

# *Structured Overlays:*



*Attacks, Defenses, and all things Proximity*

*April 27th, 2006*

*Wyman Park 4<sup>th</sup> Floor Conference Room*

*Presentation by:*

**Jay Zarfoss**

JOHNS HOPKINS  
UNIVERSITY

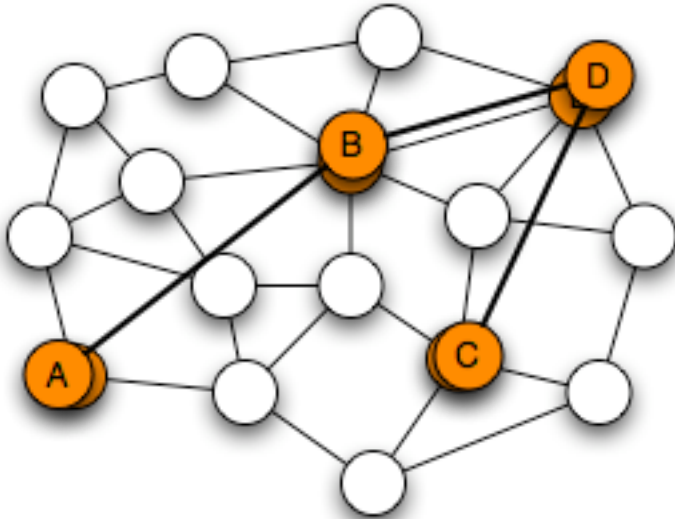
Information Security  
Institute 

# Our roadmap



- General overview of Overlays / DHTs
  - Chord [13]
  - Pastry [7]
- Location, Location, Location
- General Attacks and Defenses
- Eclipse Attacks: Churn as Shelter [4]
- Targeted Attacks: LocationGuard [11]

# What is an Overlay Network?



Logical Network that sits  
“on top of” another network.

Can be *structured*, or  
*unstructured*.

Used for P2P systems,  
multicast broadcasts, etc.

# Distributed Hash Table (DHT)

- Decentralized network where each node takes responsibility for a certain portion of a *keyspace*. Example:  $[0, 2^{160}-1]$ .
- Given a *key*, any member of the DHT should be able to efficiently lookup whatever node is responsible for that key.
- For our purposes, we can think of a DHT as being an overlay network on top of the Internet, with Overlay **NodeID = hash(IP address)**

# The “Big 4” DHTs



- **Chord** [13] (MIT)
  - **Pastry** [7] (Microsoft)
  - **CAN** [9] (UC Berkeley)
  - **Tapestry** [14] (UC Santa Barbara)
- 
- All released roughly the same time ~ 2001
  - Numerous others variants since then...

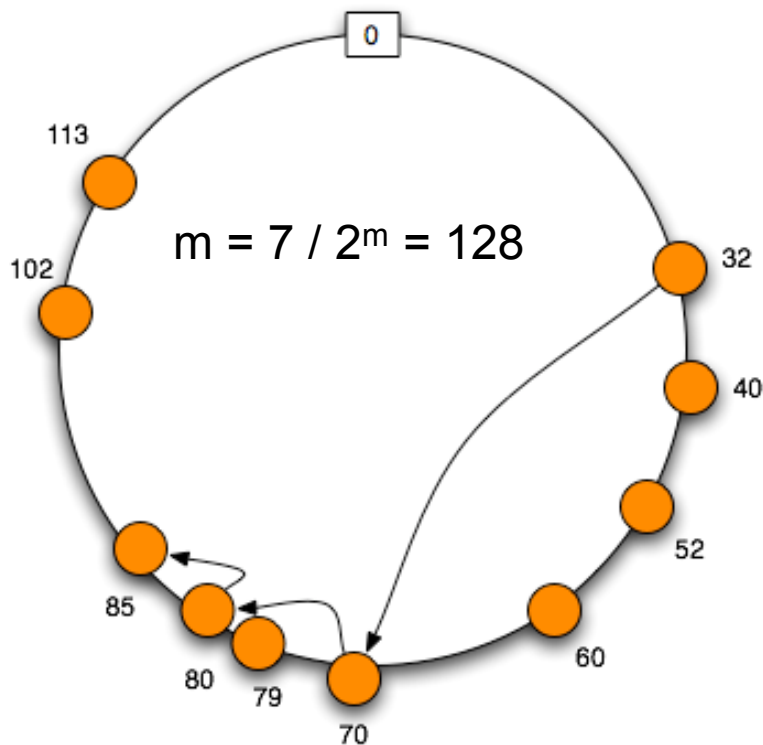
# Chord



- Use a hash function to map each node and key into an  $m$ -bit identifier circle, modulo  $2^m$ .
- Key  $k$  is assigned to the first node whose identifier is equal to or greater than  $k$ .
- Nodes keep track of their successors, predecessors along the ring, in addition to  $\log(n)$  other nodes on the ring.

# Chord Routing Example

Node 32 looks up key 82



Node 32 Finger Table		
Start	Interval	Node
33	33-33	40
34	34-35	40
36	36-39	40
40	40-47	40
48	48-63	52
64	64-95	70
96	96-31	102
Successor		40
Predecessor		113

# Chord Overview



- Very nice, easy-to-analyze properties:
  - $O(\log(n))$  **overlay** hops to perform lookup
  - $O(\log(n))$  sized routing tables
  - $O(\log^2(n))$  steps to join a network
- Extremely reliable in event of node **failure**
  - Can record multiple predecessors, successors
  - Handles concurrent joins/leaves well
- No fudging with extra parameters!

# Pastry



- Also uses a ring structure
- Performs lookups with:
  - Routing Table (Chord's Finger Table)
  - Leaf Set (Chord's Successors/Predecessors)
  - Neighborhood Set (More on this later)
- Keys thought of as sequence of digits with base  $2^b$ . Route lookups to “numerically closest” node, rather than successor node.

# Pastry Routing, Single Hop

b = 2 Example

Base =  $2^b = 2^2 = 4$

11032111

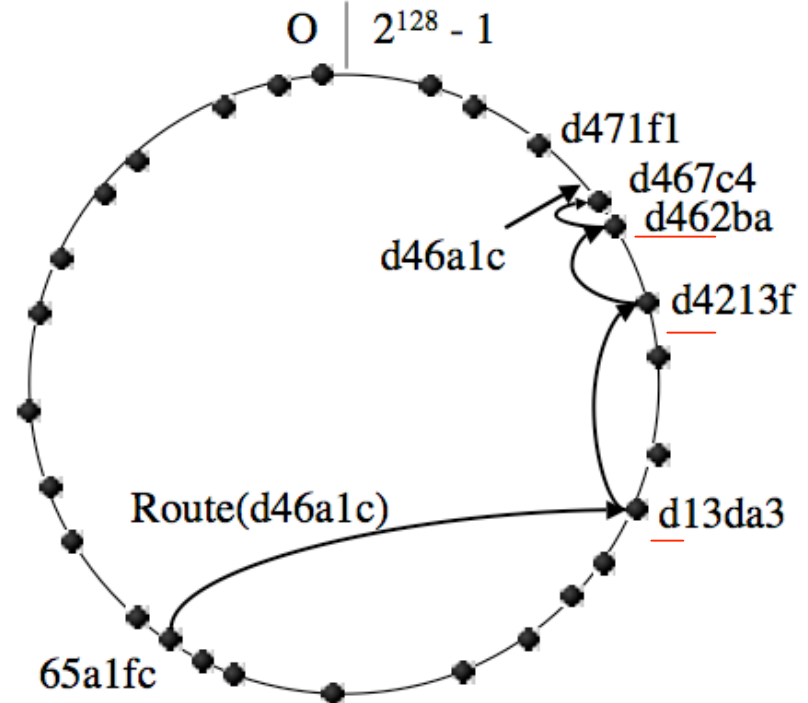
31123001

10233033

Nodeid 10233102			
<b>Leaf set</b>	SMALLER	LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232
<b>Routing table</b>			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

# Pastry Routing, Total Path

- Different view with  $b = 4$
- Lookup key  $d46a1c$  from node  $65a1fc$
- At each step, matching prefix gets larger



# Pastry Overview

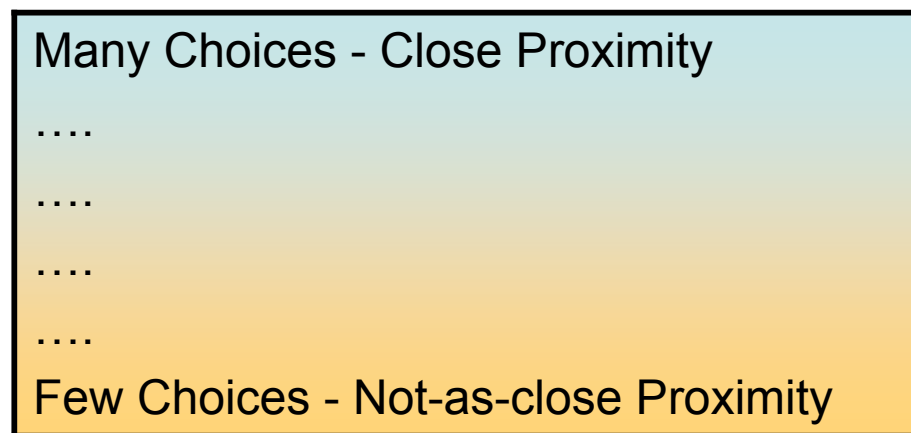


- Configuration parameters
- Slightly harder analysis (still reasonable)
  - $O(\log_{2^b}(n))$  overlay hops
  - $\log_{2^b}(n)(2^b-1)$  sized routing tables
  - $2^{b+1}(\log_{2^b}(n))$  messages for a proper join
- Routing tables are **proximity-optimized**
  - Potentially faster lookups in practice
  - Standard Chord makes no such optimizations

