

Encrypted Distributed Hash Tables

Archita Agarwal, Seny Kamara



[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



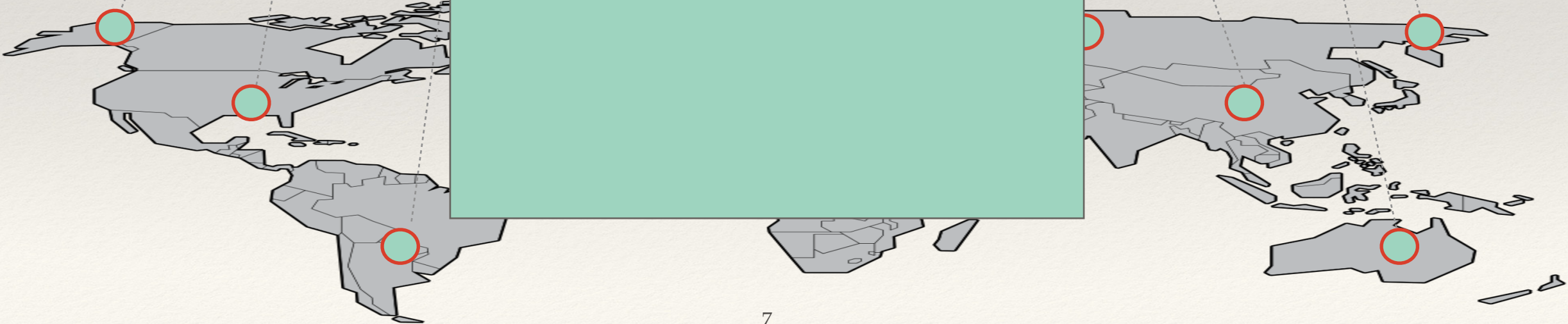
DHT

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



DHT

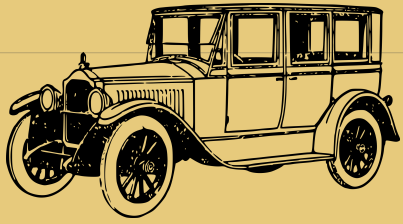
- ❖ Decentralised Systems
- ❖ Distribute (ℓ, v) pairs to nodes
- ❖ Supports $\text{Get}(\ell)$, $\text{Put}(\ell, v)$ operations



DHT

- ❖ Decentralised Systems
- ❖ Distribute (ℓ, v) pairs to nodes
- ❖ Supports $\text{Get}(\ell)$, $\text{Put}(\ell, v)$ operations
- ❖ Overlay network
- ❖ Routing protocol





Classic Applications of DHTs

Democratizing content publication with Core Content Delivery Networks

Michael J. Freedman, Eric Freudenthal, David Mazières
New York University

Faster Content Access in KAD

Moritz Steiner, Damiano Carra, and Ernst W. Biersack
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
6100 Main Street
Houston, TX 77005, USA
druschel@rice.edu

ABSTRACT

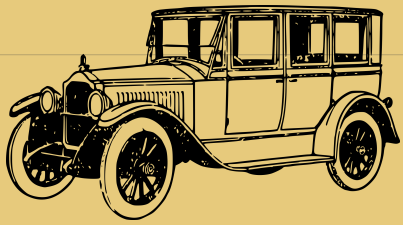
This paper presents a decentralized, peer-to-peer web cache called Squirrel. The key idea is to enable web browsers on desktop machines to share their local caches, to form an efficient and scalable web cache, without the need for dedicated hardware and the associated administrative cost. We propose 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 node failures, while imposing low overhead on the participants.

There is substantial literature in web caching [3, 6, 9, 20, 23, 24] and characterization [4]. This paper demonstrates a desirable and efficient way to adopt a web caching in a corporate LAN type environment in a single geographical region. Under this model, it shows how most of the functions of a traditional web cache can be implemented in a self-organizing system that needs no central administration, and is fault-resilient. We 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



BitTorrent™

Faster Content Delivery
 Moritz Steiner, Damiano
 Eurécom, Sophi


ation with Cor
 nal, David Mazières
 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 Row
 Microsoft Res
 7 J J Thoms
 Cambridge, C
 antr@micos

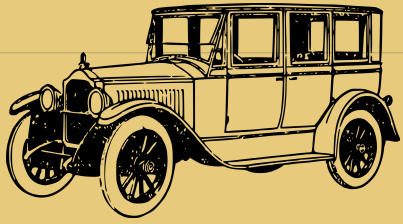
Peter Druschel
 Rice University
 6100 Main Str
 Houston, TX 7
 uschel@c



: A Dynam
 Content Dis
 Efficient
 Work

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

Wide-area cooperative storage with CFS

Frank Dabek, M. Frans Kaashoek
MIT Laboratory for Computer Science
<http://pdos.csail.mit.edu/cfs/>

Content Delivery Networks

Ivy: A Read/Write Peer-to-Peer File System

Athicha Muthitacharoen, Robert Morris, Thomer M. Chong
{athicha, rtm, thomer, benjie}@lcs.mit.edu
MIT Laboratory for Computer Science
32 Vassar Street, Cambridge, MA 02139.

P2P File Sharing

Distributed File Systems

Pond: the OceanStore Prototype*

Sean Rhea, Patrick Eaton, Dennis Geels,
Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz
University of California, Berkeley
{srhea,eaton,geels,hweather,ravenben,kubit

machines have not been compromised by outsiders; thus there should be a way to ignore or un-do some or all modifications by a participant revealed to be untrustworthy.

PAST: A large-scale, persistent peer-to-peer storage utility

Peter Druschel
Rice University
Houston, TX 77005-1892, USA*
druschel@cs.rice.edu

Antony Rowstron
Microsoft Research Ltd.
Cambridge, CB2 3NH, UK
antr@microsoft.com

Abstract

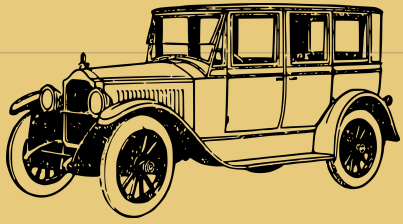
OceanStore is an Internet-scale, persistent data store designed for incremental scalability, secure sharing, and long-term durability. Pond is the OceanStore prototype; it contains many of the features of a complete system including location-independent routing, Byzantine fault tolerance, and high durability. This paper describes Pond, a large-scale, Internet based, persistent peer-to-peer storage utility.

high durability through a multi-tier architecture in this high-speed network environment. hosts which are in the lower tier of the storage hierarchy including storage redundancy and fault tolerance.

Abstract

11
This paper describes PAST, a large-scale, Internet based, persistent peer-to-peer storage utility.

While PAST offers persistent storage services, its access semantics differ from that of a conventional filesystem. Files stored in PAST are associated with a *fileId* that



Classic Applications of DHTs



Dynamo Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

Content Delivery Networks

P2P File Sharing

Distributed File Systems

Key-Value Stores

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce companies in the world; even the slightest outage has significant consequences and impacts customer trust. Dynamo is a platform, which provides services for many workloads and is implemented on top of an infrastructure of thousands of servers and network components located in multiple data centers around the world. At this scale, small and large failures occur continuously and the way persistent state is managed in the presence of these failures drives the reliability and scalability of the software systems.

This paper presents the design and implementation of a highly available key-value storage system that some of our core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of versioning and application-assisted conflict resolution to provide a novel interface for developers to use.

Categories and Subject Descriptors
D.4.2 [Operating Systems]...



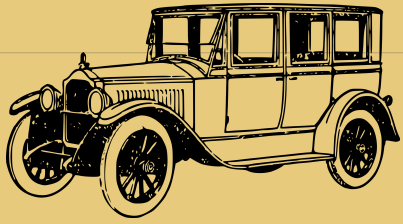
Serving Large-scale Batch Computed Data with Project Voldemort

Roshan Sumbaly Jay Kreps Lei Gao Alex Feinberg Chinmay Soman Sam Shah
LinkedIn

Abstract

Current serving systems lack the ability to bulk load massive immutable data sets without affecting serving performance. The performance degradation is largely due to the overhead of index creation and modification as CPU and memory usage increase. We have extended our existing serving system to support bulk loading of data sets.

You May Know" feature on LinkedIn runs on hundreds of terabytes of offline data daily to make these predictions. Due to the dynamic nature of the social graph, this derived data changes extremely frequently—requiring an almost complete refresh and bulk load of the data, while continuing to serve existing traffic with minimal additional latency. Naturally, this batch update should be performed during off-peak hours to avoid engendering frequent pushes.



Classic Applications of DHTs



Cassandra - A Decentralized Structured Storage System

Avinash Lakshman
Facebook

Prashant Malik
Facebook

Content Delivery Networks

P2P File Sharing

Distributed File Systems

Key-Value Stores

NoSQL Databases

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-

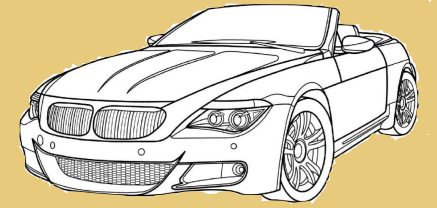
achieved scalability and l... provides a different interfac... does not support a full rel... provides clients with a sir... dynamic control over dat... lows clients to reason abo... data represented in the m...



Amazon DynamoDB: A Seamlessly Scalable Non-relational Datastore

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

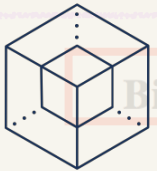
Recent Application of DHTs



Off-Chain Storage in Blockchains



ethereum



enigma



STORJ.IO



Filecoin

Content Delivery Networks

P2P File Sharing

Distributed File Systems

Key-Value Stores

NoSQL Databases

Cassandra: A Decentralized Structured Storage System

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

{fayc, dean, sanjay, wilson, kenton3b, tushar, fikes, gruber}@google.com

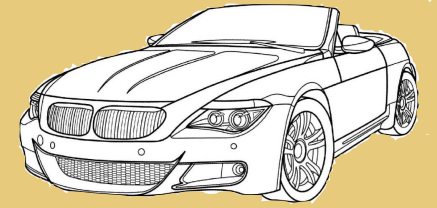
Google, Inc.



