Identity Theft Protection in Structured Overlays

Lakshmi Ganesh
Ben Y. Zhao
University of California, Santa Barbara

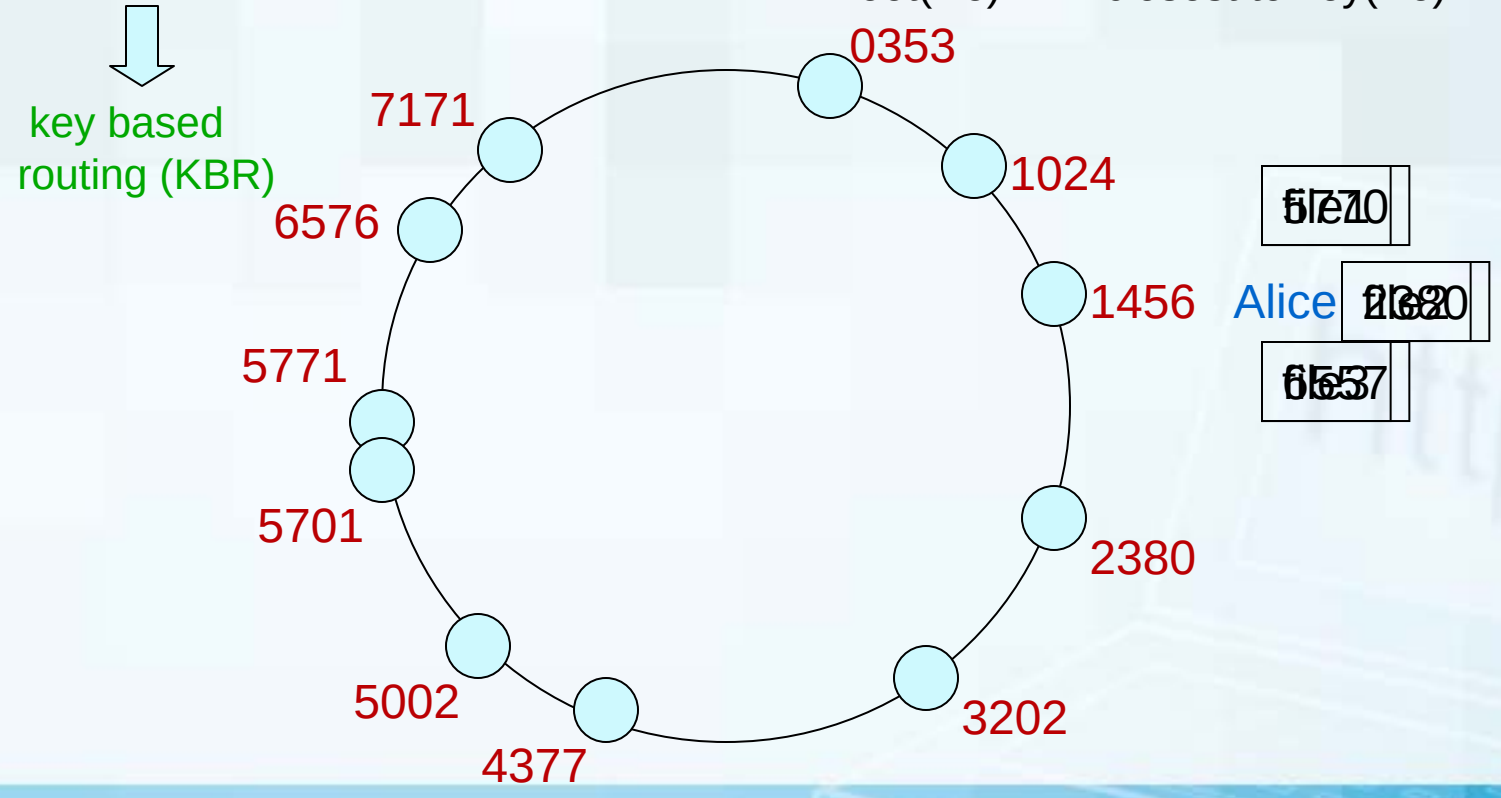
NPSec 2005

Background: Structured Overlays

- network abstraction – IDs, not IPs
- app level entities assigned keys from same namespace as IDs
- keys mapped to nodes (IDs)

key based routing (KBR)