# Proximity Neighbor Selection

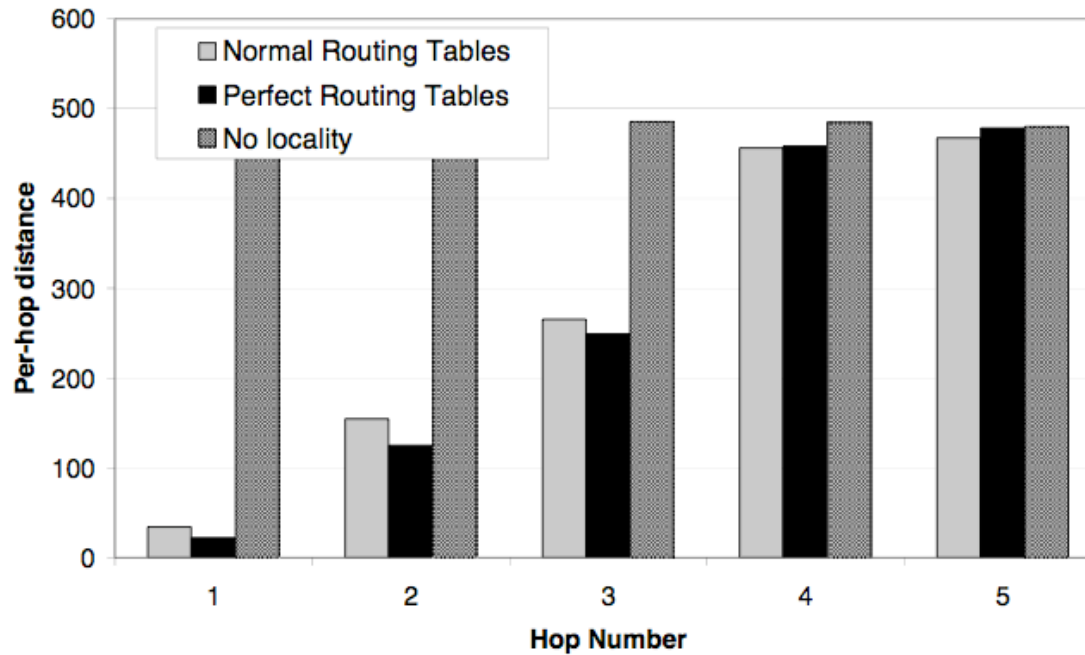
- Many choices for upper entries of the routing tables, which node do we pick?
  - Pick the one closest to us **in the network**
  - Use proximity metric: networks hops / latency

Routing  
Table



# Proximity Affects Hops [3]

Many nodes to choose from for the initial hops, so we can probably get very close neighbors.



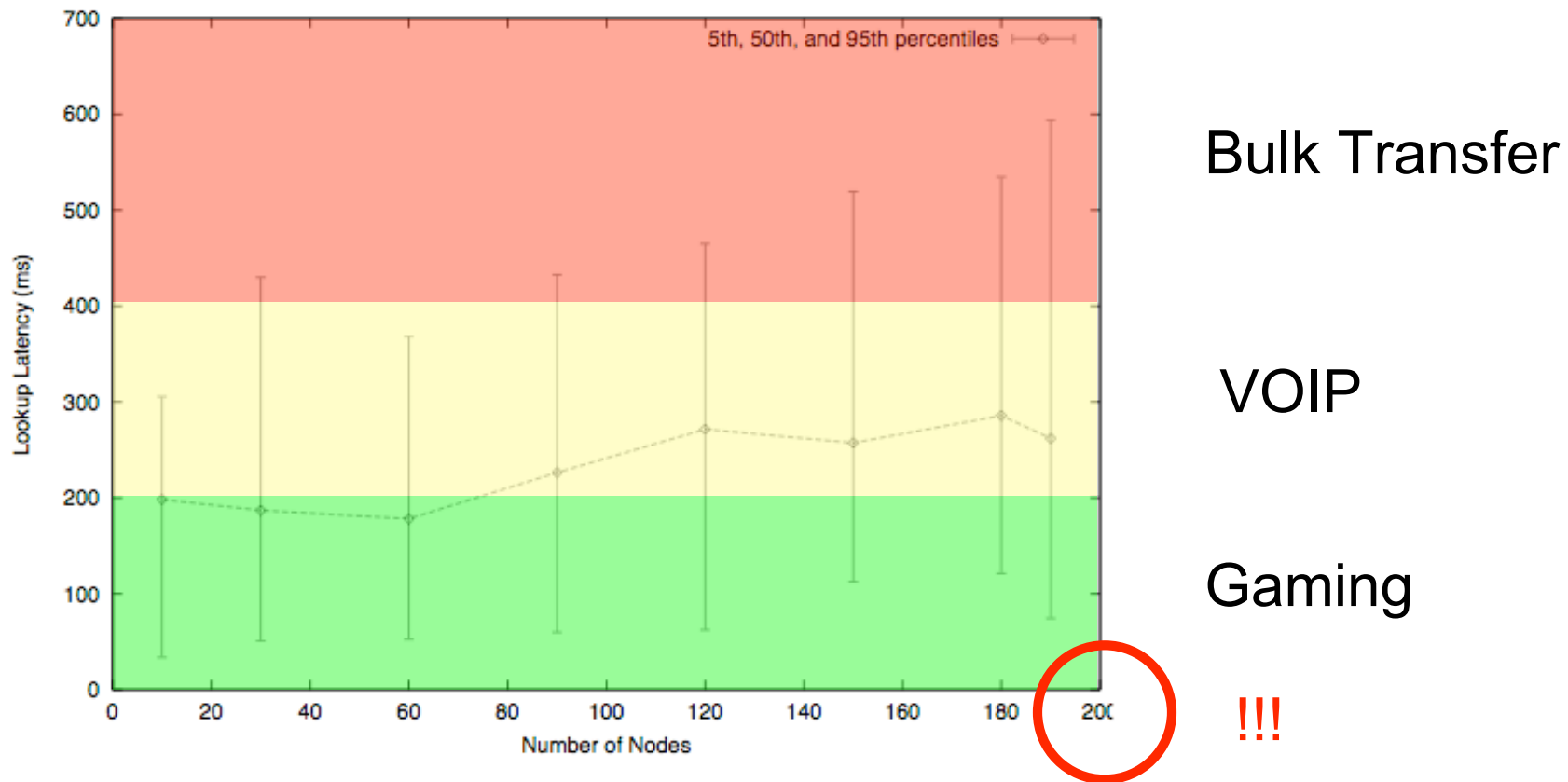
# No Proximity for Chord?



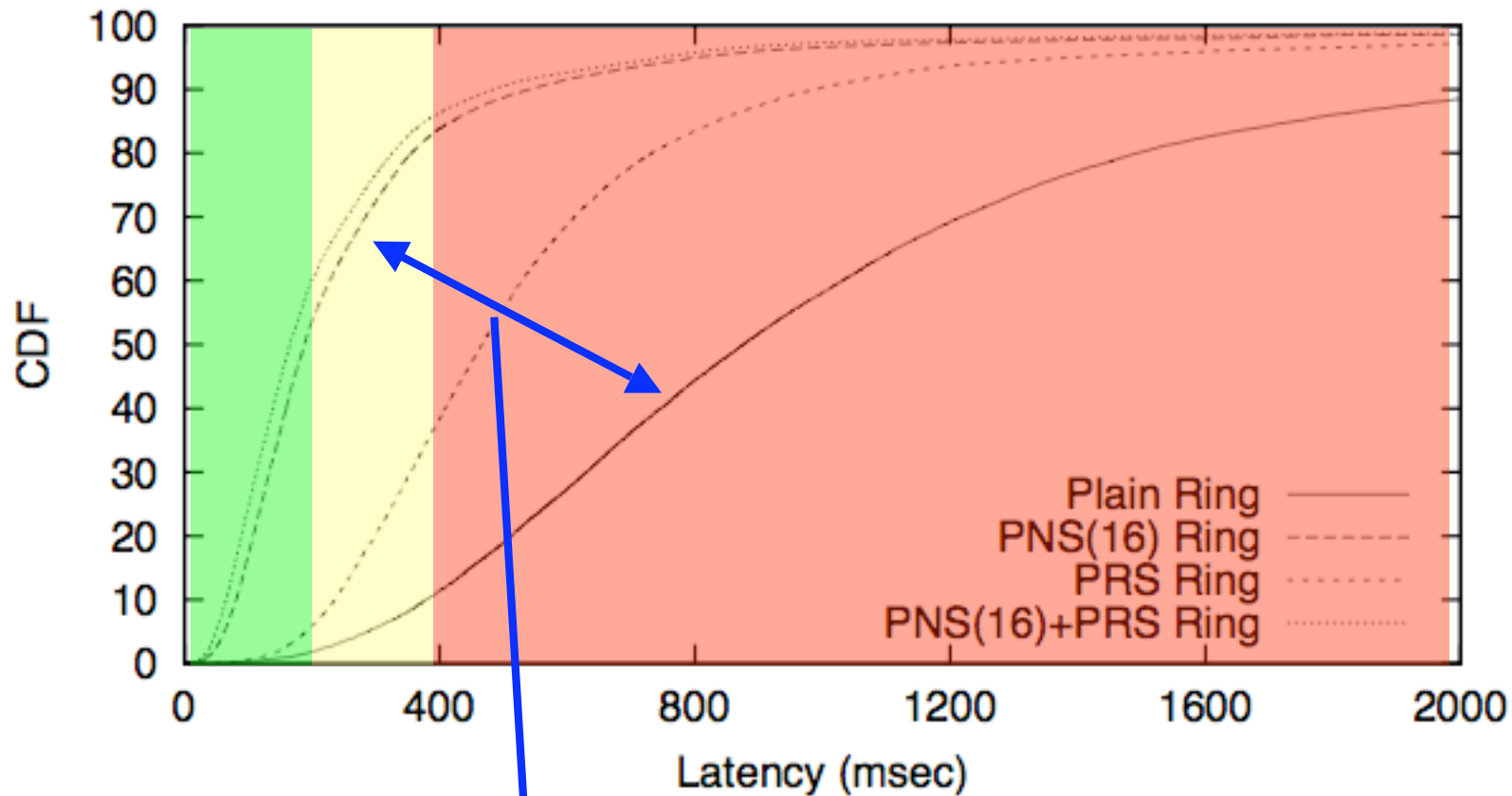
- Chord uses a **constrained** table
  - No wiggle room to proximity optimize table entries without violating the rules
  - Can we still use proximity even if we're stuck with a constrained table?
- Proximity Route Selection
  - At route-time, compromise some progress in the overlay lookup if shorter network trip

# Is constrained “good enough”?

Latency data from [13], no proximity consideration



# How about a 2nd opinion? [6]



16k nodes

Using a proximity-optimized table has  
**DRASTIC** effect on lookup time!