Amazon DynamoDB: A Seamlessly Scalable
Non-relational Datastore

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

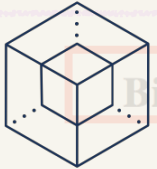
Recent Application of DHTs



Off-Chain Storage in Blockchains



ethereum



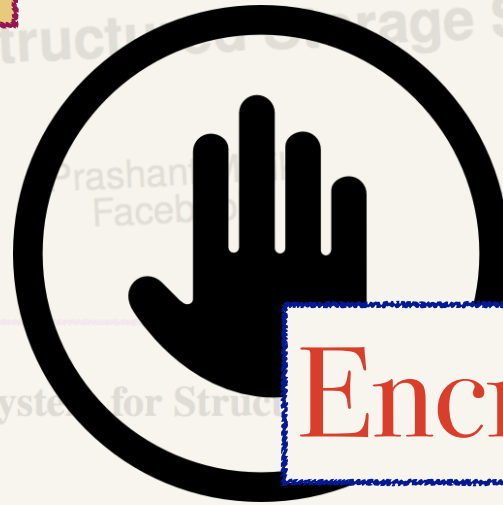
enigma



STORJ.IO



Filecoin



Encryption

Content Delivery Networks

P2P File Sharing

Distributed File Systems

Key-Value Stores

NoSQL Databases



Amazon **DynamoDB: A Seamlessly Scalable Non-relational Datastore**

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

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

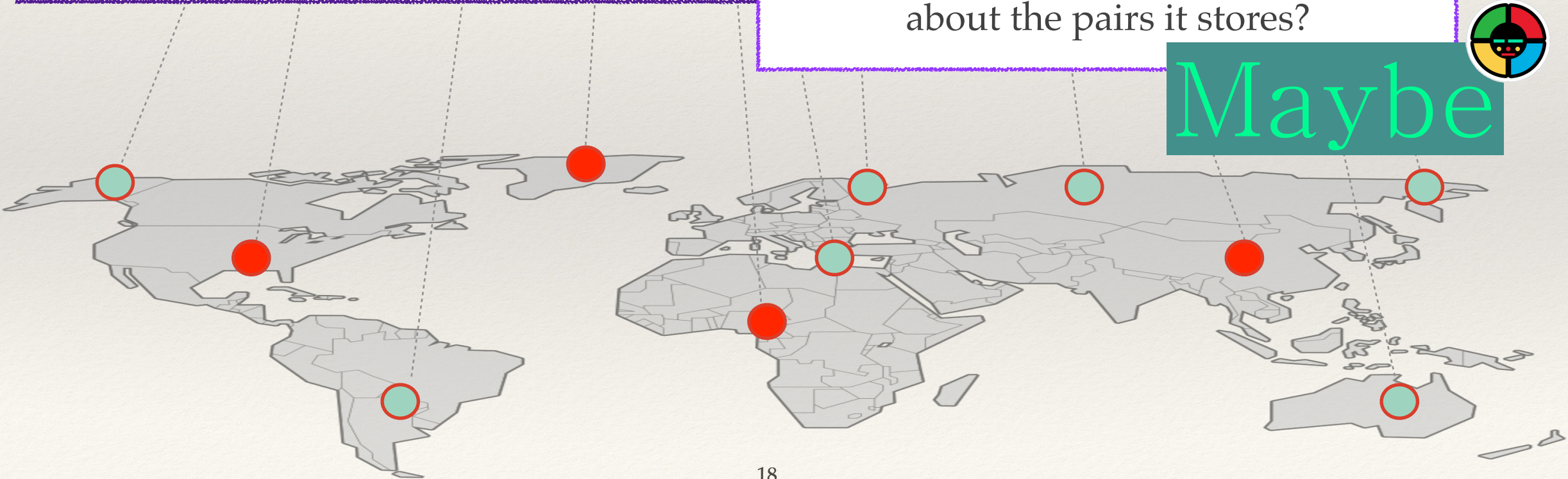
Q: What is the security of this standard scheme?

Leakage Preview

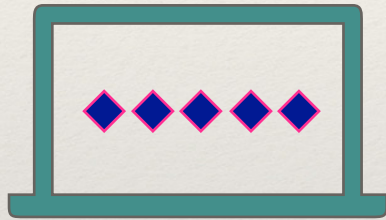
Q1: What information is learnt by **Adversary** about these pairs?

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

Maybe



Relation to Structured Encryption

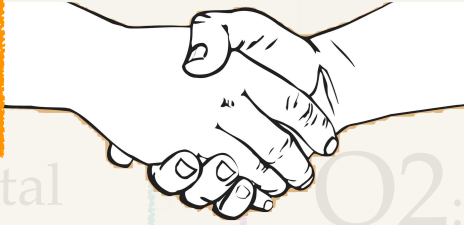


Learns about all the
pairs

Leakage Preview

System architecture

Infer a good approximation of total number of pairs!



Security

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

Maybe

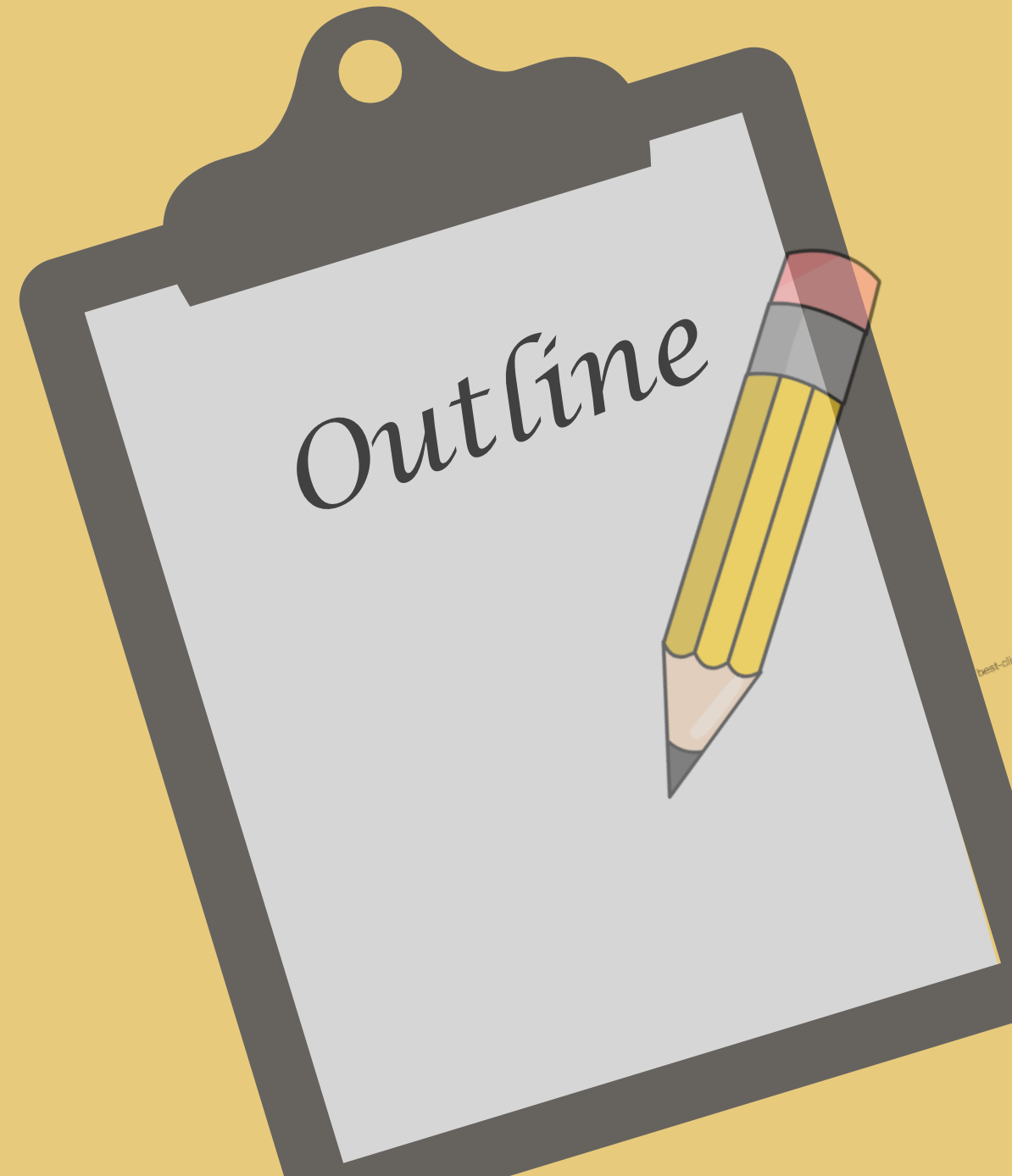
Analyzing leakage in Distributed Systems is tricky!

❖ if DHTs are load balanced

NO

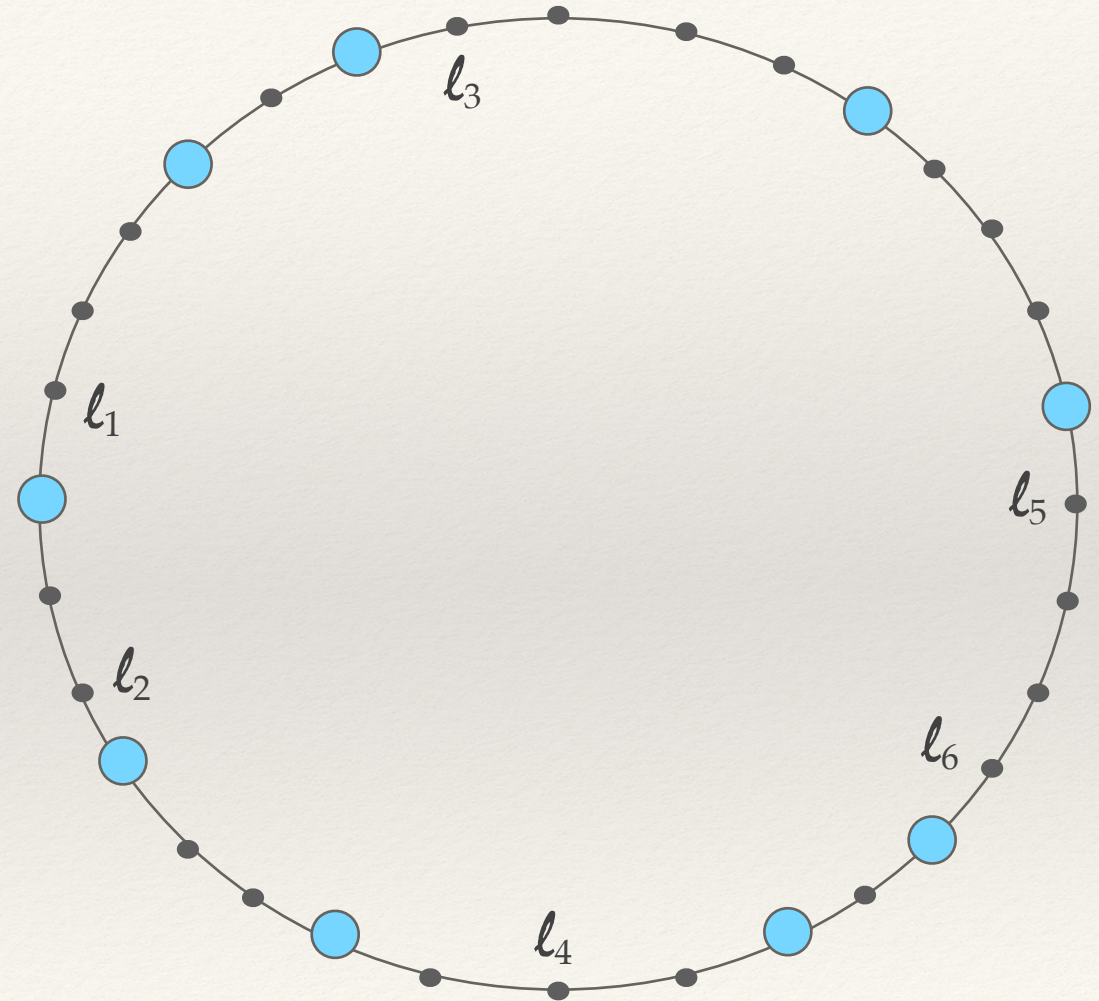
Formalize the use of end-to-end
encryption in DHTs