- many applications – eg, dist. db
- Alice wants to store her files
- key(file) = hash(meta info)
- root(file) = ID closest to key(file)

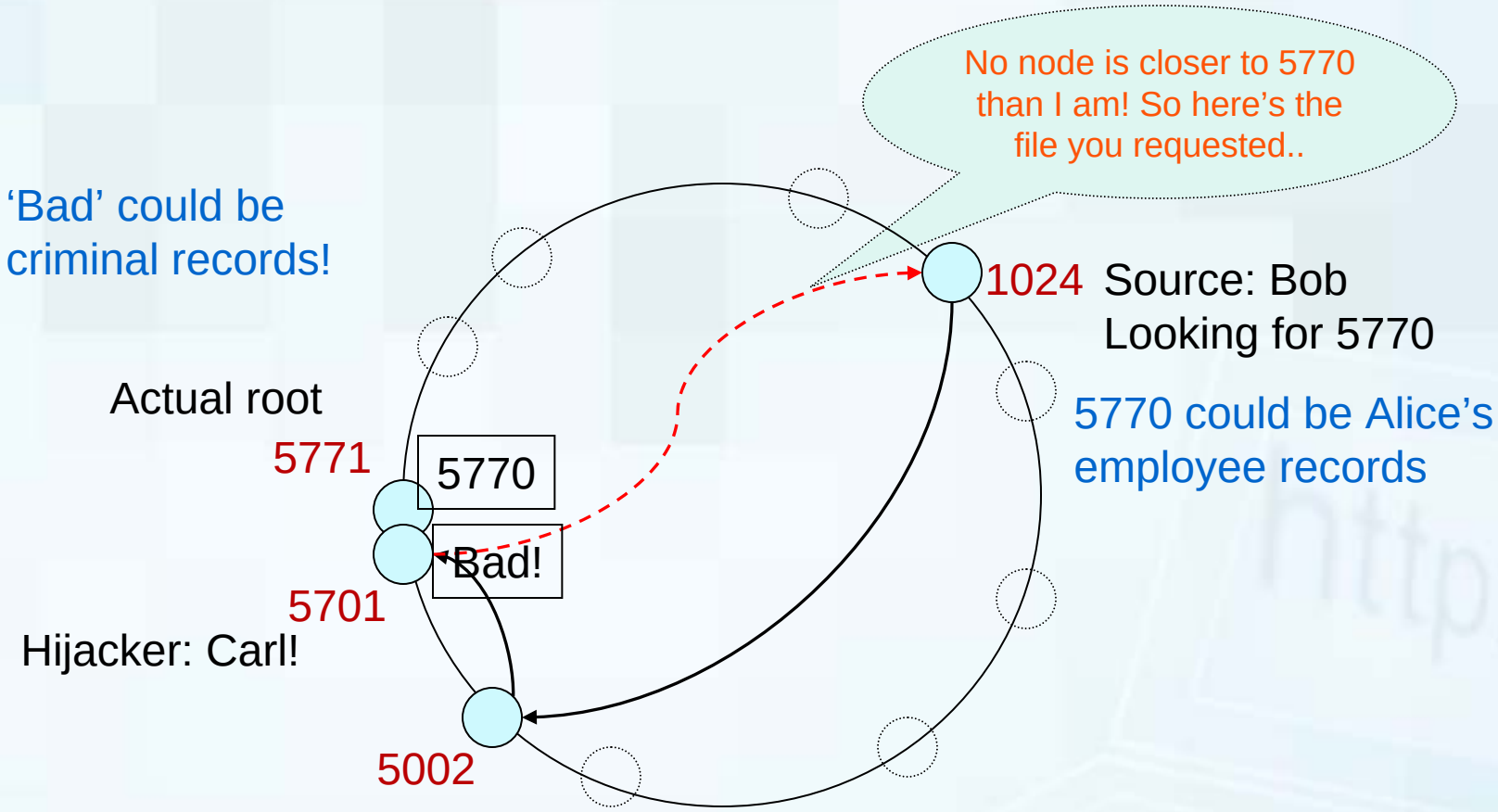


The Identity Attack

- Structured overlays rely on Key-based Routing
  - Nodes maintain limited state
    - Route messages by forwarding progressively closer to destination
    - Routing stops when a node claims it is the closest to the key
  - Relies on KBR for setting up connections between application nodes
- Identity Attacks hijacks key → node mappings
  - A hijacker would (falsely) claim that it is the closest node to the given key.
  - Hijack responsibility for storing / retrieving data, forwarding data, or any other application level responsibility

Identity Theft: Illustration

