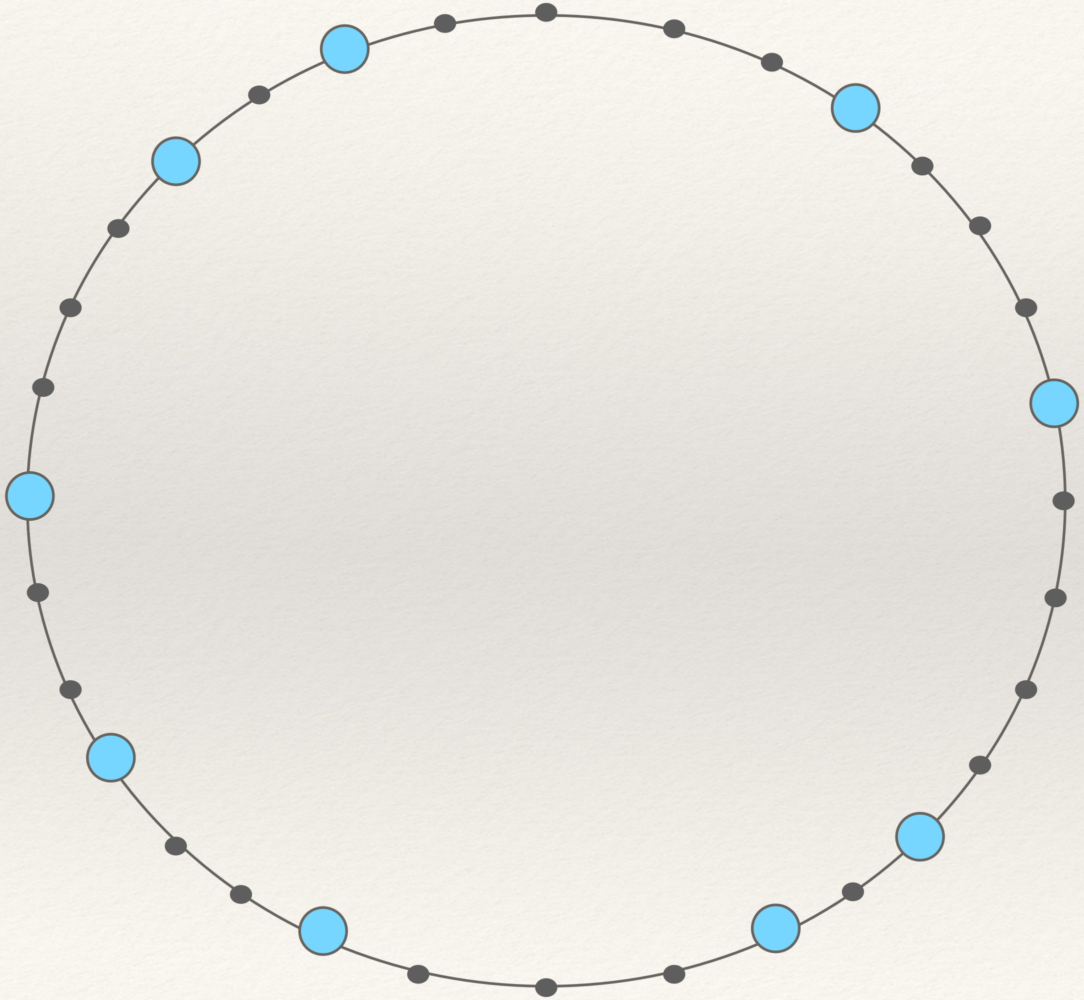
- route(a_1, a_2):

  Determined by H_1

  - logarithm... table
  - logarithm...