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



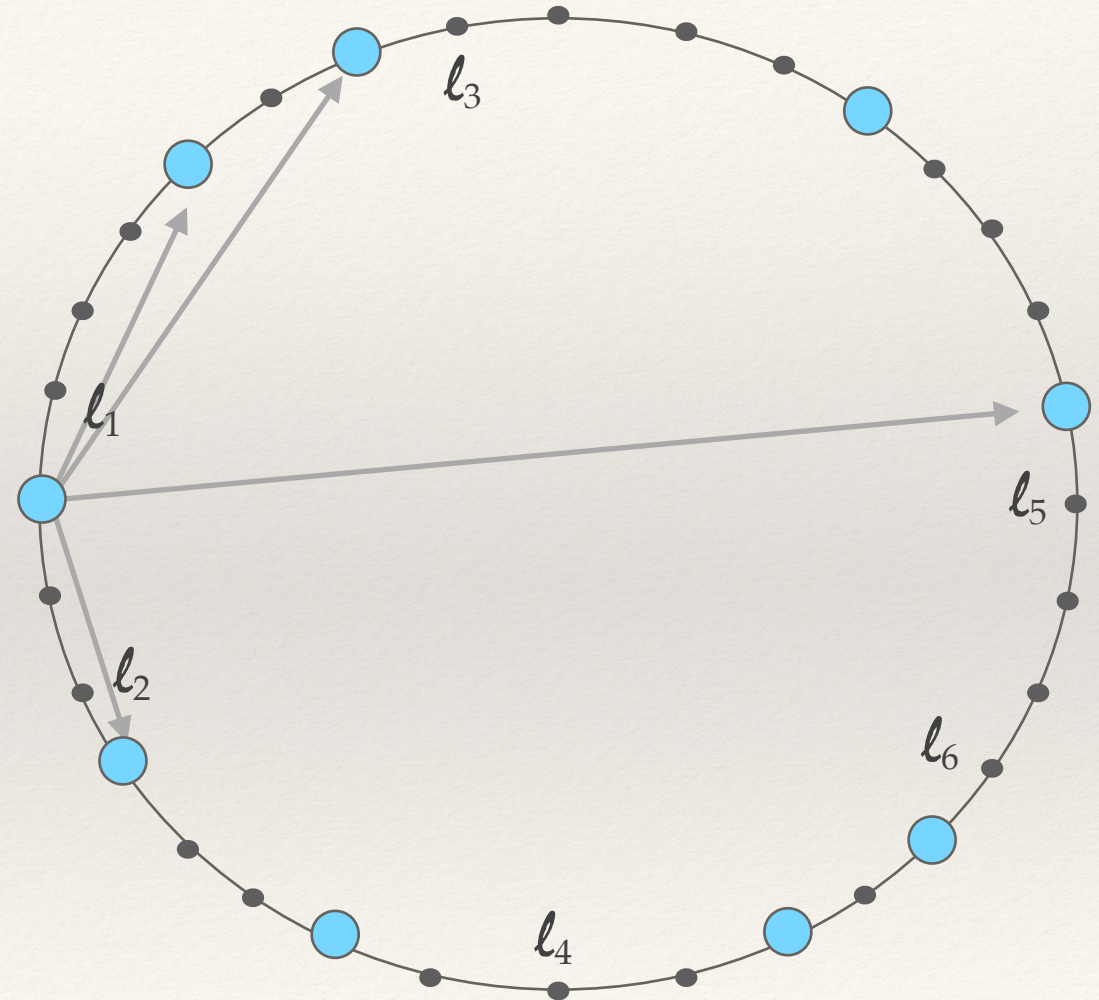
Chord DHT

- ❖ Address Space : A
- ❖ Two hash functions:
 - ❖ H_1 : hashes node ids to addresses
 - ❖ H_2 : hashes labels to addresses
- ❖ $\text{server}(l)$: $\text{successor}(H_2(l))$



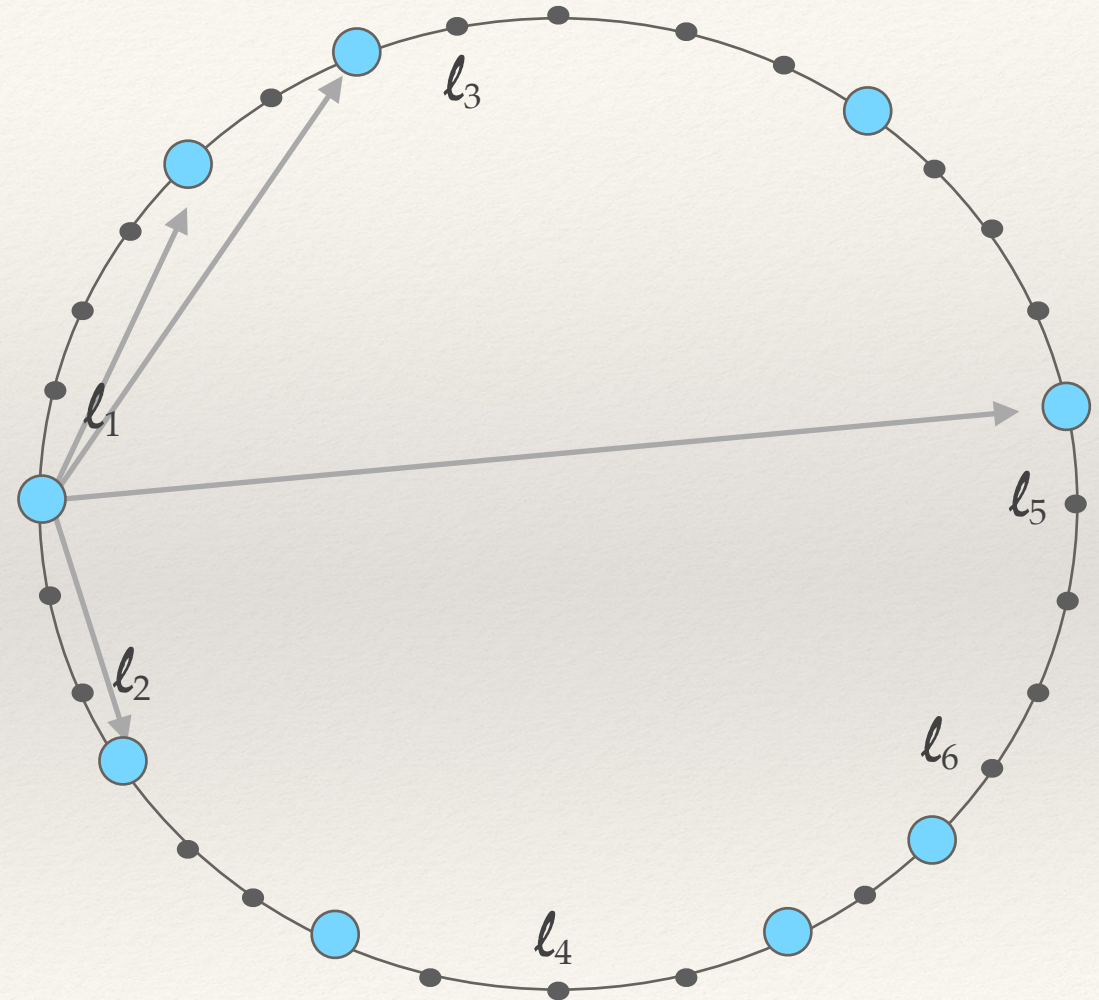
Chord DHT

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



Chord DHT

❖ Address Space : A

❖ Two hash functions:
Overlay param Allocation param

❖ H_1 : hashes node ids to addresses

❖ H_2 : hashes labels to addresses

❖ $\text{server}(l)$: $\text{successor}(H_2(l))$

❖ $\text{route}(a_1, a_2)$:

