

# Routing in the Dark: Pitch Black

Nathan S. Evans

Christian Grothoff

`Nathan.S.Evans@du.edu`

`christian@grothoff.org`

Colorado Research Institute for Security and Privacy

University of Denver



# 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?

# 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?

Nope!

# Freenet 101

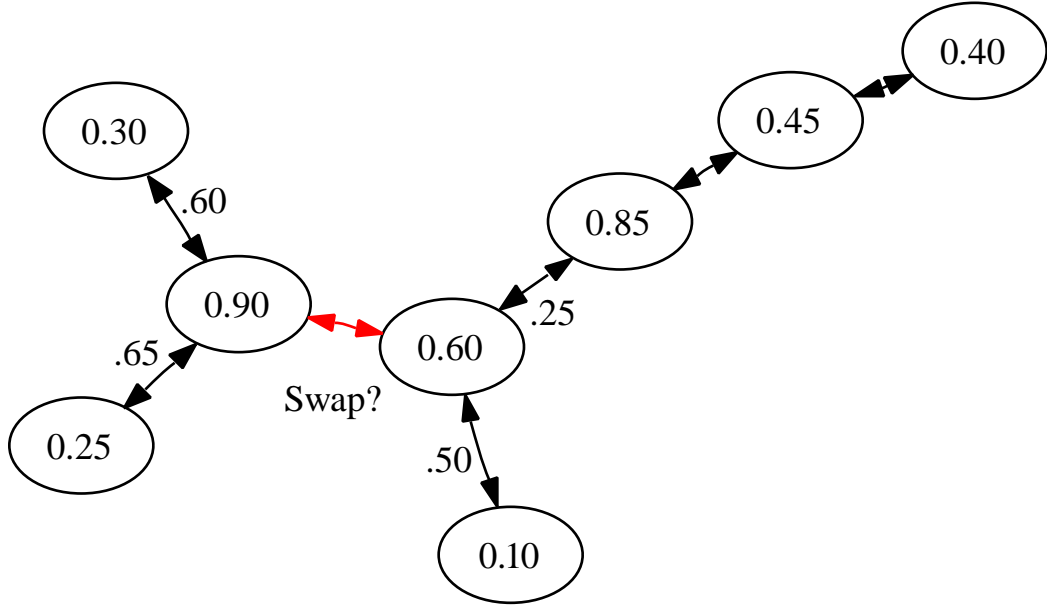
- Freenet is a 'anonymous' peer-to-peer network
- Overlay based on cyclic address space of size  $2^{32}$
- Nodes have a constant set of connections (F2F)
- All data identified by a key (modulo  $2^{32}$ )
- Data assumed to be stored at closest node
- Routing uses depth first traversal in order of proximity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



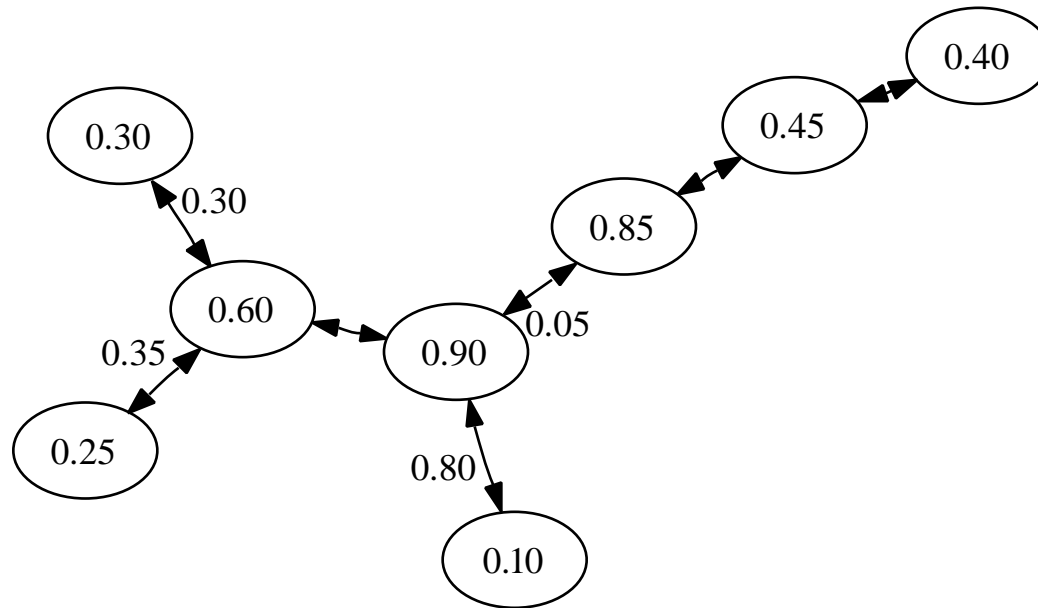
# Routing in the Dark

- 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:

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

- 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)
- ⇒ Depth-first routing in order of proximity to key.



