

Self Protection + Self* in Small World Networks

Felix Halim, Yongzheng Wu, Roland Yap
NUS

Overview

- Background and Motivation
 - Why Small World Networks?
- Small-world models
 - Watts and Strogatz
 - Kleinberg
- Routing performance
 - Sandberg's Distributed routing
- Self protection problems
 - Possible attacks
 - Protection mechanisms

P2P Issues

- Problems with P2P
 - Structured network maintenance is high
 - Vulnerable to churn
 - Unstructured network
 - Minimal assumptions
 - Inefficient query mechanism (flooding)
 - Many attacks
 - Routing attacks
 - Churn attacks
 - Sybil nodes

Small World Networks

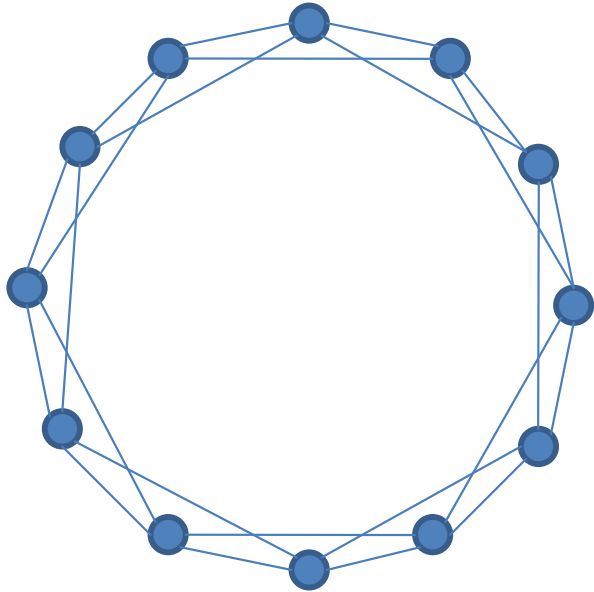
- Small world networks:
 - Social connection between nodes
 - trust relationship (Social Networks)
 - can exploit darknet
 - Navigability
 - able to route messages between two nodes in a short hops
 - Protection against Sybil Attack
 - Graph topology is relatively static comparing to DHT
 - Natural algorithms which are inherently local (fits some SELFMAN objectives):
 - naturally Self configured using local properties
 - self tuning + self healing
 - **self protection?** We are investigating protection mechanisms ... some initial positive results but more research needed ...

Social Networks

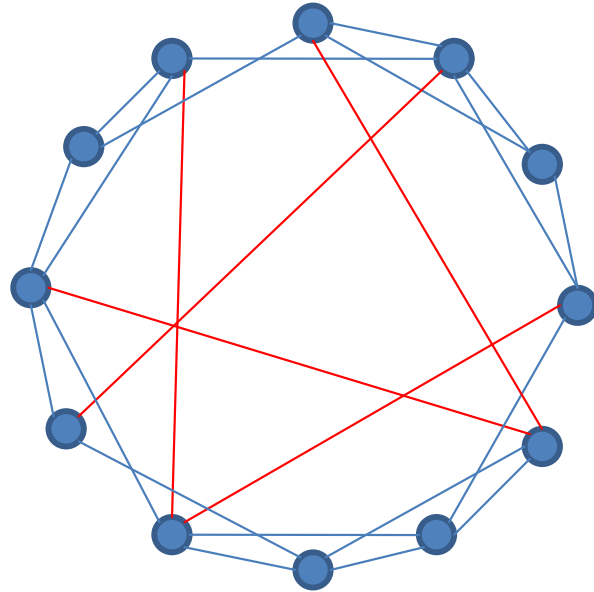
Other factors: (maybe opportunities for SELFMAN)

- Actual Social Networks correlate with Small World Networks (some experiments with real social network data)
- Increasingly popular: Facebook, LinkedIn, MySpace, ...
- Wikis
- Google Friend Connect (open infrastructure)
- Can everybody be connected? (Milgrom experiment, 6 Degrees of Separation)
- Natural hierarchical/organizational boundaries
- Used by Freenet: exploits darknet to increase privacy
- Doesn't have to be an Overlay Network ... may be the actual network (unlike structured overlay networks)