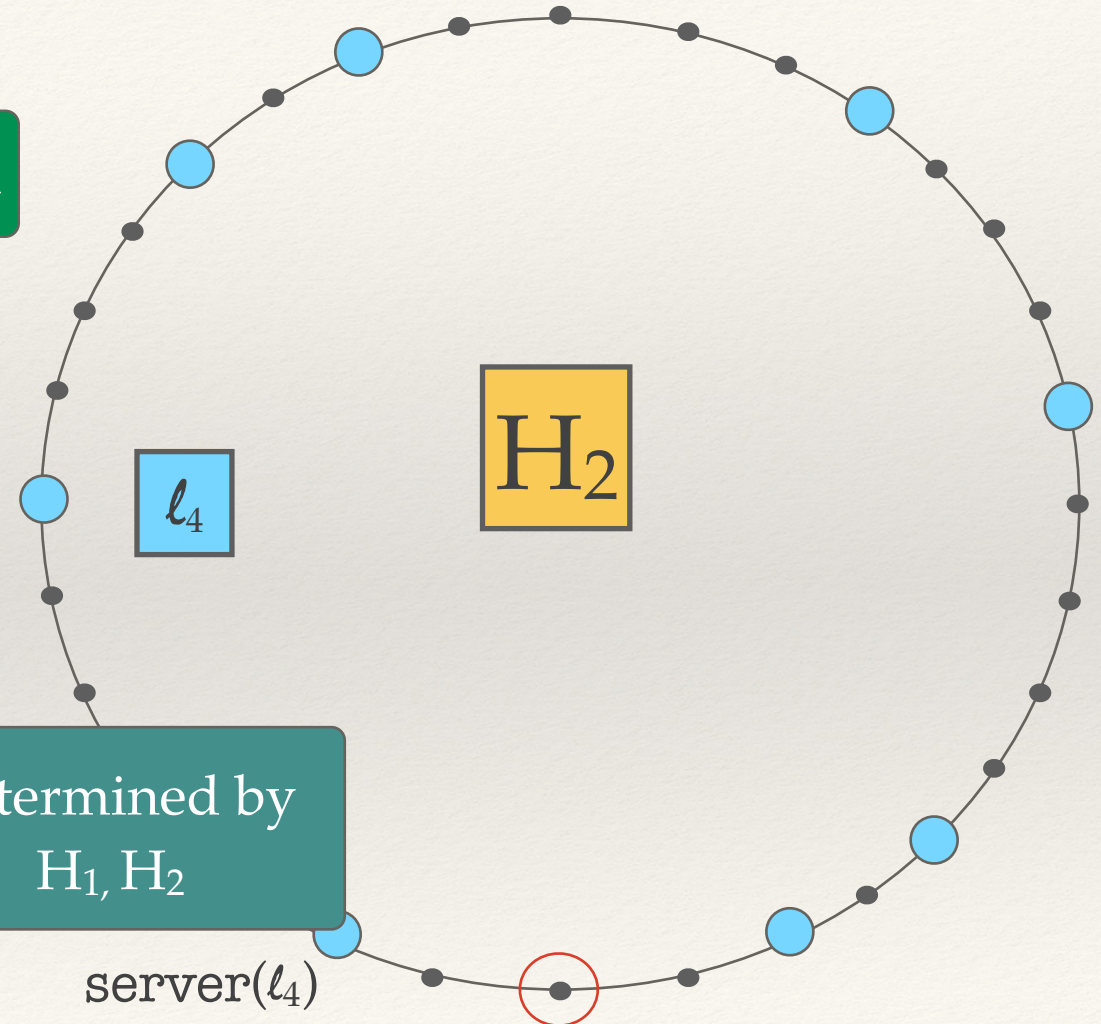
❖ logarithmic routing table

Determined by

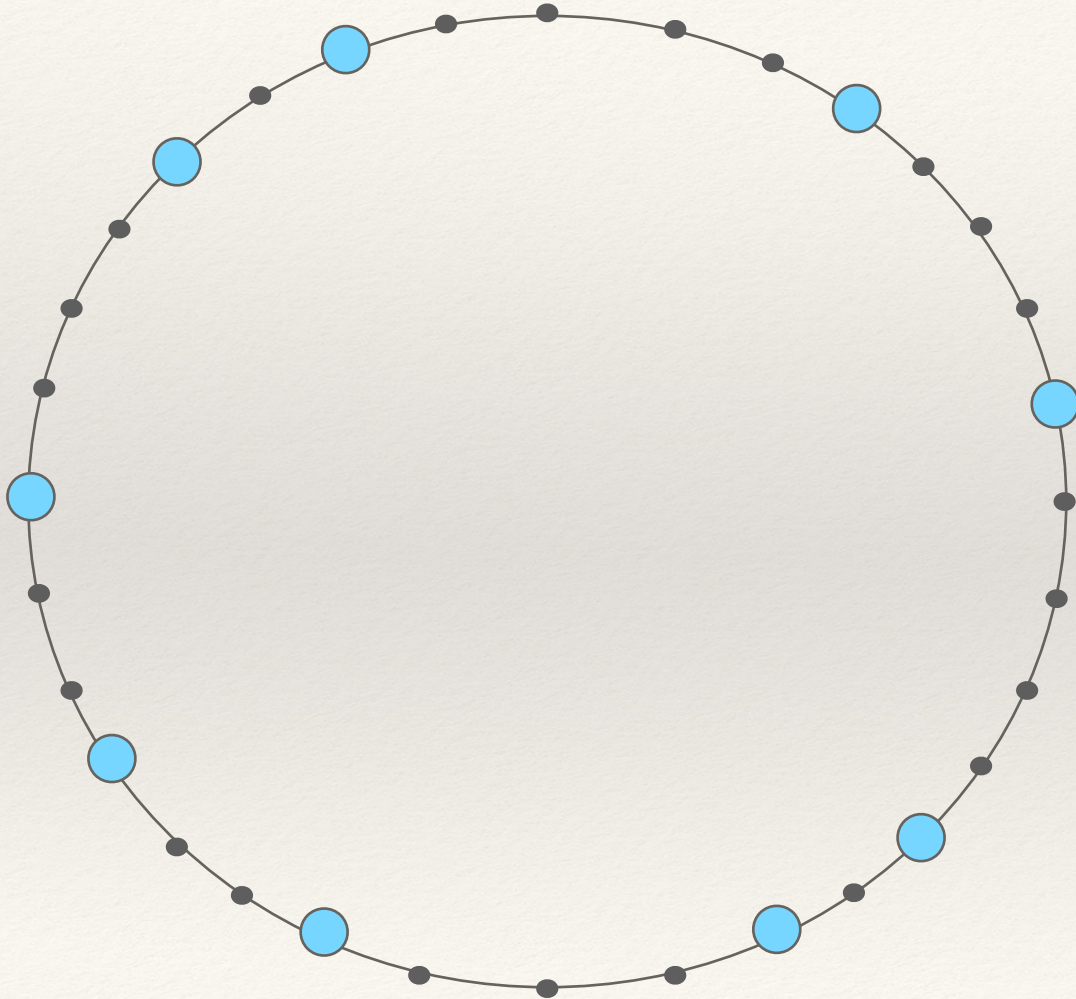
H_1

Determined by
 H_1, H_2

❖ logarithmic routing table

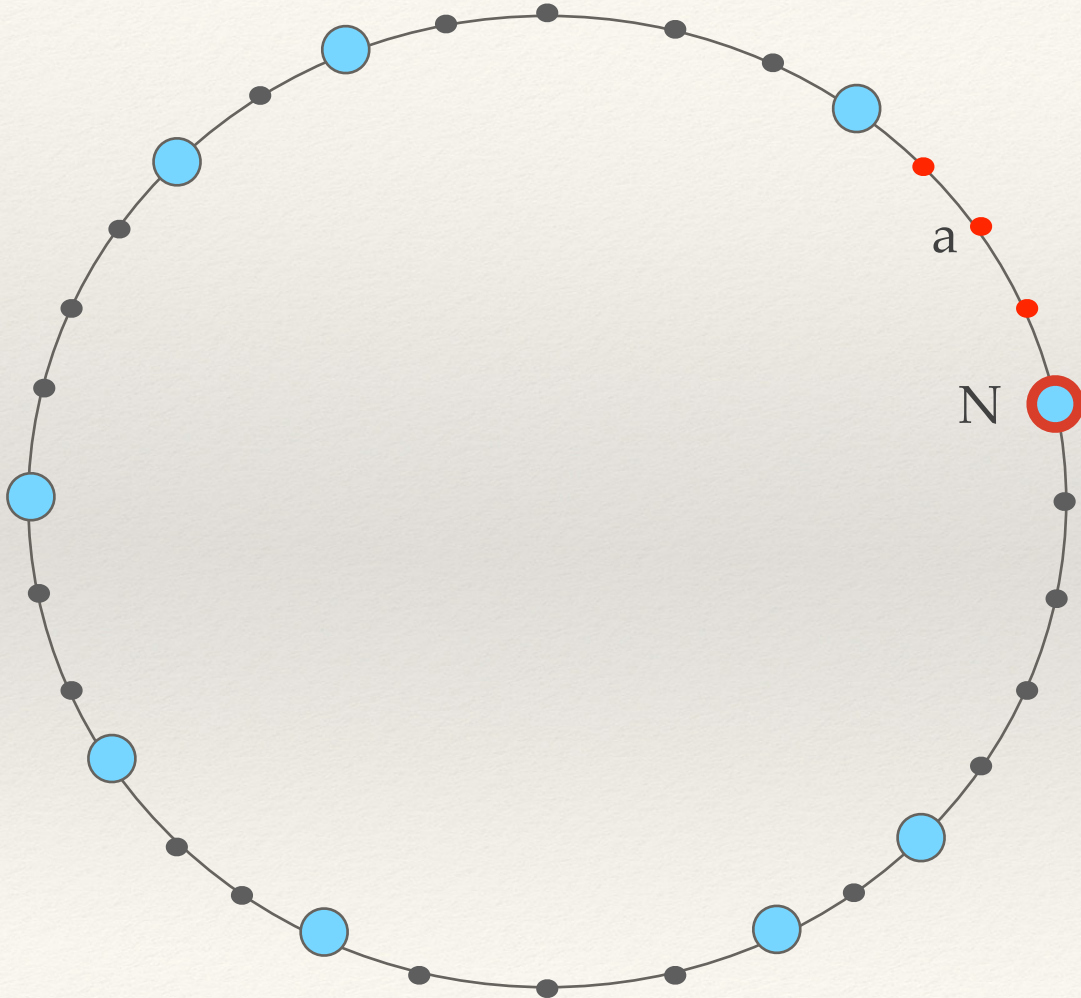


Chord: Visible addresses



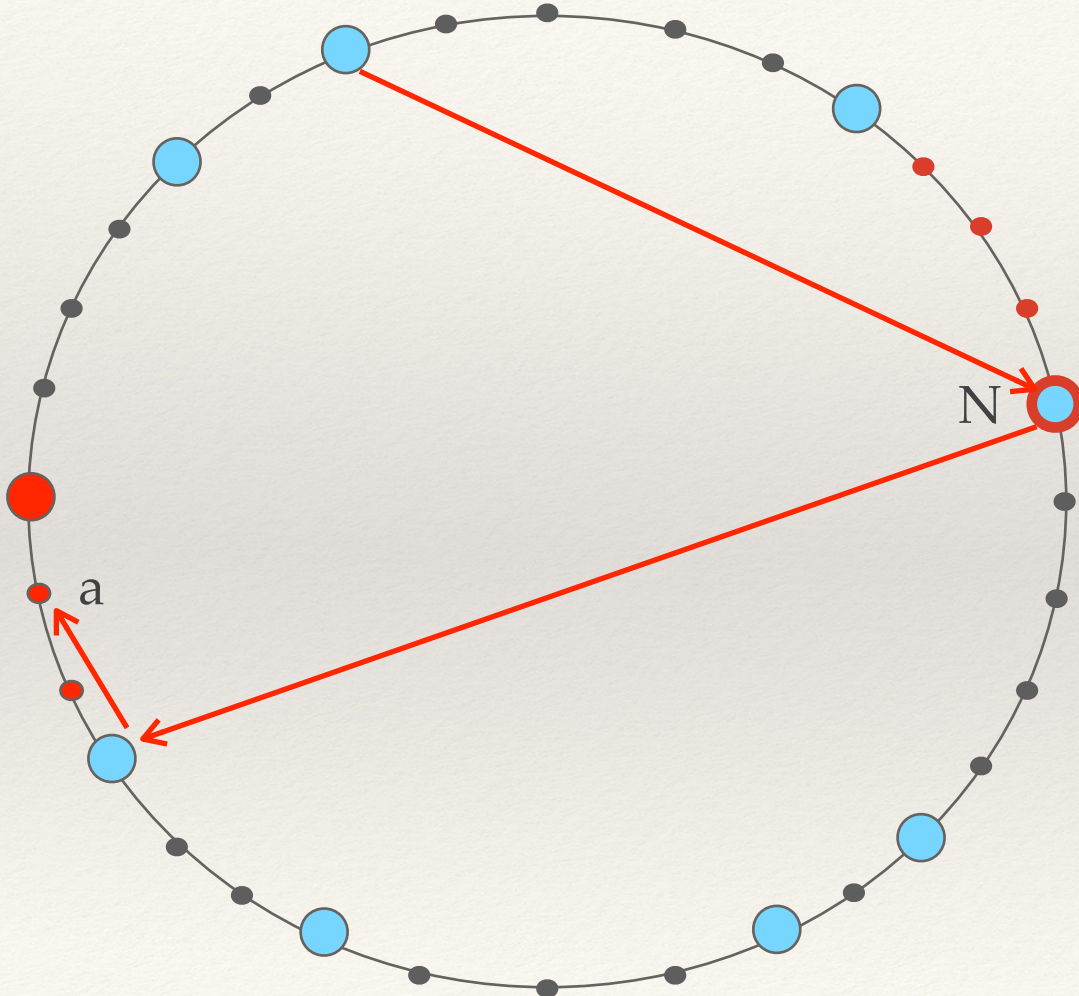
- ❖ $\text{Vis}(N)$: set of all addresses s.t. if $H_2(\ell) = a$ then either
 - ❖ $\text{server}(\ell) = N$
 - ❖ $N \in \text{route}(a)$

Chord: Visible addresses



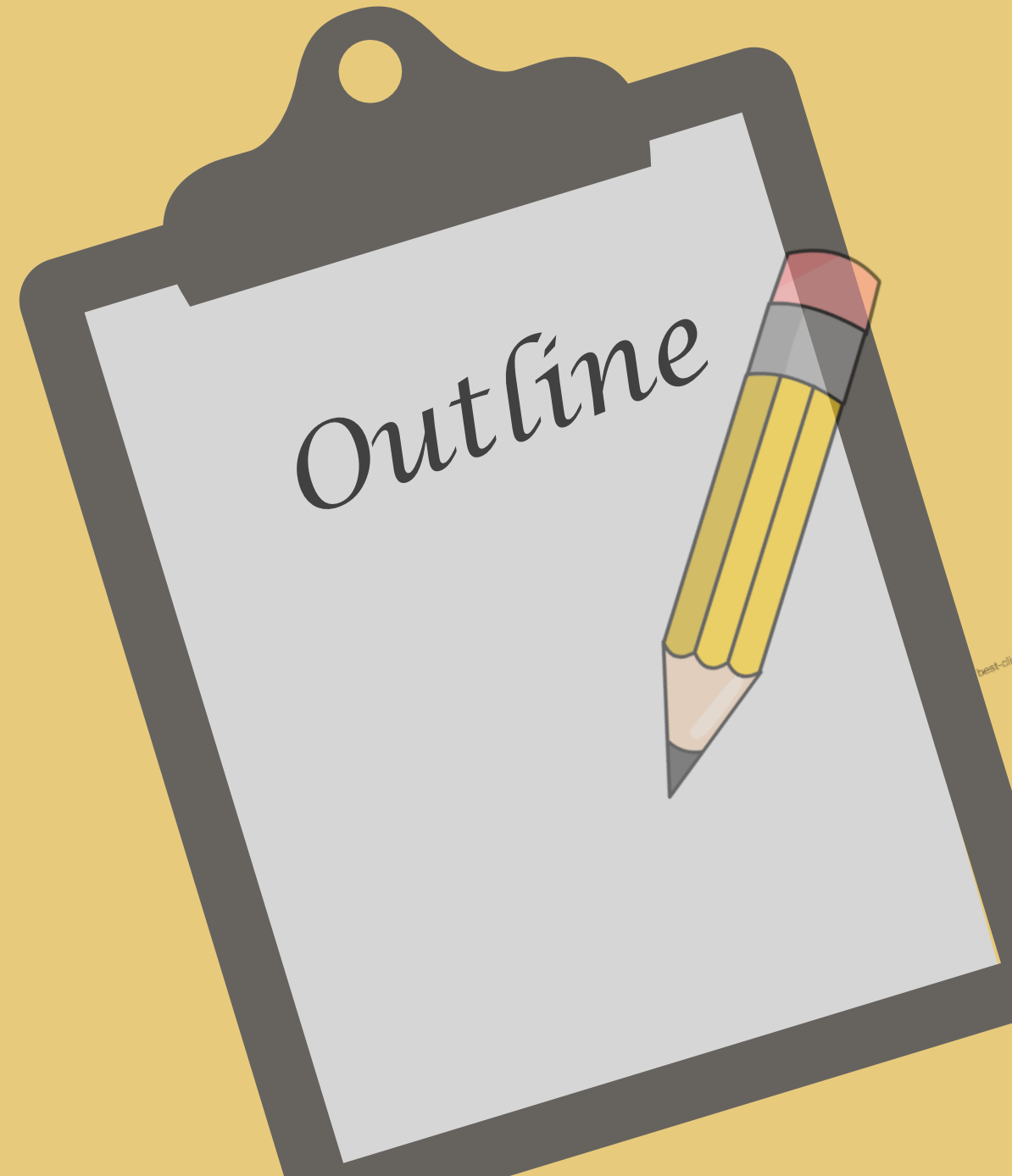
- ❖ $\text{Vis}(N)$: set of all addresses s.t. if $H_2(\ell) = a$ then either