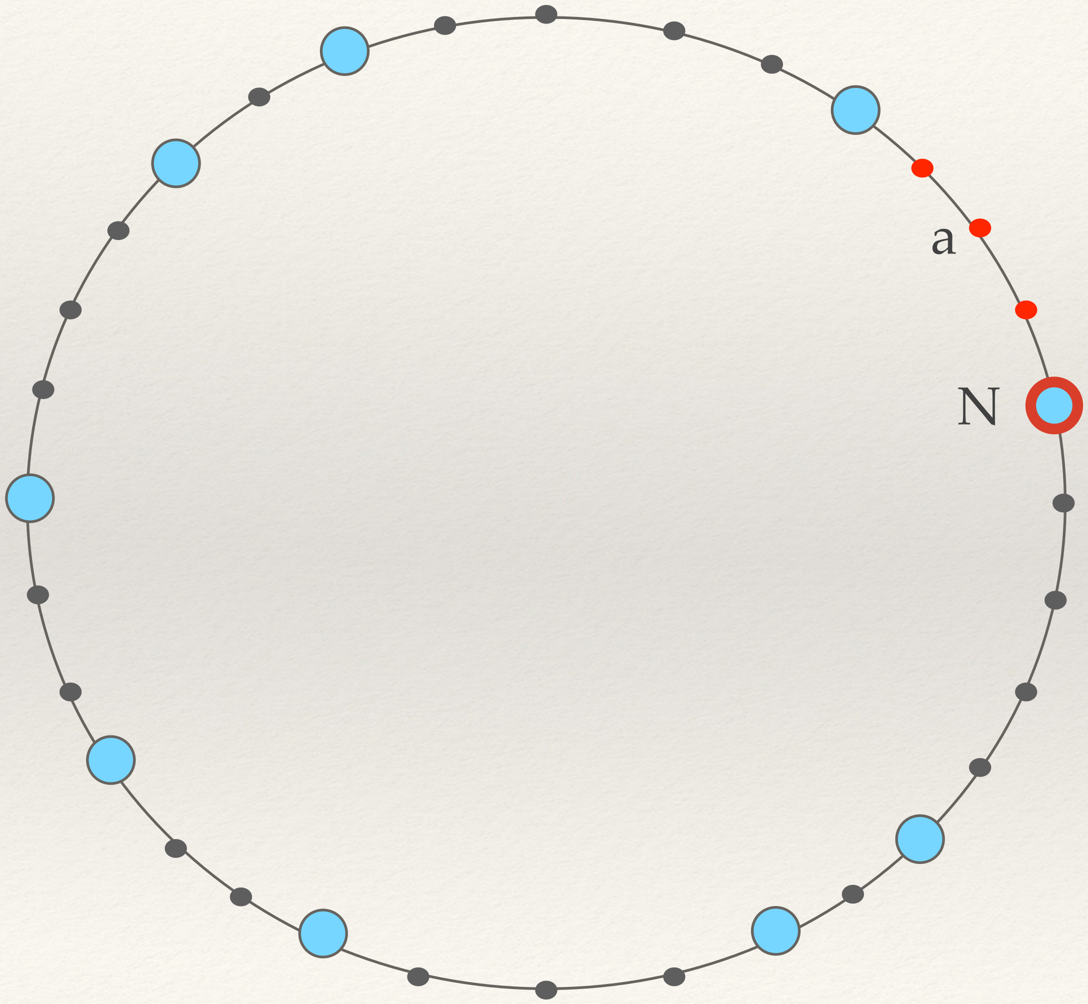


Chord: Visible addresses



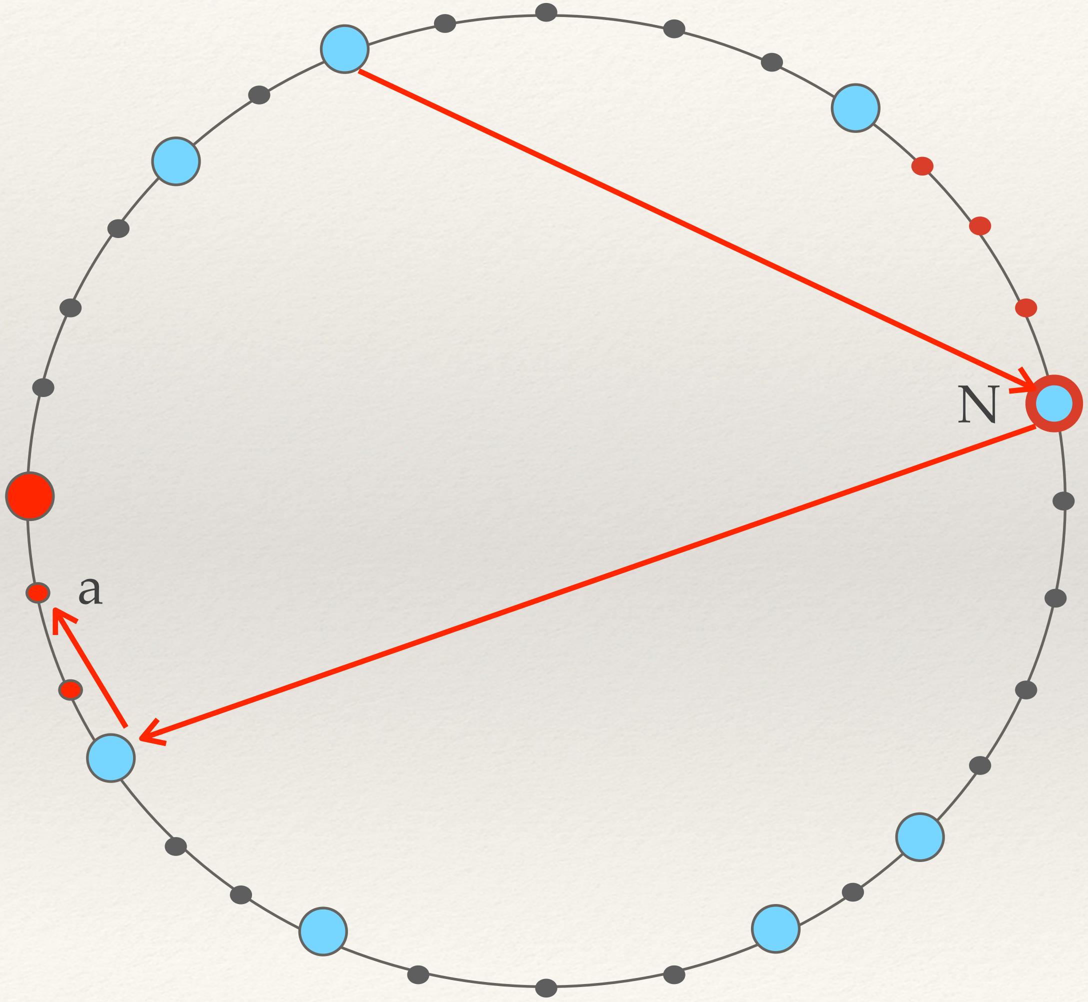
- Vis(N) : set of all addresses s.t. if $H_2(\ell) = a$ then either
  - server(ℓ) = N
  - N ϵ route(a)

Chord: Visible addresses



- Vis(N) : set of all addresses s.t. if $H_2(\ell)$ = a then either
  - server(ℓ) = N
  - N ϵ route(a)

Chord: Visible addresses



- Vis(N) : set of all addresses s.t. if $H_2(\ell) = a$ then either
  - server(ℓ) = N
  - N ∊ route(a)

- Chord DHT
- **Formalize DHTs**
- Formalize EDHTs
  - Syntax
  - Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions

Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs ω
- Alloc outputs ψ

Formalizing DHTs

DHT = (Overlay, Alloc, **Daemon**, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs ω
- Alloc outputs ψ

- Executed by all nodes
- all the time
- sends/receives messages
- stores/retrieves (label, value) pairs

Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs ω
- Alloc outputs ψ

- Executed by all nodes
- all the time
- sends/receives messages
- stores/retrieves (label, value) pairs

- Executed by client
- to store/retrieve (label/value) pair in/from network

Formalizing DHTs

DHT = (Overlay, Alloc, Daemon, Put, Get)

- Executed only once
- At the time of setup
- Overlay outputs ω
- Alloc outputs ψ

