Peer-to-Peer Systems and Security

Attacks!

Christian Grothoff

Technische Universität München

April 13, 2013

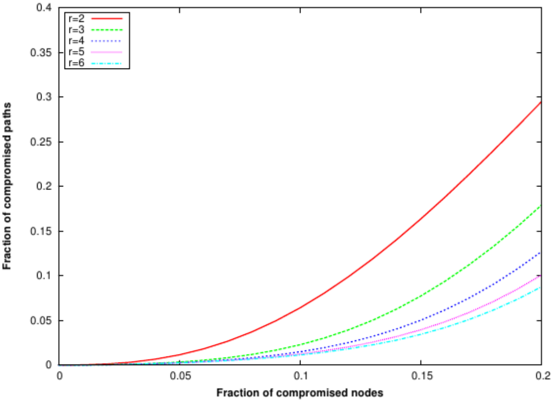
Salsa & AP3

- Goal: eliminate trusted blender server
- Idea: Use DHT (AP3: Pastry, Salsa: custom DHT) to find peers
- Sybil defense with trusted authority (AP3) or IP-based hash (Salsa)

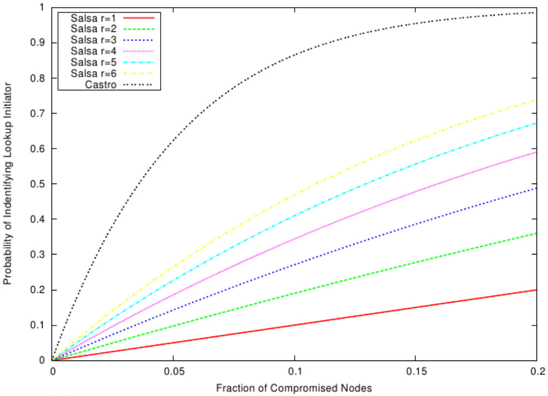
Attacks on Salsa & AP3 [2]

- Passive attack: detect lookup, then correlate with path construction later
- Active attack: return malicious peers during lookup

Defenses against Active Attack: Redundant lookups

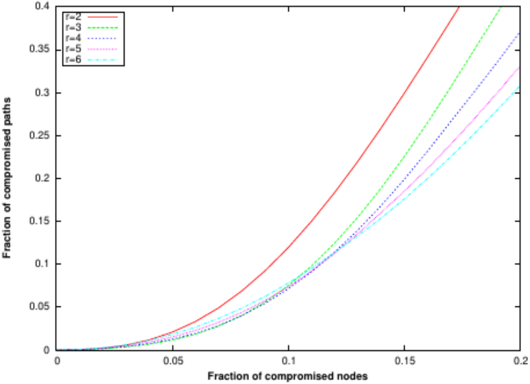


Defenses against Passive Attack: Minimize lookup footprint



(a) Information leak from secure lookups

Defending against the Combined Attack



$r = 3$ is optional for $f < 0.1$,then $r = 6$ becomes optimal.

An Attack on Tor [1]

Result:

All the Tor nodes involved in a circuit can be discovered, reducing Tor users level of anonymity and revealing a problem with Tor's protocol

An Attack on Tor [1]

Result:

All the Tor nodes involved in a circuit can be discovered, reducing Tor users level of anonymity and revealing a problem with Tor's protocol

Note: this was fixed since the attack was published in 2009.

Key Tor Properties

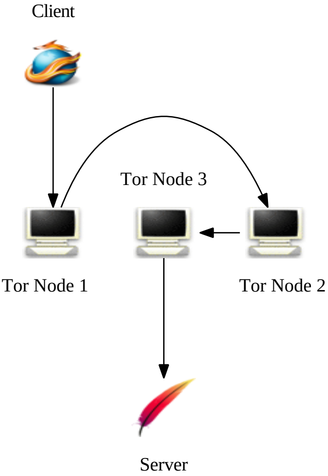
- Data is forwarded through the network
- Each node knows only the previous hop and the next hop
- Only the originator knows all the hops
- Number of hops is hard coded (currently set to three)

Key security goal: No node in the path can discover the full path

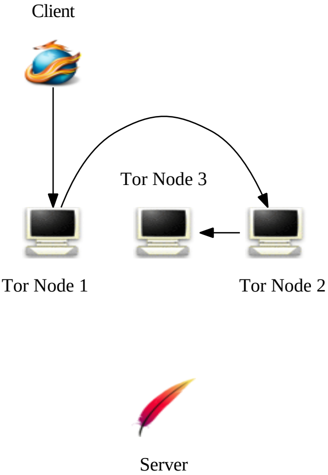
Our Basis for Deanonymization

- Target user is running Tor from 2009 with default settings
- Three design issues enable users to be deanonymized
  1. No artificial delays induced on connections
  2. Path length is set at a small finite number (3)
  3. Paths of arbitrary length through the network can be constructed

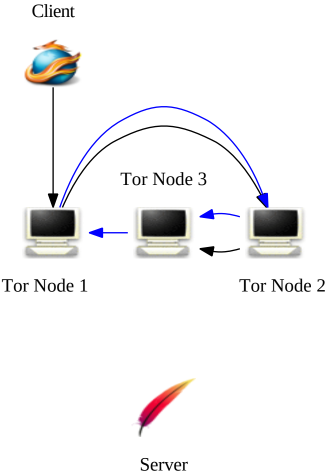
Regular Path Example



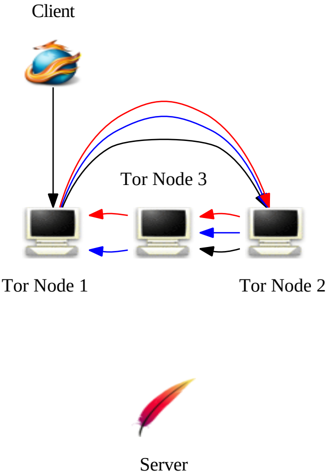
Circular Path Example 1/5



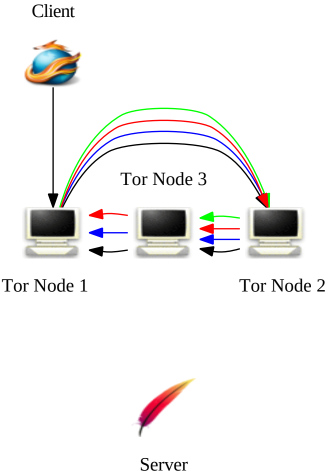
Circular Path Example 2/5



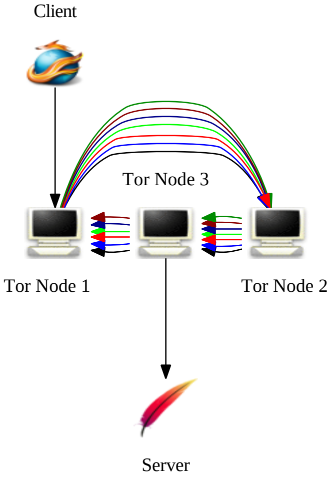
Circular Path Example 3/5



Circular Path Example 4/5



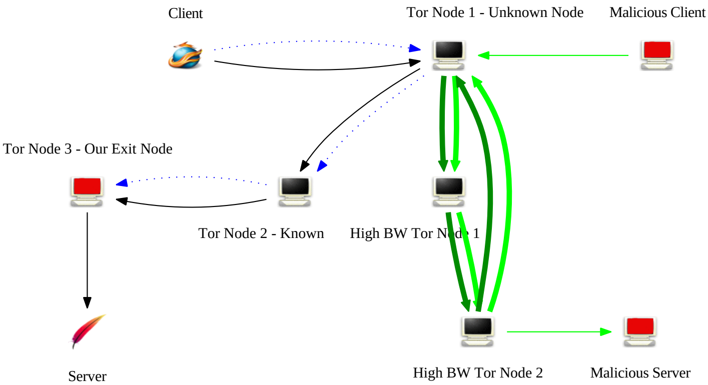
Circular Path Example 5/5



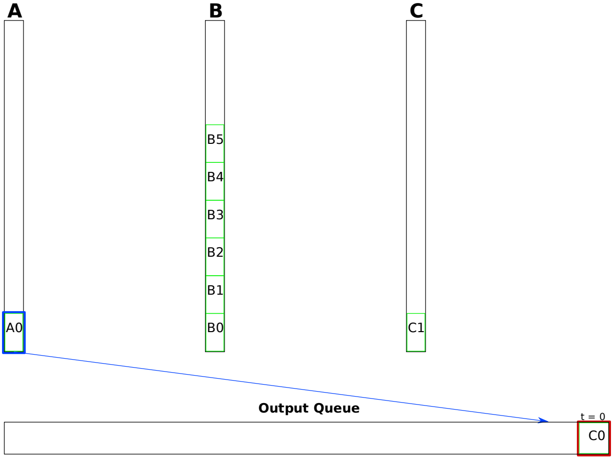
Attack Implementation

- Exit node “injects” JavaScript “ping” code into HTML response
- Client browses as usual, while JavaScript continues to “phone home”
- Exit node measures variance in latency
- While continuing to measure, attack strains possible first hop(s)
- If no significant variance observed, pick another node from candidates and start over
- Once sufficient change is observed in *repeated* measurements, initial node has been found

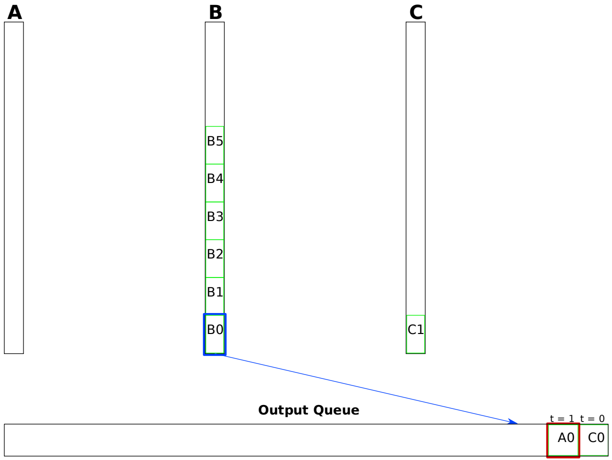
Attack Example



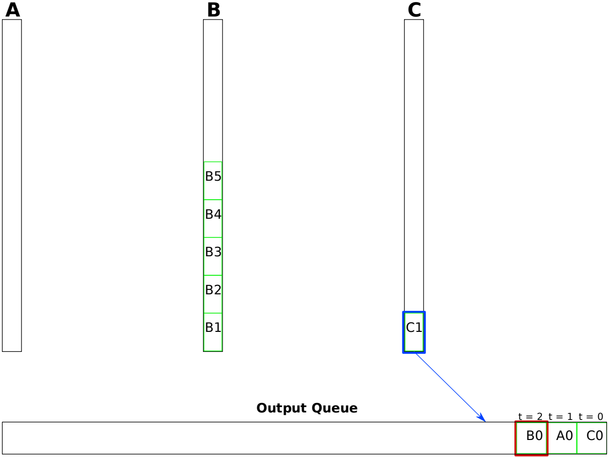
Queue example 1 (3 circuits)