- Eg: Network with namespace length = 4 and base = 8



Structured Peer-to-Peer Security

- Sybil Attack
  - Relies on: nodeIDs are free
  - Obtain large number of nodeIDs
  - Resulting virtual identities can collude as a group
  - Defense: centralized Certificate Authority
- Eclipse Attack
  - Relies on: routing table optimization for performance
  - Leverage Sybil, then fill victim's routing table with colluders
- Now what…
  - Sybil or Eclipse get you close to the victim, now what?
  - DoS easily detected, Identity Attack more powerful
  - Can also perform Identity attack independently w/ single node, hence more general than Sybil or Eclipse

Outline

- What we've covered
  - Background
    - Structured Overlays
    - Attacks
      - Identity, Sybil, Eclipse
      - Preventing Identity attacks will disempower Sybil, Eclipse attacks
- Now we'll see
  - How to detect Identity Attacks
  - Analysis of our solution

How to detect Identity Attacks

- First we get suspicious, then we seek proof
- Step 1: Figure out *when* some responder is suspicious
  - How far must the responder's ID be from the key to warrant suspicion?
- Step 2: So now you're suspicious: how to verify?
  - Ask others? But how do they know?
    - Certification!
    - Based on its ID, each node picks some nodes to certify itself with
    - You can now ask these 'proof managers'

Geographical Analogy: Intuition for Step 1

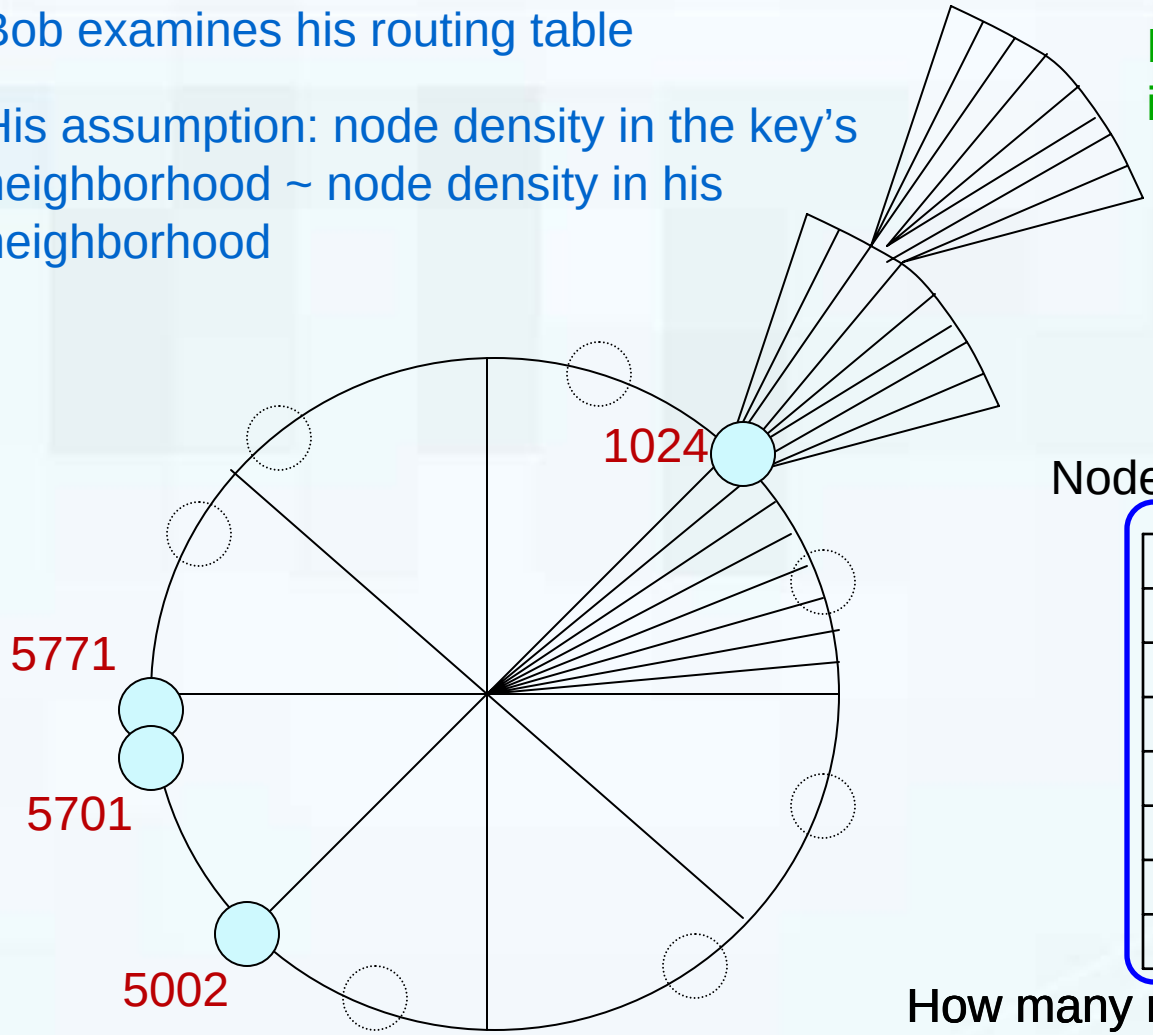
I look at my address book: there are 3 ppl in my country – isn't there even one in Egypt?? I'm unconvinced!