# The Adversarial Model



- Assume the network layer is secure.
- Freeloader Model
  - Not coordinated with other adversaries
  - Simply drops routing requests
  - Can handle adversarial nodes as failed nodes
- How many freeloaders before our lookups begin to fail?

# Try One Lookup...



Fraction of lazy nodes

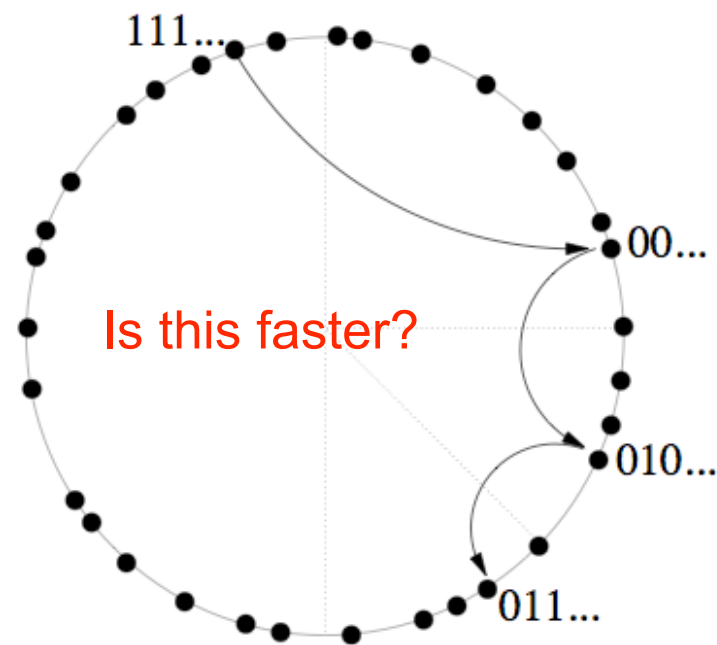
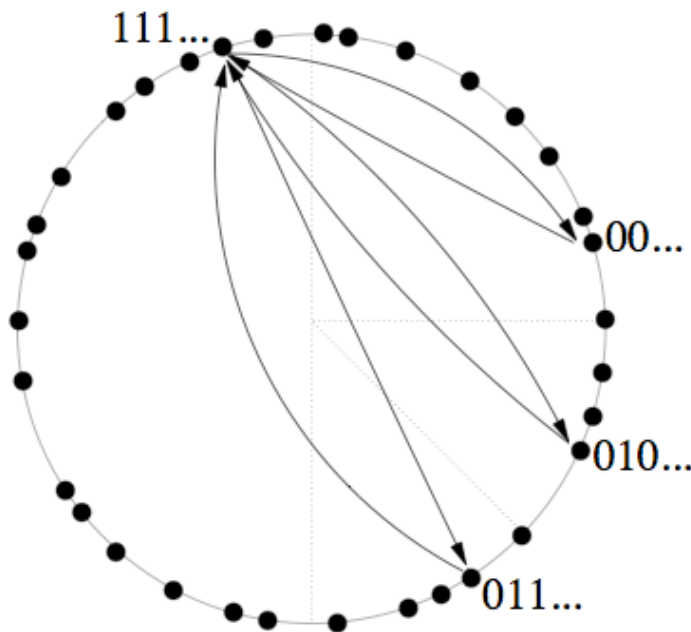
$$\Pr(\textit{success}) = (1 - f)^M$$

Expected number of overlay hops

$$\Pr(\textit{failure}) = 1 - (1 - f)^M$$

# Discover a lookup failure

- How long until we realize lookup failed?
  - Depends - Iterative or Recursive Routing?



# Try, try again

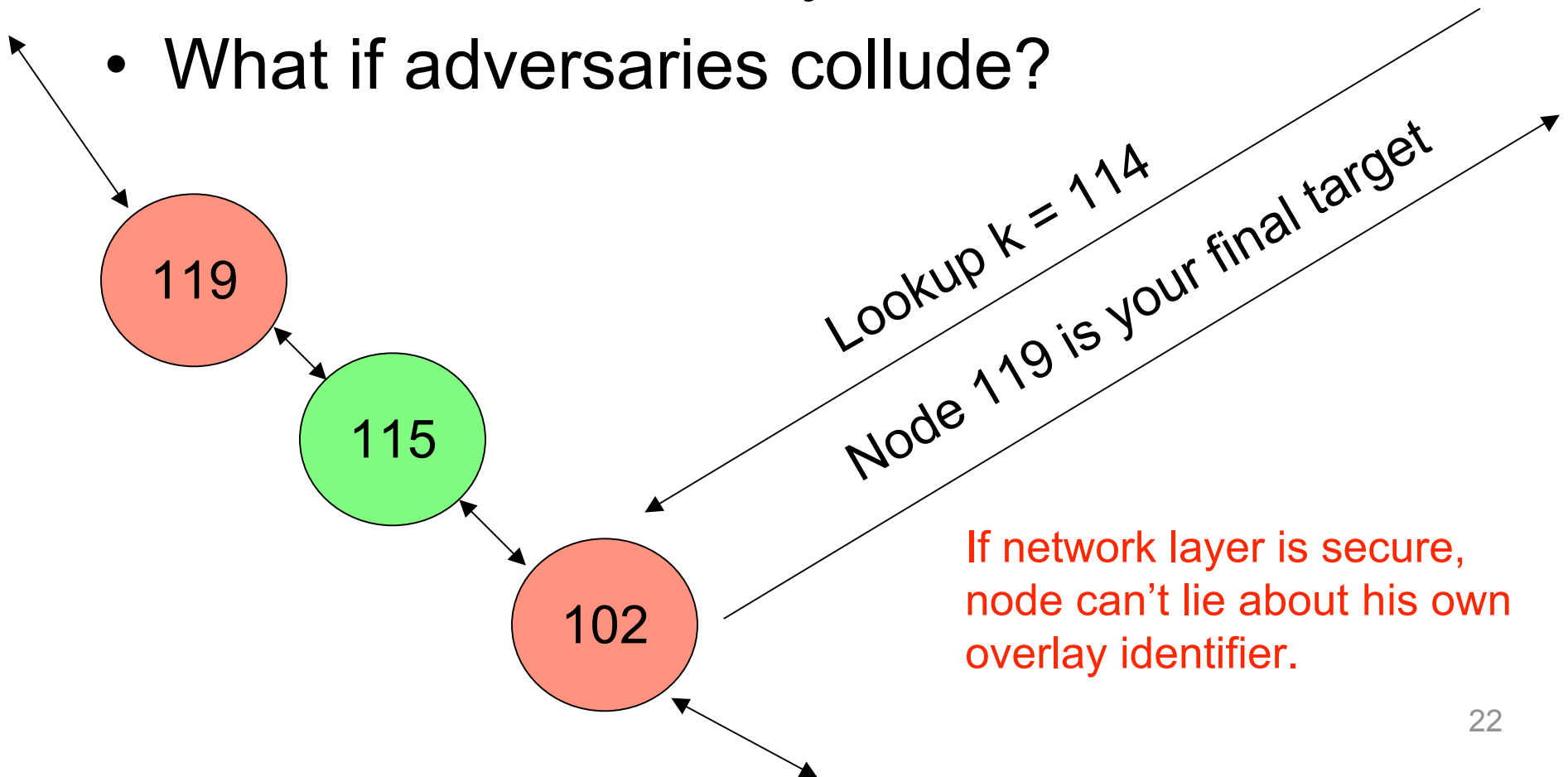
- If a single lookup fails, hand to a neighbor and let him try: *redundant routing*.
- Final success rests on overlay structure and resulting *independent paths* to target

$$\Pr(\textit{failure}) \leq (1 - (1 - f)^M)^I$$

Assumes we don't care about latency!!!

# Stronger Adversary

- What if our adversary can lie?
- What if adversaries collude?



# How to detect lying

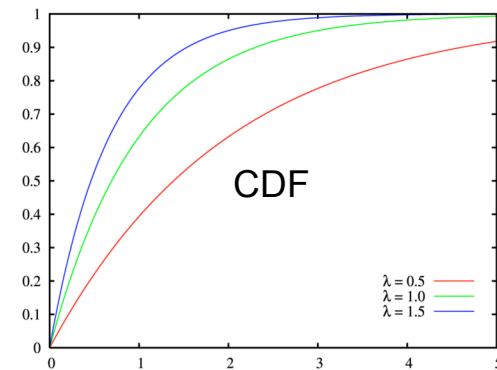


- At intermediate hop, next hop always needs to get closer the key
- At final hop, the final node ID should be *reasonably* close to the lookup key.
  - Assuming uniform hash, distance between nodes follows an **exponential distribution**.
  - Declare shenanigans if lookup result is not close enough to the key value.

# DHT Probability

Distance is **Exponential Distribution**

$$\text{PDF } f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & , x \geq 0 \\ 0 & , x < 0 \end{cases}$$



Reason: Walking along outer ring, frequency of node occurrences is a **Poisson Process**

$$p(x; \lambda) = \frac{e^{-\lambda} \lambda^x}{x!} \quad \lambda = 1 \quad \leftarrow \text{Rate w/ interval of one } n^{\text{th}} \text{ of the ring}$$

↑

Probability of “x” occurrences within one interval

# Simplistic Lie Detection

Pr(Travel around  $(1/n)^{\text{th}}$  of ring without seeing a node) =  $e^{-1}$

Pr(Travel around  $(T/n)^{\text{th}}$  of ring without seeing a node) =  $e^{-T}$

Simple algorithm:

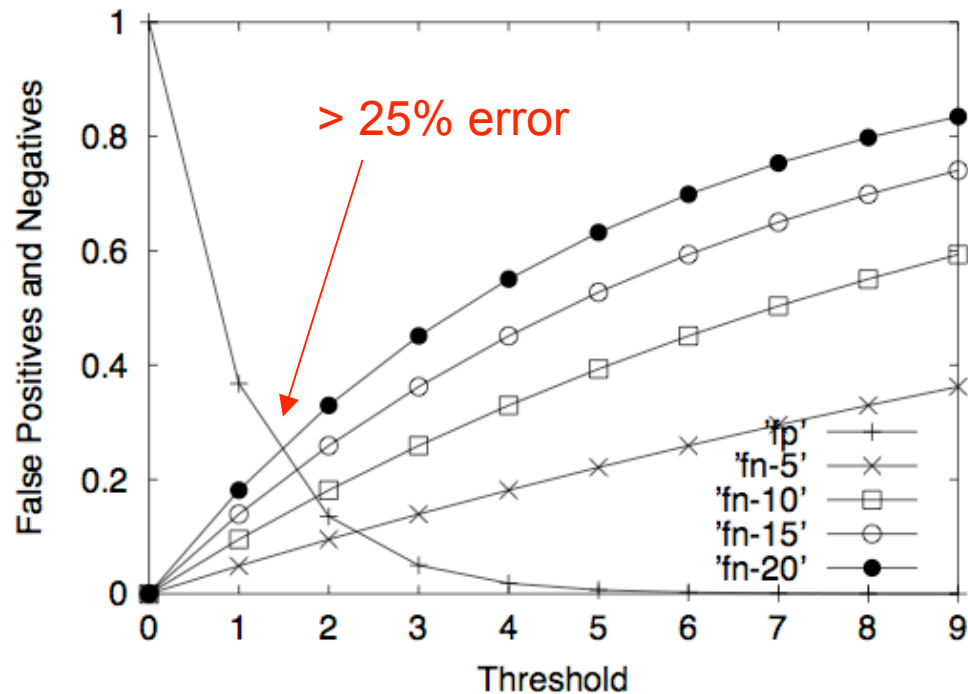
Pick a threshold,  $T$ .

