Security for Peer-to-Peer Networks

Dan S. Wallach, *Rice University*

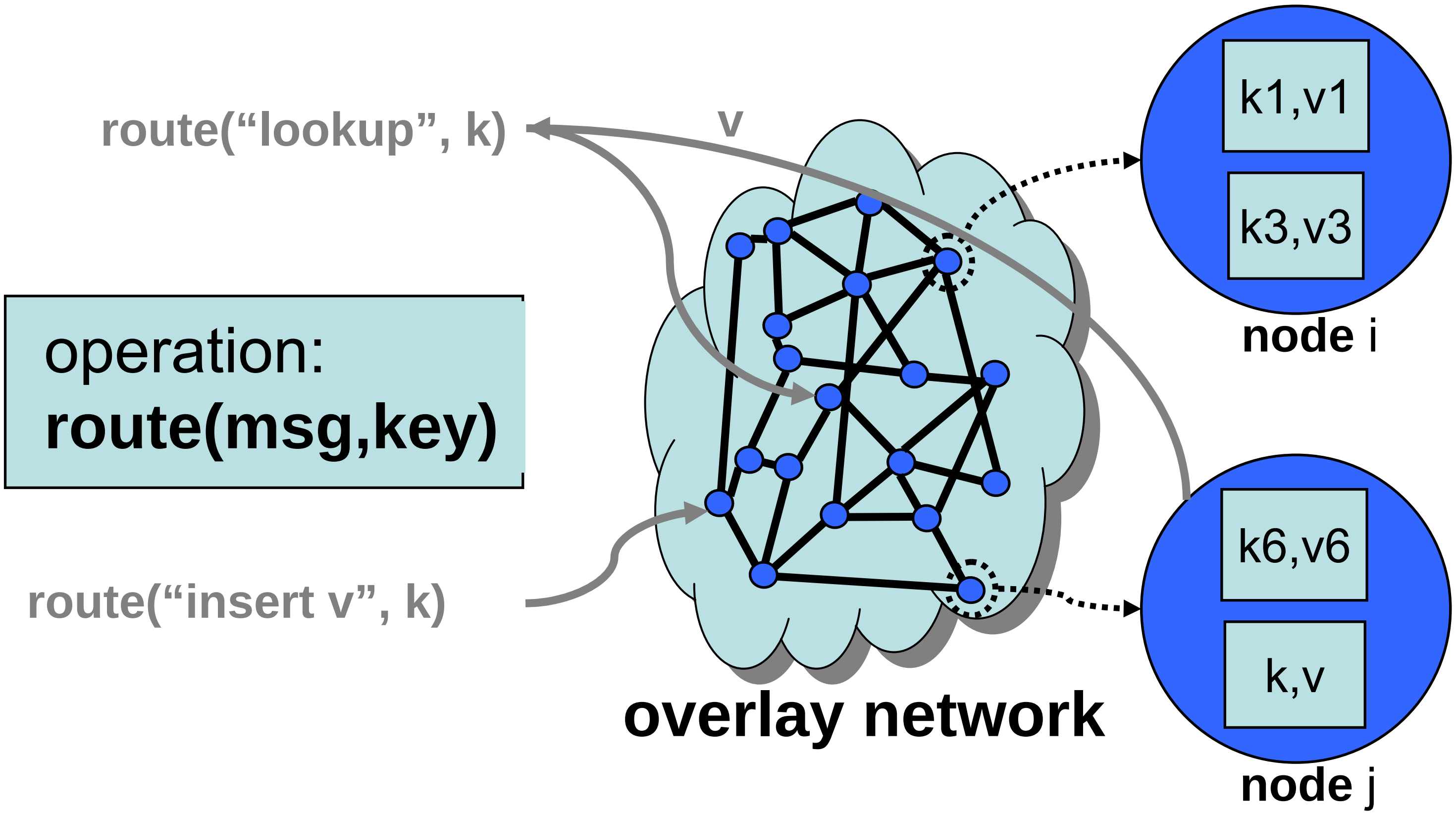
Collaborators:

Scott Crosby, Peter Druschel, Alan Mislove, Animesh Nandi, Seth Nielson, Tsuen Wan Ngan, Ansley Post, Atul Singh, *Rice University / Max Planck Institute*

Antony Rowstron, Miguel Castro, Ayalvadi Ganesh

Microsoft Research Cambridge, UK

Structured p2p overlay networks



- structured **overlay network maps keys to nodes**
- **routes messages to keys**; can implement hash table

[CAN, Chord, Kademlia, Pastry, Skipnets, Tapestry, Viceroy]

Why structured overlays?