 - ❖ $\text{server}(\ell) = N$
 - ❖ $N \in \text{route}(a)$

Chord: Visible addresses



- ❖ $\text{Vis}(N)$: set of all addresses s.t. if $H_2(\ell) = a$ then either
 - ❖ $\text{server}(\ell) = N$
 - ❖ $N \in \text{route}(a)$

- 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

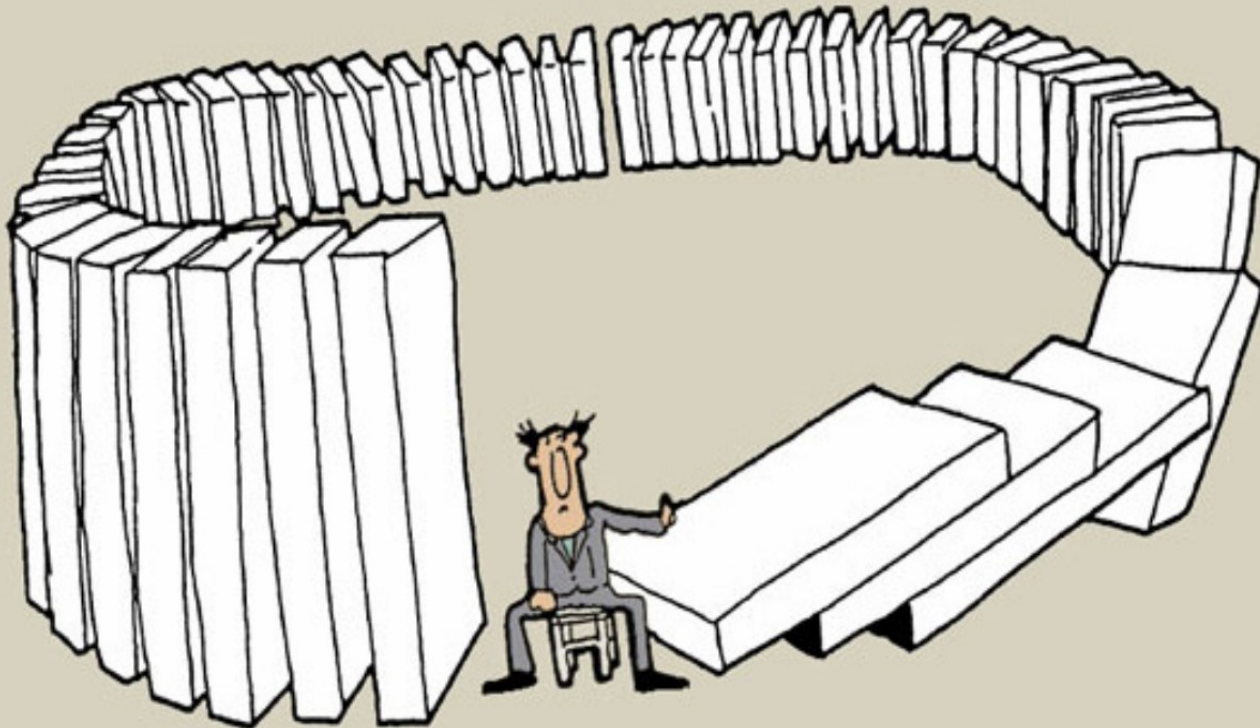
- ❖ $\text{addr}_\omega : \mathbf{N} \rightarrow \mathbf{A}$
- ❖ $\text{server}_{\omega, \psi} : \mathbf{L} \rightarrow \mathbf{A}$
- ❖ $\text{route}_\omega : \mathbf{A} \times \mathbf{A} \rightarrow 2^{\mathbf{A}}$