If  $\text{distance}(\text{node}, \text{key}) \geq T/n$

Declare Shenanigans!

# Simplistic Lie Detection [12]

False Positives and False Negatives are substantial



Other (more complicated) ways to significantly reduce the error.

# Final Word on Lie Detection



- If we need to lookup one key in particular, error rates are probably too high
- If we can replicate functionality among many nodes ( $r$  file replicas), unlikely that:
  - False negatives on all lies
  - False positives on all well-behaved nodes
- Works for optimized or constrained routing

# More Powerful Adversaries



- Until now, assumed that if  $f$  fraction of the overlay is malicious means  $f$  fraction of my routing table points to adversaries.
- What if adversaries can “poison” routing tables to increase their influence?

# The Sybil Attack [5]



- Without a trusted third party, one attacker may assume an unbounded number of identities on the overlay network.
- Chord and Pastry imply trust of IANA to somewhat mitigate this
  - Owners of large IP space yield more power
  - Will really become a problem with IPv6

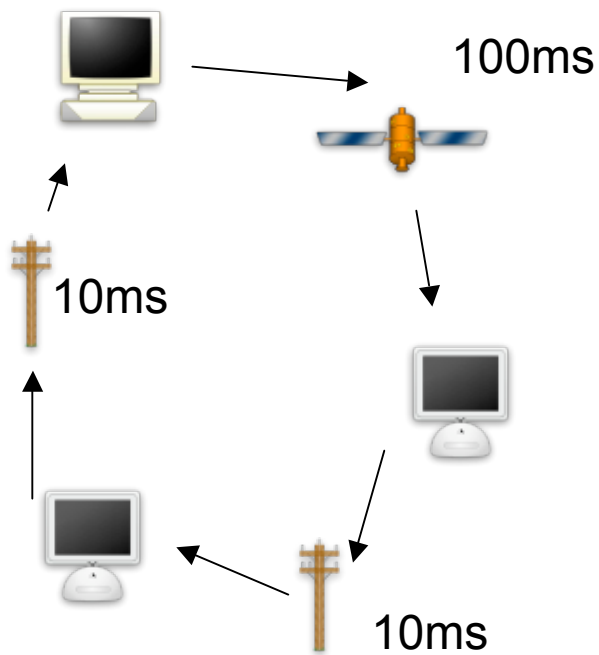
# The Eclipse Attack [10]



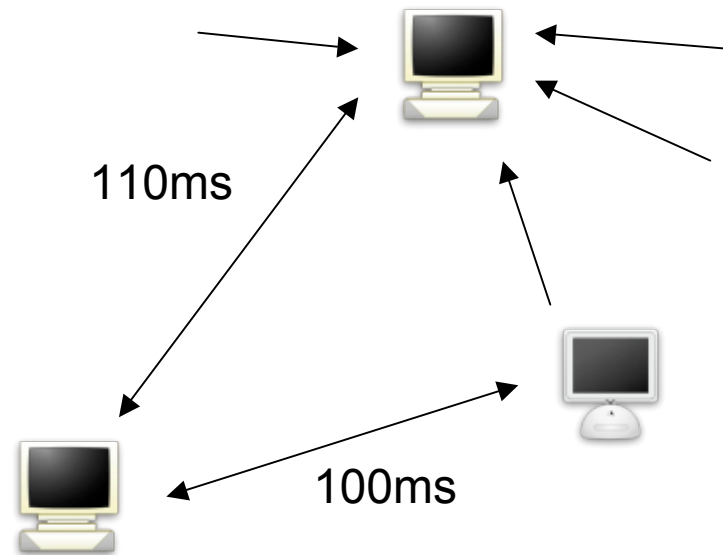
- If an attacker can “appear” to be closer than good nodes in the underlying network, the attacker will be chosen to populate the **proximity-optimized** table.
- Not nearly as effective on a **constrained** routing table, since routing table IDs are chosen by strict rules.

# How to “appear” closer

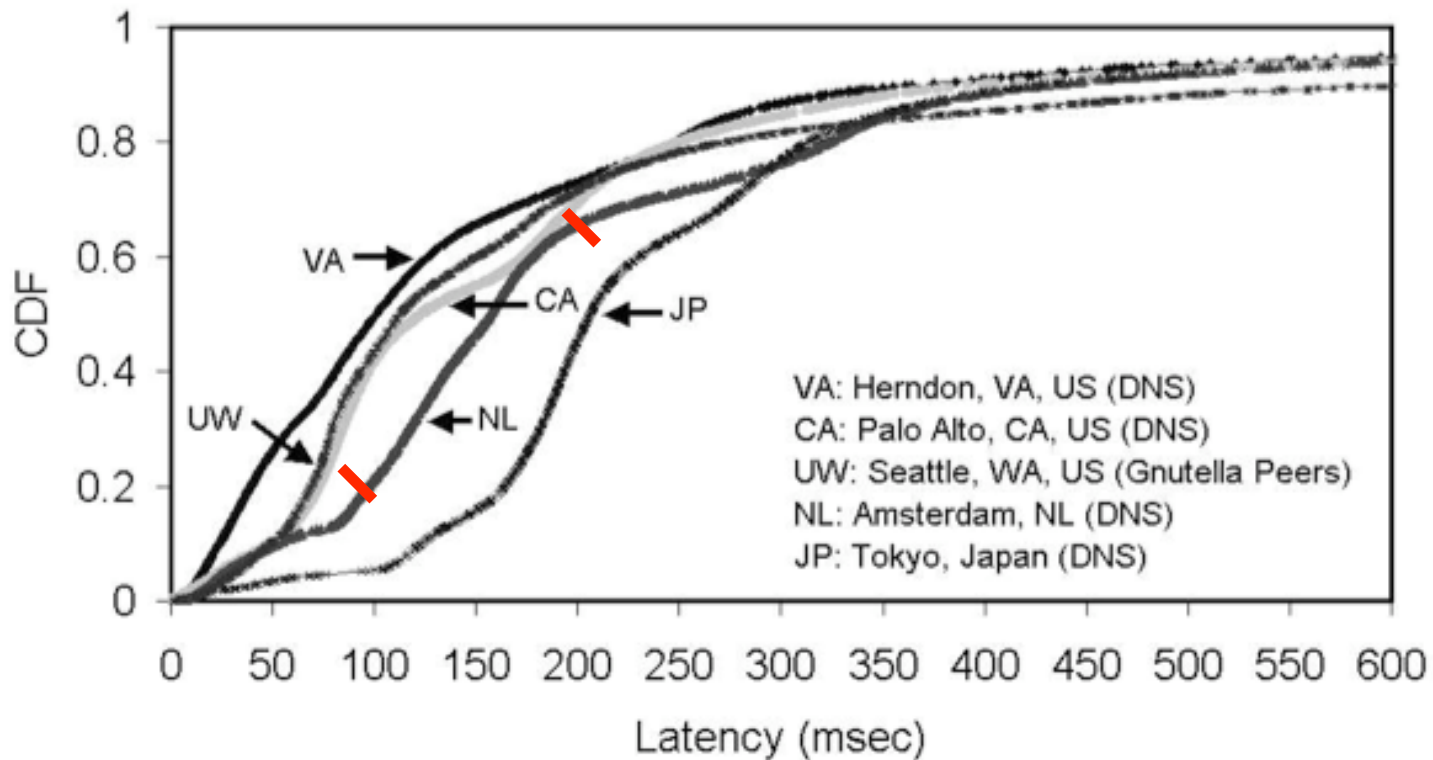
The Internet isn't a Euclidian space, use alternate reply routes!



Launch DoS attacks against good nodes to slow them down slightly



# C'mon -- is this feasible? [6]



Over 40% of requests within 100ms latency

# Feasibility Test



- From home cable modem, performed 50 pings of `www.google.com`
  - Average Round Trip: 29.429ms
- 1 minute later, 50 pings again, this time performing two downloads over SSL.
  - Average Round Trip: 127.107ms

Difference ~ 100ms -- This attack is VERY feasible!!

# The Eclipse Attack



- This attack is DEVASTATING against overlays with **optimized routing tables**
- If we assume malicious nodes can always use proximity in their favor, initial tests show adversary can achieve 100% routing table control with  $f = 20\%$
- More details tomorrow by Dan

# Shelter from Eclipse Attack



- Solution #1 [2]
  - Use **optimized routing table** unless we detect lying. Then switch to highly redundant and **constrained routing table**.
- But...
  - Redundancy causes a lot of overhead.
  - No proximity considerations may cause unacceptable delay.