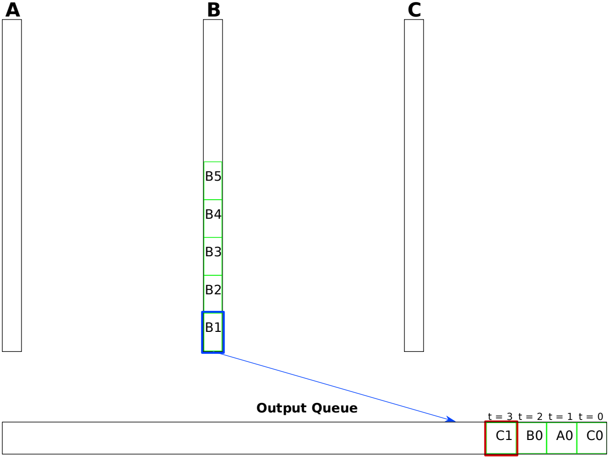
Queue example 2 (3 circuits)



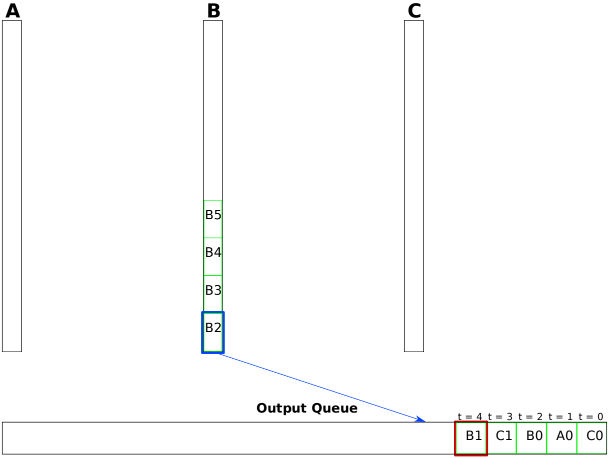
Queue example 3 (3 circuits)



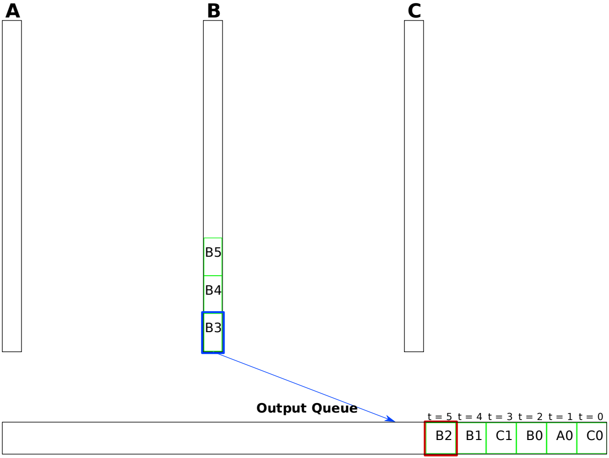
Queue example 4 (3 circuits)



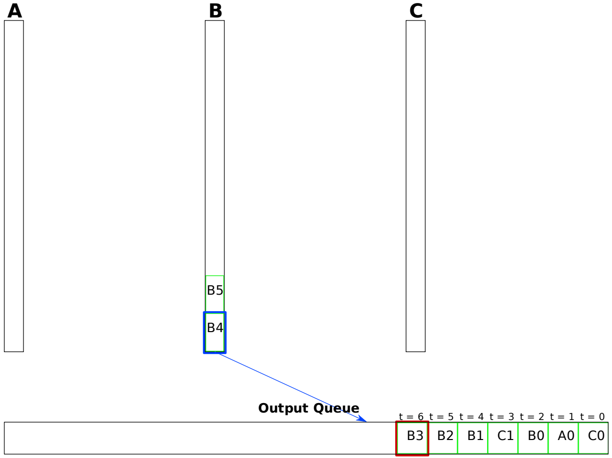
Queue example 5 (3 circuits)



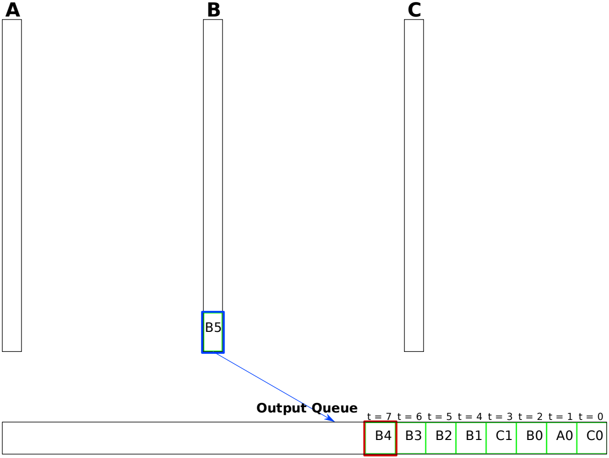
Queue example 6 (3 circuits)



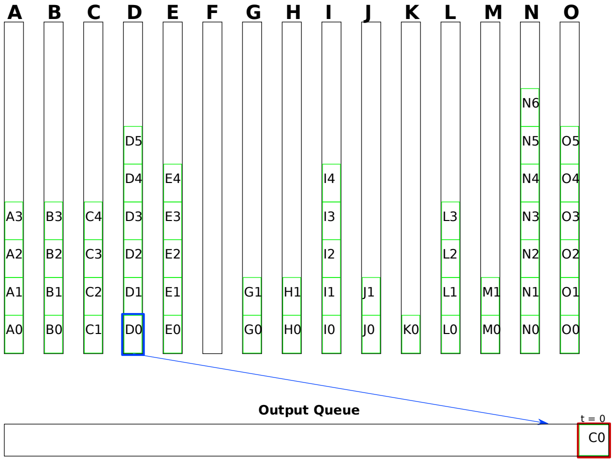
Queue example 7 (3 circuits)



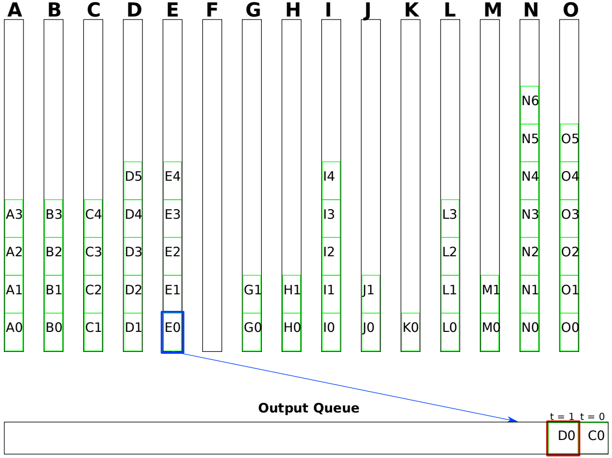
Queue example 8 (3 circuits)



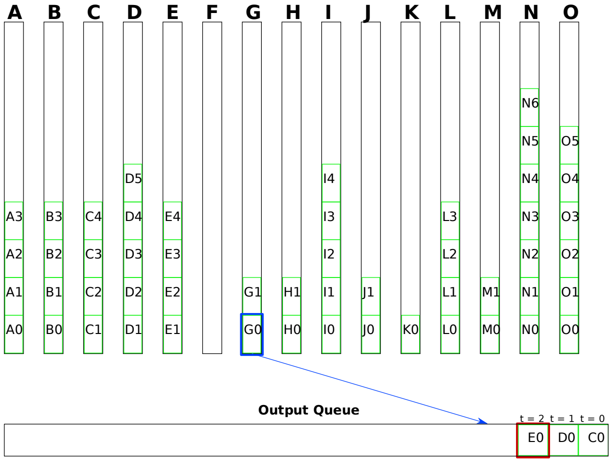
Queue example 1 (15 circuits)



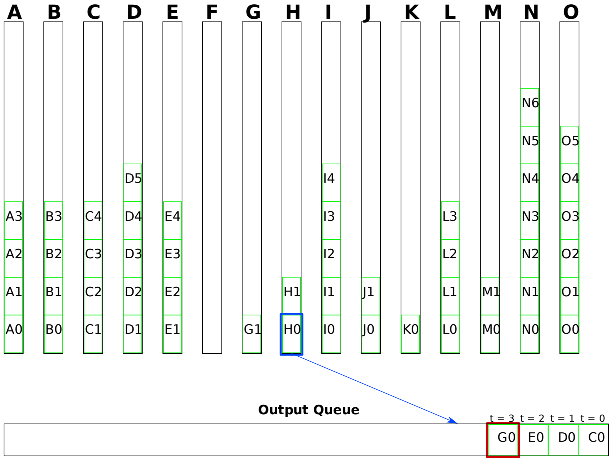
Queue example 2 (15 circuits)



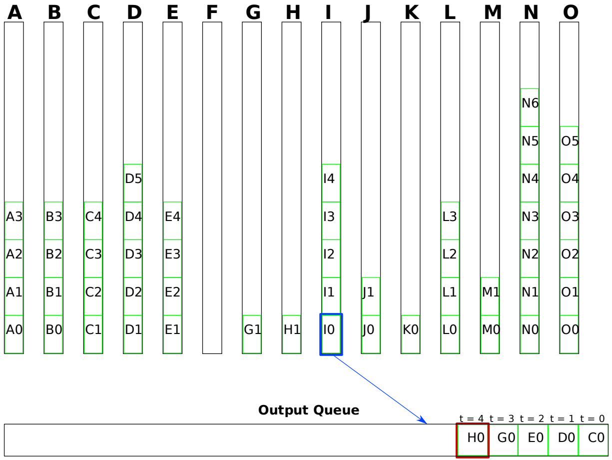
Queue example 3 (15 circuits)