- scalable
  - route in O(log *N*) hops with O(log *N*) node state
  - balance routing and key management load
- self-organizing
  - fix overlay when nodes join or leave
  - redistribute load when nodes join or leave
  - completely decentralized with no administrators

Good substrate for distributed applications

Problem 1: attacks on routing

- some overlay nodes are likely to be malicious
  - large scale
  - distributed open environment
  - no special administration

- malicious nodes can attack routing
  - corrupt messages, stored data, and services
  - **drop messages**
  - **misroute messages**

Problem 2: fair-sharing of resources

- Why should node *A* do work on behalf of node *B*?
  - *Tragedy of the commons*
    - Why contribute resources if it’s not necessary?
  - Example: Most Gnutella users do not contribute disk space to the network
  - BitTorrent exactly addresses this problem!

In this talk

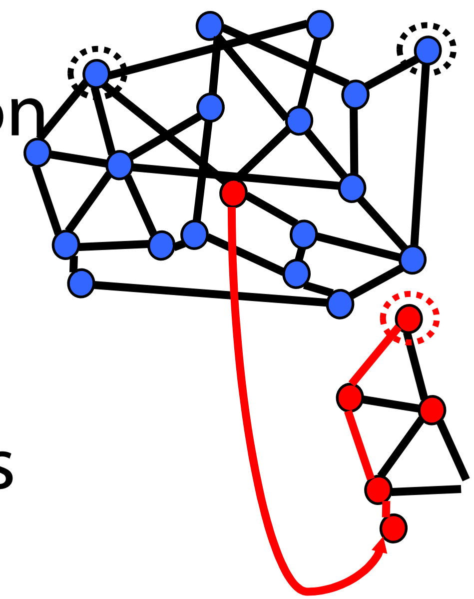
- Routing security
  - Improve robustness of p2p primitives
  - Tollerate some fraction of malicious nodes

The next talk

- Application-level fairness
  - Auditing mechanisms that enforce fairness
    - Economic incentives to participate correctly

Traditional security ideas?

- Integrity and authenticity guarantees
  - self-certifying data and services
  - Byzantine fault tolerant replication



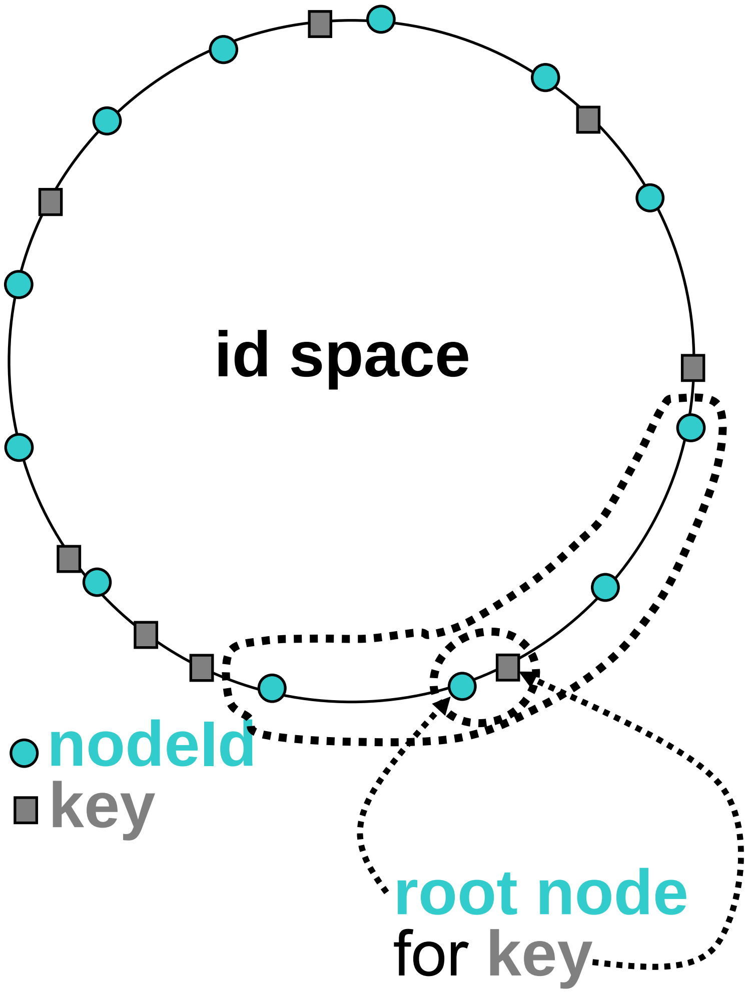
- Denial-of-service
  - easy to detect dropped messages
  - **hard to detect misrouting**
    - sender does not know message destination
    - overlay structure determines message destination
    - attacker can misroute to credible destination