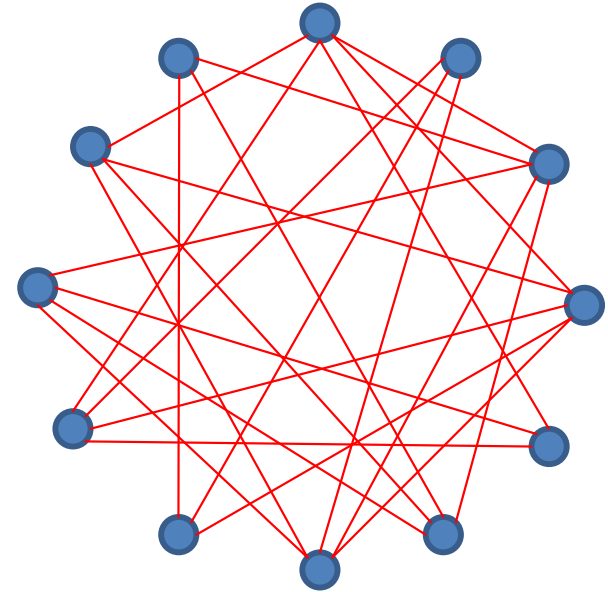
Watts and Strogatz Small World Model



$p = 0$, no shortcuts
Regular network
High Clustering
Large diameter



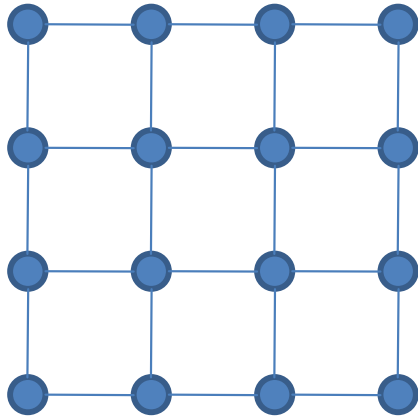
$0 < p < 1$, few shortcuts
Small-world network
Medium clustering
 $\text{Log}(n)$ diameter



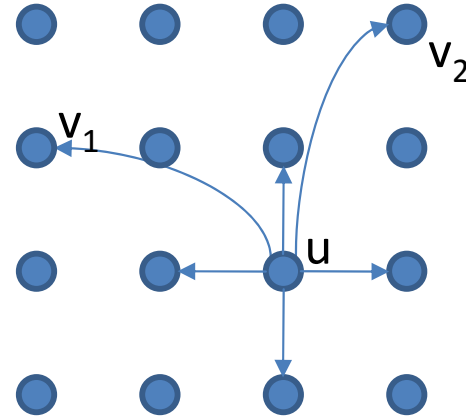
$p = 1$, many shortcuts
Random network
Low clustering
 $\text{Log}(n)$ diameter

Random rewiring procedure does not alter the number of vertices or edges in the graph.
With probability p , *reconnect this edge to a vertex chosen uniformly at random over the entire ring.*

Kleinberg model



$n = 4, p = 1, q = 0$



$n = 4, p = 1, q = 2$

- local node connections from lattice (left pic)
- non-local connections: add q shortcuts with probability p proportional to $d(u,v)^{-r}$ (right pic)
- node ids based on lattice coordinates
- **Greedy routing** works in expected $O(\log^2(n))$ steps.
- Expected diameter is $\log(n)$.

Routing Issues

- Kleinberg assumed that the individual nodes are aware of their own coordinates as well as those of their neighbors and the destination node.
 - In peer-to-peer networks this assumption is not valid
 - How can short-paths be found in this case?
- Can the ***embedding*** be recovered [Sandberg]?
 - Fully distributed only local knowledge at each node
 - Possible to make greedy routing work