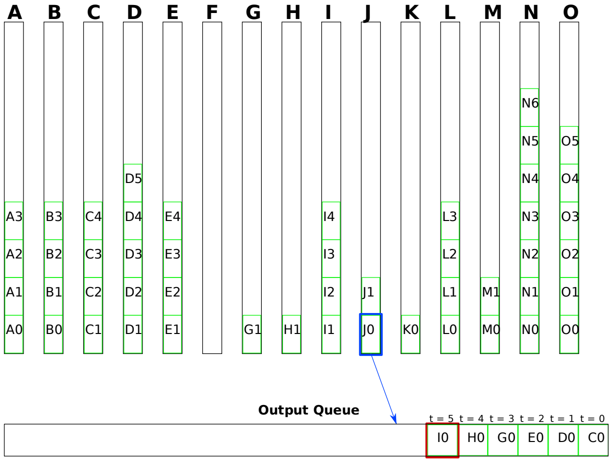
Queue example 4 (15 circuits)



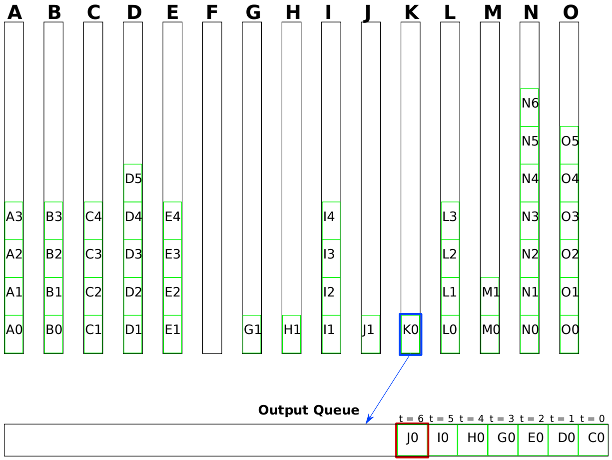
Queue example 5 (15 circuits)



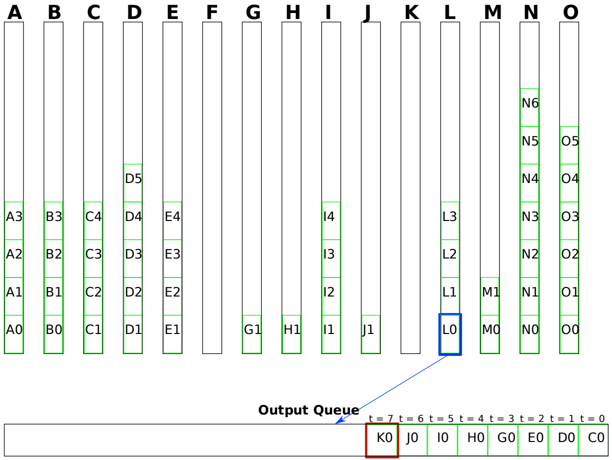
Queue example 6 (15 circuits)



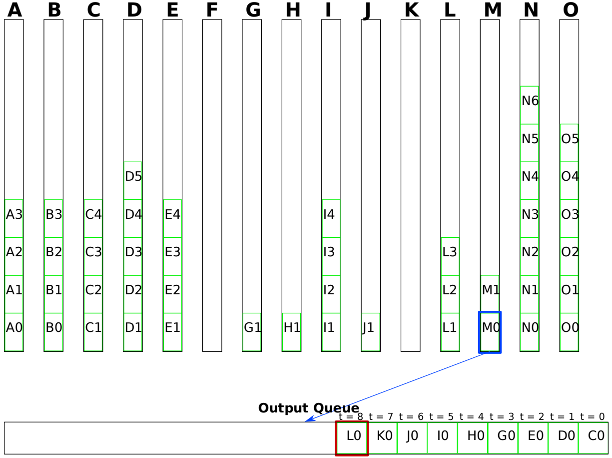
Queue example 7 (15 circuits)



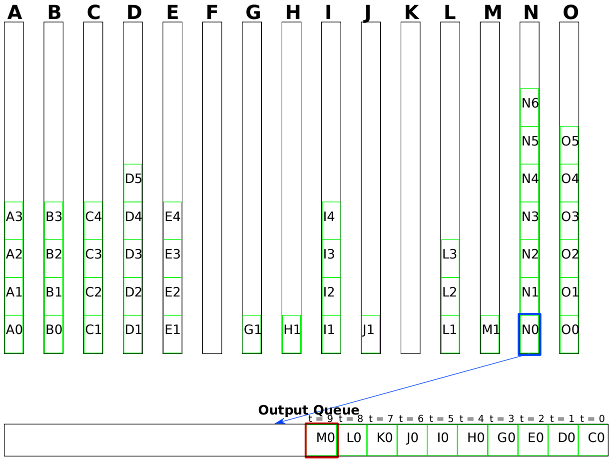
Queue example 8 (15 circuits)



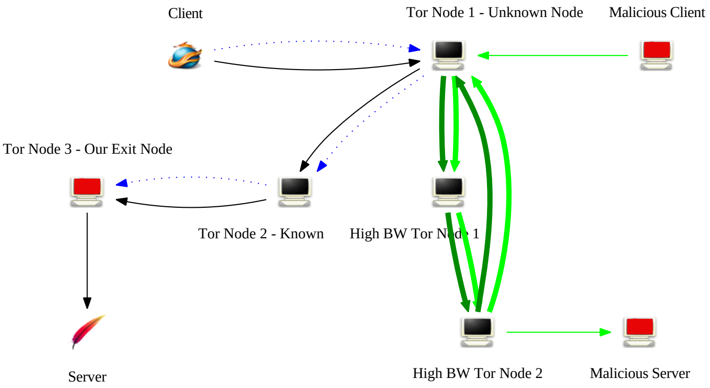
Queue example 9 (15 circuits)



Queue example 10 (15 circuits)



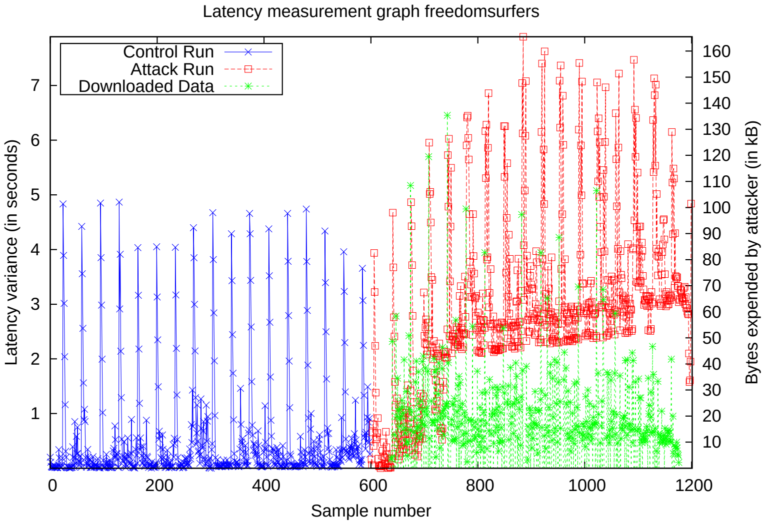
Attack Example



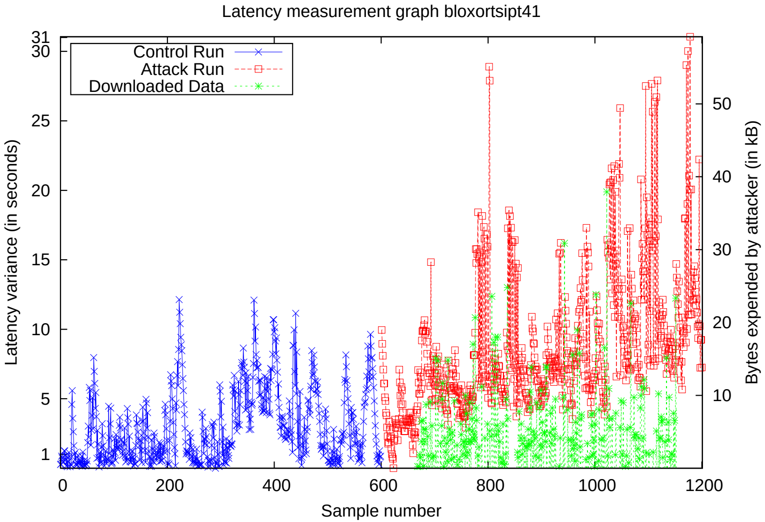
Attack Implementation

- Modified exit node
- Modified malicious client node
- Lightweight malicious web server running on GNU libmicrohttpd
- Client side JavaScript for latency measurements
- Instrumentation client to receive data

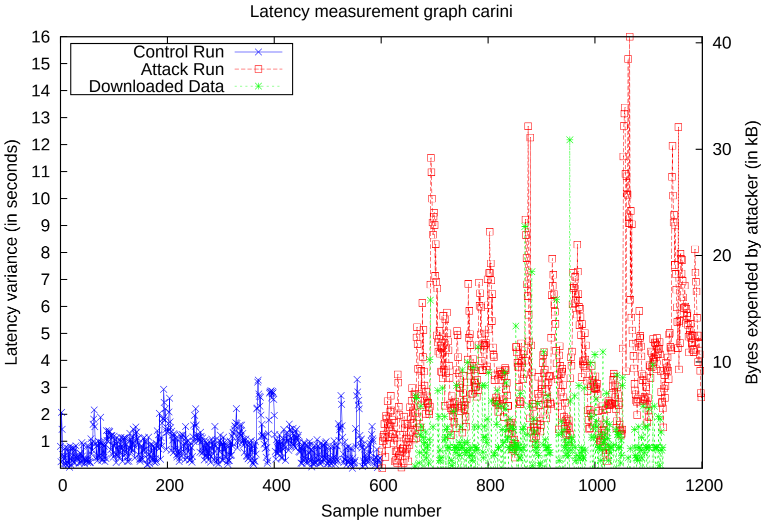
Gathered Data Example (1/8)



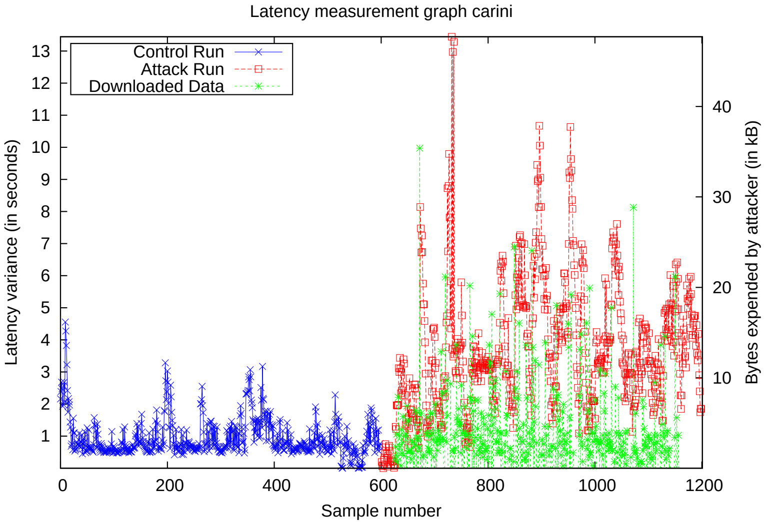
Gathered Data Example (2/8)