Structured routing example

- Pastry p2p substrate
  [*Rowstron, Druschel ’01*]

Techniques generalize to other p2p systems

Mapping keys to nodes

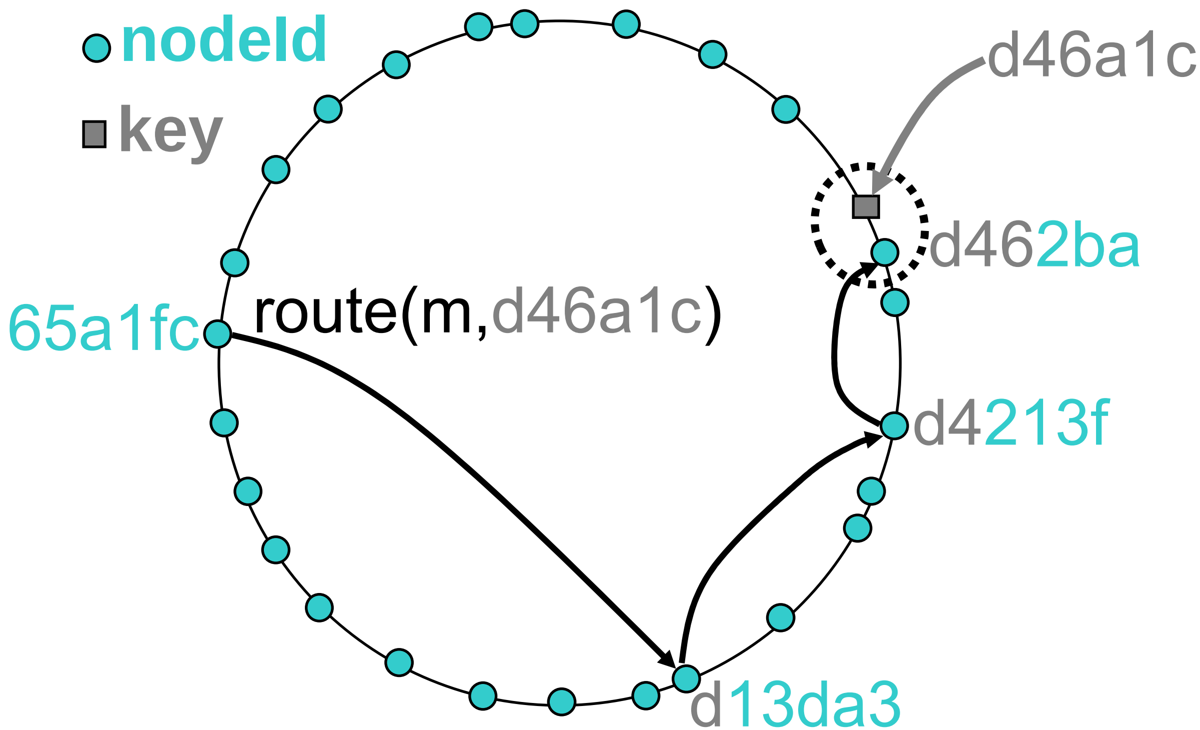


- large **id space** (128 bit integers)
- **nodeIds** picked randomly from space
- **keys** picked randomly from space
- key is managed by its **root node**:
  - live node with id closest to the key
- key is replicated by its **replica roots**:
  - ***r*** nodes with ids closest to key

Node routing state

- ids and keys are 128-bit numbers in base 2^{b}
  - typically, b=4 (hexadecimal, base 16)
- **topology aware routing table**
  - matrix with 128/4 rows and 16 columns
  - entry in row *i* and column *j* contains a
    - nodeId that matches current nodeId in first *i* digits
    - and has value *j* in the next digit
    - id is among the closest in underlying network
- **neighbor set:** *L*/2 closest ids left and right
  - typically, *L*=16 or *L*=32

Pastry: routing

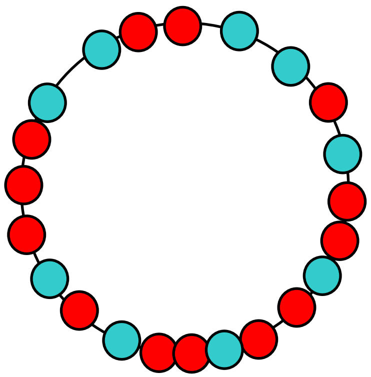


- prefix matching: each hop resolves extra key digit
- neighbor set used to find root node in last hop
- properties: $\log_{16} N$ hops with low delay routes