- ppl I know
- me

Step 1: How it looks for Bob

Bob examines his routing table

His assumption: node density in the key's neighborhood ~ node density in his neighborhood



He sees up to 3 nodes in the 102x range

Is it likely that there isn't even one node in the 577x range? NO!

Node 1024's Routing Table

0353	1024	1001	1020
1024	1177	10	102
2380	1254	1024	102
3202	1300	10	102
4377	1456	1044	1024
5002	1570	1051	102
6576	1606	10	1026
7171	1711	10	102

How many nodes in XXXX range?

Step 2: Proof Certificates

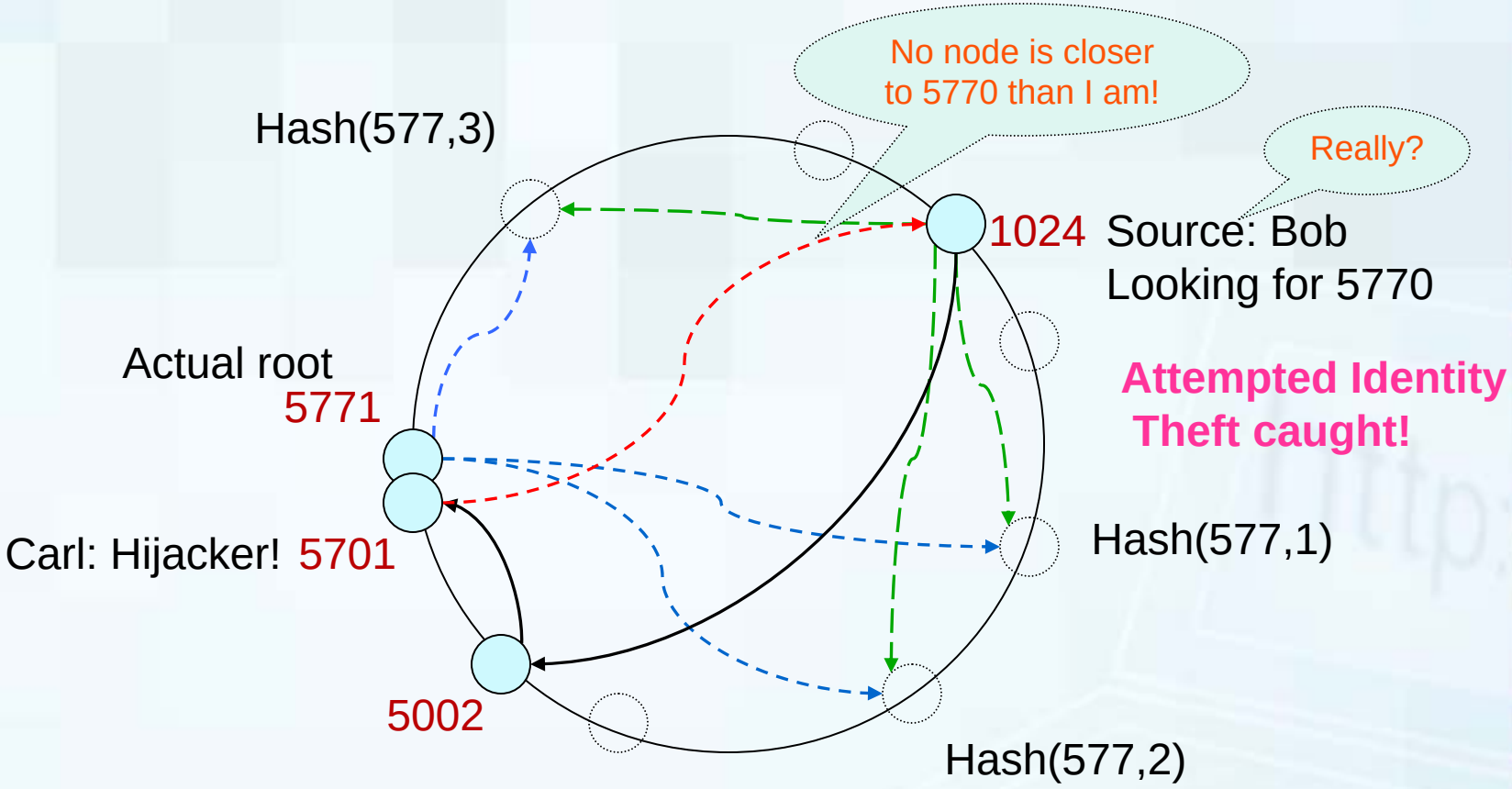
- Ok, we're suspicious.. Now what?
- Intuition:
  - Each node 'tells' a set of other nodes about its existence
    - Periodic certification (signed, time-stamped certificates)
  - These 'proof managers' can now be contacted for proofs
  - You verify the *neighborhood* you are in
    - 5770 verifies 57xx and 577x (for example)
  - Proof manager (PM) set computed based on prefix certified
    - PMs for 57xx = {hash(57,1), hash(57,2), hash(57,3)}
    - PMs for 577x = {hash(577,1), hash(577,2), hash(577,3)}
  - Bob would now ask hash(577,x) for proofs
- Scalable
  - You don't verify every possible neighbourhood
    - You don't need to

Verification

- Clients requesting verification
  - Estimate several prefixes of key that “should” exist
  - E.g., key = 5770
    - Test prefixes 577x, then 57x
  - For each prefix
    - Calculate location of proof managers by hashing prefix
    - Issue request to proof managers for certificates
  - If certificates exist
    - Proof of attack