Gathered Data Example (3/8)

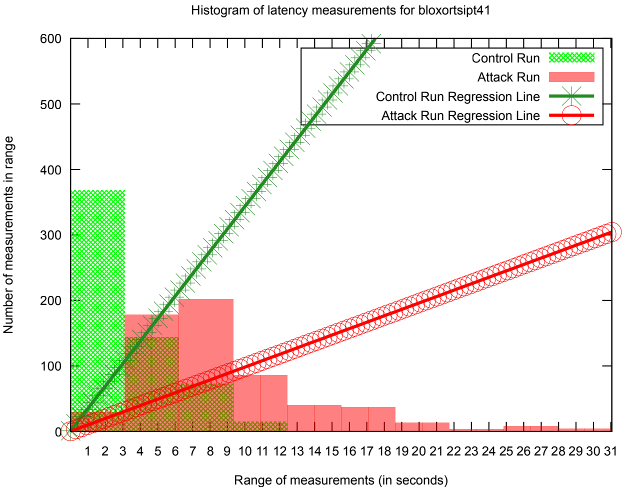


Gathered Data Example (4/8)

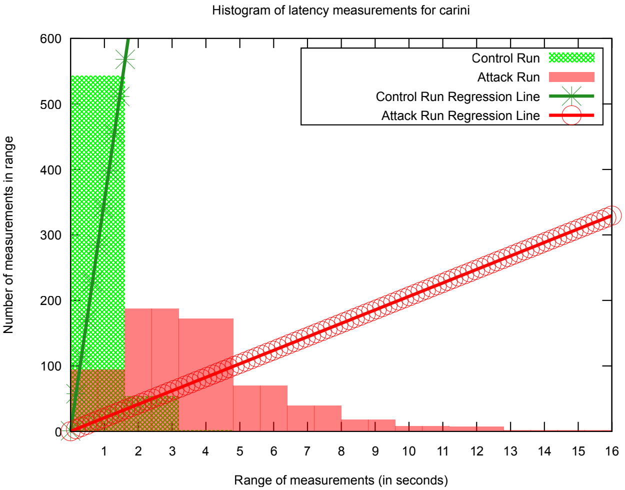


Gathered Data Example (5/8)

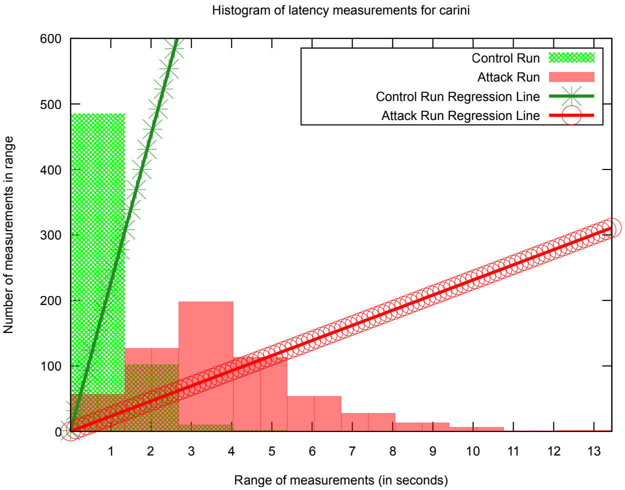
Gathered Data Example (6/8)



Gathered Data Example (7/8)



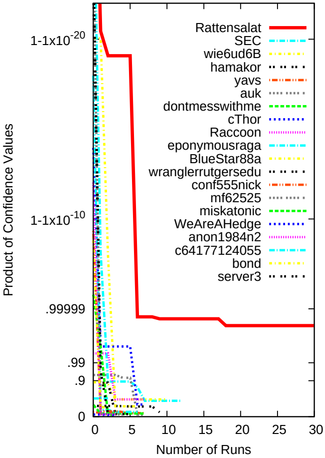
Gathered Data Example (8/8)

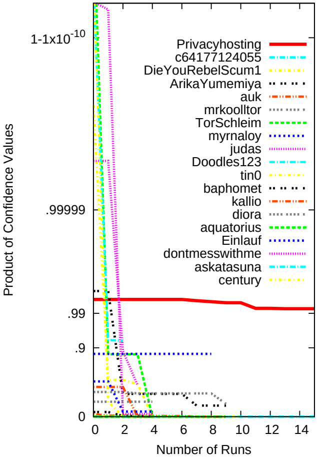


Statistical Analysis

- Use modified χ^2 test
- Compare baseline distribution to attack distribution
- High χ^2 value indicates distribution changed *in the right direction*
- Product of χ^2 confidence values over multiple runs
- Iterate over suspect routers until single node stands out

Cumulative Product of χ^2 p-values





What We Actually Achieve

- We do identify the entire path through the Tor network
- We do achieve this on the 2009 Tor network
- Attack works on routers with differing bandwidths
- This means that if someone were performing this attack from an exit node, Tor becomes as effective as a network of one-hop proxies

Why Our Attack is Effective

- Since we run the exit router, only a single node needs to be found
- Our multiplication of bandwidth technique allows low bandwidth connections to DoS high bandwidth connections (solves common DoS limitation)

Fixes

- Don't use a fixed path length (or at least make it longer)
- Don't allow infinite path lengths (this is fixed in Tor now!)
- Induce delays into connections (probably not going to happen)
- Monitor exit nodes for strange behavior (been done somewhat)
- Disable JavaScript in clients
- Use end-to-end encryption

Attack Improvements/Variants

- Use meta refresh tags for measurements instead of JavaScript
- Parallelize testing (rule out multiple possible first nodes at once)
- Improved latency measures for first hop to further narrow possible first hops

Conclusion

- Initial Tor implementation allowed arbitrary length paths
- Arbitrary path lengths allow latency altering attack
- Latency altering attack allows detection of significant changes in latency
- Significant changes in latency reveal paths used

Motivation

- Efficient fully decentralized routing in restricted-route topologies is important:
  - Friend-to-friend (F2F) networks ("darknets")
  - WiFi ad-hoc and sensor networks
  - Unstructured networks
- Clarke & Sandberg claim to achieve $O(\log n)$ routing in the dark (Freenet 0.7)
- Is this new routing protocol reasonably resistant against attacks?

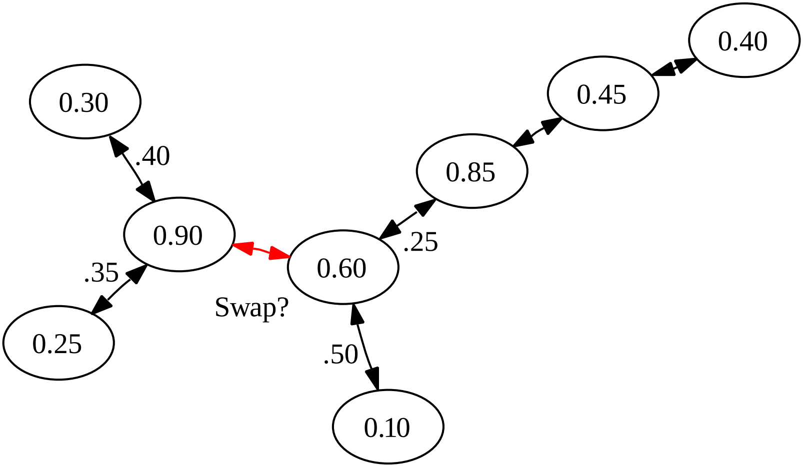
Freenet 101

- Freenet is a 'anonymous' peer-to-peer network
- Overlay based on cyclic address space of size 2^{32}
- Nodes have a constant set of connections (F2F)
- All data identified by a key (modulo 2^{32})
- Data assumed to be stored at closest node
- Routing uses depth-first traversal in order of proximity to key

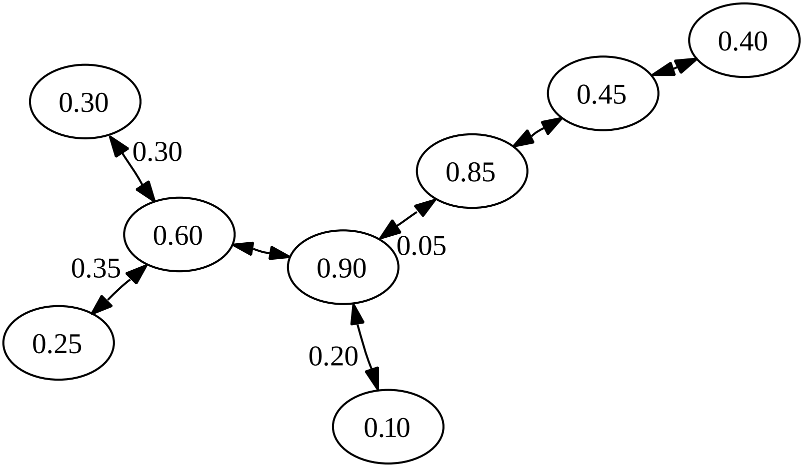
Routing in the Dark

- Small world network assumption
  - Sparsely connected graph
  - There exists a short path (O(log N)) between any pair of nodes
  - Common real world phenomenon (Milgram, Watts & Strogatz)
- Freenet's routing algorithm attempts to find short paths
  - Uses locations of nodes to determine proximity to target
  - Uses swapping of locations to structure topology

Swap Example



Result of Swap



Location Swapping

- Nodes swap locations to improve routing performance
- Each connected pair of nodes (a, b) computes:

$$P_{a,b} := \frac{\prod_{(a,o)\in E} |L_a - L_o| \cdot \prod_{(b,p)\in E} |L_b - L_p|}{\prod_{(a,o)\in E} |L_b - L_o| \cdot \prod_{(b,p)\in E} |L_a - L_p|} \tag{1}$$

- If $P_{a,b} \geq 1$ the nodes swap locations
- Otherwise they swap with probability $P_{a,b}$

Routing of GET Requests

GET requests are routed based on peer locations and key:

1. Client initiates GET request
2. Request routed to neighbor with closest location to key
3. If data not found, request is forwarded to neighbors in order of proximity to the key
4. Forwarding stops when data found, hops-to-live reaches zero or identical request was recently forwarded (to avoid circular routing)

$\Rightarrow$ Depth-first routing in order of proximity to key.