- Executed by all nodes

...es

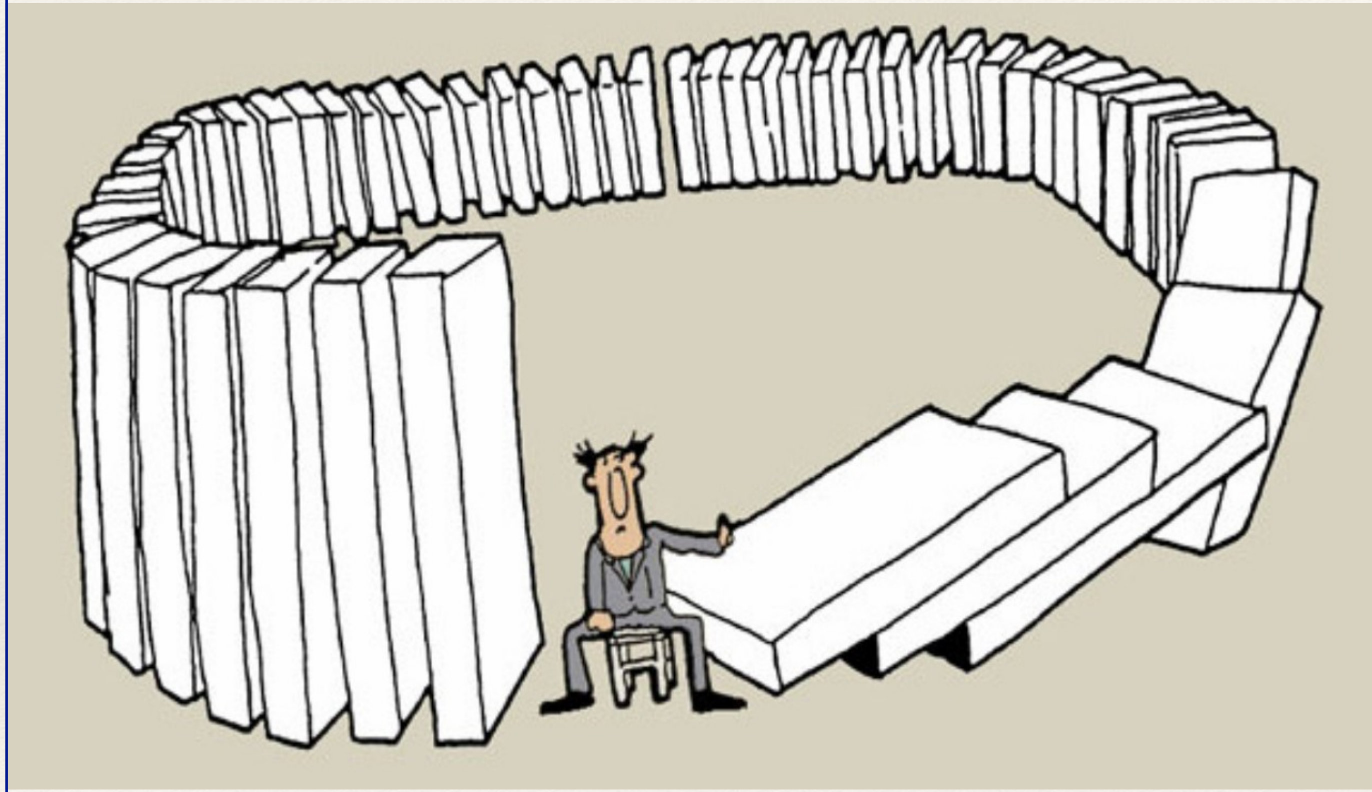
- $addr_\omega : \mathbf{N} \rightarrow \mathbf{A}$
- $server_{\omega,\psi} : \mathbf{L} \rightarrow \mathbf{A}$
- $route_\omega : \mathbf{A} \times \mathbf{A} \rightarrow 2^{\mathbf{A}}$

- Executed by client
- to store/retrieve (label/value) pair in/from network

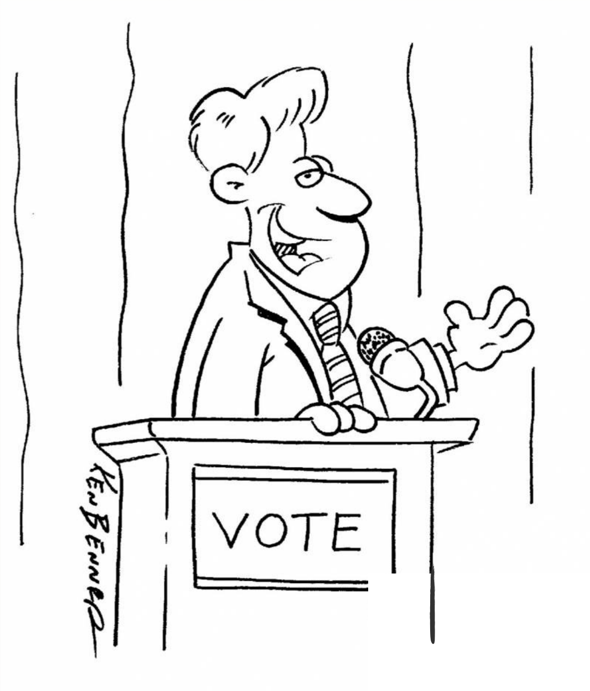
Properties of DHTs

Properties of DHTs

P1: Balance



P2: Non-committing allocations



“And if elected, I promise to keep making promises.”

P1: Balance

- Overlay ω is ε-balanced if ∀ labels ℓ, and all nodes N
  - $\Pr[\mathsf{server}(\ell) \in \mathsf{Vis}(N)] \leq \varepsilon$
  - Prob over choice of ψ

Prob of a label being visible to a node is bounded

- A DHT is (ε, δ)-balanced if
  - $\Pr[\ \omega \text{ is } \varepsilon\text{-balanced}] \geq 1 - \delta$
  - Prob over choice of ω

w/ prob 1-δ the sampled overlay is balanced

P1: Balance for Chord

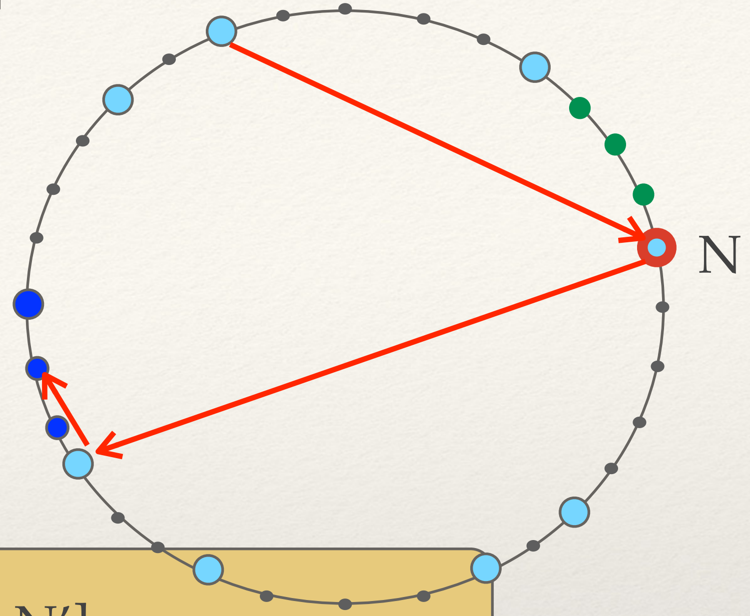
$\Pr[\text{server}(\ell) \in \text{Vis}(N)]$

$= \Pr[\text{server}(\ell) = N] + \Pr[N \in \text{route}(\text{server}(\ell))]$

$\propto$ length of arc of N

with high prob max arc $\leq (4\ |\mathbf{A}|\log n)/n$

- $\Pr[\text{server}(\ell) = N']$
- Pr [N on log sized path to N′]



Balance of Chord

Th :

Chord is (ε, δ)-balanced for

$$\varepsilon = \frac{(4 \log n)}{n} + \frac{4n \log^2 n}{|A|} \quad \text{and} \quad \delta = 1/n$$

- If $|A| = 2^{512} \Longrightarrow n^2 \log n < |A|$, even for $n = 2^{250} \Longrightarrow \varepsilon = O(\log n / n)$

P2: Non-committing allocations

- be able to change/program ψ
  - given a label ℓ and an address a
  - set $\psi(\ell) = a$