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

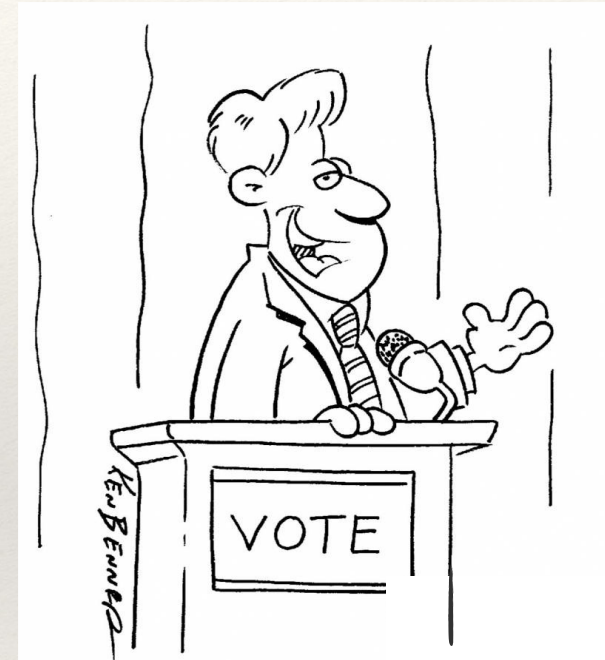
Properties of DHTs

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 \forall labels ℓ , and all nodes N

- ❖ $\Pr[\text{server}(\ell) \in \text{Vis}(N)] \leq \varepsilon$

Prob of a label being visible to a node is bounded

- ❖ Prob over choice of ψ

- ❖ A DHT is (ε, δ) -balanced if

- ❖ $\Pr[\omega \text{ is } \varepsilon\text{-balanced}] \geq 1 - \delta$

w/ prob $1-\delta$ the sampled overlay is balanced

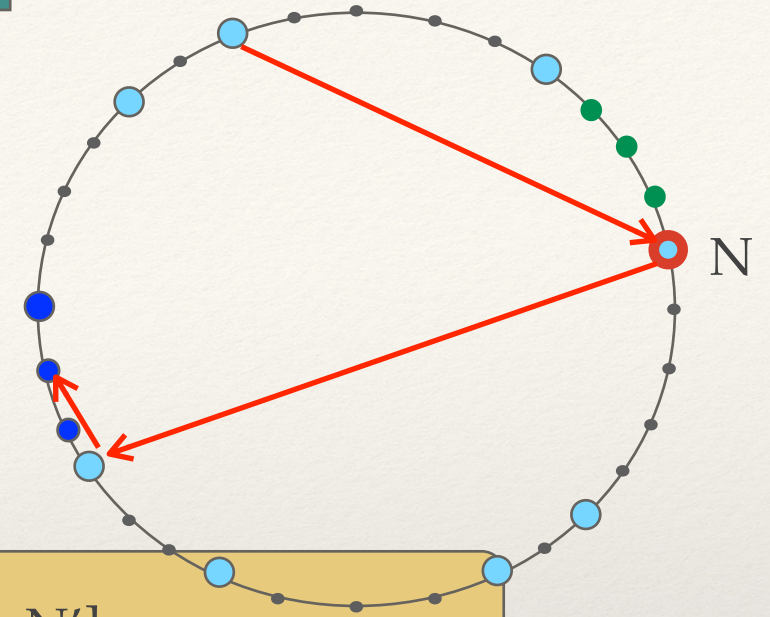
- ❖ Prob over choice of ω

P1: Balance for Chord

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

\propto length of arc of \mathbf{N}

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



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

Balance of Chord

Th :

Chord is (ϵ, δ) -balanced for

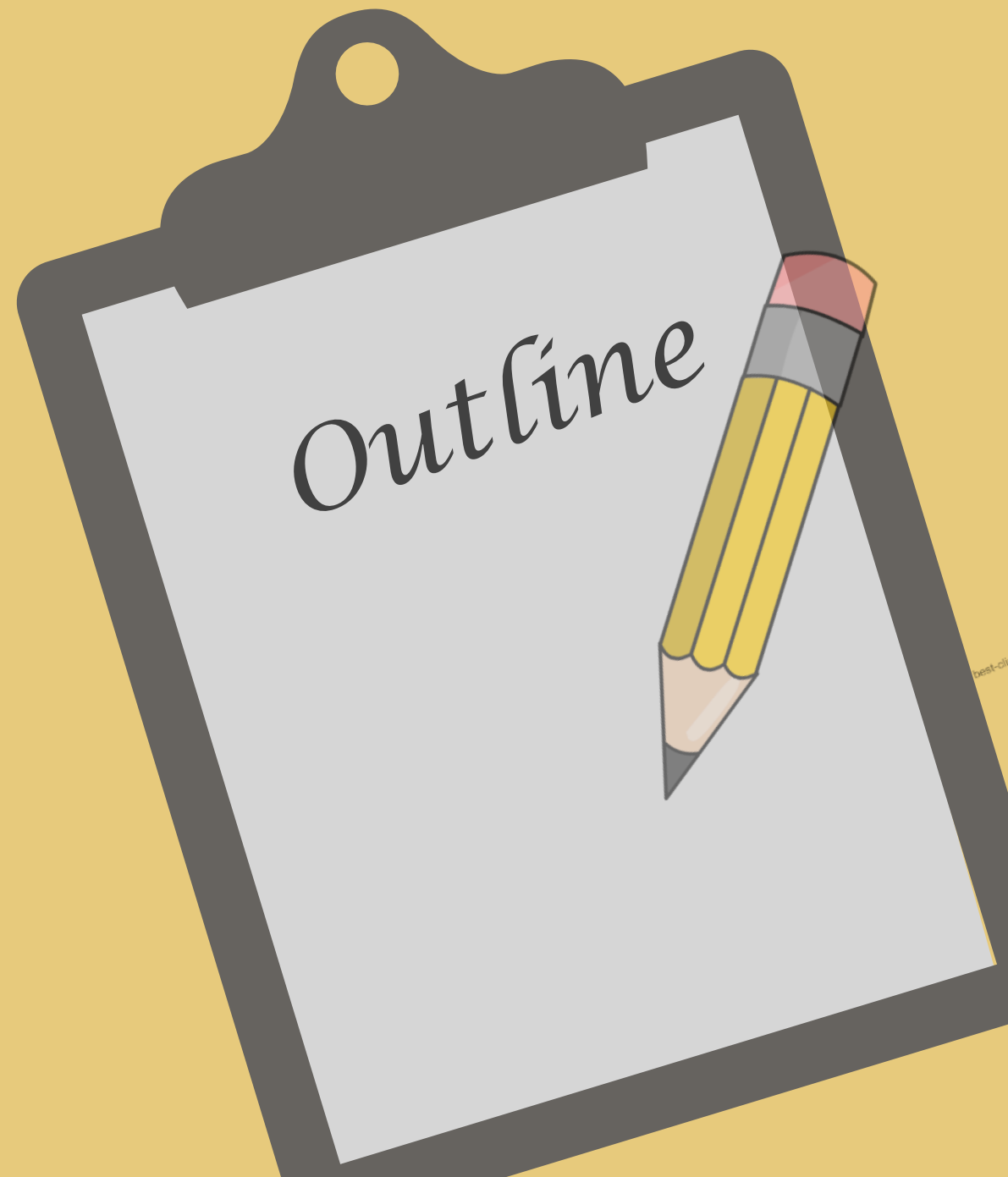
$$\epsilon = \frac{(4 \log n)}{n} + \frac{4n \log^2 n}{|A|} \quad \text{and} \quad \delta = 1/n$$

❖ If $|A| = 2^{512} \Rightarrow n^2 \log n < |A|$, even for $n = 2^{250} \Rightarrow \epsilon = O(\log n / n)$

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)

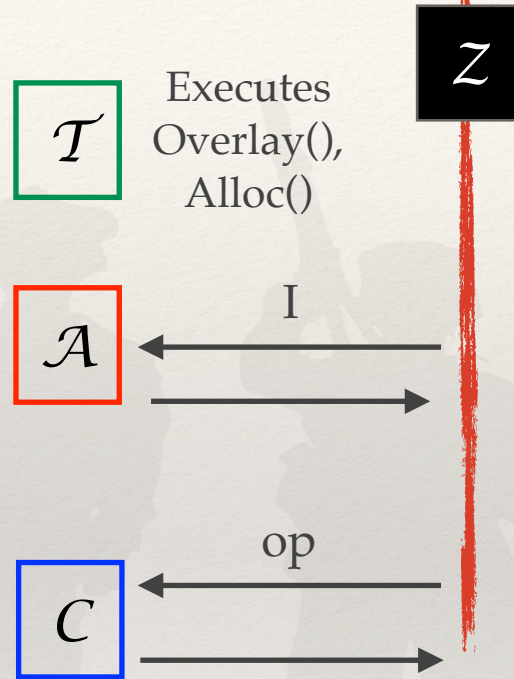
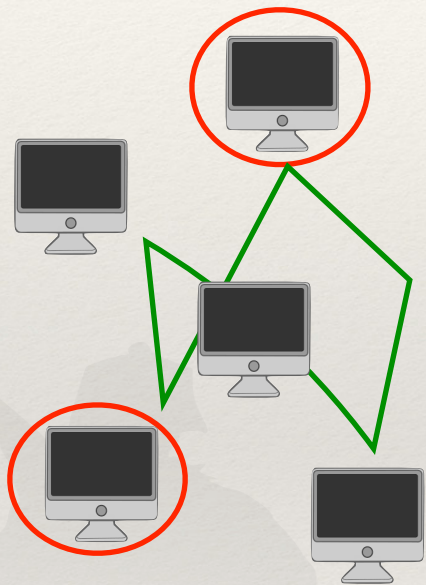
Same as before