# GET 1/7

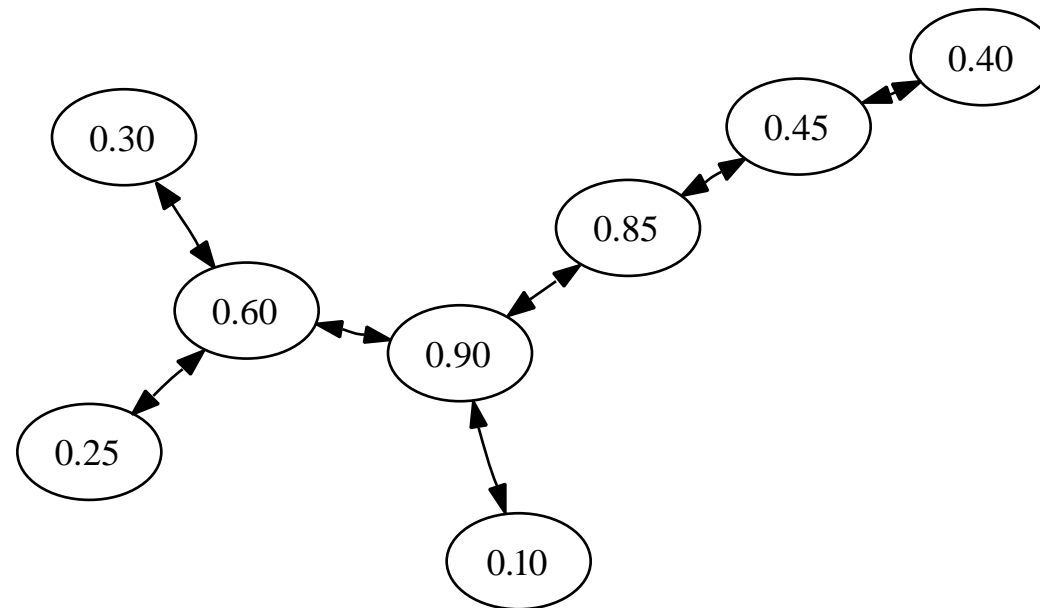


Figure 1: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# GET 2/7

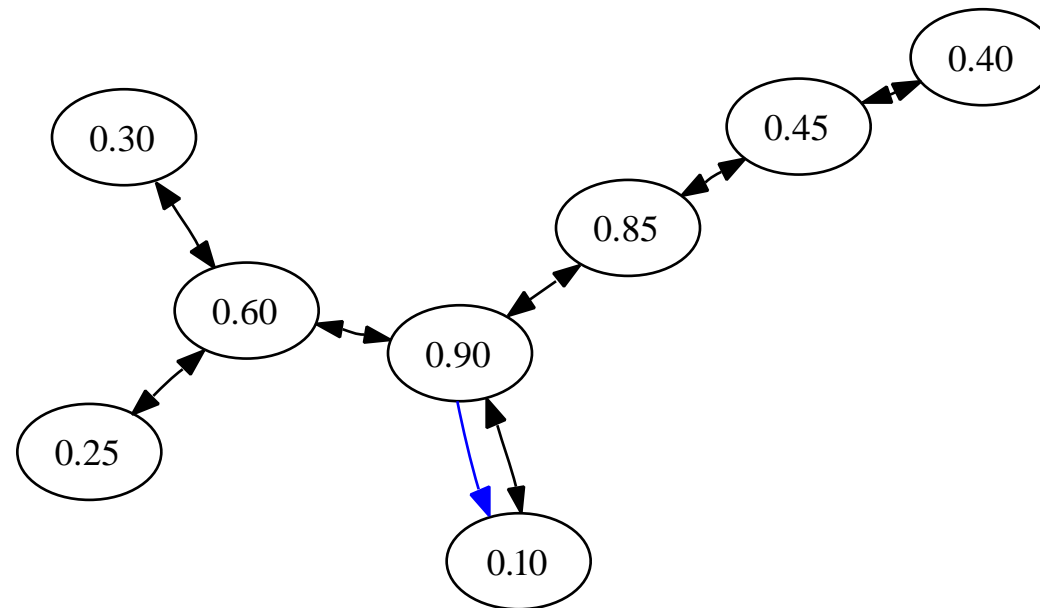


Figure 2: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# GET 3/7

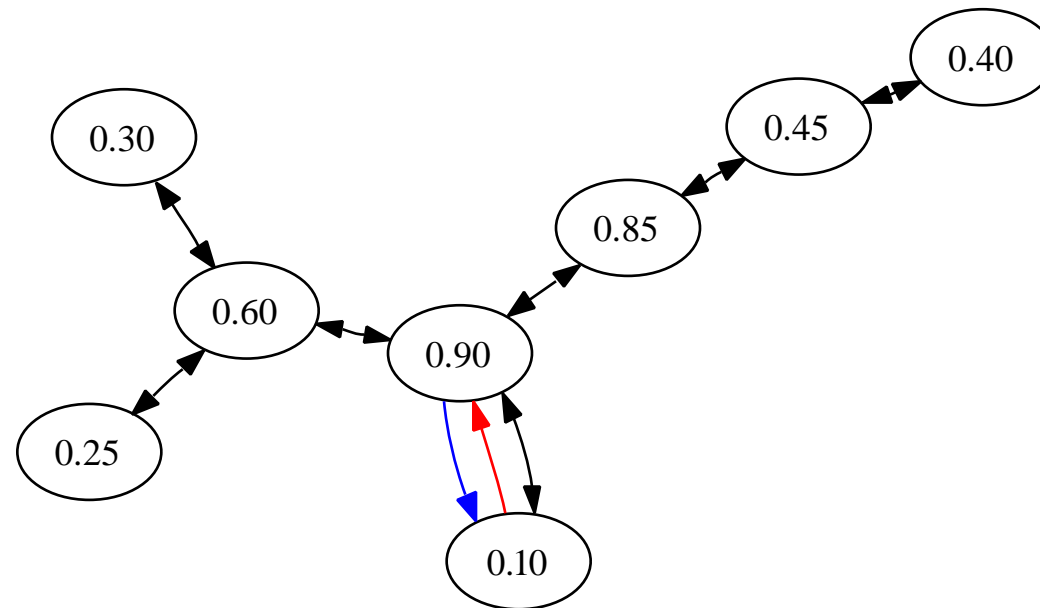