- Chord DHT
- Formalize DHTs
- **Formalize EDHTs**
  - Syntax
  - Security
- Analyze Standard Scheme
- Extend to Transient Setting
- Takeaways & Open Questions

Formalizing EDHTs : Syntax

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get)

Formalizing EDHTs : Syntax

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get)

Same as before

Formalizing EDHTs : Syntax

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get)

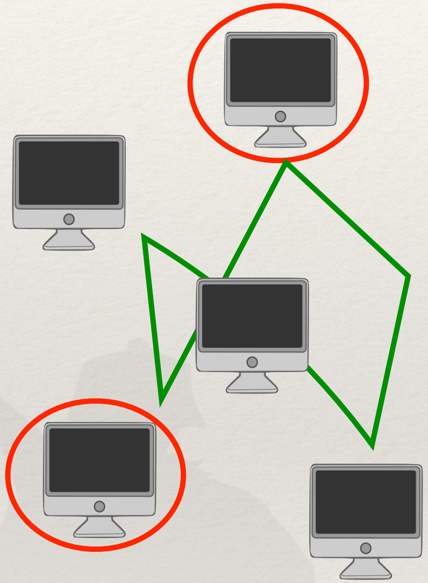
- Executed by Client
- Generates reqd. keys for client

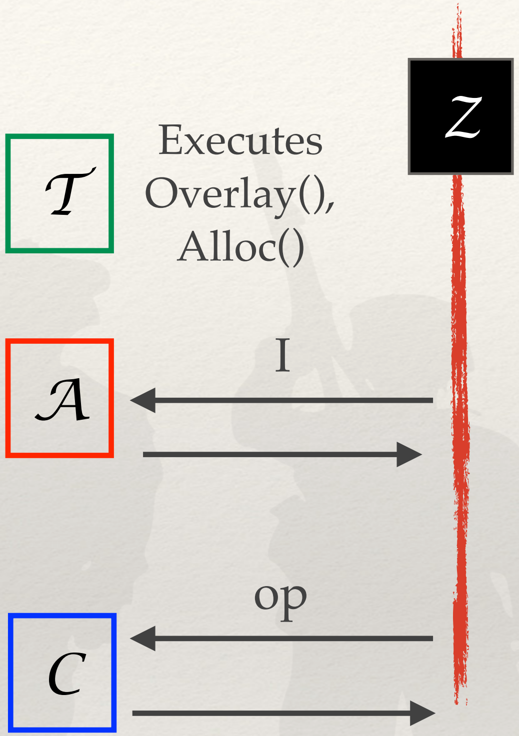
Same as before

EDHTs Security

Real

Ideal

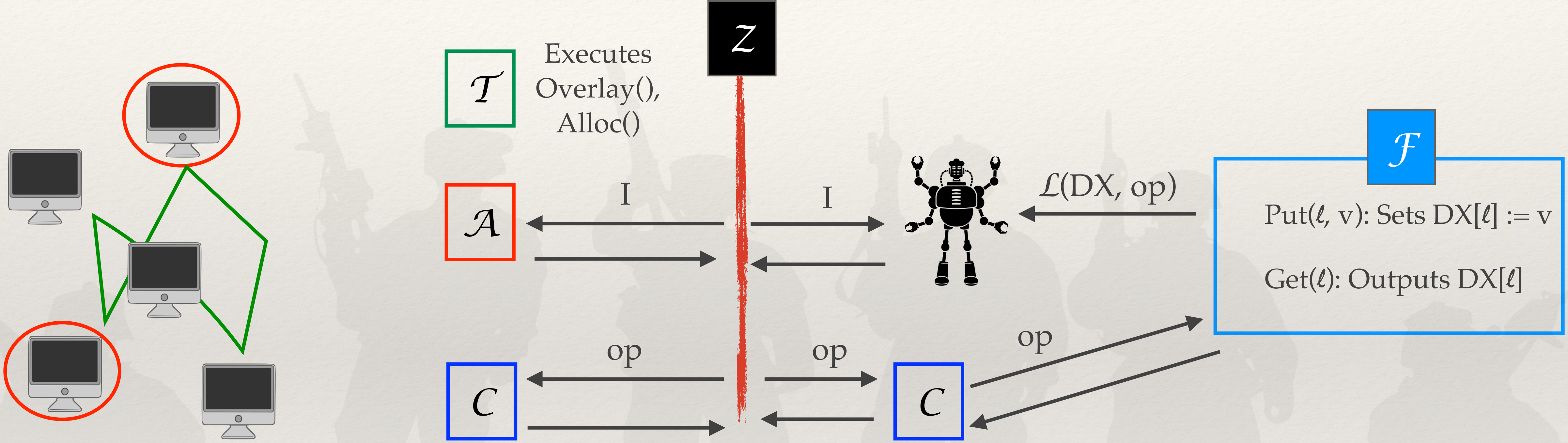




EDHTs Security

Real

Ideal



Real $\approx$ Ideal

- Chord DHT
- Formalize DHTs
- Formalize EDHTs
  - Syntax
  - Security
- **Analyze Standard Scheme**
- Extend to Transient Setting
- Takeaways & Open Questions

Standard Scheme : Construction

$Gen(1^k)$

- Sample $K_1 \leftarrow \{0, 1\}^k$
- $K_2 \leftarrow SKE.Gen(1^k)$
- Output (K_1, K_2)