Secure routing

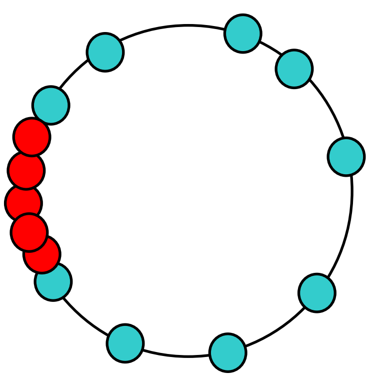
- *sec-route(m,k,r):*
  - delivers message *m* to all the correct replica roots of key *k* with high probability
  - *r* is the number of replica roots
- assumed security model
  - Byzantine faults: arbitrary behavior
  - bound *f* on fraction of faulty overlay nodes

Attacks on nodeId assignment



attacker can obtain many nodeIds

- control arbitrary fraction *f*
  - a.k.a. Sybil attacks [*Doceur '02*]



attacker can pick ids closest to a key

- control all replica roots (targeted attack)
- break Pastry invariant on neighbor sets

Secure nodeId assignment

- certified nodeIds
- trusted certification authorities
  - assign random nodeIds
  - certificates binding id with node public key
  - charge money for certificates or check identities
- nodes in small overlays must be trusted

distributed assignment has fundamental weakness

Routing table maintenance

- routing table maintenance should ensure:
  - If attacker controls nodes with probability *f*,
  - entries in routing tables are bad with probability *f*

- attacks on routing table maintenance
  - malicious seed nodes for joining
  - bad routing updates
    - exploit locality to bias choice of routing entries
    - exploit flexibility to bias choice of routing entries

Routing updates on Pastry

- source of update correct with prob. $1 - f$
  - bad routing entry in update with prob. f
- source of update malicious with prob. f
  - bad routing entry in update with prob. 1
- without strong, verifiable constraints on entries
  - updated entry is faulty, prob. $f(1 - f) + f > f$
  - fraction of bad entries grows over time

Locality vs. security

- Flexibility to choose routing table entries
  - Example: Pastry and Tapestry
  - Low delay routes
  - Vulnerable to previous attack
- Constrained routing table entry choice
  - Example: Chord
  - High delay
  - More secure