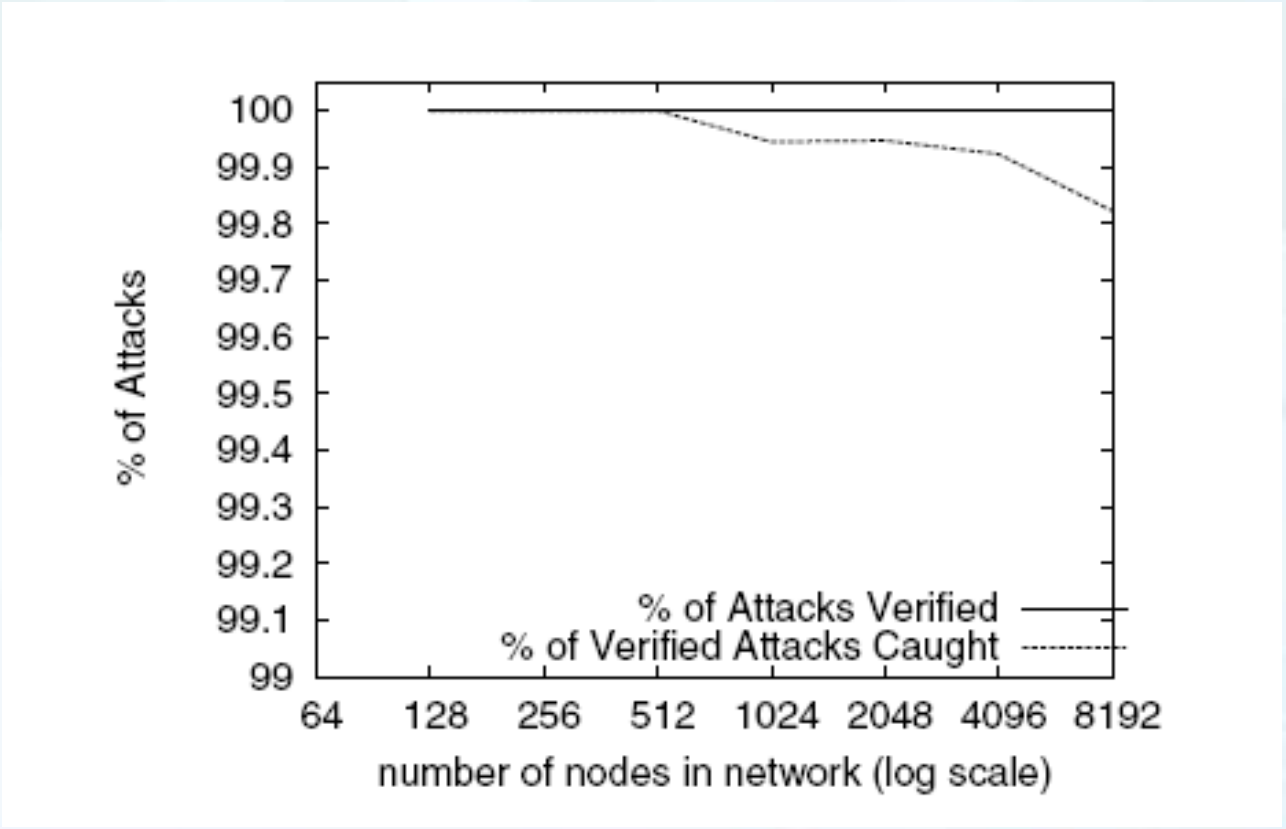
Tying it all together: Illustration

1. Certification (of 577x, say)
2. Routing
3. Verification



System Analysis: Performance

Effectiveness of the verification system under ideal conditions (no denials, no certificate hijacks, no node churn).