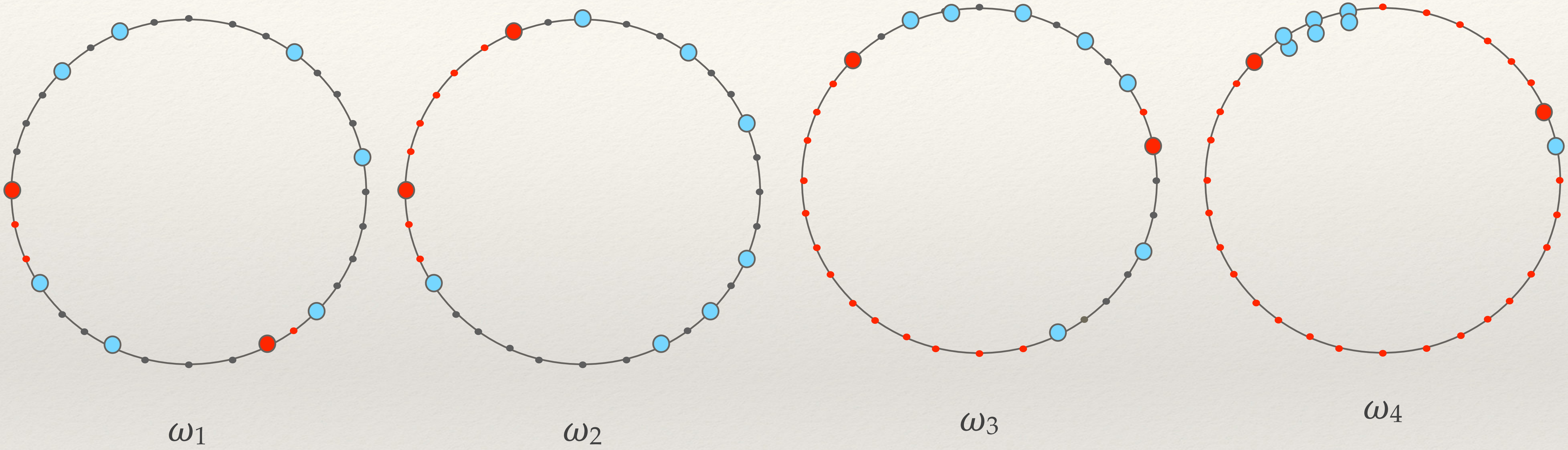
$Put(K, \ell, v)$

- $K = (K_1, K_2)$
- $t = F_{K1}(\ell)$
- $e = SKE.Enc_{K2}(v)$
- $DHT.Put(t, e)$

$Get(K, \ell)$

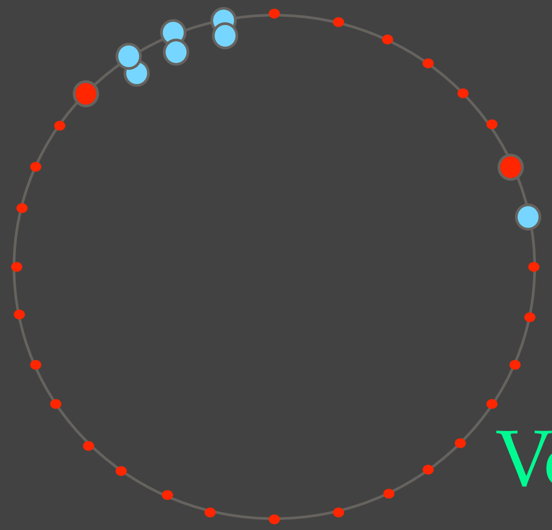
- $K = (K_1, K_2)$
- $t = F_{K1}(\ell)$
- $e \leftarrow DHT.Get(t)$
- $v \leftarrow SKE.Dec_{K2}(e)$

Understanding Leakage



Q: Is there any gain over STE leakage?

YES



Very unlikely*

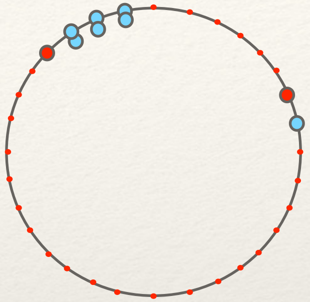
Understanding Leakage

DHT is (ε, δ)-balanced if

Pr[ω is ε-balanced] ≥ 1 - δ

Pr[label being visible to a node] ≤ ε

Sampling a “bad” overlay is unlikely



$\mathcal{L}$:

leaks qeq(ℓ) with probability min(1, t·ε)

probabilistic

affected by balancing properties of DHT

Standard Scheme: Security

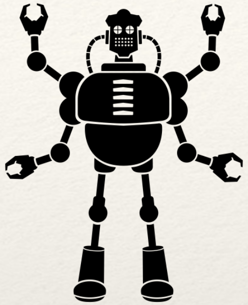
Th :

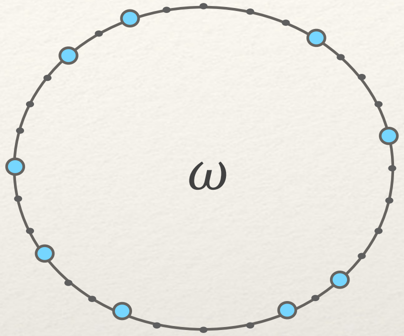
If DHT is (ε, δ)-balanced and
has non-committing overlays, then
EDHT is $\mathcal{L}_{\varepsilon}$-secure
with prob at least $1 - \delta - \text{negl}(k)$

