Security for Structured Peer-to-peer Overlay Networks

By Miguel Castro et al. OSDI'02
Presented by Shiping Chen in IT818

1

Acknowledgement

- Some of the following slides are borrowed from talks by Yun Mao (University of Pennsylvania)

2

Outline

- Background & Model of P2P Network
- How to achieve Secure Routing
- How to use Secure Routing
- Conclusions

3

What is the paper about and not about

- Not about traditional attacks
  - SYN flood, IP Spoofing, Buffer overflow, DoS attacks on resource access
  - Keep in mind these attacks still work
- About unique security problems in P2P
- Goal: Secured Routing
  - Ensure that when a correct node sends a message to a key, the message reaches all correct replica roots for the key with very high probability.

4

What's new in P2P for security?

- Nodes are MUCH more powerful
  - Assign nodeID themselves
  - Act as a router: has routing table, forwarding messages..
- Fully Decentralized
  - no authorization and authentication
  - You can trust nobody
  - Dynamic & self-organizing

5

Typical Routing Model

- Routing
  - Given a key, locate the corresponding node with high probability
- Pastry, Tapestry
  - Internet topology-aware in routing-table
- Chord, CAN
  - Routing-table constrained
- Performance and security assumption: nodeID uniform random distribution

6

Fault model

- Byzantine failure model
- N: number of total nodes
- f: ($0<=f<1$) the fraction of nodes that may be faulty
- cN: ($1/N<=c<=f$) bound size of independent coalitions

7

Network model

- Assumption: no NAT, no DHCP
- Network level and Overlay level
- Adversary has complete control over network-level communication
- Adversaries may delay messages between correct nodes, but we assume that any message will go through a no faulty route in time D with Prob. P_D

8

Possible Attacks and Counter Measures

- Attackes
  - On nodeID assignment
  - On routing maintenance
  - On using routing table to forward messages
- Counter measures
  - Securely nodeId assignment
  - Secure routing table maintenance
  - Secure message forwarding

9

(1) NodeID Assignment Attack

- What if attacker can choose nodeID?
  - Surround a victim node
  - Partition a p2p network
  - become the key holder (root)
- Self ID generation
  - Random (freenet), bad
  - hash IP (chord), bad(?)
- Fundamental assumption: there is a uniform random distribution of nodeIds that cannot be controlled by attacks

10

Solution: Certified NodeIDs

- Move ID generation to trusted CAs
  - Centralized: survival under *Sybil* attack
  - Multiple CAs
  - working offline, open a connection when needed
  - Include network address
  - money or puzzles: prevent attacker get too many nodeIds or too quickly

11

Review CA solutions

- Not a new problem.
- Pros
  - Weaker than those used to verify web sites. Don't have to bind with real-world ID
- Cons
  - Doesn't work with small overlay network
  - Doesn't work with dynamic nodeID (CAN ☹)

12

(2) Attacks on maintaining routing table

- **Fake the closest node**
  - **Intercept probe message, let a near node to reply**
  - **This attack is harder when c is small**
  - **Can be ruled out if bind IP addr to nodeId**
- **Supply faulty routing update**
  - **Faulty info propagate**
  - **(1-f)*f+f*1>f**
  - **Routing algorithm related**
  - **Pastry VS Chord**

13

Solution: Constrained Routing Table

- **Use two routing tables: Pastry+Chord**
  - **First: normal locality-aware Pastry routing table**
    - **Slot(*l*,*d*): share first *l* digits, has value *d* in *l+1* digit**
  - **Second: Constrained Pastry routing table**
    - **Slot(*l*,*d*): closest nodeId to a point p**
    - ***p*: share first *l* digits, has value *d* in *l+1* digit, and has the same remaining digit**
  - **First is efficient, second is for backup**

14

New Algorithms to Initialize and Maintain the routing table

- **Bootstrap nodes**
  - **Use a set of diverse bootstrap nodes**
  - **It's big enough to ensure one is correct**
- **Procedure**
  - **Pick up a set of bootstrap nodes and ask them to route using its nodeId as the key**
  - **No-faulty bootstrap node uses secure forwarding techniques**
  - **Collects all the proposed neighbor set from each of bootstrap nodes, pick the "closed" as its neighbor**
  - **Pick the route entry with minimal delay as the locality-aware routing table**
  - **Initialize each entry of constrained routing table as the live nodeId closest to the desired point p in the id space (secure forwarding)**

15

Review: constraints on routing tables

- Idea: trade complexity (2 routing tables) to both security and performance, but…
- Complexity implies low performance
  - Maintain more routing tables
  - Expensive when building constrained routing table if b>1 unless more complex
- Complexity implies insecurity
  - How to guarantee bootstrap node secure?
- Faulty routing tables still diffuse: (1-f)*f+f*1>f

16

(3) Attack on forwarding

- Simply ignore forwarding msgs
  - Failed if ANY one in routing is faulty
  - Prob. of success routing between two correct nodes: $(1-f)^{h-1}$ (No conspiring needed)
- Root node for a key maybe faulty
  - Prob. of success routing to a root : $(1-f)^h$ (h bounded to O(logN))
  - E.g. f=10%, successful routing=65%

17

Probability of Routing in Pastry

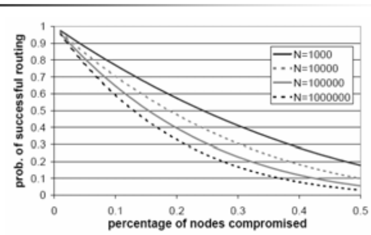


Figure 3: Probability of routing to a correct replica.