GET 1/7

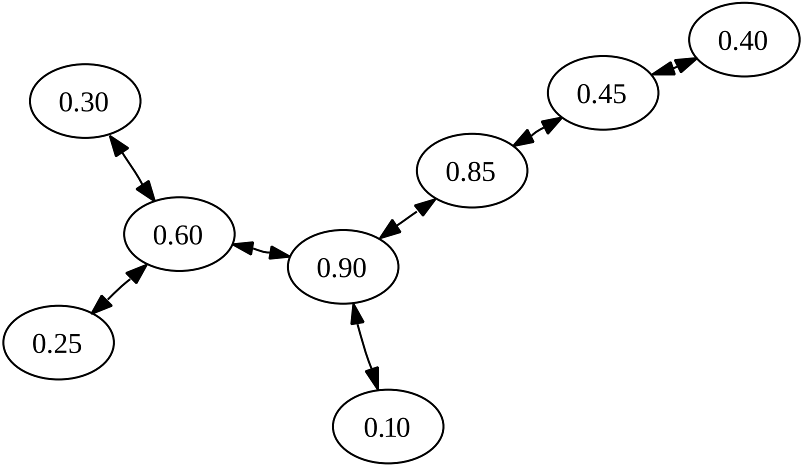


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

GET 2/7

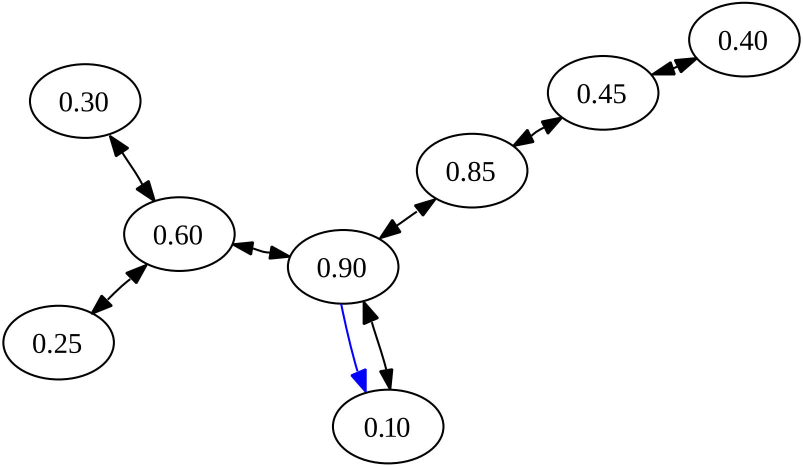


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

GET 3/7

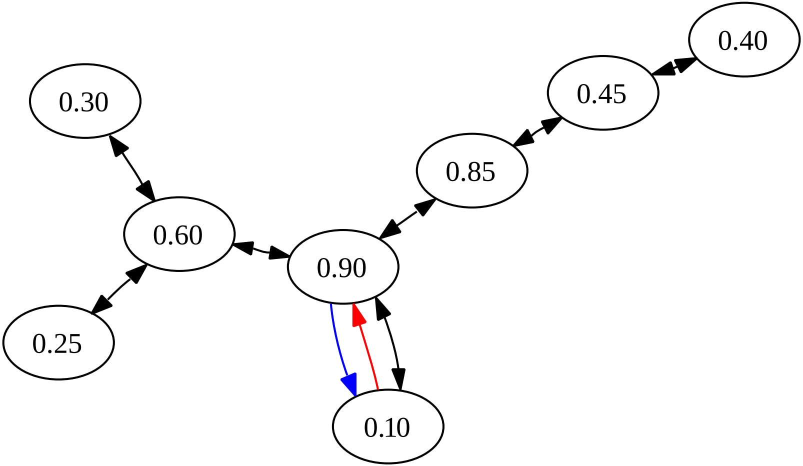


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

GET 4/7

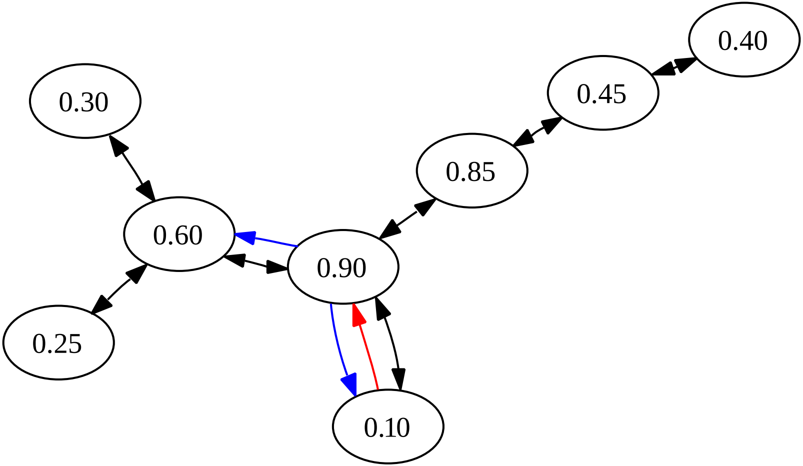


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

GET 5/7

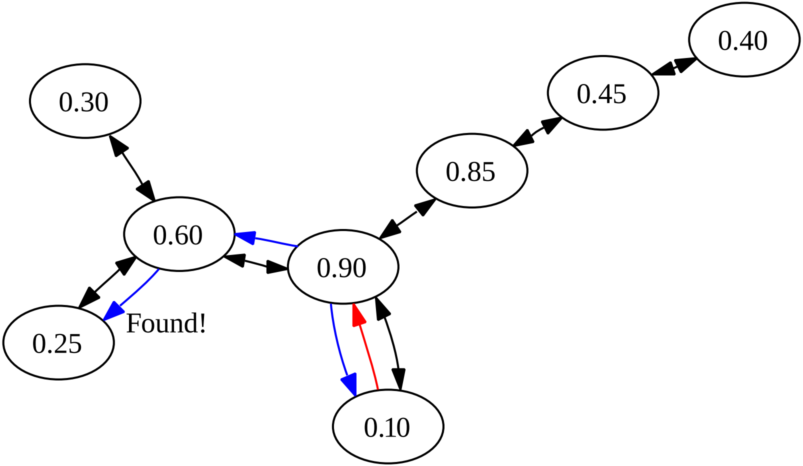


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

GET 6/7

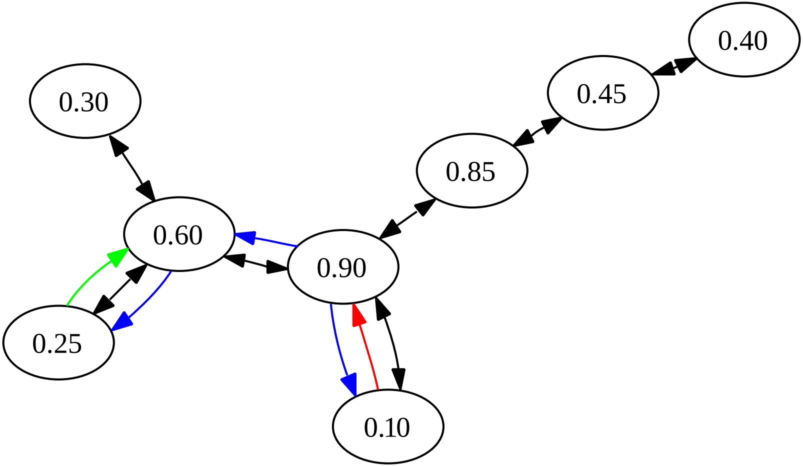


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

GET 7/7

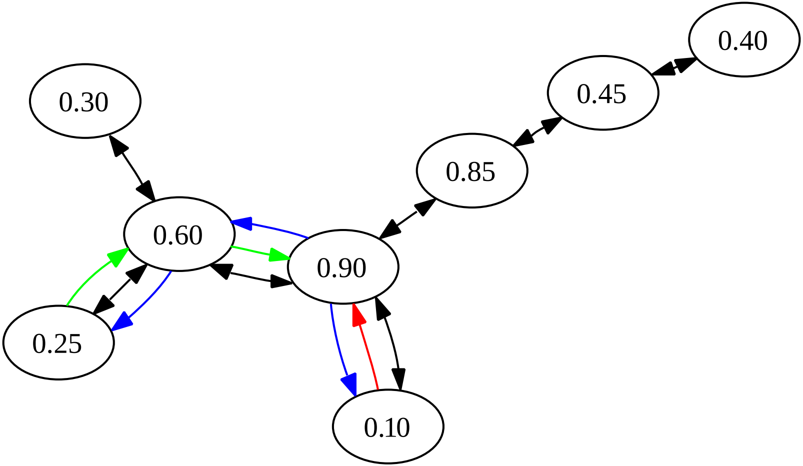


Figure: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

PUT Requests

PUT requests are routed the same as GET requests:

1. Client initiates PUT requests
2. Request routed to neighbor closest to the key
3. If receiver has any peer whose location is closer to the key, request is forwarded
4. If not, the node resets the hops-to-live to the maximum and sends the put request to all of its' neighbors
5. Routing continues until hops-to-live reaches zero (or node has seen request already)

Put Example

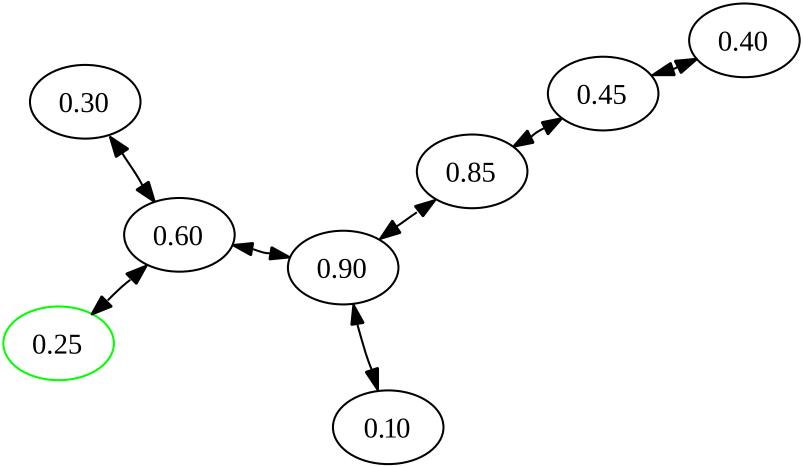


Figure: Put example from node with ID 0.25 inserting data identified by the ID 0.93