Challenges in Proof

$\mathcal{L}$

needs to generate leakages compatible with ω

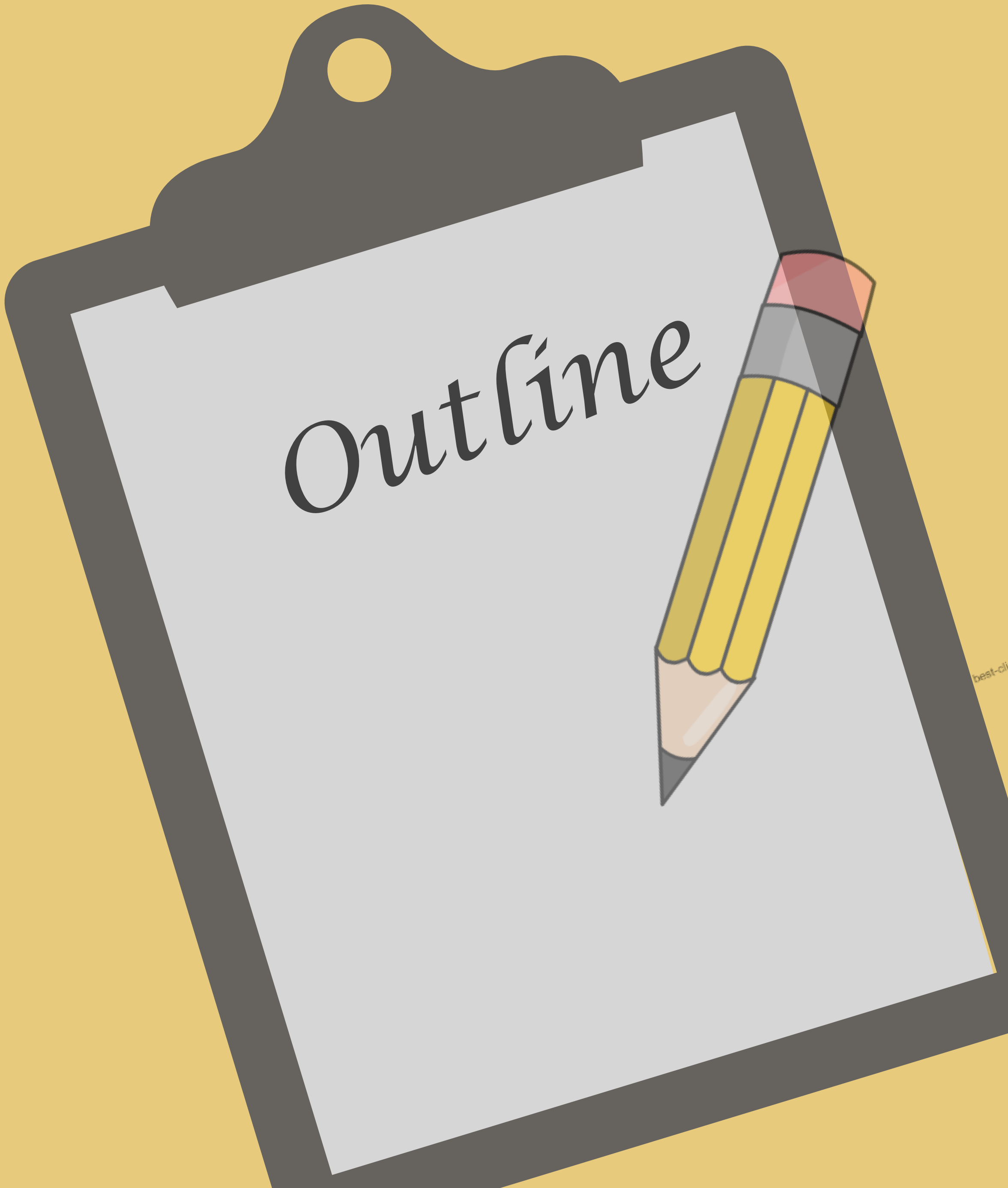




Two options:

- leak all the queries
- $\mathcal{L}$ generates ω

- Chord DHT
- Formalize DHTs
- Formalize EDHTs
    - Syntax
    - Security
- Analyze Standard Scheme
- **Extend to Transient Setting**
- Takeaways & Open Questions



Transient Setting

Nodes can leave/enter the network

Syntax

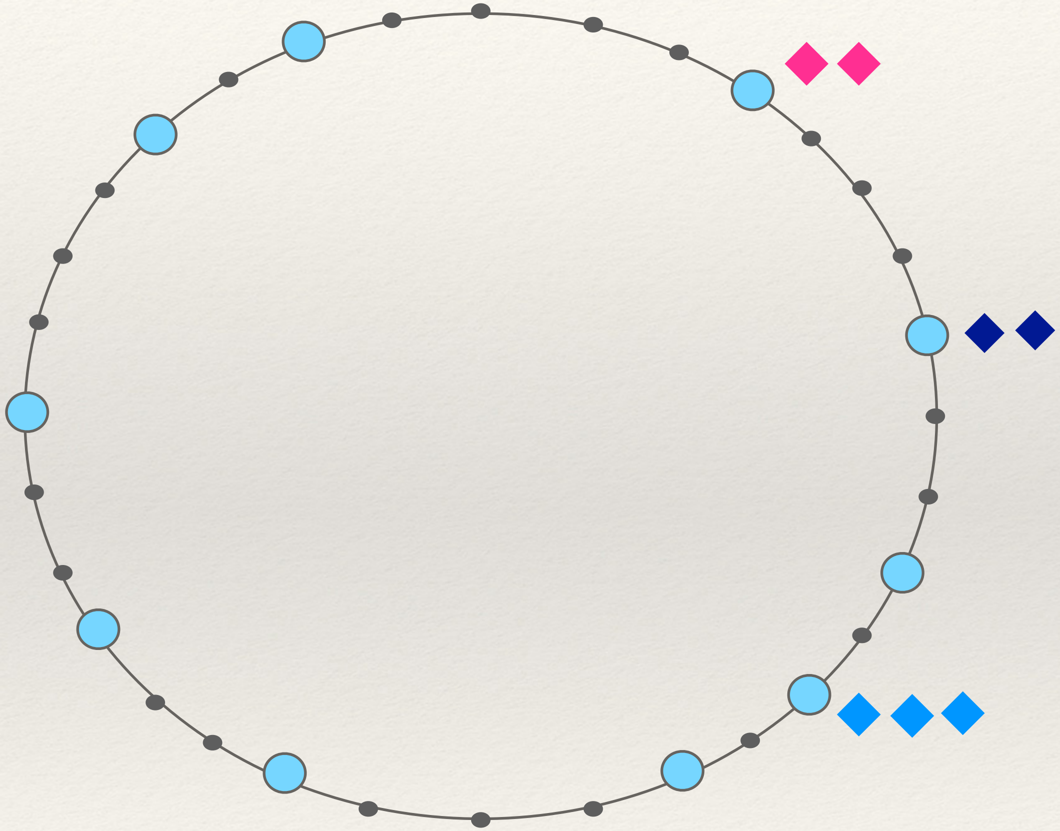
DHT = (Overlay, Alloc, Daemon, Put, Get, **Leave, Join**)

EDHT = (Gen, Overlay, Alloc, Daemon, Put, Get, **Leave, Join**)