Figure 3: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# GET 4/7

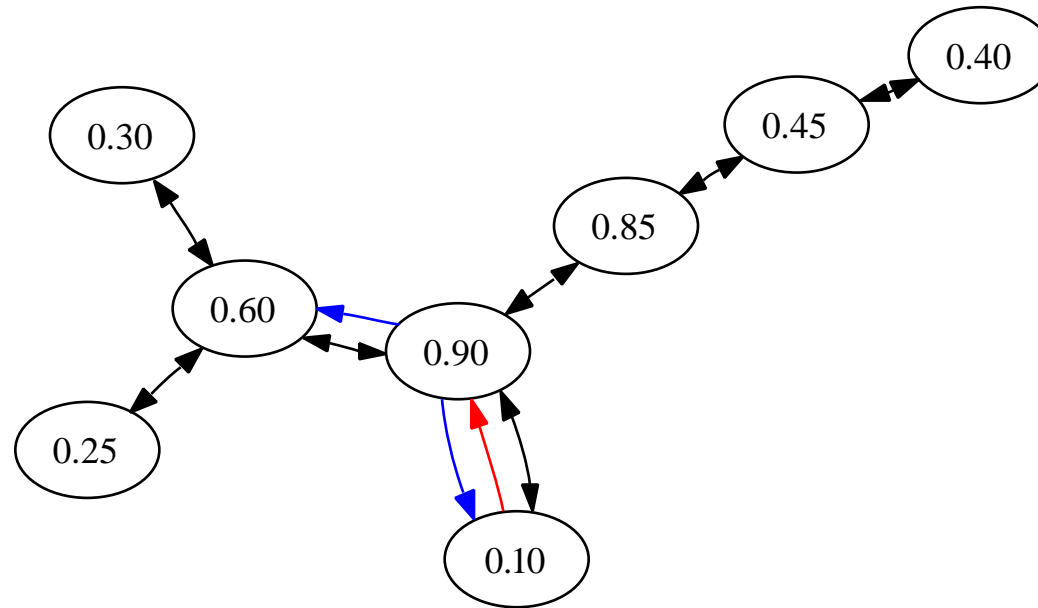


Figure 4: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# GET 5/7

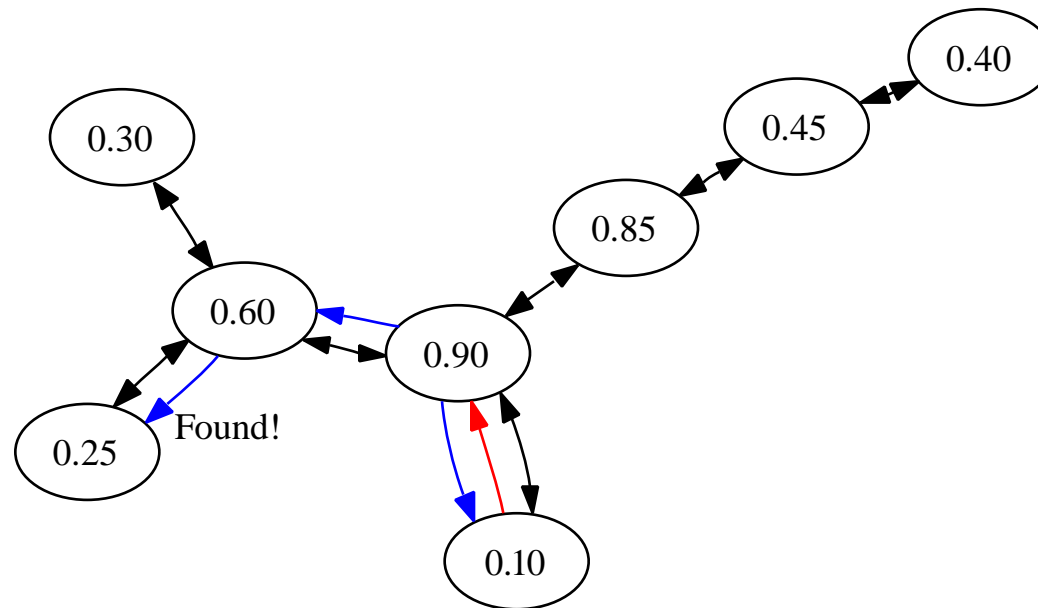


Figure 5: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# GET 6/7