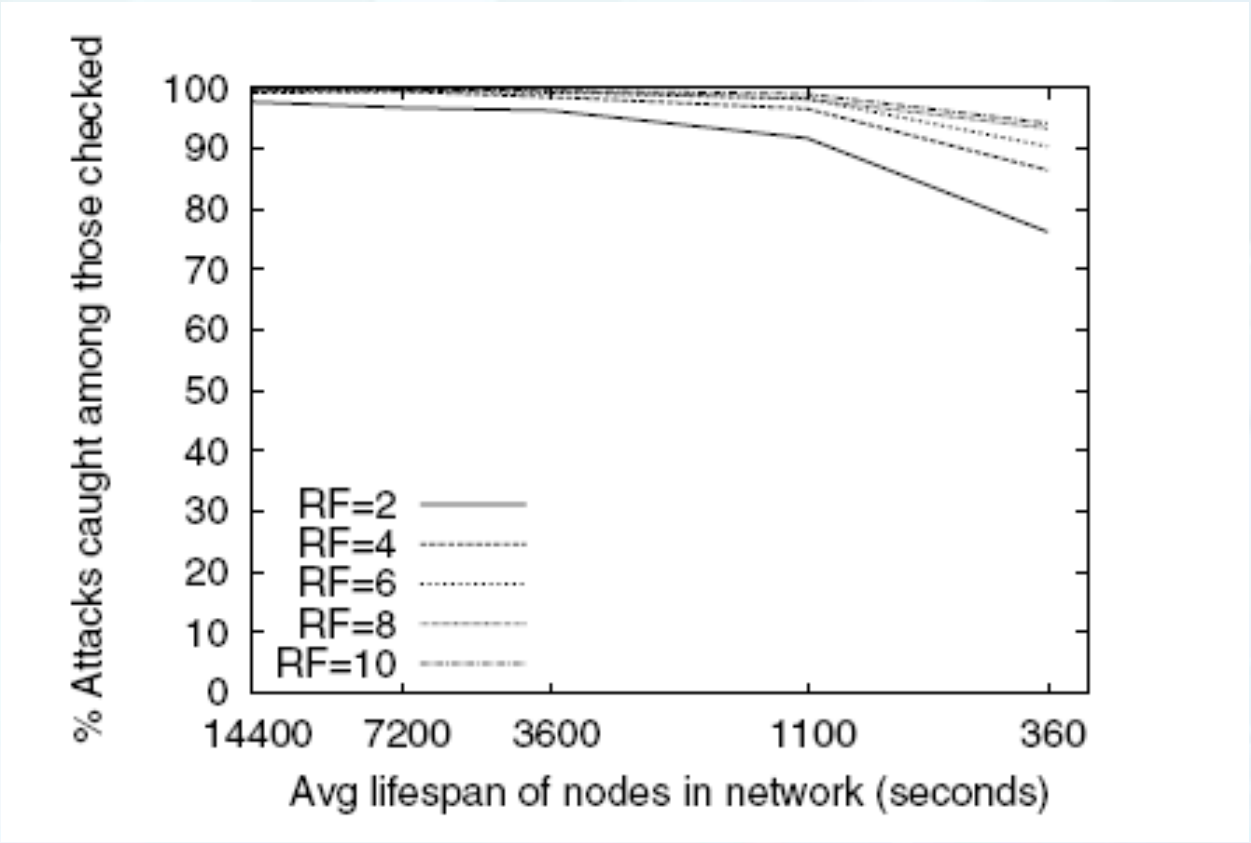


System Analysis: Factors

- Message Hijacks
  - The Identity attack
- Certificate Hijacks
  - Malicious node on path between node and its proof manager
- Verification Denials
  - Malicious proof manager
- Node churn
  - Nodes come and go

System Analysis: Performance

Use of replication factor to increase verification effectiveness (certificate denials, hijacks, and node churn assuming 20% malicious nodes).



System Analysis: Overhead

- Verification Overhead
  - Bandwidth: certs sent per node per second

    P*RF/T (=3*4/500 = 0.024 certs/sec ~ 1.25 bytes/sec)
  - Storage: certs stored per node

    P*RF (=3*4 certs ~ 600 bytes)

where *P = num PGs certified by each node*

RF = replication factor,

T = certification interval (secs)

cert size ~ 50 bytes

- Small price to pay to keep Alice happy ☺

Looking ahead…

- Dynamic computation of prefixes to verify
- Load balancing among members of a prefix group
- Extension to other protocols
  - For protocols that do not use prefix routing
  - Replace prefix groups with ‘range specifiers’
    Each specifier includes central point and range on each side
    i.e. 123X → range(1234.5, 4.5)

Thank You!

Questions?

Certification: Scalability

The probability of the cusp including more than 2 routing levels is

$P \leq b/e^b$, where b = base of the prefix digit.

$P < 0.07$ for $b = 4$ and

$P < 1.8 * 10^{-6}$ for $b = 16$.