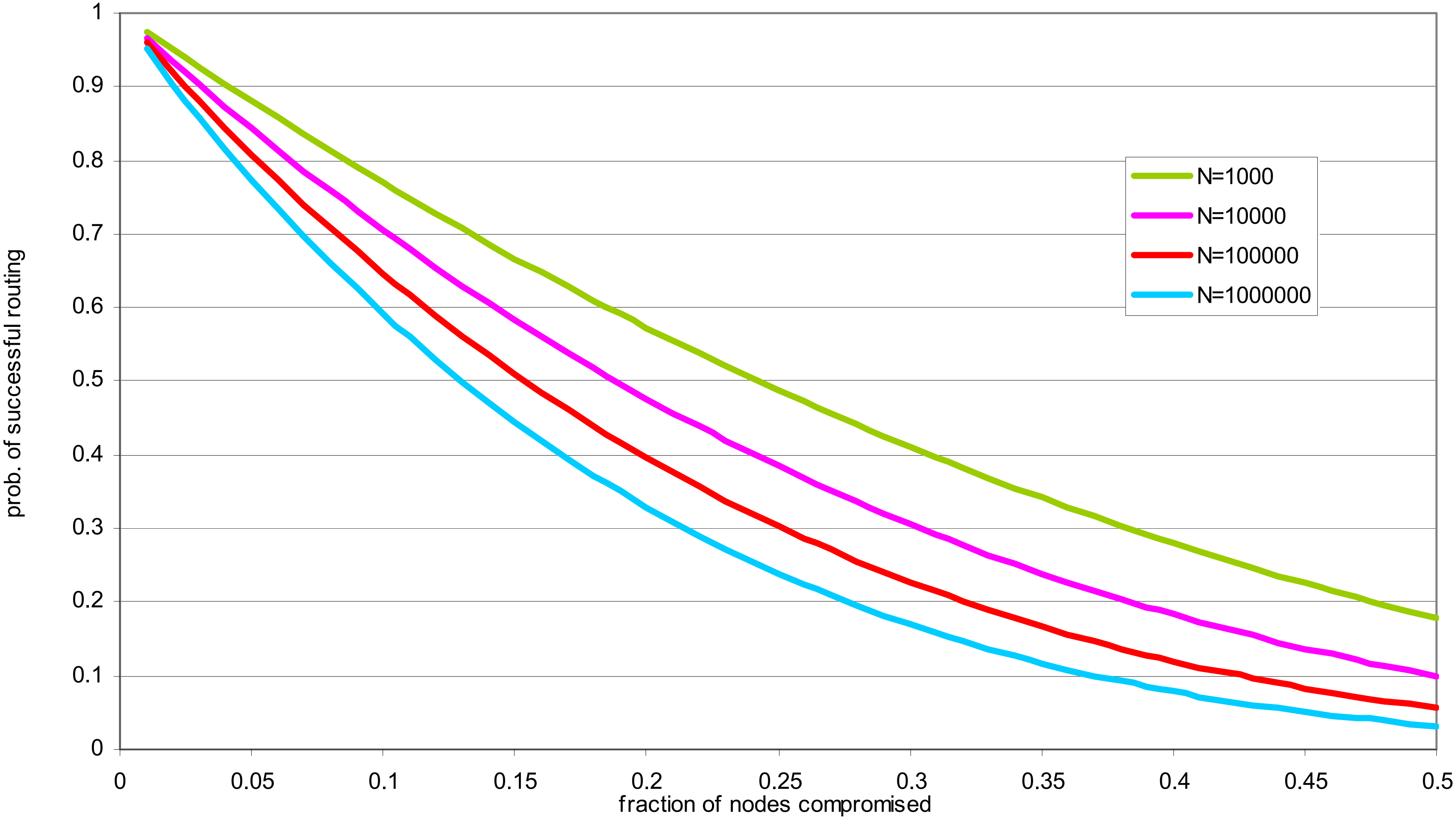
Secure routing tables

- two routing tables: locality aware and
- constrained routing table
  - strong, verifiable constraints on routing entries
  - each entry has live nodeId closest to point in id space
  - attacker controls nodeId closest to point with prob. *f*
  - entries bad with probability *f* (with certified nodeIds)
- node joining
  - secure routing from multiple seed nodes
  - obtain neighbor set with high probability
  - build constrained routing table from neighbors' tables

Attacks on forwarding

- attacker
  - controls fraction f of nodes
  - controls fraction f of routing entries
  - can drop or misroute messages
- probability of routing correctly drops fast
  - when number of hops increases
    - Larger p2p ring → more hops to destination
  - when fraction of compromised nodes f increases

Probability of routing correctly

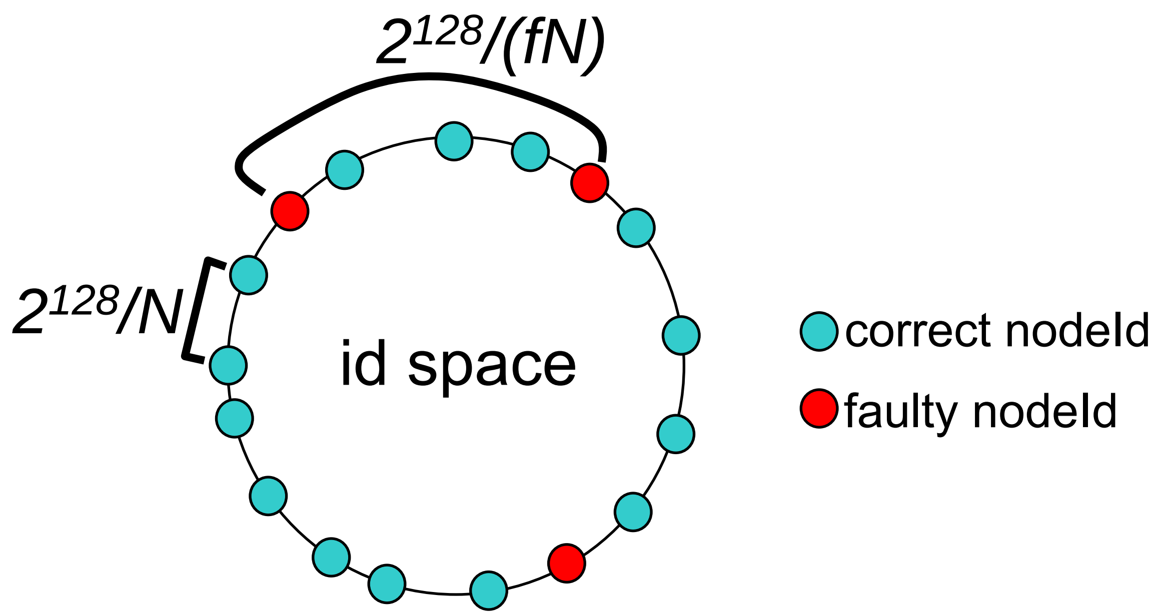


Secure forwarding

- route efficiently with topology aware routing
- run routing failure test
  - if no failure, done
- use redundant routing with constrained table