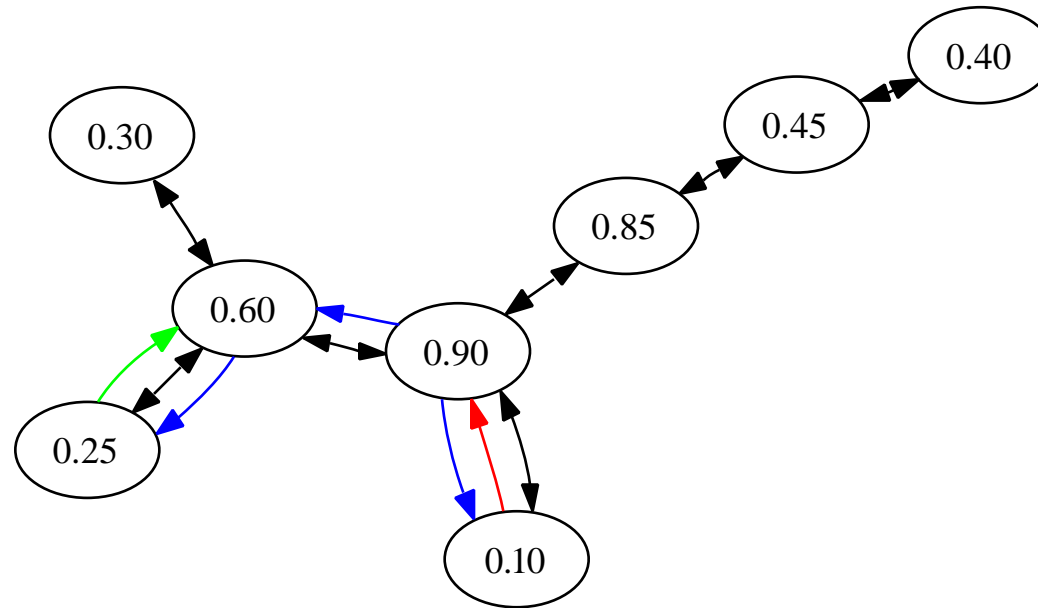


Figure 6: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# GET 7/7

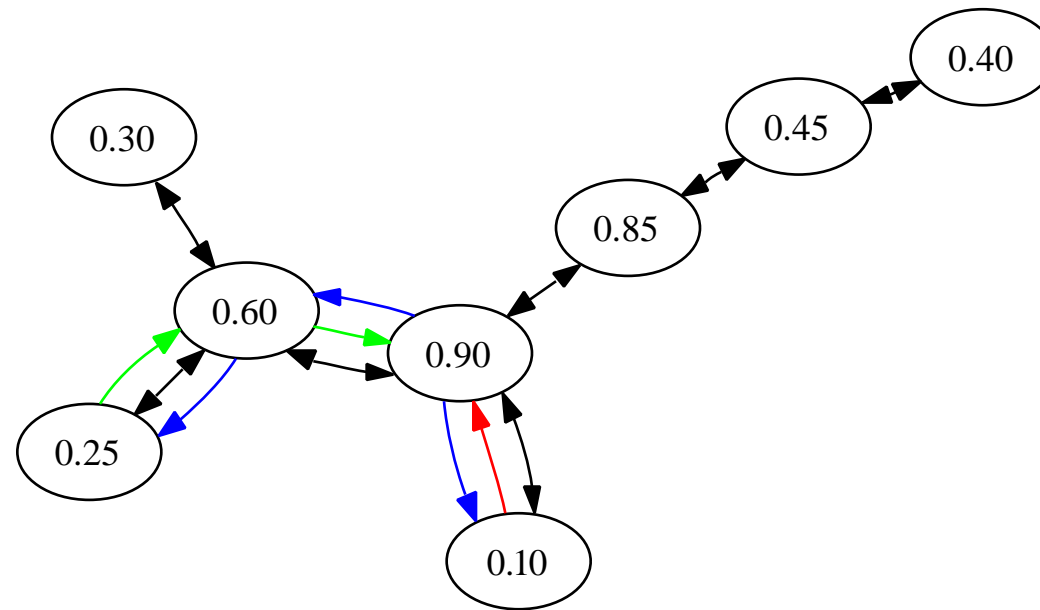


Figure 7: A GET request from node 0.90 searching for data with identifier 0.22 (which is stored at node identified by 0.25)