Put Example 1/3

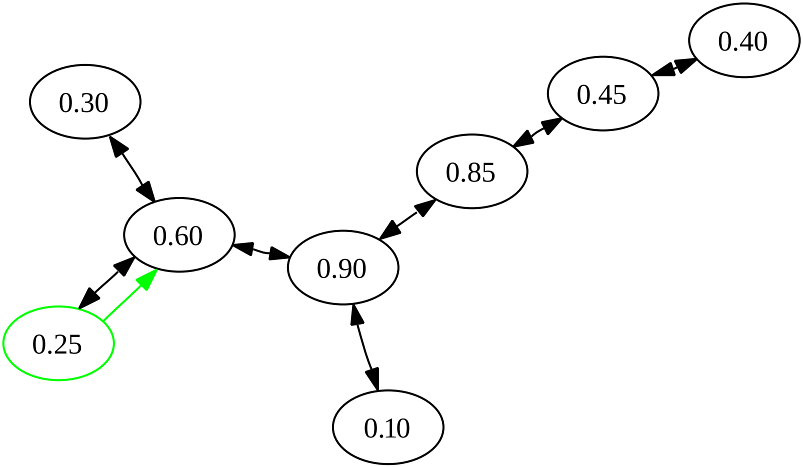


Figure: Put example from node with ID 0.25 inserting data identified by the ID 0.93

Put Example 2/3

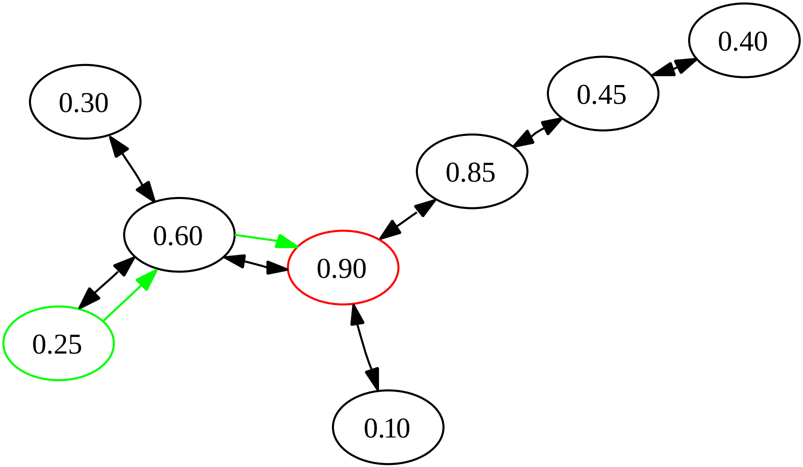


Figure: Put example from node with ID 0.25 inserting data identified by the ID 0.93

Put Example 3/3

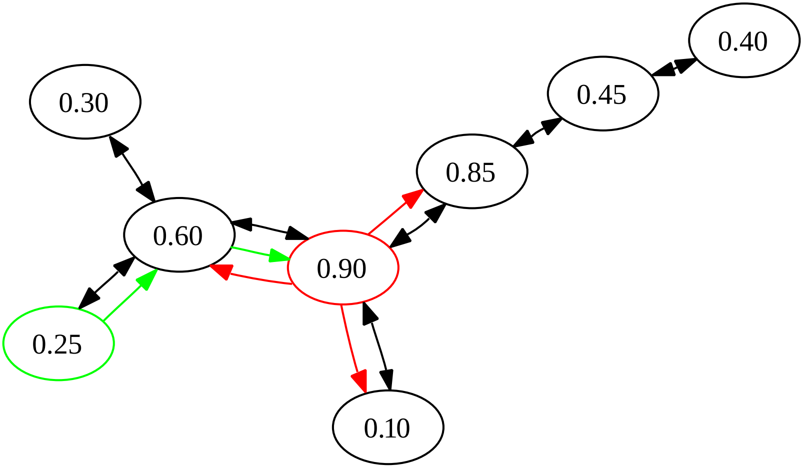


Figure: Put example from node with ID 0.25 inserting data identified by the ID 0.93

Basic Idea for the Attack

- Freenet relies on a balanced distribution of node locations for data storage
- Reducing the spread of locations causes imbalance in storage responsibilities
- Peers cannot verify locations in swap protocol, including location(s) they may receive

$\Rightarrow$ use swap protocol to reduce spread of locations!

Attack Details

- Initialize malicious nodes with a specific location
- If a node swaps with the malicious node, the malicious node resets to the initial location (or one very close to it)
- This removes the "good" node location and replaces it with one of the malicious nodes choosing
- Each time any node swaps with the malicious node, another location is removed and replaced with a "bad" location
- Bad location(s) spread to other nodes through normal swapping behavior
- Over time, the attacker creates large clusters of nodes around a few locations

Attack Example 1/11

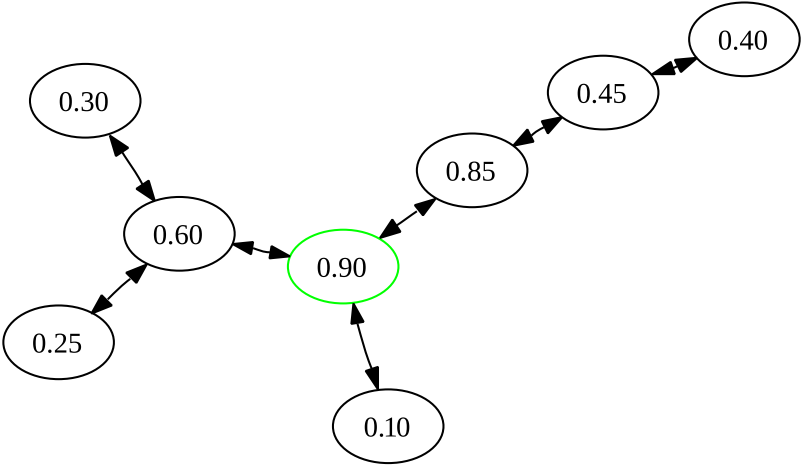


Figure: Node 0.90 is sent a signal to become malicious

Attack Example 2/11

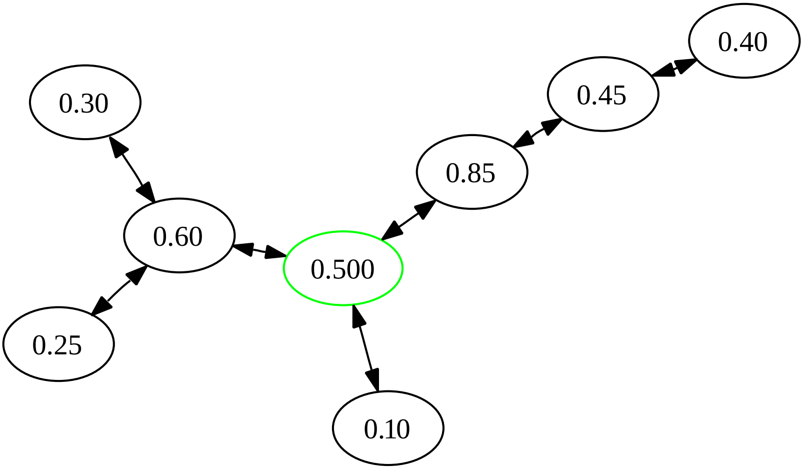


Figure: Malicious node resets its location to malicious location (0.500)

Attack Example 3/11

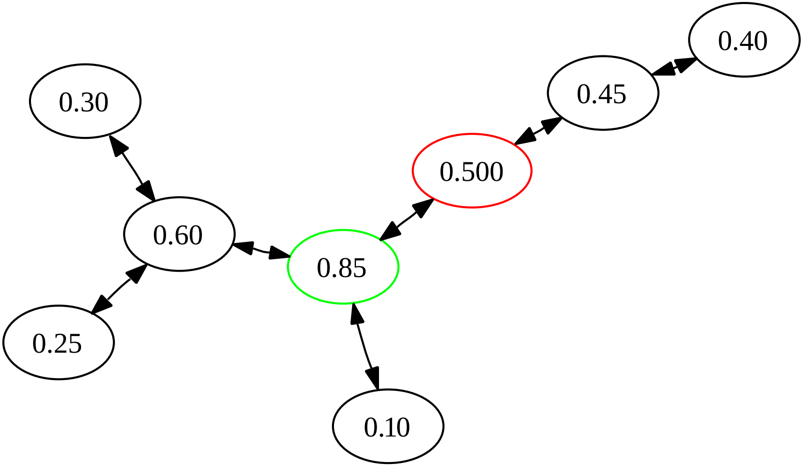


Figure: Malicious node forces a swap with 0.85

Attack Example 4/11

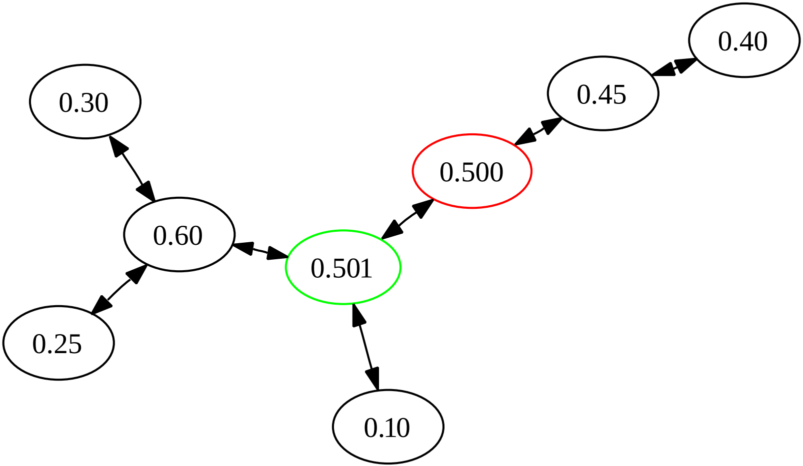


Figure: Malicious node resets its location to malicious location (0.501) removing "good" location 0.85