Routing failure test: idea

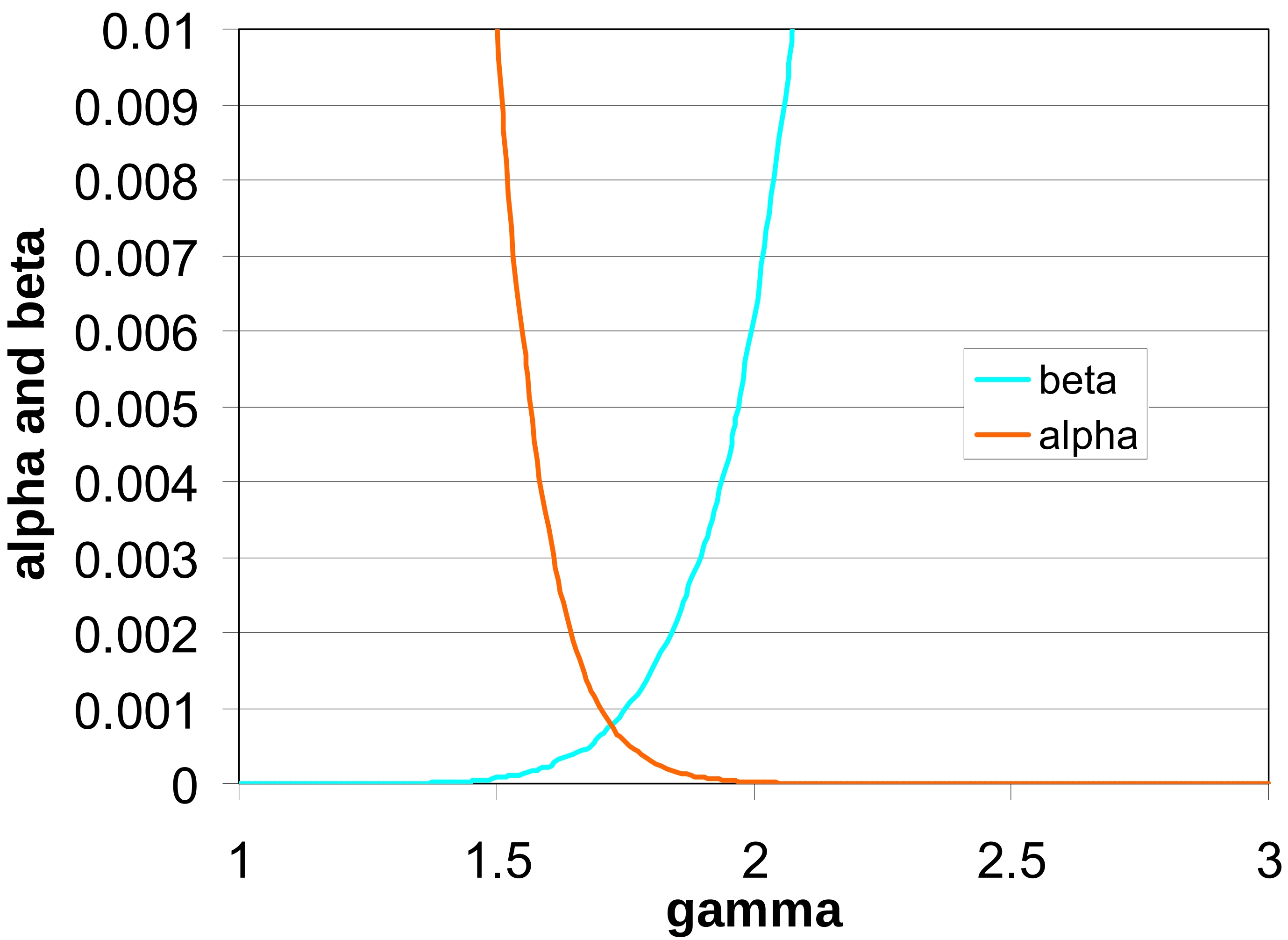
- density of faulty nodeIds is lower
  - average distance between nodeIds: $2^{128} / N$
  - average distance between faulty nodeIds: $2^{128} / (f\,N)$