# PUT Requests

PUT requests are routed in a similar fashion 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, requests 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

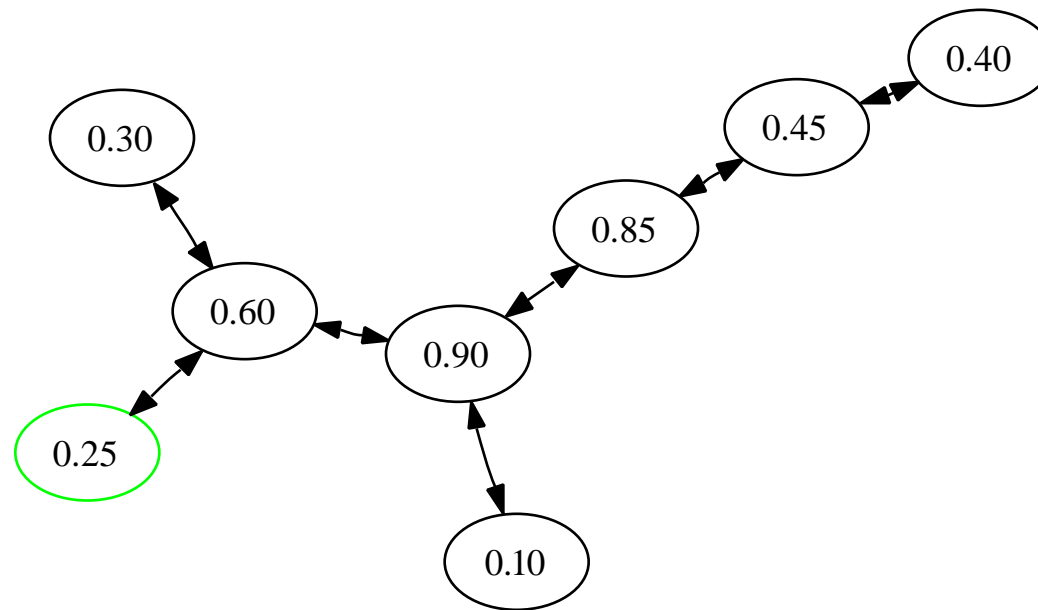


Figure 8: Put example from node with ID 0.25 inserting data identified by the ID 0.93