Sandberg model

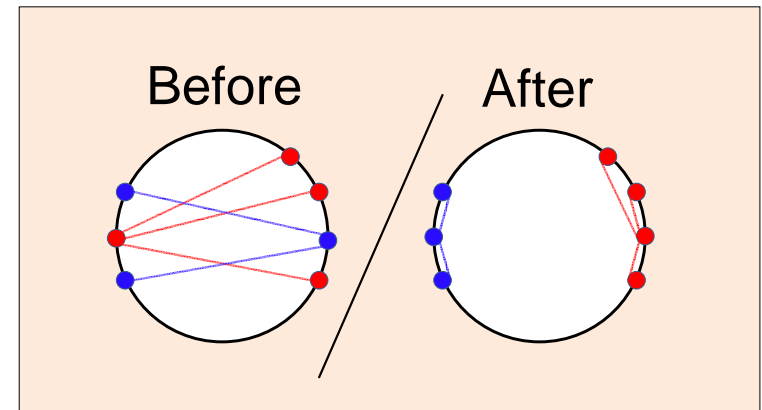
- Ring network topology (1-D)
- More links - up to $6 \cdot \log(n)$ links per node:
 - $\log(n)$ routing steps instead of $\log^2(n)$ steps
 - minimizing the dead-end queries
- Assumes nodes do not know any global topology/coordinate information

Recovering the embedding

- The positions are uniformly and uniquely generated for each node.
- For every iteration, each node will attempt to switch its position with another node found using random walk through their neighbors.

- The switch probability is defined as

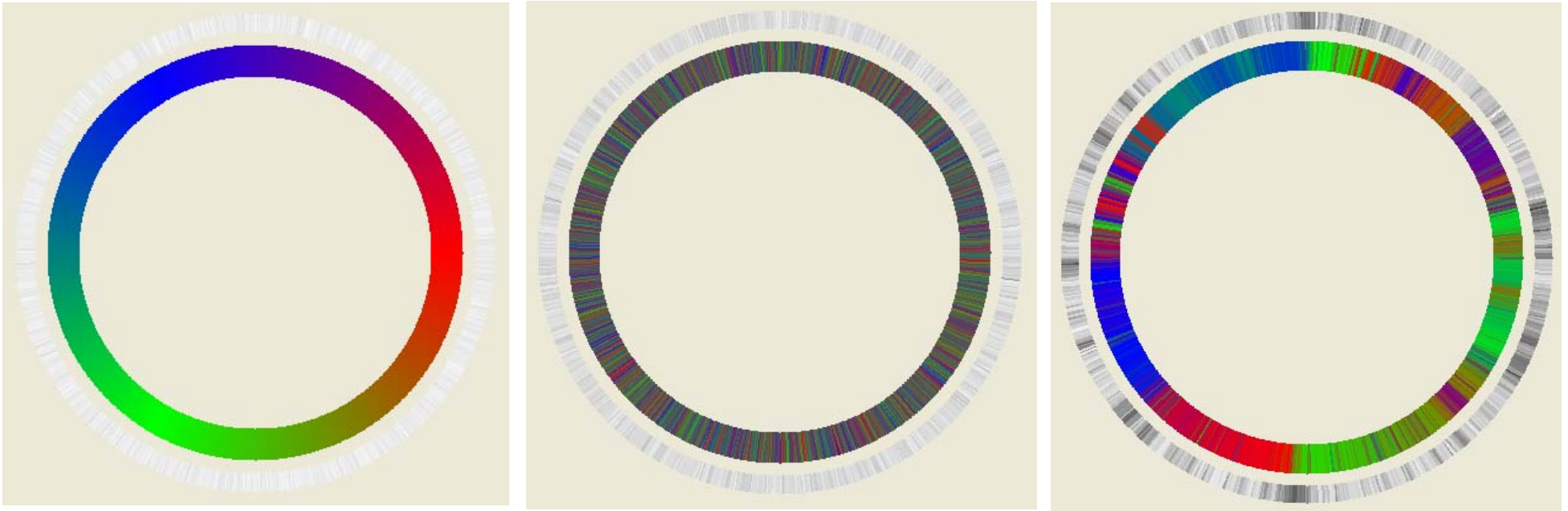
$$\beta(\phi_1, \phi_2) = \min \left(1, \prod_{i=1}^m \frac{d(\phi_1(x_i), \phi_1(y_i))^k}{d(\phi_2(x_i), \phi_2(y_i))^k} \right)$$



- In plain English: The switch probability is counted by having ***the product of edges before switch divided by the product of edges after switch*** (if its larger than 1, then the probability is 1)

- Good convergence routing results from simulation

Recovering the embedding



- Imperfect embedding recovery (cannot expect perfect recovery)
- Nevertheless, simulation shows routing in $\log(n)$ hops