- ❖ Executed by Client
- ❖ Generates reqd. keys for client

EDHTs Security

Real

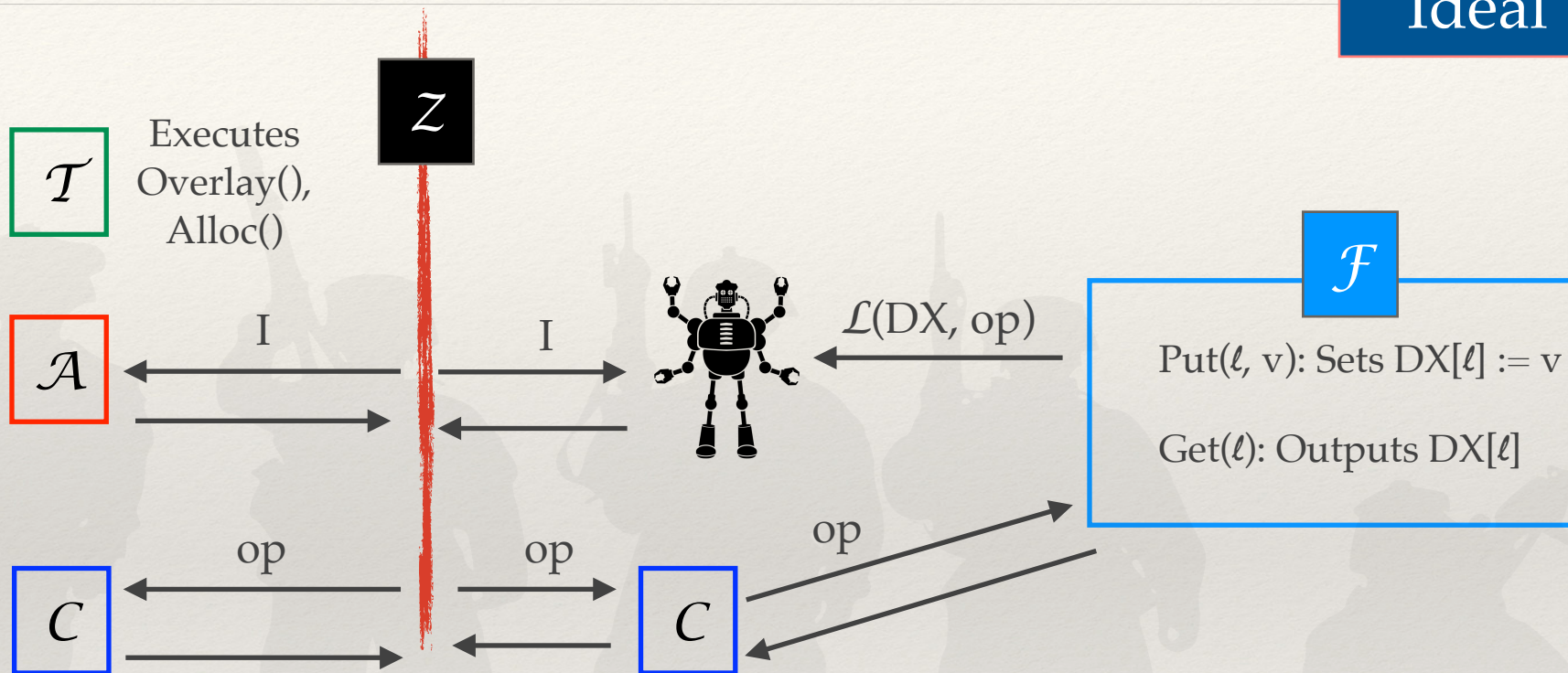
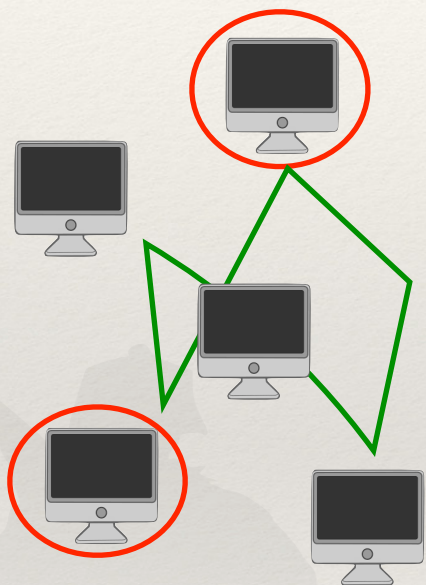
Ideal



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

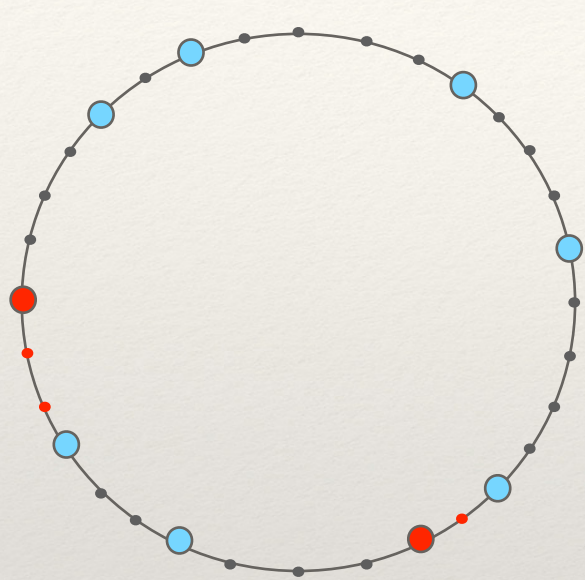
Put(K, ℓ, v)

- ❖ $K = (K_1, K_2)$
- ❖ $t = F_{K_1}(\ell)$
- ❖ $e = \text{SKE.Enc}_{K_2}(v)$
- ❖ $\text{DHT.Put}(t, e)$

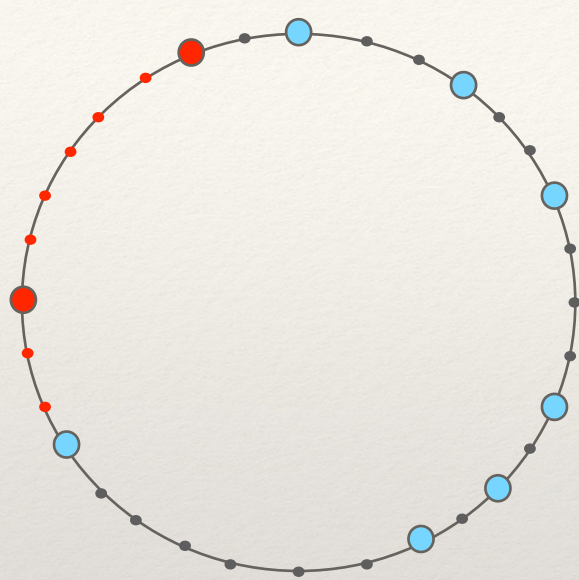
Get(K, ℓ)

- ❖ $K = (K_1, K_2)$
- ❖ $t = F_{K_1}(\ell)$
- ❖ $e \leftarrow \text{DHT.Get}(t)$
- ❖ $v \leftarrow \text{SKE.Dec}_{K_2}(e)$

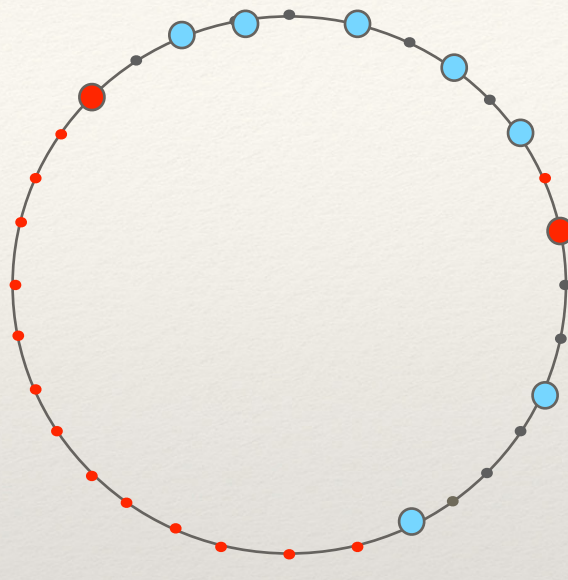
Understanding Leakage



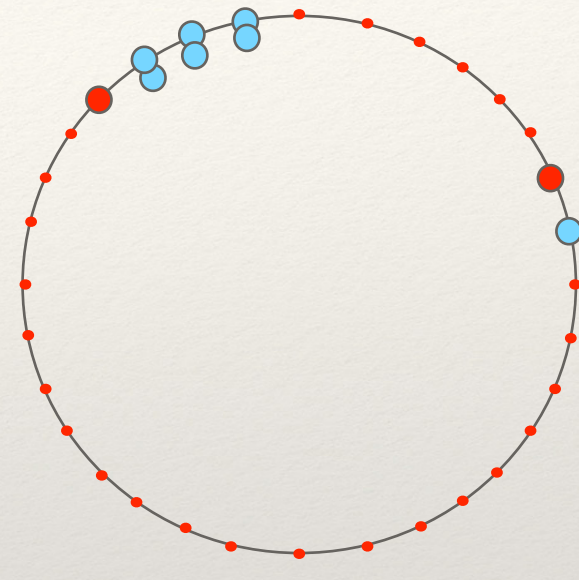
ω_1



ω_2



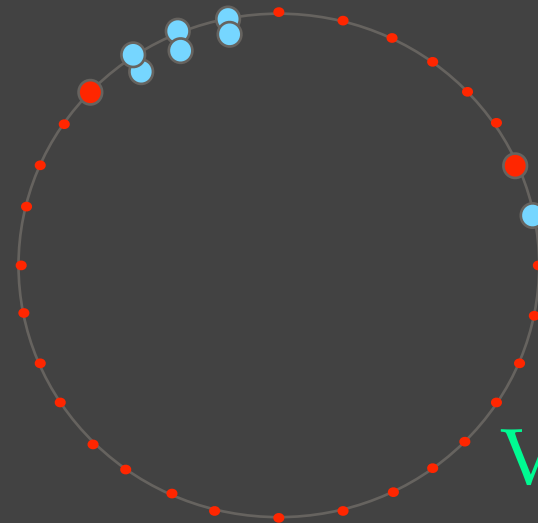
ω_3



ω_4

Q: Is there any gain over STE leakage?