# Put Example 1/3

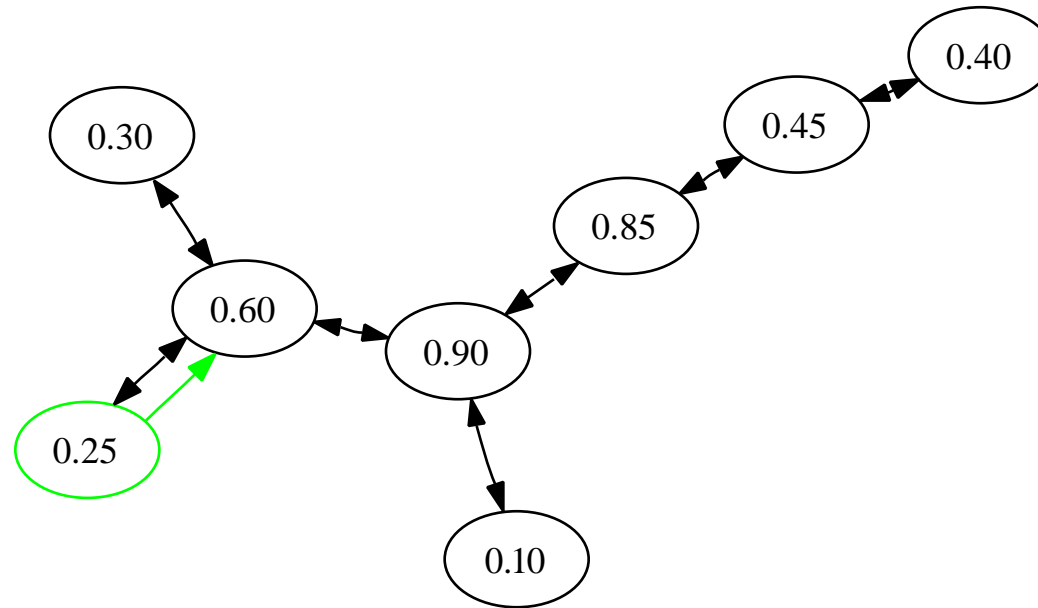


Figure 9: Put example from node with ID 0.25 inserting data identified by the ID 0.93

## Put Example 2/3

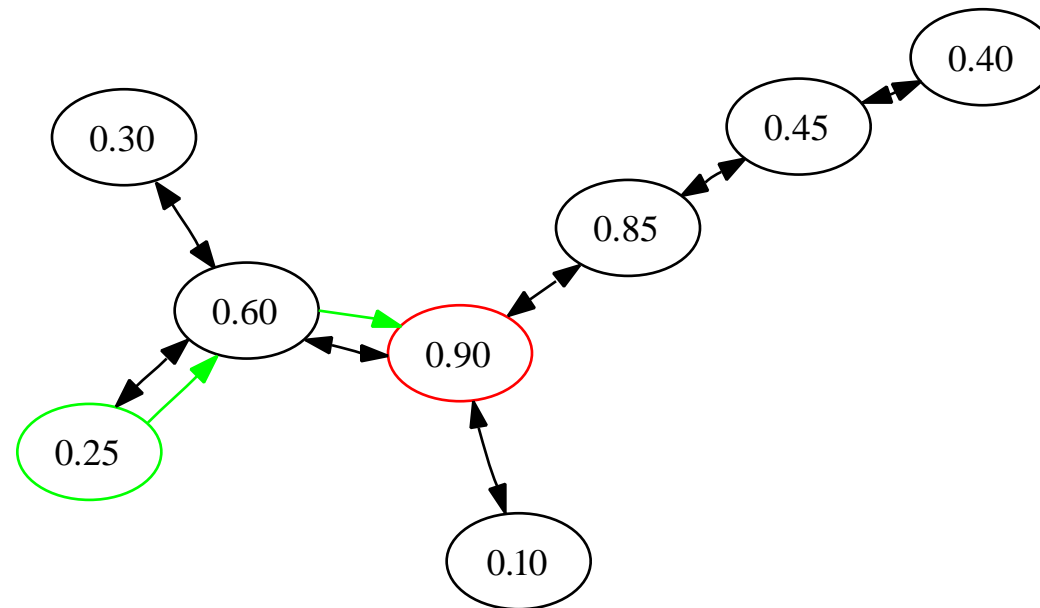


Figure 10: Put example from node with ID 0.25 inserting data identified by the ID 0.93