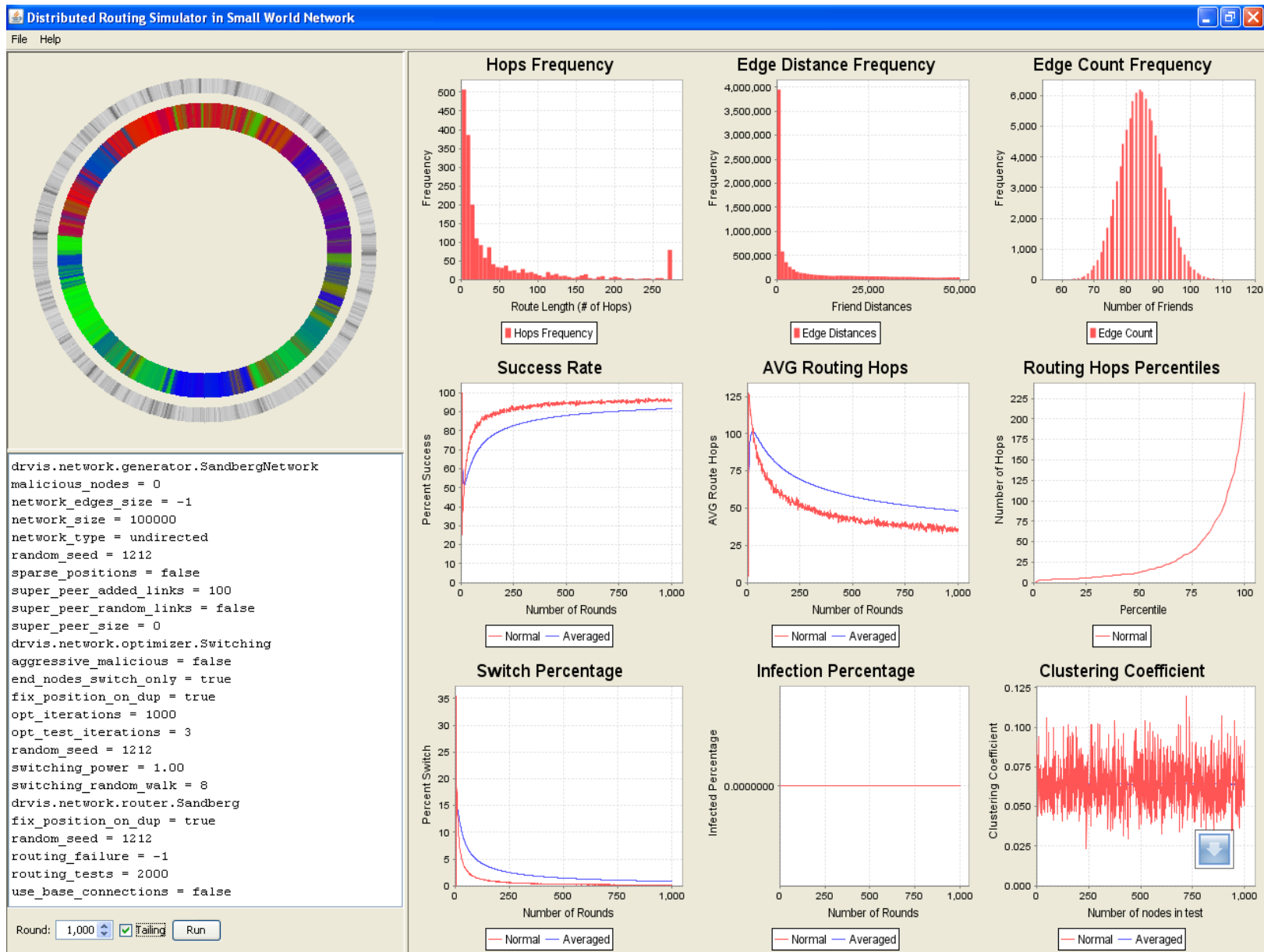
Switching strategies

- Two strategies of switching:
 1. Switch a node with another node found using S steps of random walk [Sandberg].
 2. Switch a node with any node that are the best to switch along the S steps of random walk.
- Our current simulations:
 - Strategy (2) converges faster
 - Final restoration less good than (1)