- Run by node wishing to leave the network

- Run by node wishing to join the network

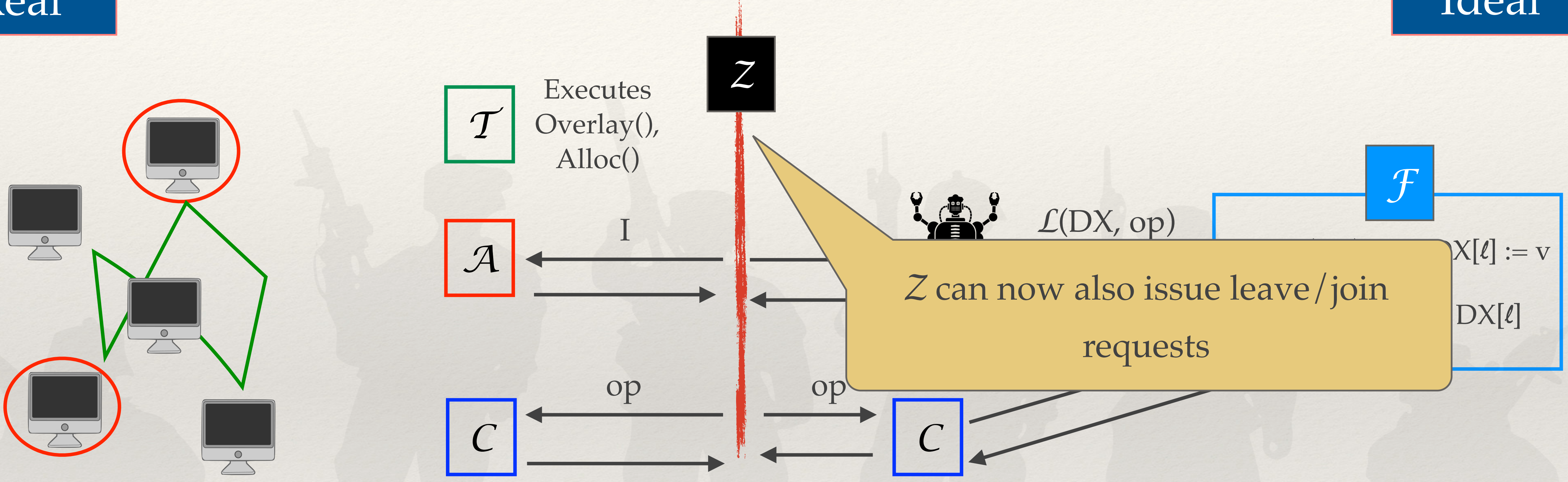
Leave/Join in Chord



Security : Transient EDHTs

Real

Ideal



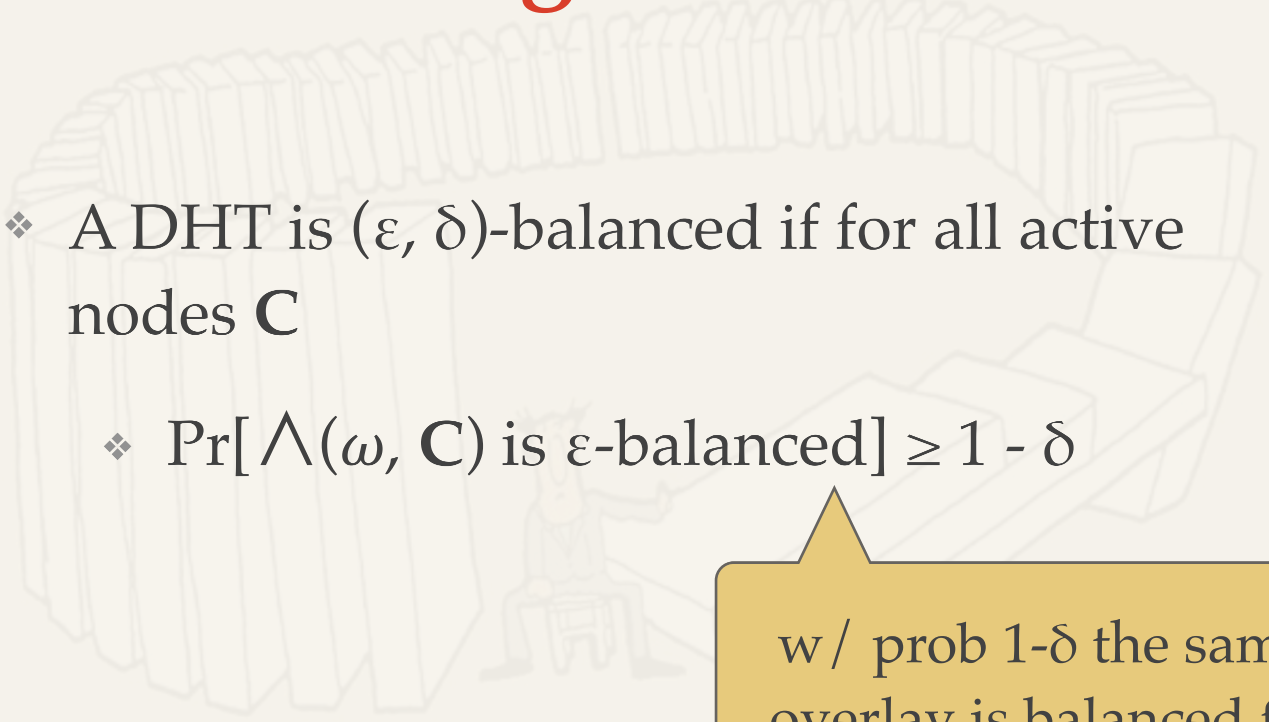
$\mathcal{Z}$ can now also issue leave/join requests

Real $\approx$ Ideal

Properties of DHTs

P1: Balance

Stronger notion

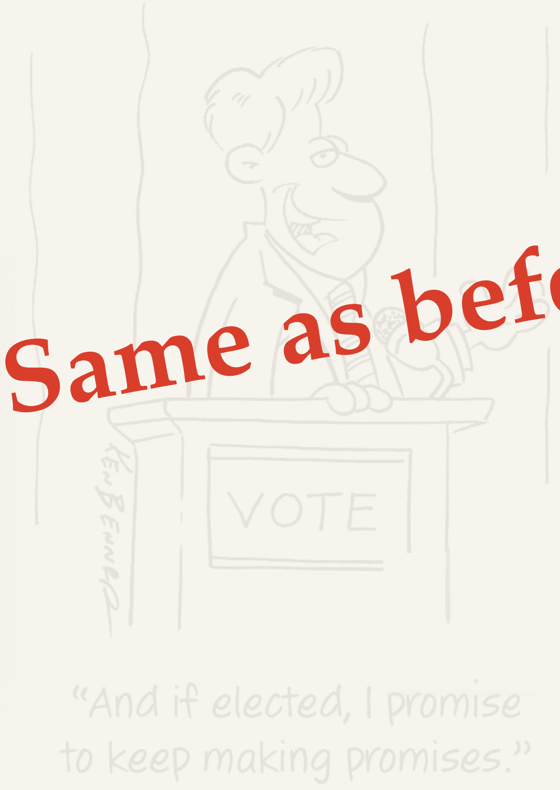


- A DHT is (ε, δ)-balanced if for all active nodes **C**
    - Pr[$\bigwedge(\omega, \mathbf{C})$ is ε-balanced] ≥ 1 - δ

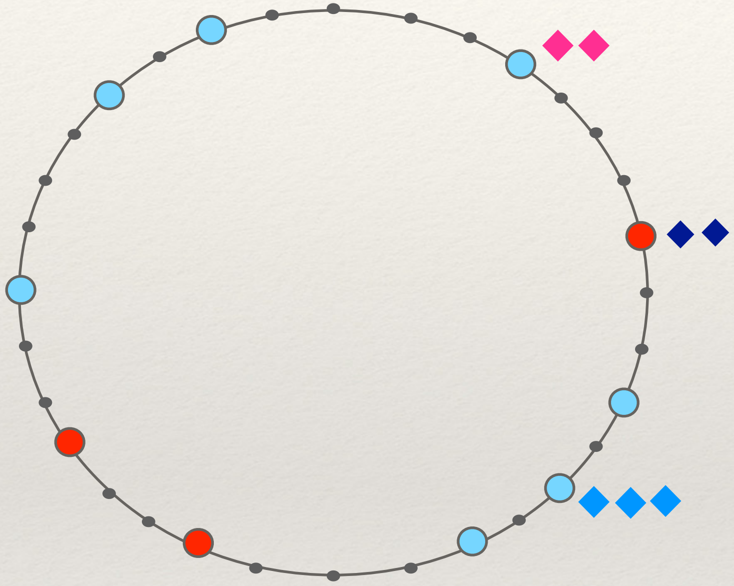
w/ prob 1-δ the sampled overlay is balanced for all nodes **C**