# Shelter from Eclipse Attack



- Solution #2 [10]
  - Perform auditing of all nodes to determine that their in-degree and out-degree are appropriate
  - May allow for us to retain the use of our optimized routing table
  - Dan will address this in depth tomorrow

# Churn as Shelter [4]



- Completely off-the-wall solution (if you ask me)
- Force nodes to leave and rejoin the overlay at regular intervals
- When nodes rejoin, Overlay **NodeID = hash(random || IP)** , so both the victim and adversary are put in new random location within the overlay.
- Rejoins are staggered to maintain stability

# Churn as Shelter

Cert(timestep 100 - nonce '0xf01b')  
Cert(timestep 101 - nonce '0xb33f')  
Cert(timestep 102 - nonce '0x4e33')  
Cert(timestep 103 - nonce '0xa30b')

At time  $t$ , group  $g$  uses:

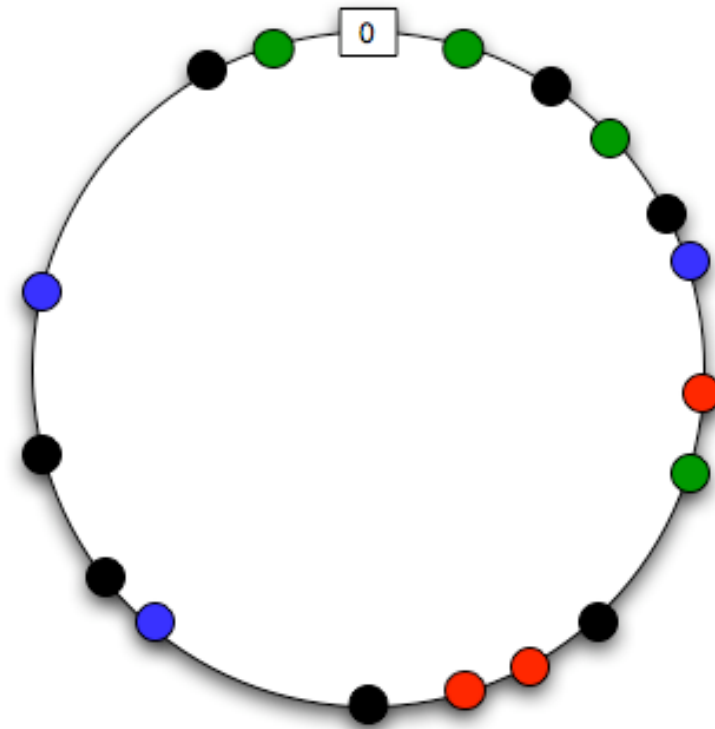
$$t - \left( \left( t - \frac{gk}{G} \right) \bmod k \right)$$

Group 0 - ID = hash('0xf01b' || IP)

Group 1 - ID = hash('0xb33f' || IP)

Group 2 - ID = hash('0x4e33' || IP)

Group 3 - ID = hash('0xa30b' || IP)



$G = 4$  total groups

$k = 4$  timesteps per epoch

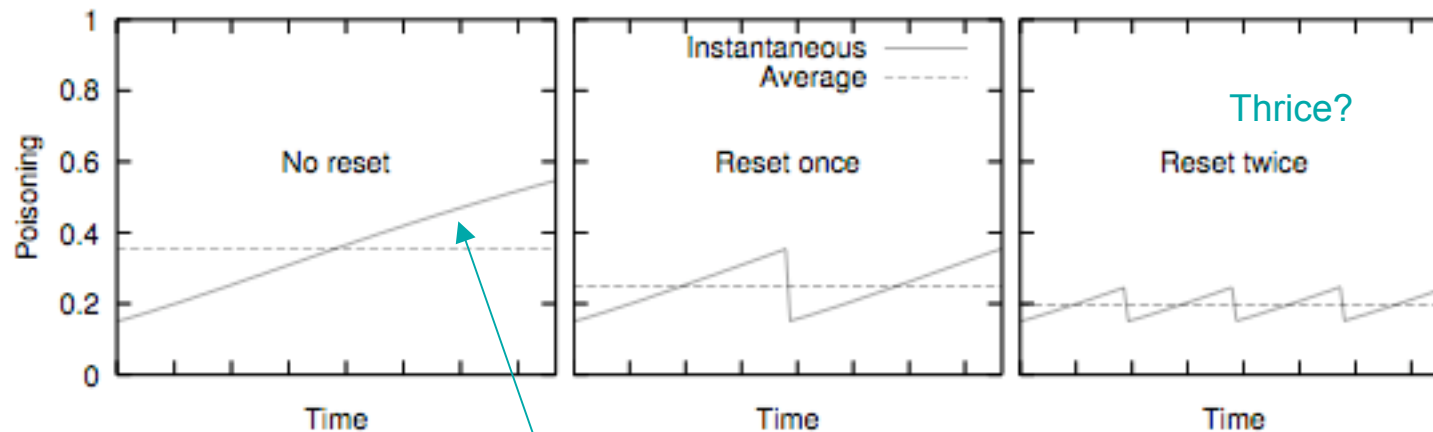
# Tale of 2 Routing Tables



- Constrained Routing Table (**CONS**)
  - No proximity consideration
  - Highly redundant
  - Can precompute **CONS** for next time epoch
- Churn-Optimized Routing Table (**CHURN**)
  - Proximity consideration
  - Flushed at the beginning of every epoch
  - Changes and Updates are rate-limited

# The Intuition

- Forced rejoins (resets) reduce average poisoning level in the **CHURN** table



Poisoning has flat slope due to limiting of the update rates

# Not all updates made equal

If you can poison one entry, which one should it be?

-0-2212102	<b>1</b>	-2-2301203	<b>-3-1203203</b>
<b>0</b>	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	<b>2</b>	10-3-23302
102-0-0230	102-1-1302	102-2-2302	<b>3</b>
1023-0-322	1023-1-000	1023-2-121	<b>3</b>
10233-0-01	<b>1</b>	10233-2-32	
<b>0</b>		102331-2-0	
		<b>2</b>	

$$\frac{1}{2^b} = \frac{1}{4}$$

routes affected

$$\left(\frac{1}{2^b}\right)^7 = \frac{1}{16384}$$

Upper entries carry exponentially more importance, both in an attack and in an optimization on node initiating a lookup on the overlay!!

# How do you measure poisoning?

Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Routing table			
-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Measure by fraction of poisoned entries:

$$\text{Poisoning(A)} = \text{Poisoning(B)} \sim 1 / 25$$

“Churn” Metric

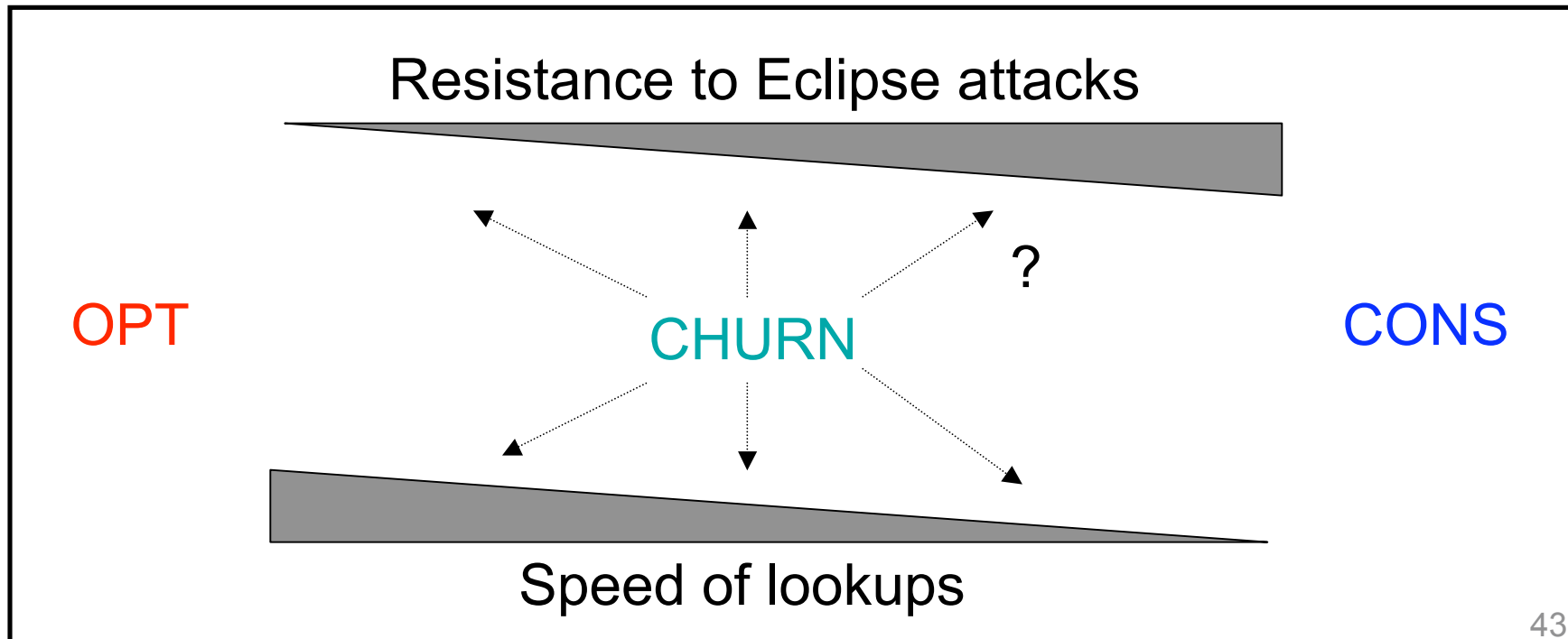
Measure by effective control over routing table:

$$\text{Poisoning(A)} = 1/4$$

$$\text{Poisoning(B)} = 1/25$$

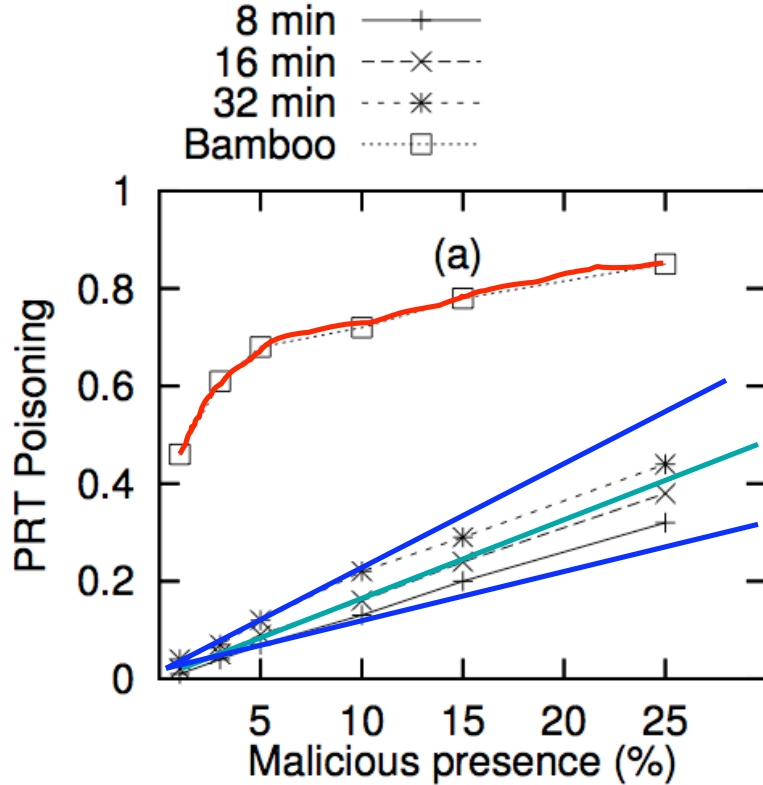
# Can “Churn” be the answer?