18

Solution on forwarding

- The most important part of Secure Routing
- Basic Idea
  - Apply *failure test* to determine if routing worked correctly.
  - If no, use *redundant* and/or *iterative routing.*
- Goal – accomplish in reasonable time/expense

19

Routing failure test

- Takes a key and a set of prospective replica roots for the key
- Return negative if the set of roots is likely to be correct for the key
- Otherwise, return positive
- Timeout to detect ignoring routing msgs

20

Routing failure test

- Observation on average density of nodeID
  - The average density of nodeIds per unit of `volumn` in the id space is greater than the average density of faulty nodeIds
- Basic idea
  - Comparing density of nodeIDs in the neighbor set of the sender VS the density of nodeIDs close to the replica roots of the destination key.

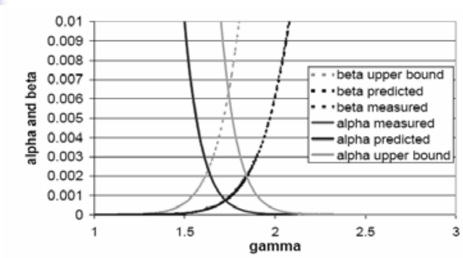
21

Failure test in Pastry

- Density (N live nodes, D=2^128 nodeId space)
  - average numerical distance *u* between consecutive nodeIDs in the neighbor set.
  - Distance between normal consecutive nodeIds
    - Independent exponential random variable with mean D/N,
  - Distance between fault consecutive nodeIds
  - Independent exponential random variable with mean : D/(c*N),
- Failure Test: $u_{rn} < u_p$ * *gamma*
  - u_p: *average numerical distance between consecutive NodeIds in neighbor set of p*
  - u_{np}: *average numerical distance between consecutive nodeIds in rn*
- Fail cases:
  - False positive: alpha<=fun (gamma, n, k)
  - False negative: beta<=fun (gamma, n, k, c)
  - n is the sample used to compute u_p, k is the number of sample to compute u_{np}
  - Independent of *N , provided k<<N*

22

Accuracy of detection



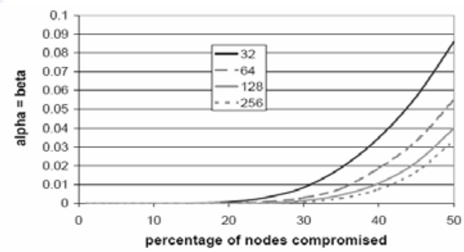
23

Exceptions

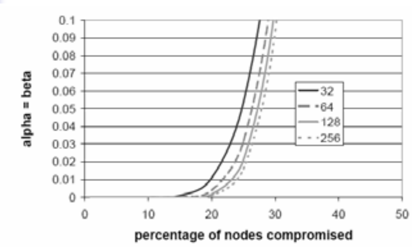
- Collect nodeID certificates that have left the overlay → increase the density
- Mix both faulty and good nodes.
  - Have to contact all prospective replicas
- nodeID suppression attack
  - Idea: make it hard to calculate on density
  - Solution: +alpha, -beta (assume more failure!)

24

Simulation: without suppression



Simulation: with suppression



Tradeoff between alpha and beta

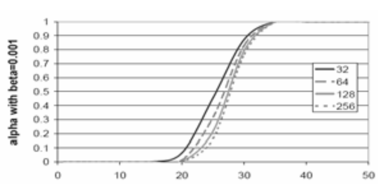
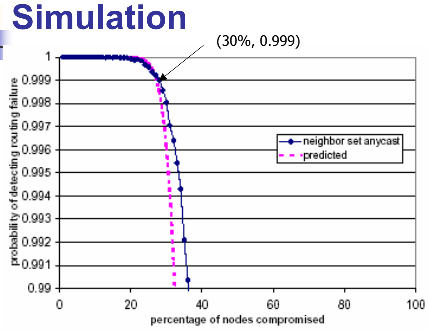


Figure 7: Routing failure test: probability of false positives for a false negative rate of 0.001 with nodeId suppression attacks and varying number of samples.

Redundant routing

- Invoked when failure test is positive
- Route copies of the msgs over multiple routes toward each of the dest key's replica roots.
  - Expectation: at least one copy is correct
- CAN & Tapestry
  - Ask the neighbors to forward the copies of the message to the replica keys.
- Pastry & Chord
  - anycast

Redundant Routing Simulation



Checked iterative routing

- Alternative to redundant routing
- Basic idea
  - The sender starts by looking up the next hop in its routing table and setting a variable n to point to that node
  - Then asks the node for the next hop and update n to point to the returned value
  - The process repeated until reach the destination key
- Expensive than recursive but more secure
- Add *hop tests* to make iterative routing stronger
  - Better for constraint routing table

Review on the solution

- No perfect solution.
- Protocol specific
- Tricky in failure test
  - But subject to more tricky attacks!
- Performance is low
  - Trade off security for improved performance

31

Now we have *Secure Routing*

- Ensure that when a correct node sends a message to a key, the message reaches all correct replica roots for the key with very high probability.
  - Slow, expensive
  - Use common routing as possible as we can

32

When to use secure routing

- Joining
  - Or point to all faulty nodes
- Inserting
  - Or adversary arrange for all replicas
- Reading
  - No. use self-certifying data.

33

Summary

	Pastry
NodeID generation	CA
Routing table maintenance	Act like chord Constrained routing table
Forwarding	Failure test redundant routing Checked Iterative routing

34

Conclusions

- Keep it as simple as possible
- It is a hard problem – no perfect solution now
- Harder when conspiracy
- Have to trust something
  - CAs, bootstrap nodes

35

Discussion: beyond this paper

- Given a nodeID, how to actively detect whether it is malicious or not?
- How to block the faulty nodes?
- Why not trust more?
  - by certification
  - by "credit history" or feedback

36