P2: Non-committing allocations

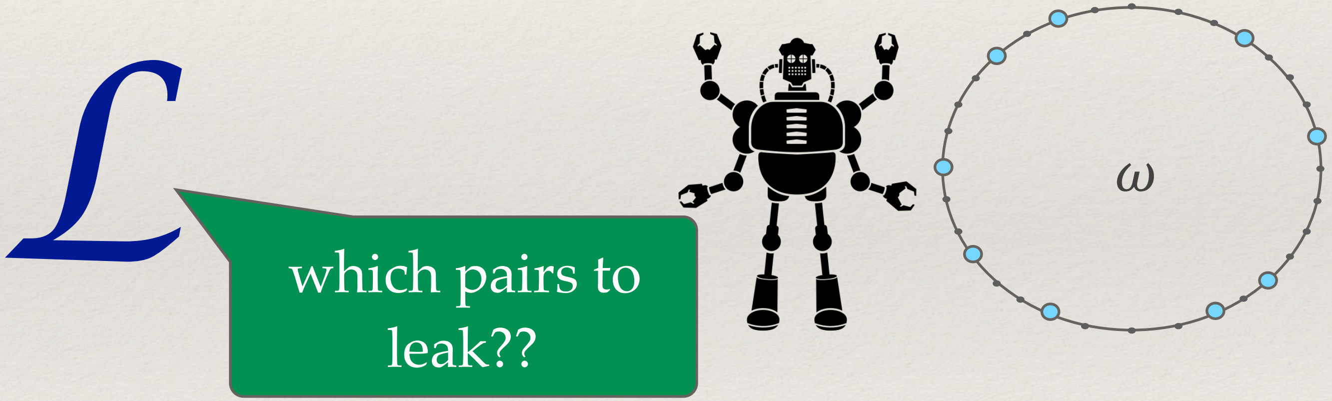
Same as before



Understanding Leakage



- Additional pairs become visible during leave/join

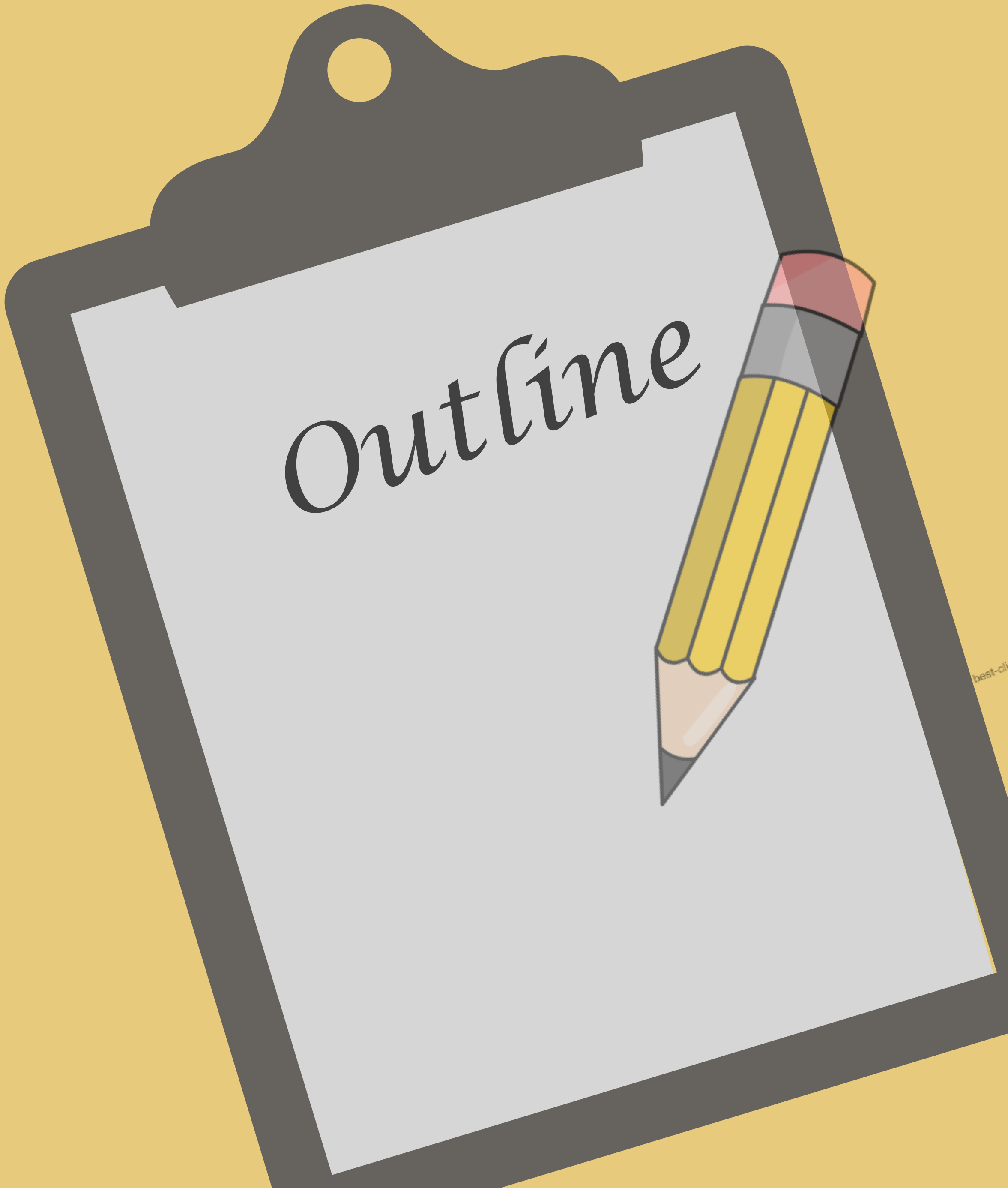


leaks qeq(ℓ) of all the previous pairs

Transient Standard Scheme: Security

Th :

If transient DHT is (ε, δ)-balanced and has non-committing overlays, then transient EDHT is $\mathcal{L}_\varepsilon$-secure with prob at least $1 - \delta - \text{negl}(k)$



- Chord DHT
- Formalize DHTs
- Formalize EDHTs
  - Syntax
  - Security
- Analyze Standard Scheme
- Extend to Transient Setting
- **Takeaways & Open Questions**

Key Takeaways

- Expected Leakage Analysis
  - Earlier : leakage functions were deterministic
  - Now : probabilistic
- Co-design distributed systems with reqd. crypto
- Building secure distributed systems can be tricky
  - Intuitions are not always right
- Distributing data can help in leakage suppression

Open Questions

- Tighter analysis of Transient Chord
- Study of (ε, δ) of other DHTs
  - Kademlia, Koorde
- Design other EDHTs
- Security in UC setting


nk you