Attack Example 5/11

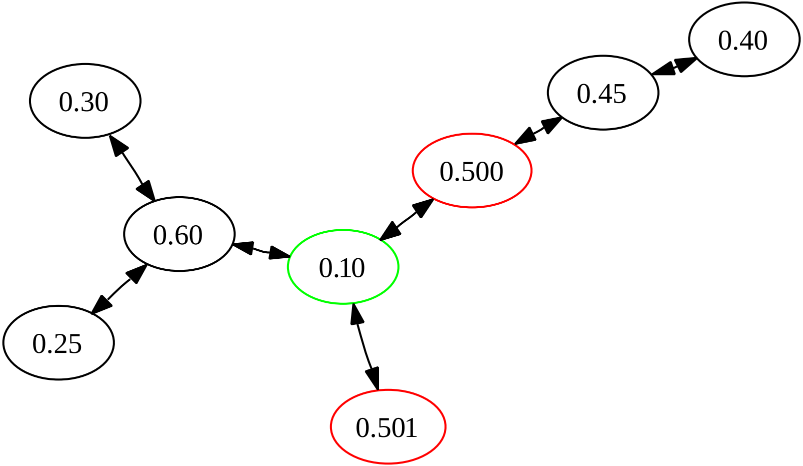


Figure: Malicious node forces a swap with 0.10

Attack Example 6/11

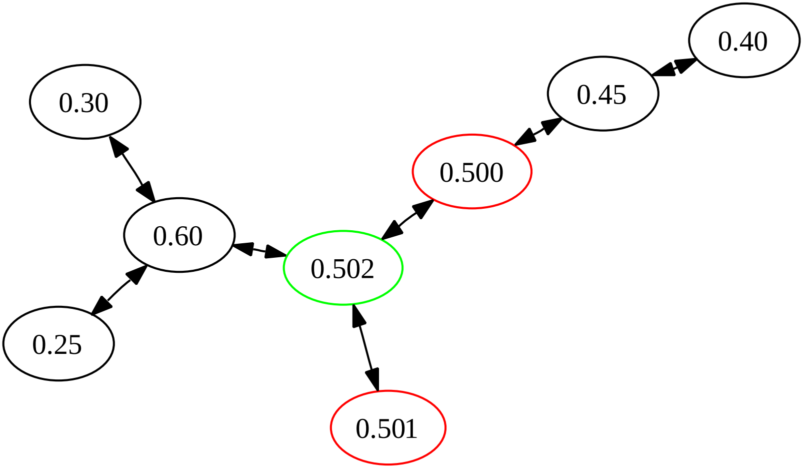


Figure: Malicious node resets its location to malicious location (0.502) removing “good” location 0.10

Attack Example 7/11

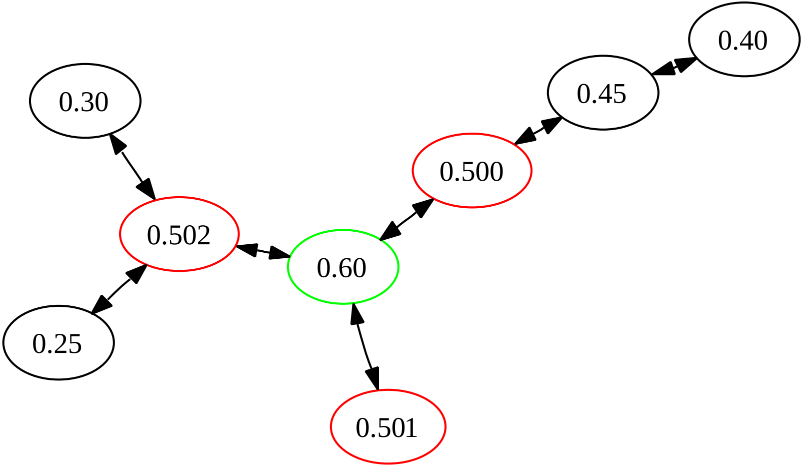


Figure: Malicious node forces a swap with 0.60

Attack Example 8/11

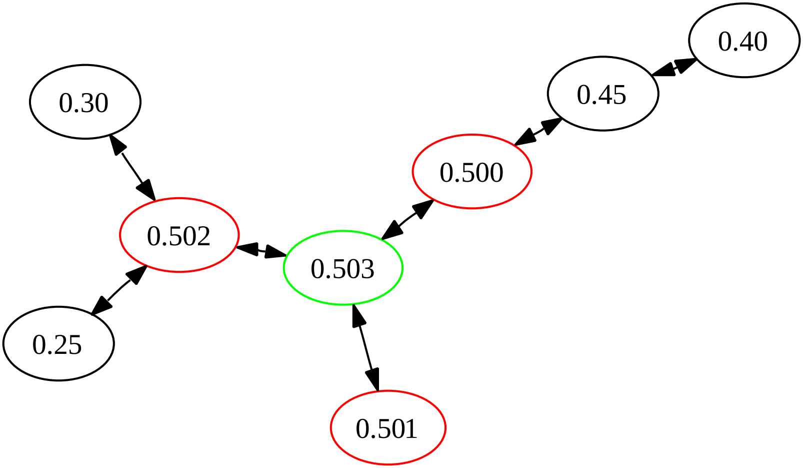


Figure: Malicious node resets its location to malicious location (0.503) removing “good” location 0.60

Attack Example 9/11

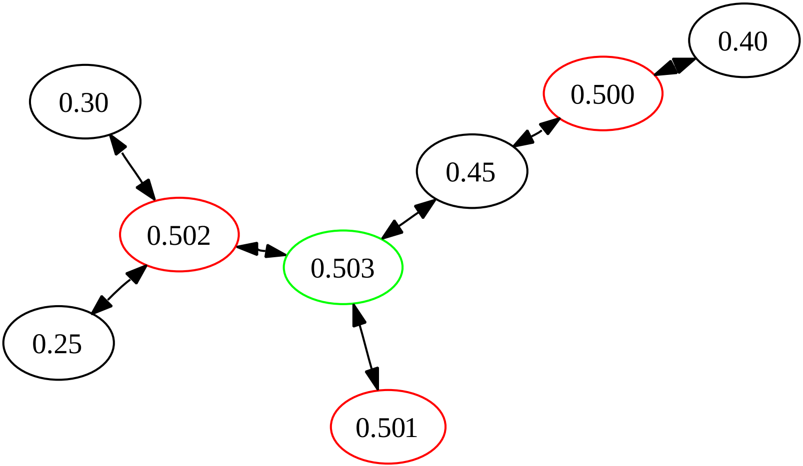


Figure: Even with low probability, a swap can occur between 0.500 and 0.45

Attack Example 10/11

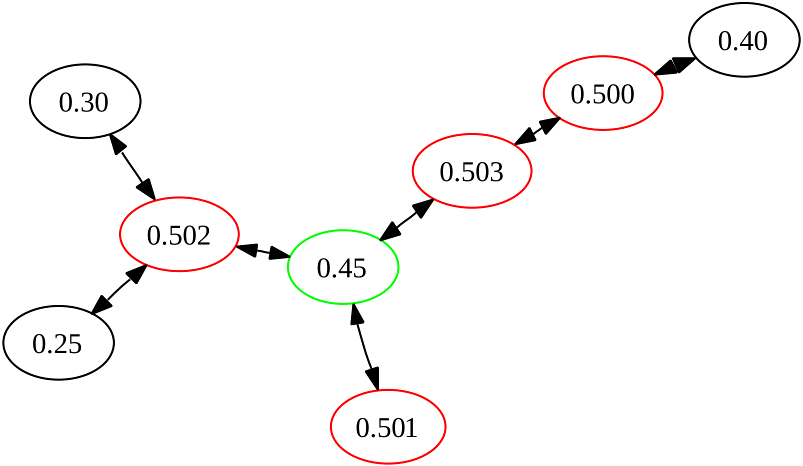


Figure: Malicious node forces a swap with 0.45

Attack Example 11/11

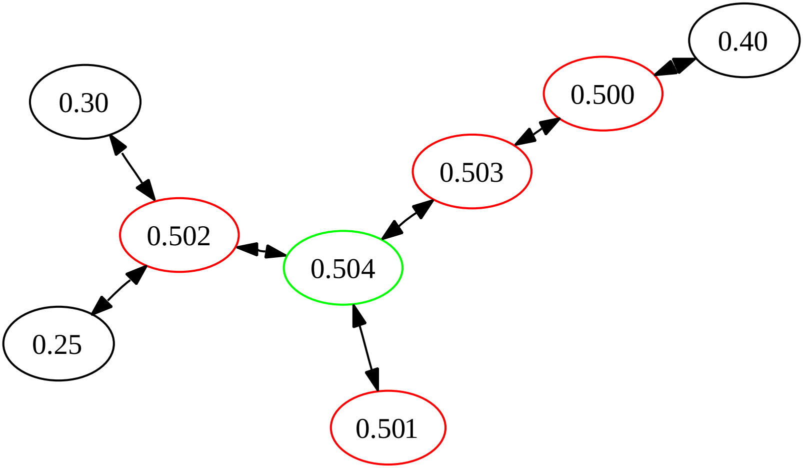


Figure: Malicious node resets its location to malicious location (0.504) removing “good” location 0.45

Attack Implementation

- Malicious node uses Freenet's codebase with minor modifications
- Attacker does not violate the protocol in a detectable manner
- Malicious nodes behave as if they had a large group of friends
- Given enough time, a single malicous node can spread bad locations to most nodes
- Using multiple locations for clustering increases the speed of penetration

Experimental Setup

- Created testbed with 800 Freenet nodes
- Topology corresponds to Watts & Strogatz small world networks
- Instrumentation captures path lengths and node locations
- Content is always placed at node with closest location
- Nodes have bounded storage space

Dispersion Example with 800 Nodes

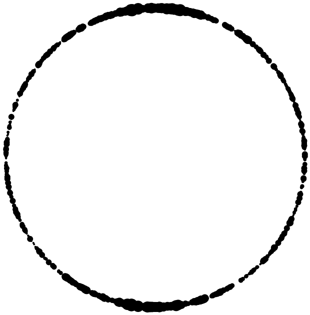


Figure: Plot of node locations before attack.

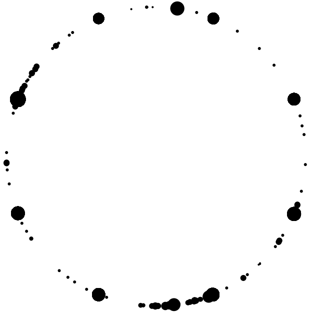


Figure: Plot of node locations after attack.

Data Loss Example (2 attack nodes)

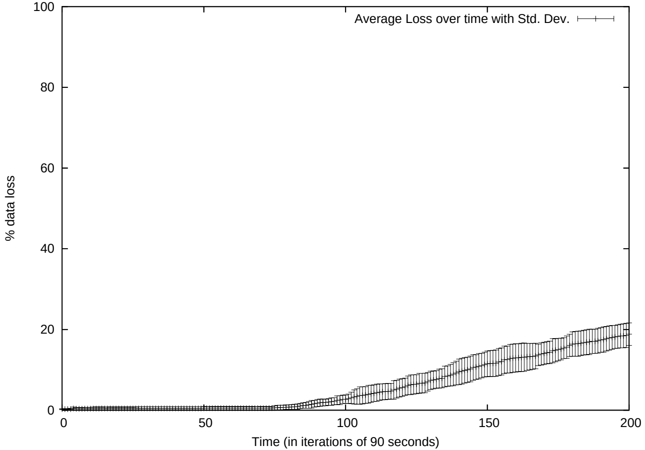


Figure: Graph showing average data loss over 5 runs with 800 nodes and 2 attack nodes using 8 bad locations with the attack starting after about 2h.