Routing failure test: how it works

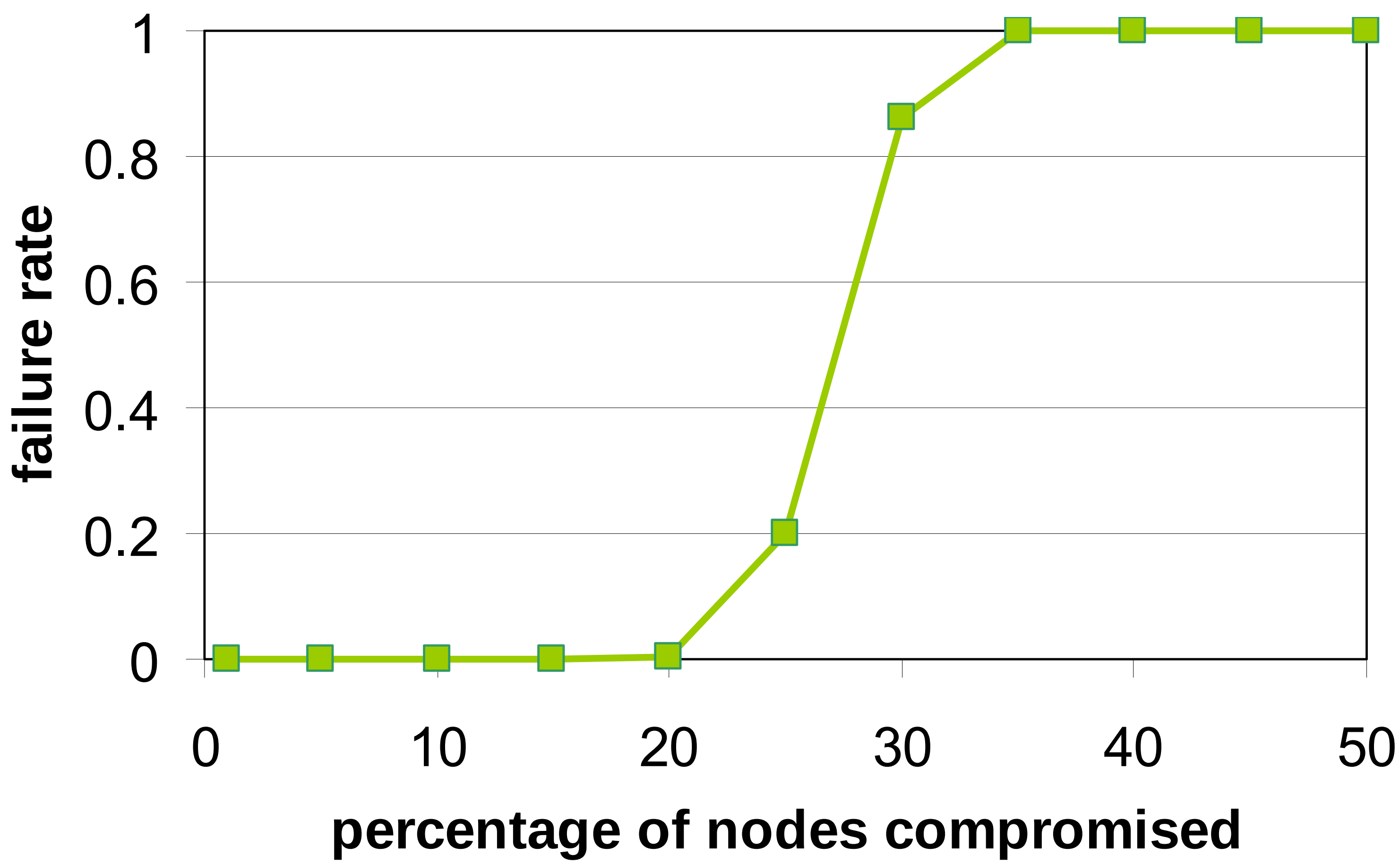
- route efficiently and get neighbor set
- compute average:
  - distance between ids in sender's neighbor set: μ_s
  - distance between ids in receiver's neighbor set: μ_R
- if $\mu_R > \mu_s \times \gamma$, signal failure
- otherwise, signal success