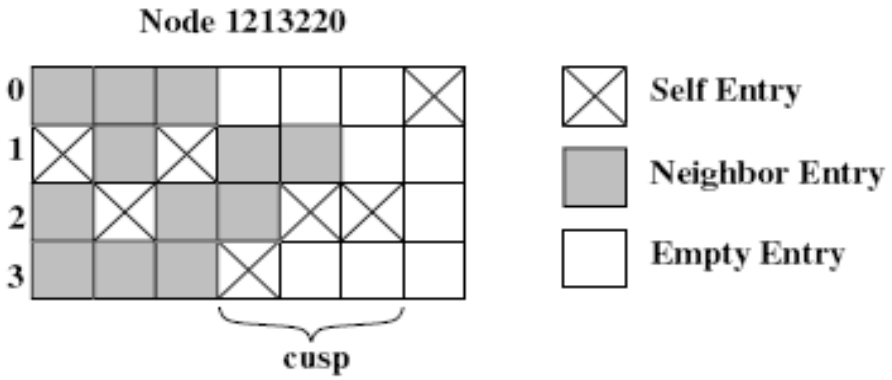
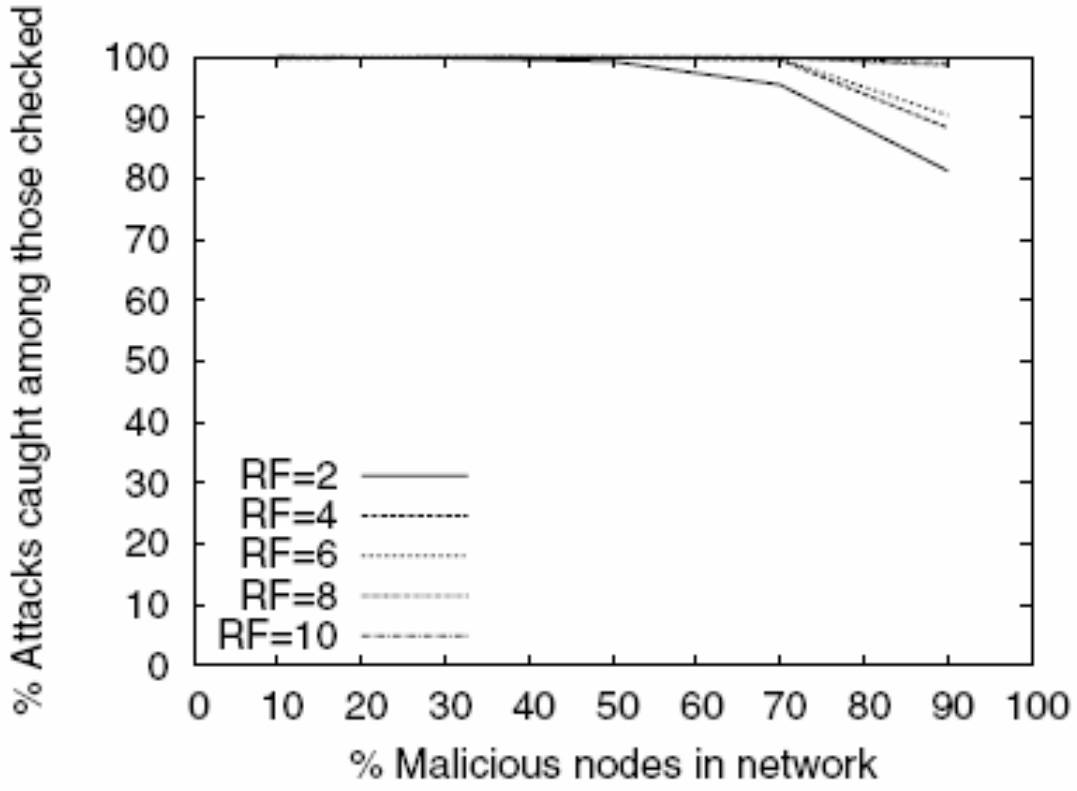


Figure 5. Routing table for node `1213220`, showing the *cusp region.*

System Analysis: Performance

Use of replication factor to increase verification effectiveness (certificate denials, no hijacks, no node churn).



System Analysis: Performance

Use of replication factor to increase verification effectiveness (certificate denials and hijacks, no node churn).