- We are essentially giving up **some** optimization for **some** protection against eclipse attacks.
- We need to quantify this trade-off!

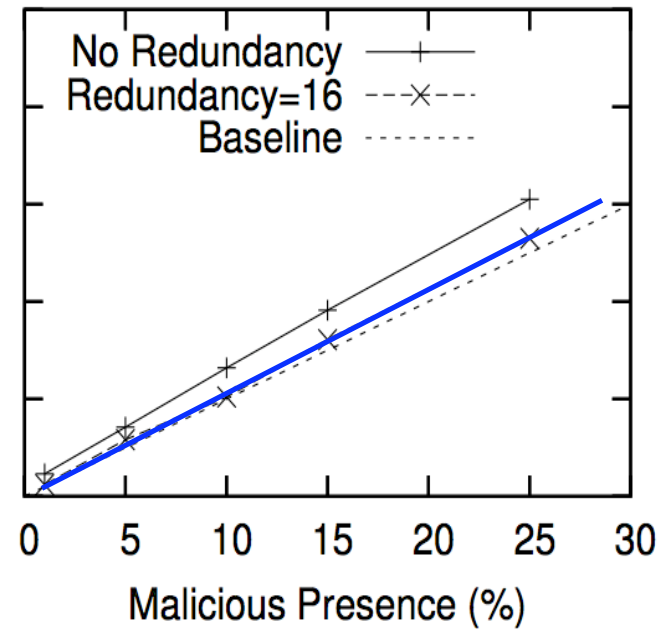


# Table Poisoning in an Attack

OPT -- CHURN -- CONS

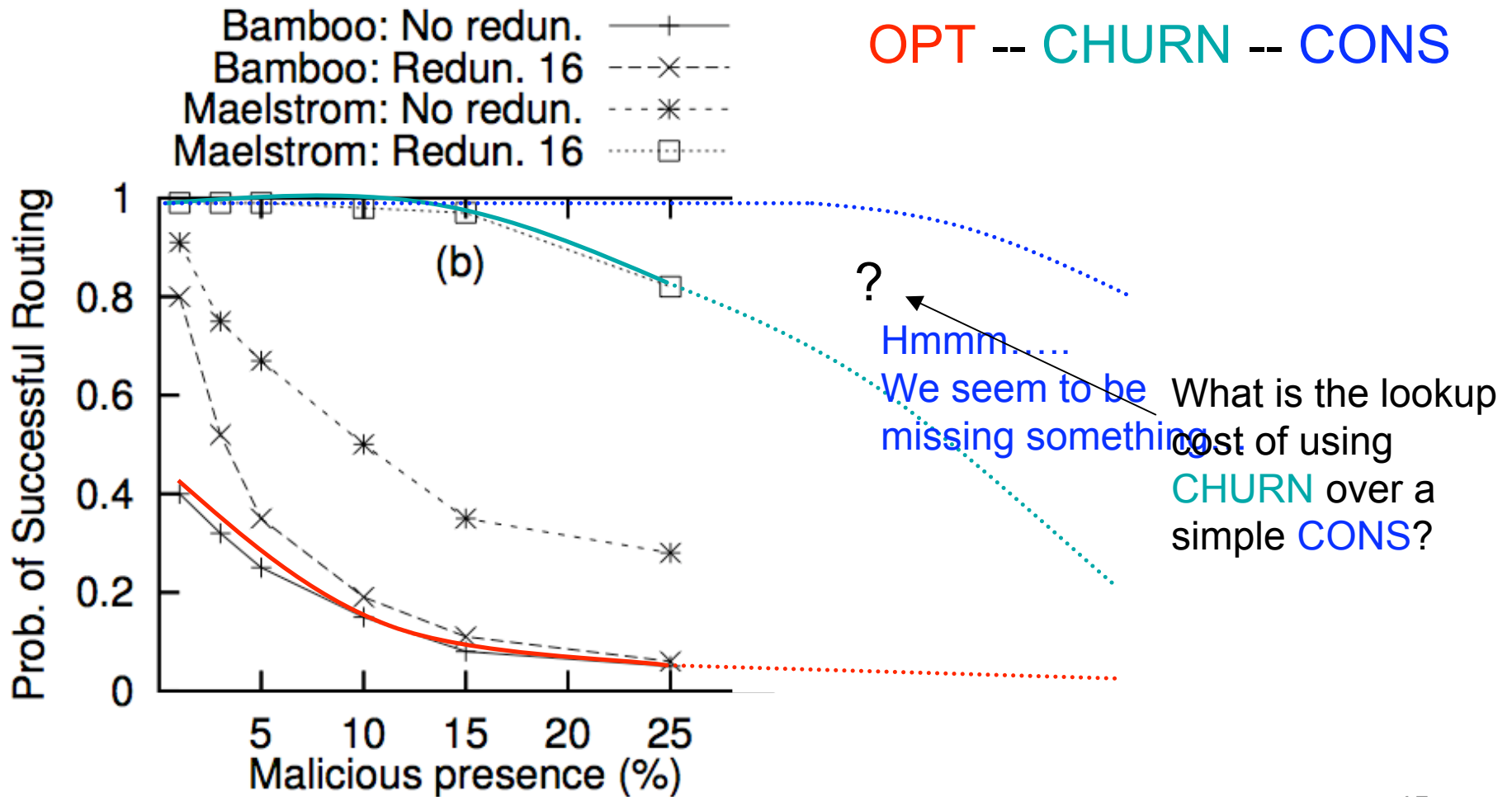


.5



Still lose additional **20%** of table entries over **CONS**

# Lookup Success in an Attack



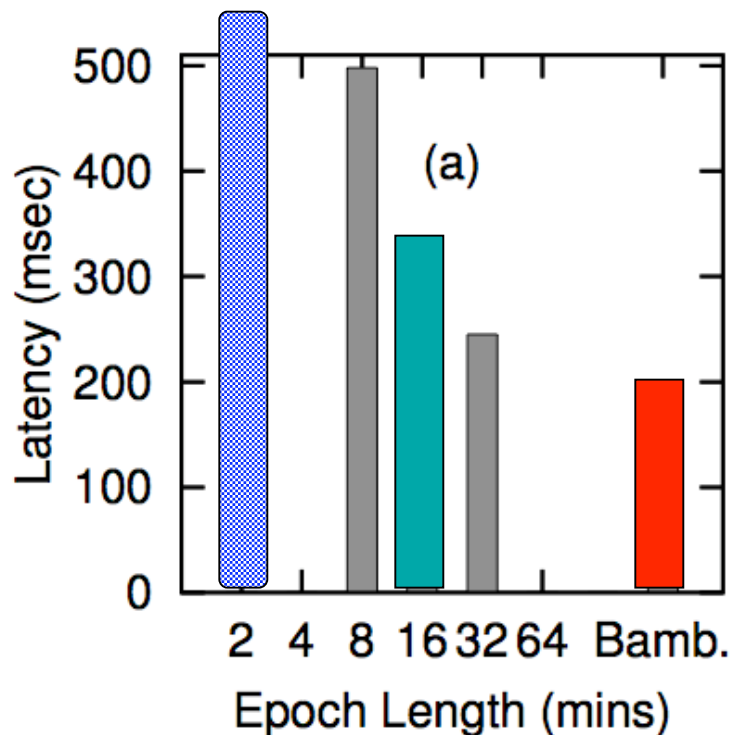
# CHURN resistance to attack



- Churn gives us “some” shelter against Eclipse attacks as compared to a fully optimized routing table (**OPT**)
- Using only a constrained (**CONS**) routing table presumably performs much better in the face of an Eclipse attack
  - Not particularly clear as to exactly where **CHURN** sits between **CONS** and **OPT** when handling Eclipse attacks

# CHURN in good conditions

- What if there is no attack? How much do we pay for using CHURN in lieu of OPT? How much better is CHURN than simply using CONS?