YES



Very unlikely*

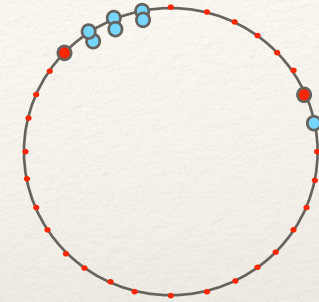
Understanding Leakage



DHT is (ϵ, δ) -balanced if

$$\Pr[\omega \text{ is } \epsilon\text{-balanced}] \geq 1 - \delta$$

$\Pr[\text{label being visible to a node}] \leq \epsilon$



Very unlikely*

Sampling a “bad” overlay is unlikely

\mathcal{L}

leaks $q_{eq}(\ell)$ with probability $\min(1, t \cdot \epsilon)$

• probabilistic

affected by balancing properties of DHT

Standard Scheme: Security

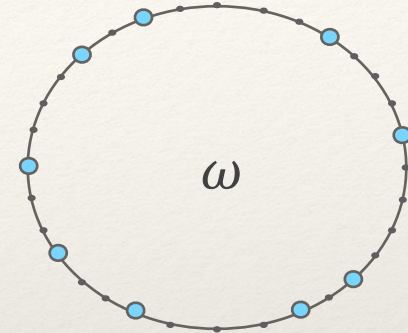
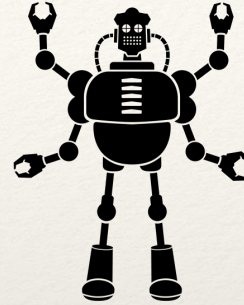
Th :

If DHT is (ϵ, δ) -balanced and has non-committing overlays, then EDHT is \mathcal{L}_ϵ -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

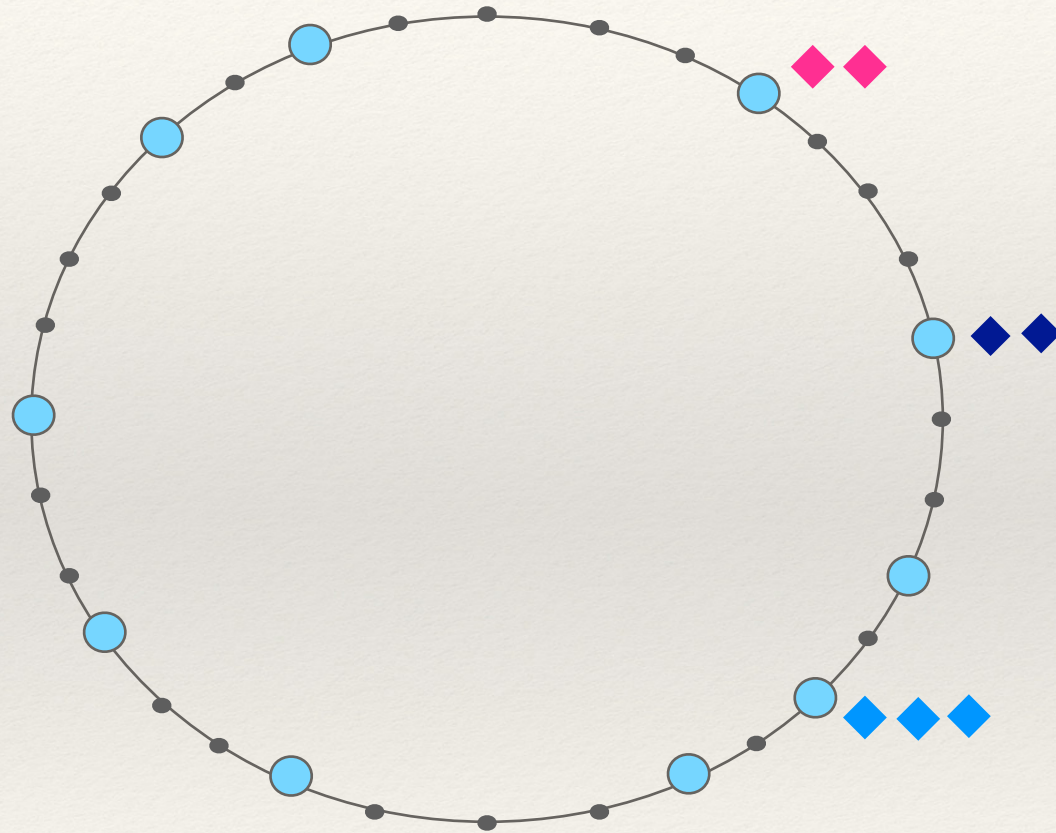
DHT = (Overlay, Alloc, Daemon, Put, Get, **Leave, Join**)

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

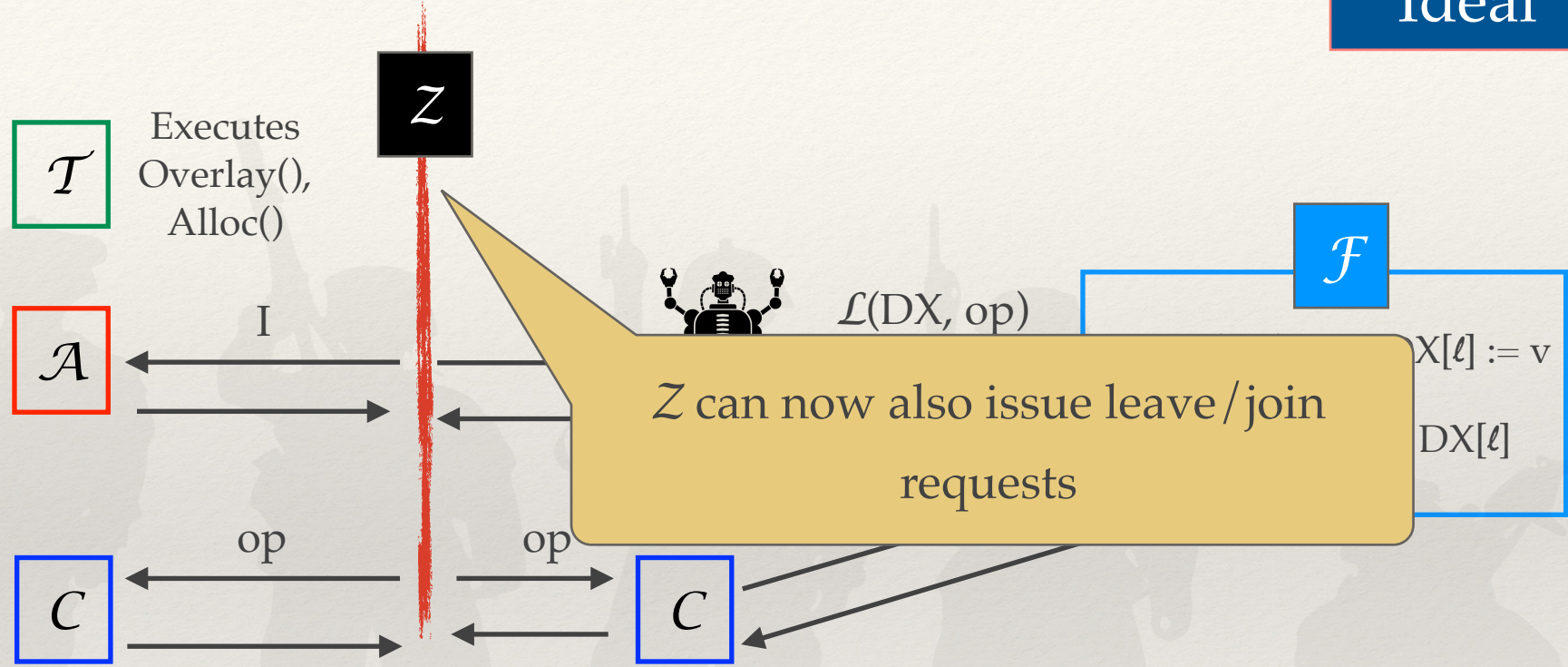
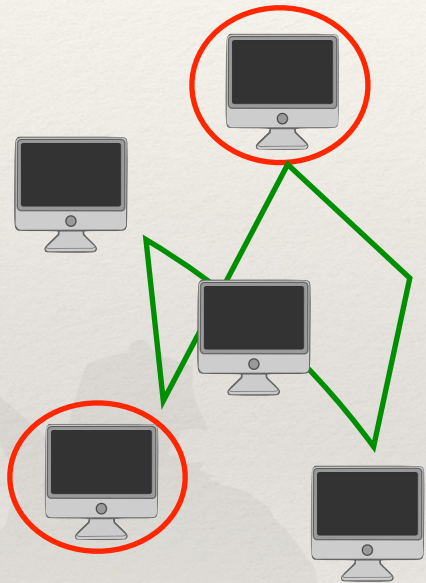
Leave/Join in Chord



Security : Transient EDHTs

Real

Ideal



Real

\approx

Ideal

Properties of DHTs

P1: Balance

Stronger notion

- ❖ A DHT is (ϵ, δ) -balanced if for all active nodes \mathbf{C}
 - ❖ $\Pr[\wedge(\omega, \mathbf{C}) \text{ is } \epsilon\text{-balanced}] \geq 1 - \delta$

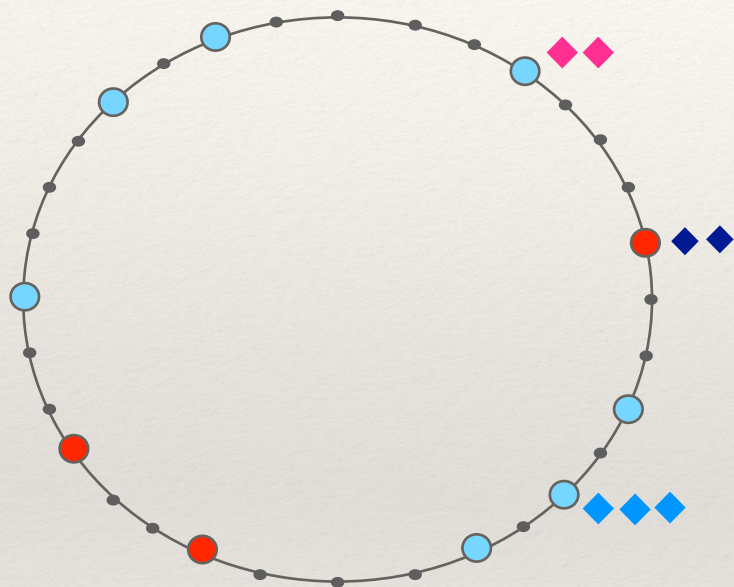
w/ prob $1-\delta$ the sampled overlay is balanced for all nodes \mathbf{C}

P2: Non-committing allocations

Same as before

Ken Benneke
"And if elected, I promise to keep making promises."

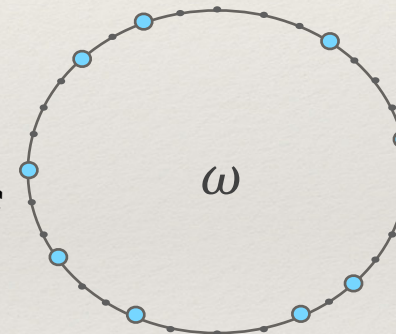
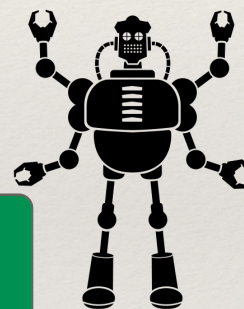
Understanding Leakage



- ❖ Additional pairs become visible during leave/join

\mathcal{L}

which pairs to leak??



leaks $qeq(\ell)$ 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}_ϵ -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**





- ❖ 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



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

nk you