Data Loss Example (4 Attack nodes)

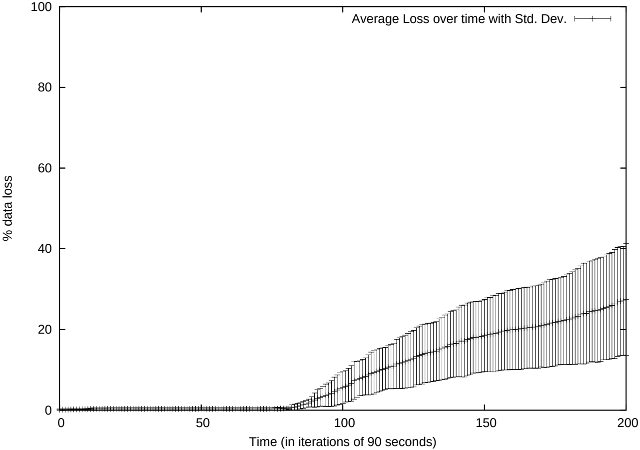


Figure: Graph showing average data loss over 5 runs with 800 nodes and 4 attack nodes using 8 bad locations with the attack starting after about 2h.

Data Loss Example (8 Attack nodes)

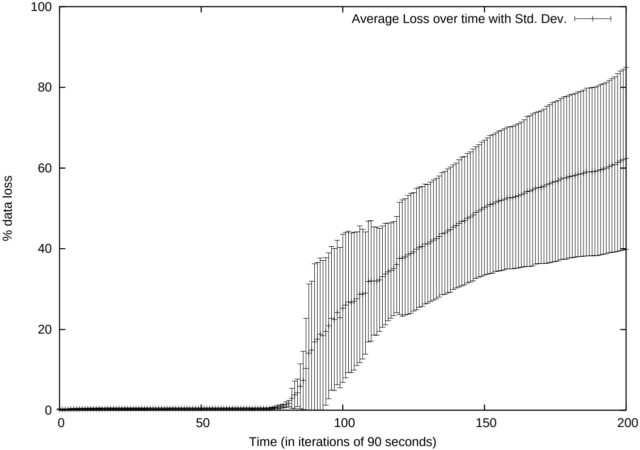


Figure: Graph showing average data loss over 5 runs with 800 nodes and 8 attack nodes using 8 bad locations with the attack starting after about 2h.

How to protect against this?

- Check how frequently a node swaps similar locations?
- Limit number of swaps with a particular peer?
- Determine a node is malicious because its' location is *too* close?
- Periodically reset all node locations?
- Secure multiparty computation for swaps?

In F2F networks, you can never be sure about the friends of your friends!

Churn

- Leave join churn
  - Nodes are not constantly in the network
  - They leave for some period of time and then come back into the network
- Join leave churn
  - Nodes join the network for a time, then disconnect permanently
- This also causes load imbalance similar to our attack

Churn Example 1/13

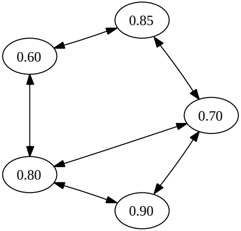


Figure: Example stable core network

Figure: Node location plot

Churn Example 2/13

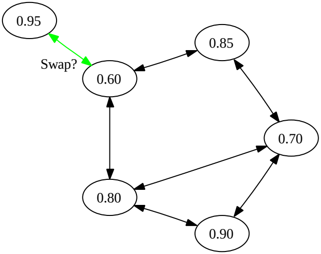


Figure: Node 0.95 joins the network

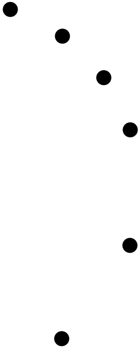


Figure: Node location plot

Churn Example 3/13

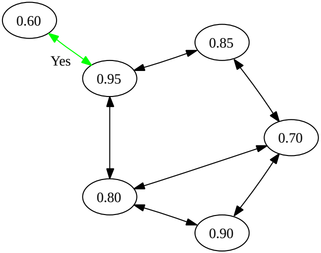


Figure: Node 0.95 swaps with 0.60

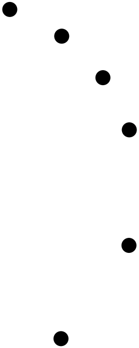


Figure: Node location plot

Churn Example 4/13

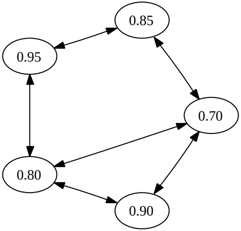


Figure: Node (now 0.60) leaves the network

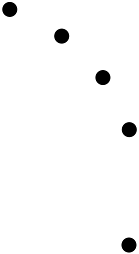


Figure: Node location plot

Churn Example 5/13

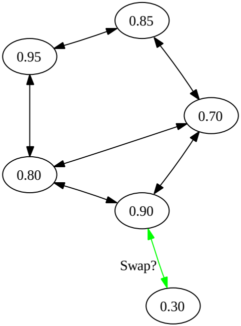


Figure: Node 0.30 joins the network

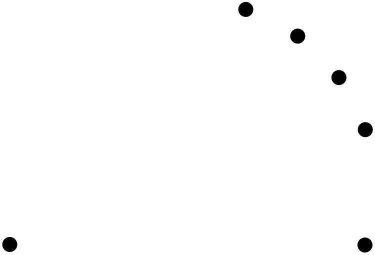


Figure: Node location plot

Churn Example 6/13

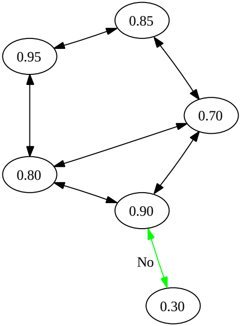


Figure: Node 0.30 does not swap

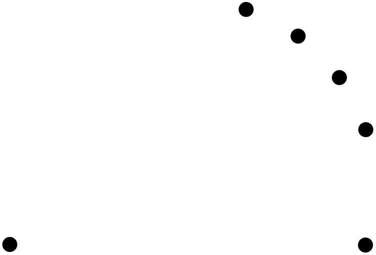


Figure: Node location plot

Churn Example 7/13

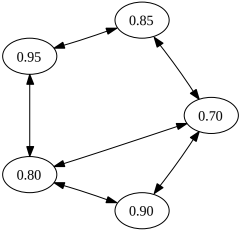


Figure: Node 0.30 leaves the network

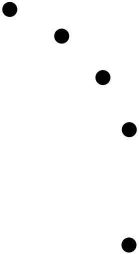


Figure: Node location plot

Churn Example 8/13

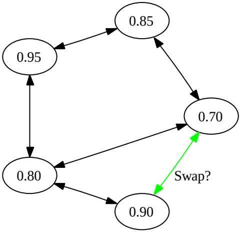


Figure: Nodes 0.70 and 0.90 consider a swap

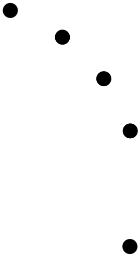


Figure: Node location plot

Churn Example 9/13

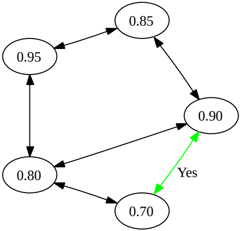


Figure: 0.70 and 0.90 swap

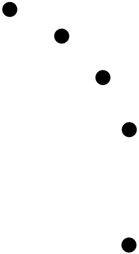


Figure: Node location plot

Churn Example 10/13

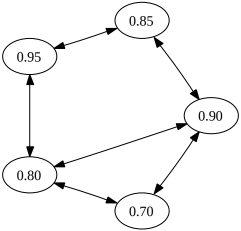


Figure: Result after the swap

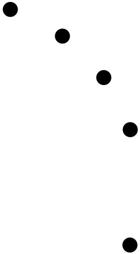


Figure: Node location plot

Churn Example 11/13

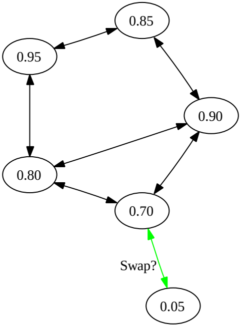


Figure: Node 0.05 joins the network

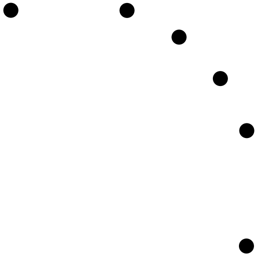


Figure: Node location plot

Churn Example 12/13

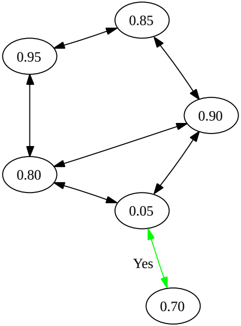


Figure: Node 0.05 swaps location with 0.70

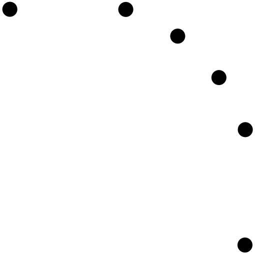


Figure: Node location plot

Churn Example 13/13

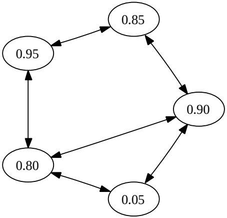


Figure: Node (now 0.70) leaves the network

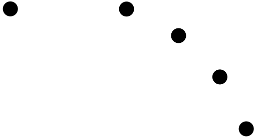


Figure: Node location plot

Churn Simulation

- Created stable core of nodes
- Simulated join-leave churn, let network stabilize
- Ran exactly the native swap code
- Repeat *n* number of times
- Revealed drastic convergence to single location

Conclusion

- Freenet's routing algorithm is not robust
- Adversaries can easily remove most of the content
- Attack exploits location swap, where nodes trust each other
- Swap is fundamental to the routing algorithm
- Natural churn causes similar results

References

Nathan S. Evans, Roger Dingledine, and Christian Grothoff.
A practical congestion attack on tor using long paths.
In *18th USENIX Security Symposium*, pages 33–50. USENIX, 2009.

Prateek Mittal and Nikita Borisov.
Information leaks in structured peer-to-peer anonymous communication systems.
In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 267–278, New York, NY, USA, 2008. ACM.