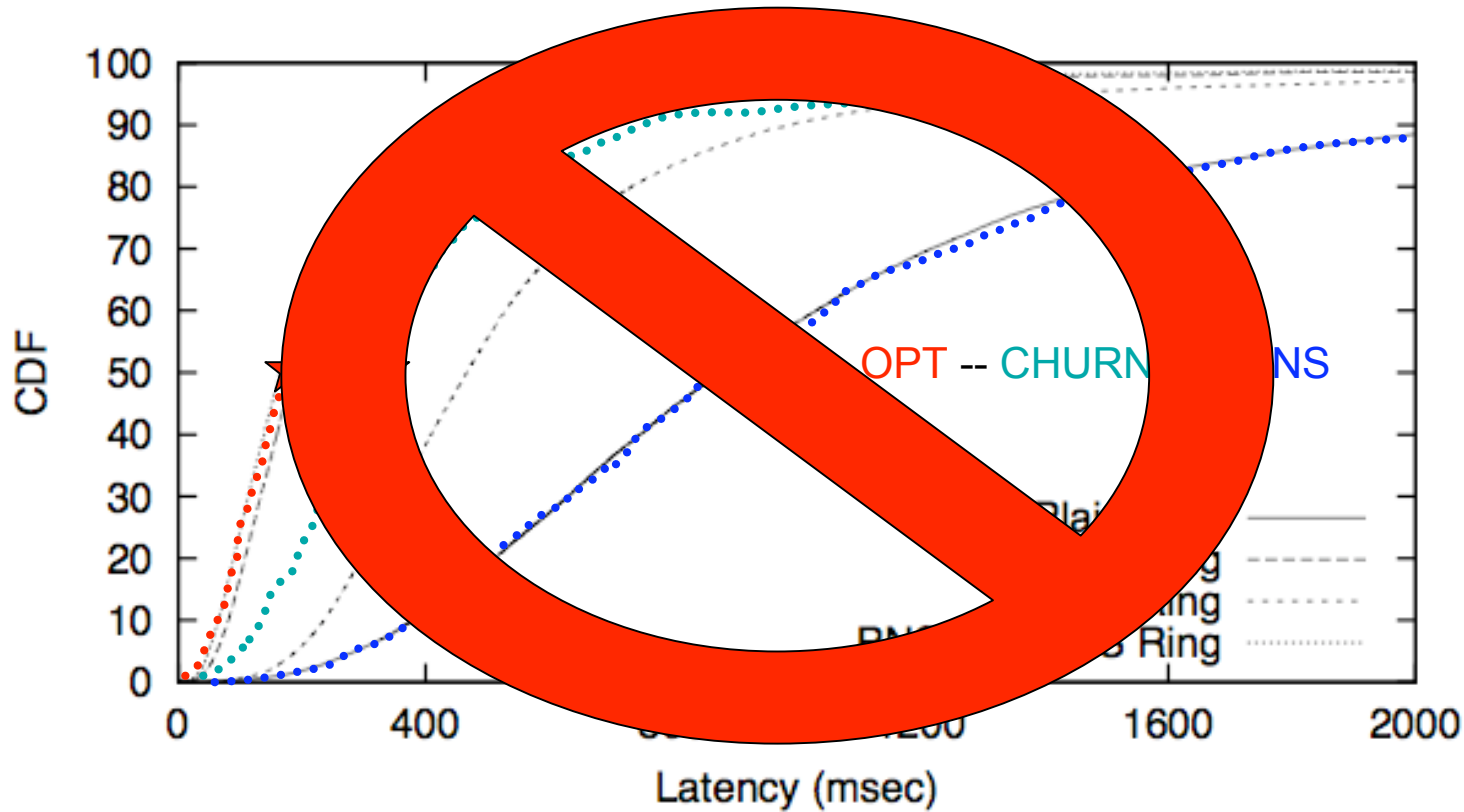


OPT -- CHURN -- CONS

Hmm....

Can we at least make an educated guess?

# Cost of CHURN



There is absolutely no reason why I should have to do this guesswork!

# Cost of CHURN



- It's just not clear how much faster this hybrid approach is over a CONS table.
- Can CHURN keep up with applications that demand low latency?
  - Difference between 200ms and 400ms is substantial for applications like VOIP.
  - If your application easily tolerates 600ms of delay with CHURN, couldn't it probably handle 1000ms just as easily?

# Final word on CHURN

- In well-behaved network:
  - How much faster is CHURN than CONS?
  - How much faster is OPT than CHURN?
- Under Eclipse attack:
  - How much safer is CONS than CHURN?
  - How much safer is CHURN than OPT?

The only question that was well addressed in the paper -- did they sell it to you?

# Different Adversarial Model



- Until now, we assumed that our adversary wanted to cause as much havoc as possible on the overlay as a whole
- What if our adversary is less greedy?
  - We consider the example where our overlay is used as a distributed file system
  - Can our adversary **target** one particular file so as to deny us access to it?

# DHT File Systems



- Farsite [1]
- OceanStore [8]
  
- Generally protect data with cryptography
- Adversary can still launch DoS attack against specific files
- An attack of this nature is analogous to censorship over a distributed file system

# LocationGuard [11]

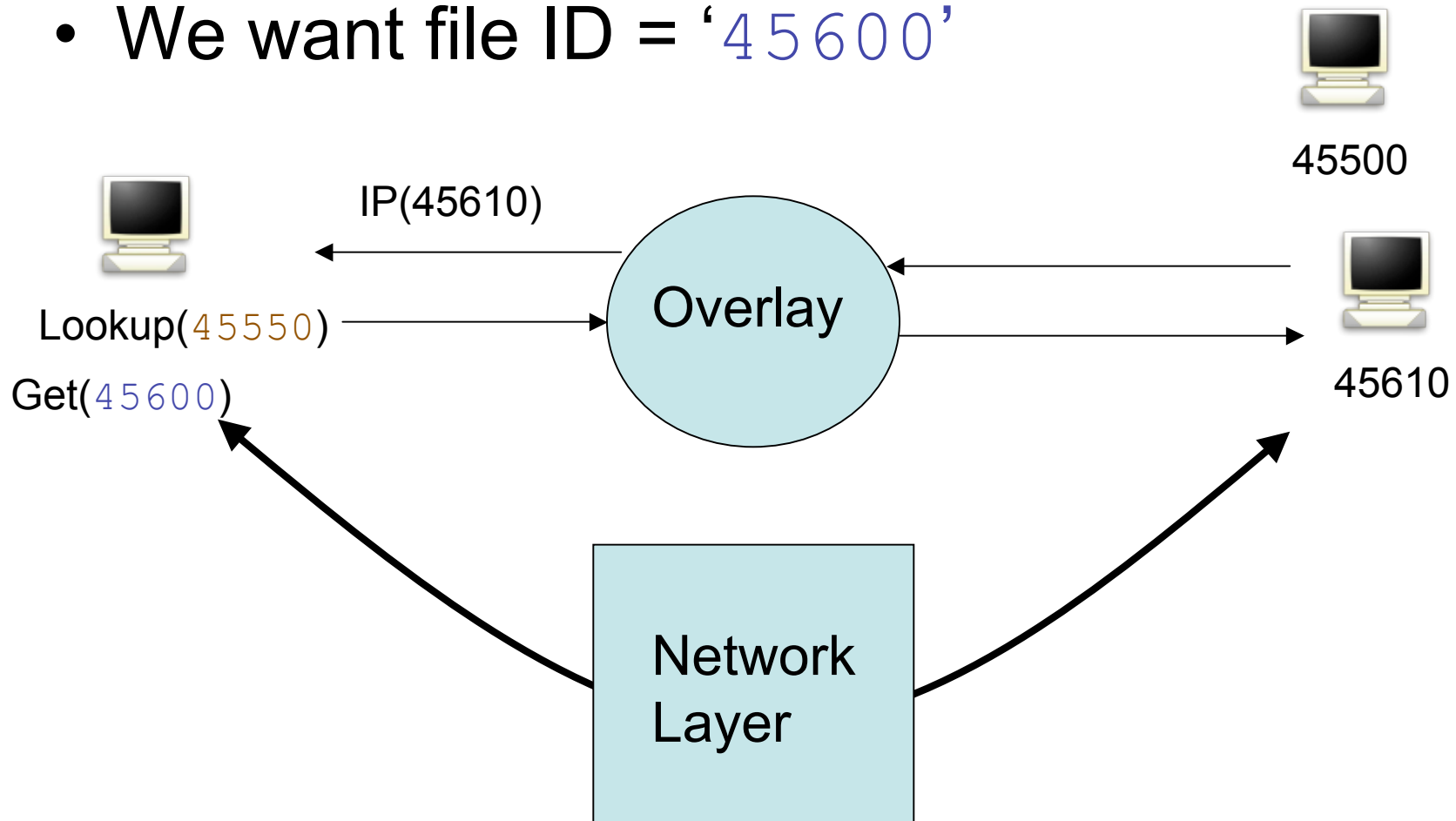
- Adversary can't target a specific file if he doesn't know what to look for
- Assume we have  $i$  replicas of a file on an overlay
- We first protect our filename with a *LocationKey*

$$Identifier_i = E_k(filename \parallel i)$$

We still don't want to perform a simple *Lookup(Identifier<sub>j</sub>)* -- why not??

# LookupGuard

- We want file ID = '45600'



# LookupGuard

- How much we can shift our lookup value and still find the right node?
- Use same probabilistic properties that were used to detect lying!

$$\Pr(\textit{safe}) = e^{-rN} \longrightarrow r \leq \frac{-\ln(\Pr(\textit{safe}))}{N}$$

Amount of (relative) shift in lookup value

Total number of nodes in the overlay

# Obfuscation Example

- $N = 1$  million ( $2^{20}$ )
- Want a correct lookup 999,999 times out of 1 million tries

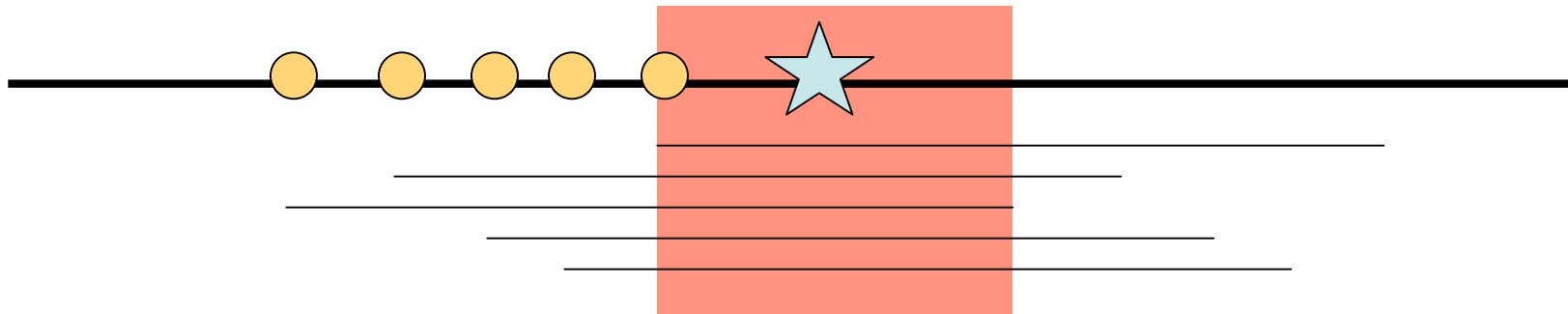
$$r \leq \frac{-\ln(1 - 2^{-20})}{2^{20}} \rightarrow r \leq 2^{-40}$$

Suppose a Chord ring of keyspace  $2^{160}$

Max safe obfuscation offset =  $(2^{160})(2^{-40}) = \mathbf{2^{120}}$

# Sieve Attack

- Given enough queries, can't the adversary *sieve* the space to discover the true token?