# Put Example 3/3

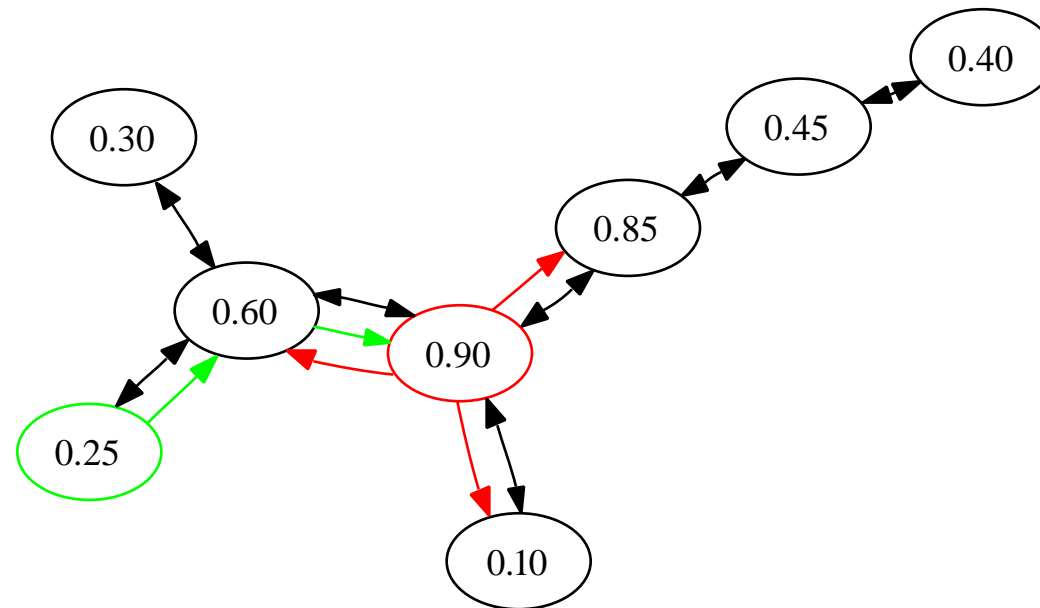


Figure 11: Put example from node with ID 0.25 inserting data identified by the ID 0.93

# Basic Idea for the Attack

- Freenet relies on wide spread 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 they may receive
- ⇒ 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 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 malicious 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

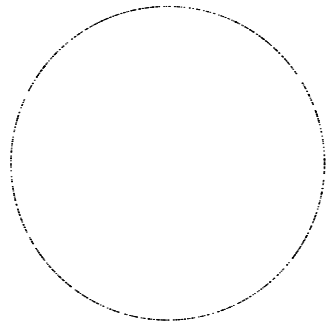


Figure 12: Plot of node locations before attack.

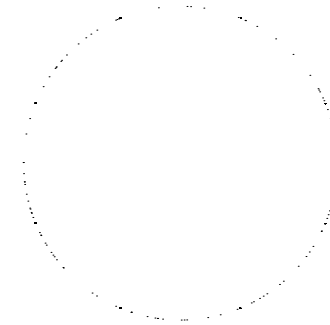


Figure 13: Plot of node locations after attack.

# Data Loss Example (2 attack nodes)

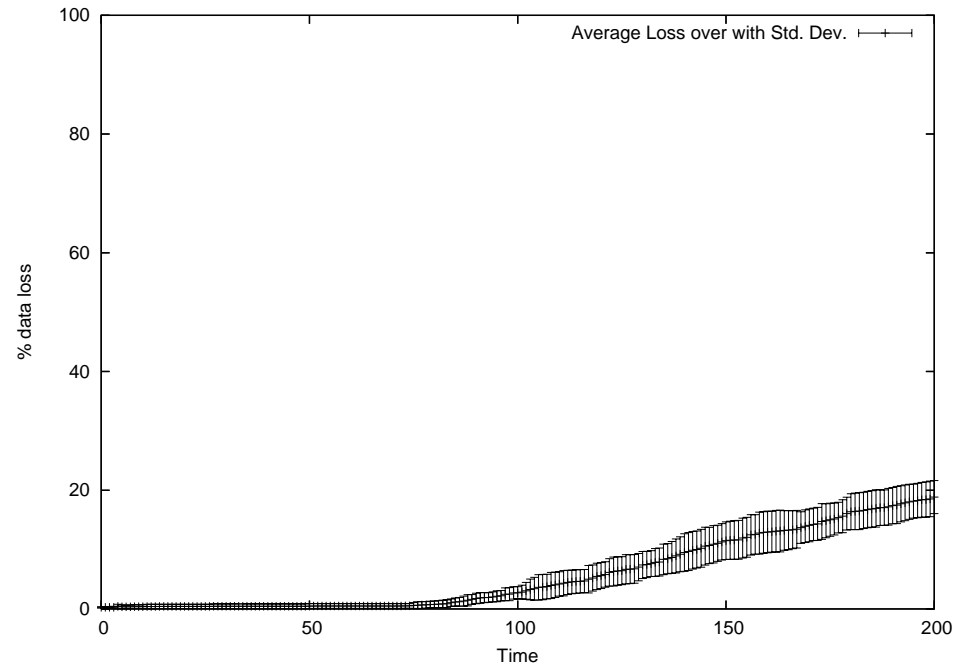


Figure 14: Graph showing average data loss over 5 runs with 800 nodes and 2 attack nodes using 8 bad locations with the attack starting after about 2h.