Routing failure test: performance



false positive rate: alpha; false negative rate: beta

Routing failure test: performance



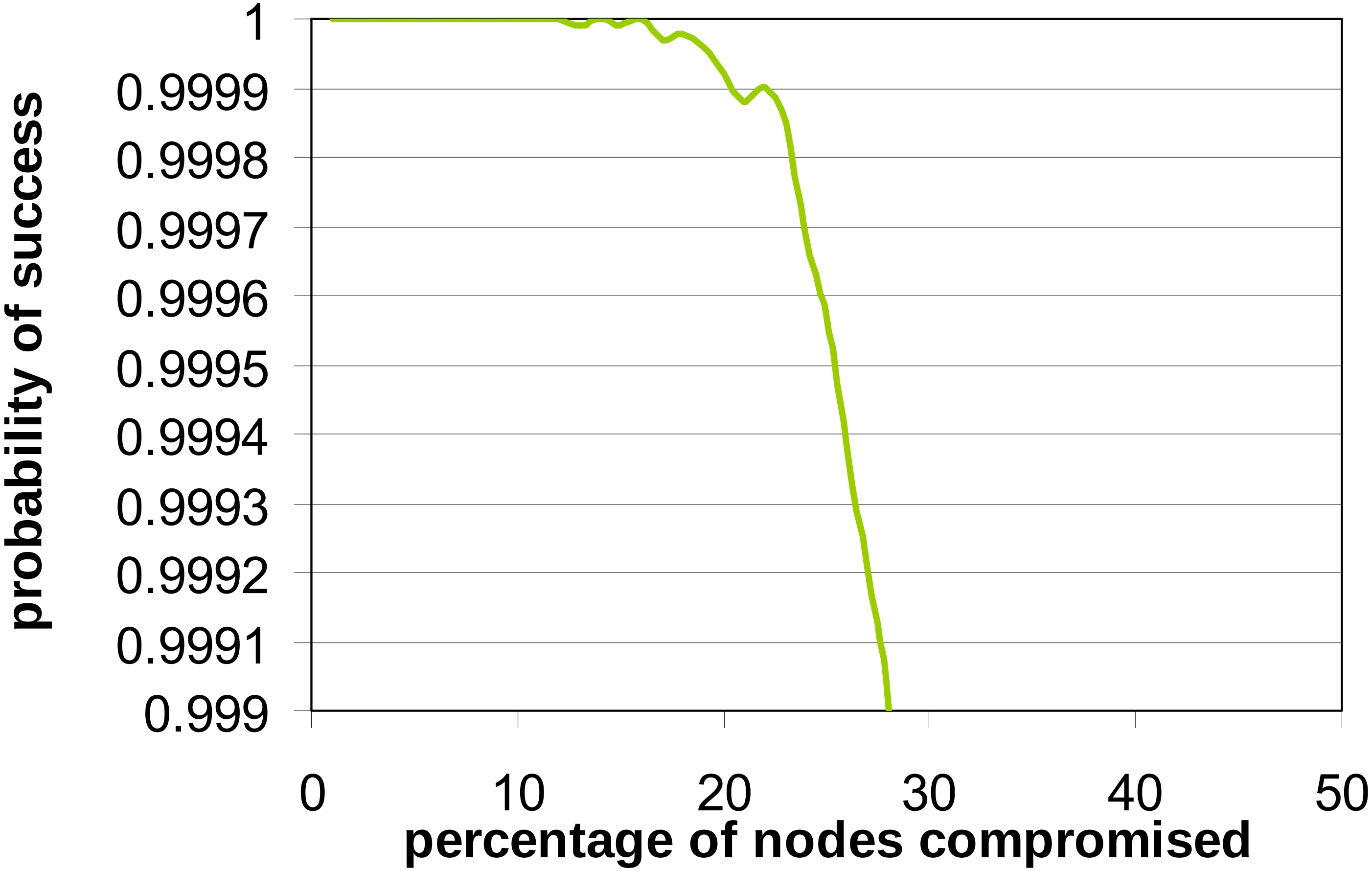
Routing failure test: attacks

- Attacker can fool test by
  1. using nodeIds of stopped correct nodes
  2. mixing nodeIds of correct and incorrect nodes
  3. suppressing faulty nodeIds
     - near sender increases α; near receiver increases β
- Solution for 1 and 2
  - talk with nodeId owners before running test
    - query/validate all nodes in a neighbor set
    - no solution for 3: reduced test accuracy

Redundant routing

- Use redundancy when routing test fails
  - send messages over diverse routes to key *k*
    - route messages through neighbors
  - neighbor set anycast
    - avoid early convergence on *k*'s root
    - delivery to first node in route with key *k* in neighbor set
  - collect neighbor set proposals
  - wait for all replies or a timeout
  - pick *r* nodeIds closest to key *k* as its replica roots

Redundant routing: performance



probability of success greater than 0.999 if *f* < 0.25

Secure routing summary

- Vulnerabilities when nodes are malicious
  - Message forwarding
  - Route updates
  - Randomness assumptions of p2p primitives
- Techniques to increase reliability
  - Certified nodeId assignment
  - Redundant routing / neighbor set density checking
  - Constrained routing (trade-off locality vs. robustness)