Given  $x$  samples, we expect an adversary to narrow the range to a size of  $(\text{initial range}) / x$

# Sieve Attack



- Obfuscation range is HUGE  $\sim 2^{120}$
- Even after 1 million lookups, adversary has only narrowed range to  $2^{100}$
- Attack has to be performed on-line, not at all feasible for keyspaces this large

Of course, all of this analysis is meaningless if our adversary owns the keyspace with our file in it...

# Inference Attack



- We should further guarantee that an adversary cannot infer statistics from:
  - Lookup frequency
  - End-user IP address
  - File replicas
  - File size
- Important that two adversaries can't tell they own copies of the same original file

# Survivability



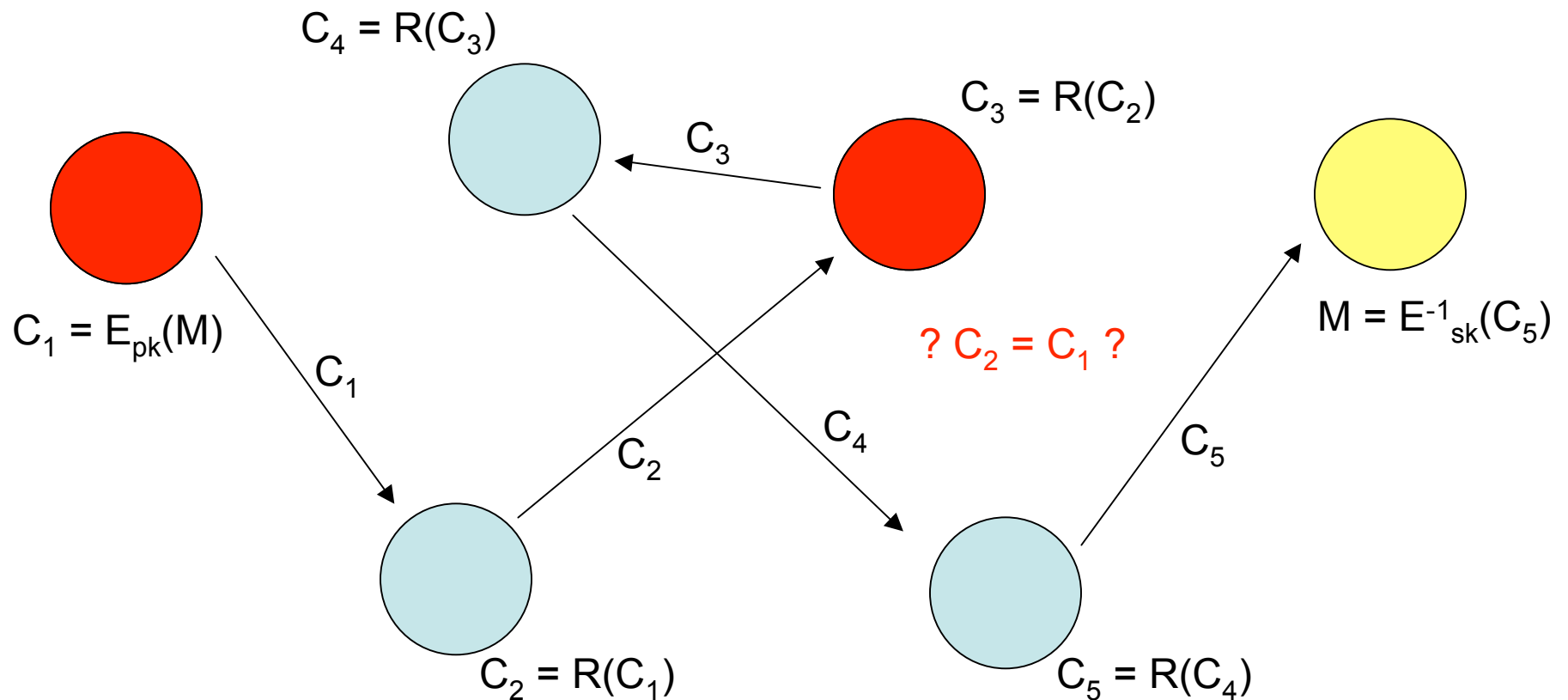
- LocationGuard mitigates the DoS attacks of today -- but data will degrade over time!
  - Malicious nodes can corrupt data they control, and churn means that over time adversaries could corrupt a lot
  - Nodes crash (maybe I wrote the code)
- LocationGuard's replication offers no means by which to replenish our data

# Universal re-encryption [15]



- Can nodes automatically re-publish data before it degrades?
- Current construction of **universal re-encryption** allows for nodes to re-encrypt without any knowledge of the public key
- Requires the definition of *semantic security under re-encryption*

# Universal Re-encryption



Can be done with variation of ElGamal -- construction is not particularly complicated, refer to [15] for more details

# The big picture



- We want to design a DHT that is:
  - Highly optimized WRT proximity
  - Resistant to Eclipse Attacks
  - Supports strong data protection (crypto)
  - Is resilient against targeted attacked
  - Has very good survivability for application-level content stored on the overlay

How much free time does everyone have?

# References



- [1] Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, R. P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment", 5th OSDI, Dec 2002.
- [2] M. Castro, P. Drushel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In OSDI '02, Boston, MA, 2002.
- [3] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Exploiting Network Proximity in Peer-to-Peer Overlay Networks. In Technical Report MSR-TR-2003-82, Microsoft Research, 2002.
- [4] T. Condie, V. Kacholia, S. Sankararaman, J. M. Hellerstein, and P. Maniatis. Induced Churn as Shelter from RoutingTable Poisoning. In Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA, USA, Feb. 2006.
- [5] John R. Douceur. The sybil attack. In Proc. of the IPTPS02 Workshop, Cambridge, MA (USA), March 2002.
- [6] R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of dht routing geometry on resilience and proximity. In Proc. ACM SIGCOMM'03, Karlsruhe, Germany, 2003.
- [7] Antony I. T. Rowstron and Peter Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in Middleware, 2001.

# References



- [8] J. Kubiatawicz, et al. OceanStore: An Architecture for Global-Scale Persistent Storage. ASPLOS, December 2000.
- [9] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In Proc. ACM SIGCOMM 2001, August 2001.
- [10] Singh, M. Castro, P. Druschel, and A. Rowstron. Defending against Eclipse attacks on overlay networks. In Proceedings of the 11th ACM SIGOPS European Workshop, pages 115--120, Leuven, Belgium, Sept. 2004.
- [11] Mudhakar Srivatsa and Ling Liu. Countering Targeted File Attacks using LocationGuard. In Proceedings of the 14th USENIX Security Symposium, to appear August 2005.
- [12] Mudhakar Srivatsa and Ling Liu, Vulnerabilities and Security Threats in Structured Overlay Networks: A Quantitative Analysis , In the Proceedings of the 20th IEEE Annual Computer Security Applications Conference (ACSAC 2004)
- [13] Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. Technical Report TR-819, MIT, March 2001.
- [14] Ben Zhao, John Kubiatawicz, and Anthony Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.
- [15] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal Re-encryption for Mixnets, 2003.



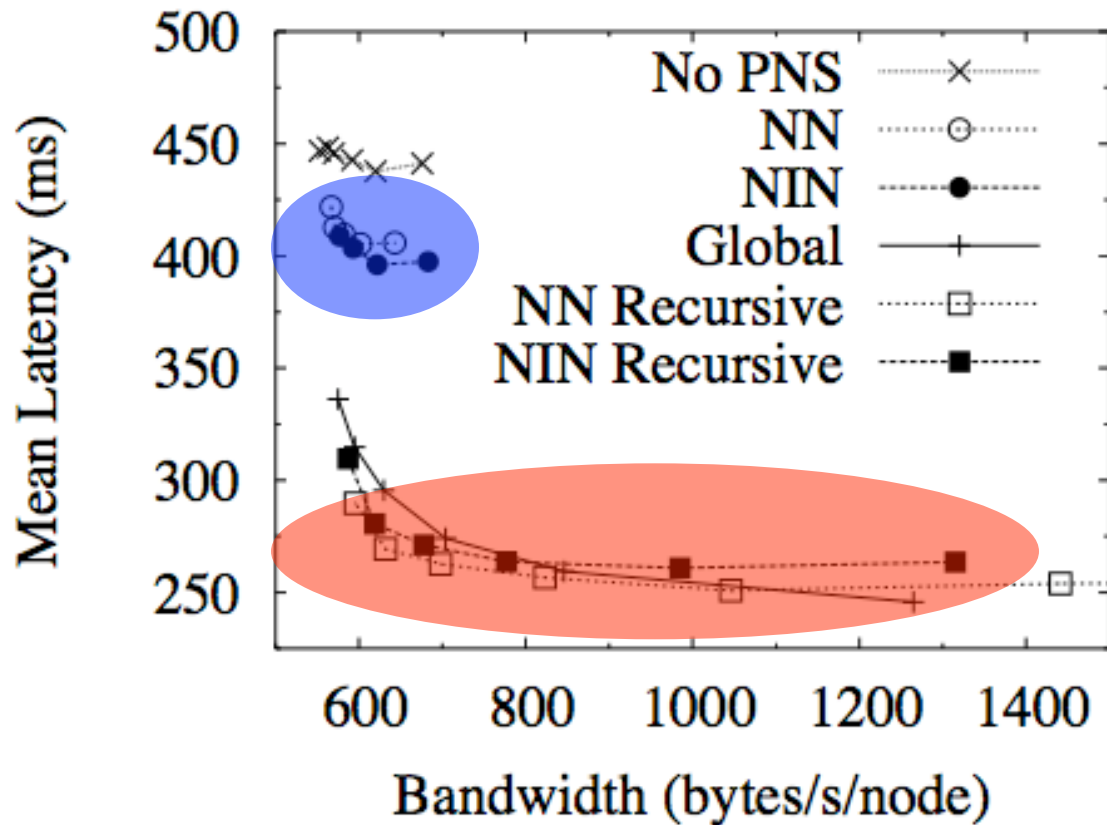
Fin.

# After reset, must update quickly



- Global Tuning
  - Perform lookups at random, test your replies
- Local Tuning (Network Neighbor)
  - Ask nodes in your routing table for their tables
- Network Inverse Neighbor
  - Ask nodes in your table for their backpointers
- Recursive variants

# Which update method is best?



Just getting our neighbors' routing tables is cheap and easy

But doing random lookups is much more effective!