# Data Loss Example (4 Attack nodes)

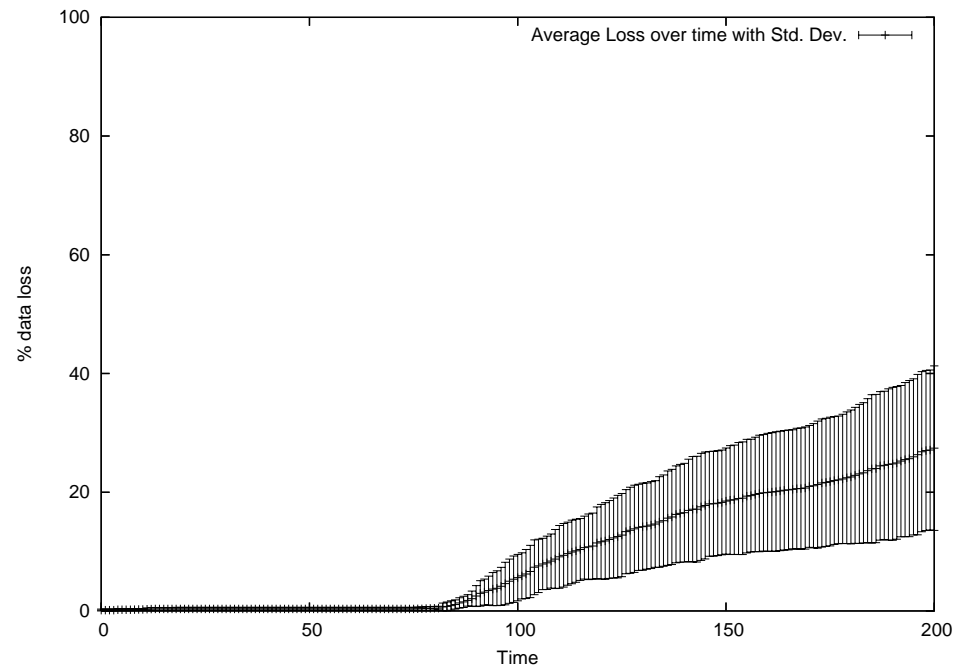


Figure 15: Graph showing average data loss over 5 runs with 800 nodes and 4 attack nodes using 8 bad locations with the attack starting after about 2h.

# Data Loss Example (8 Attack nodes)

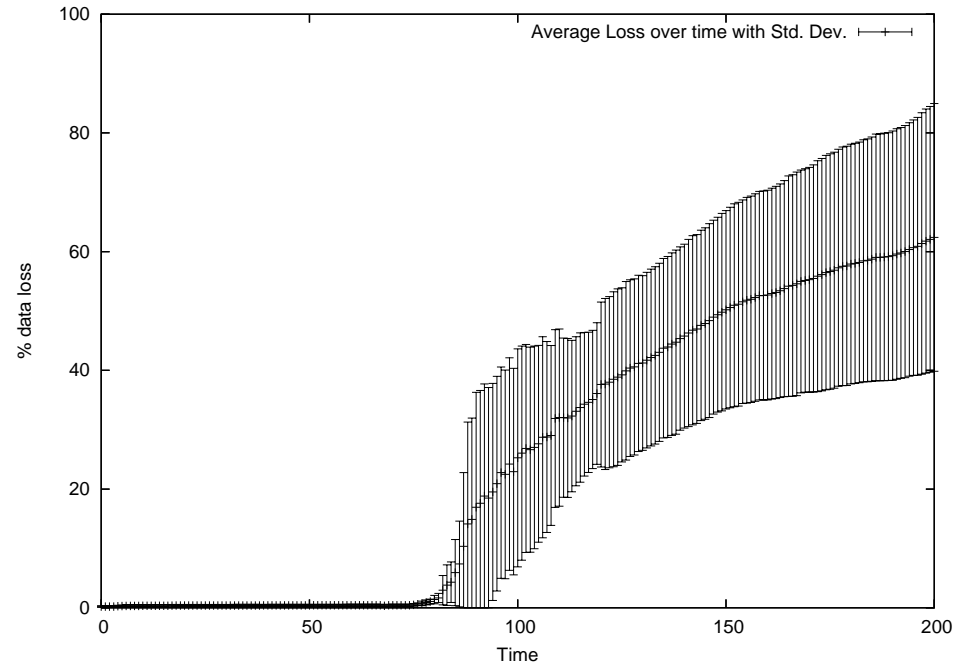


Figure 16: Graph of typical data loss from 1 run with 800 nodes and 8 attack nodes using 8 bad locations with the attack starting after about 2h.

## 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!

# 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
- Attack code available at <http://crisp.cs.du.edu/pitchblack/>

Natural churn causes similar clustering of node locations. For details, read our paper!



# Questions?



# Routing Path Lengths