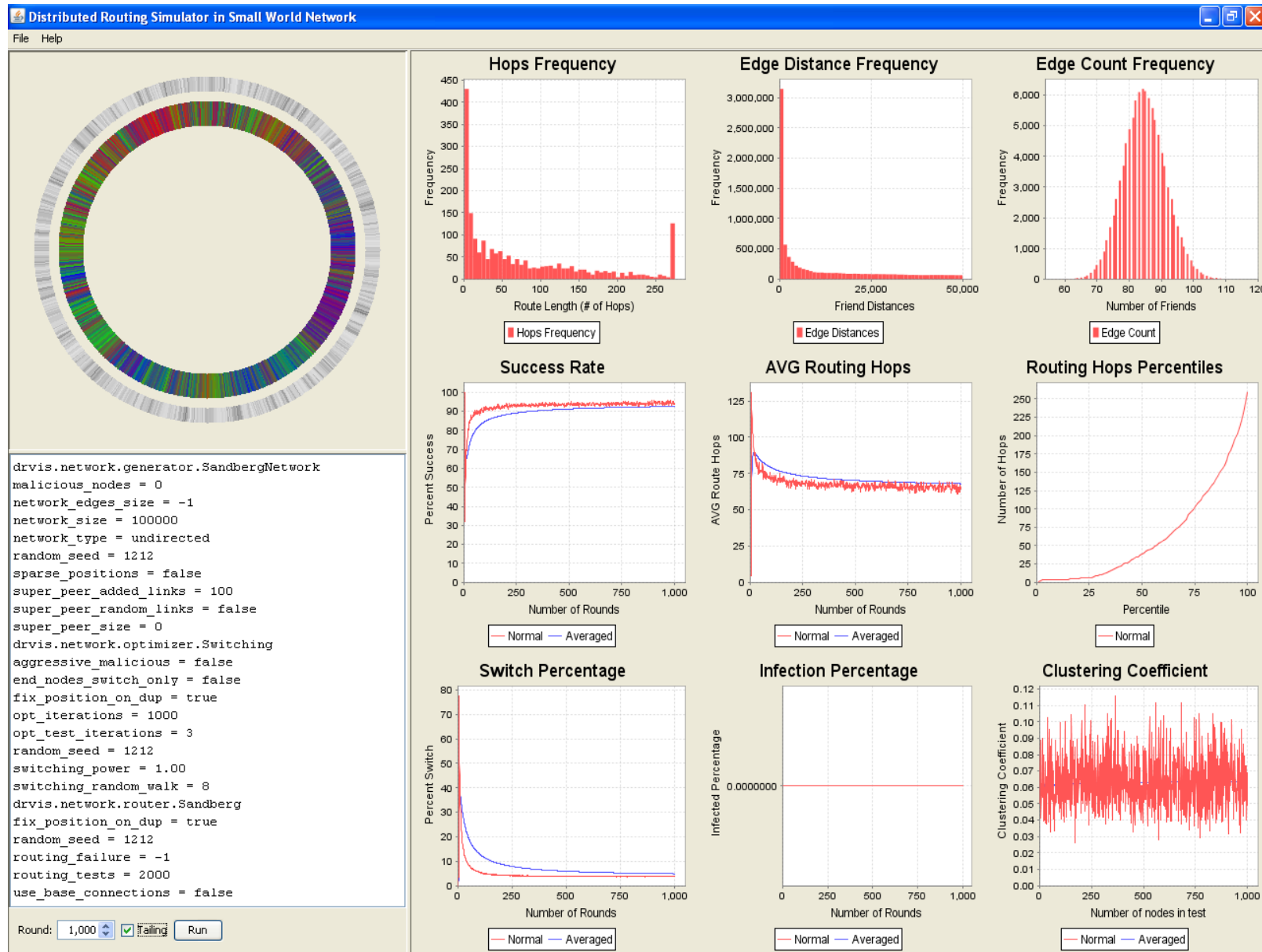
Small World Network Simulator

- Easy to test routing strategies
- Currently testbed for routing + self protection
- Animation
- Visualization:
 - seems to be useful to understanding performance
 - work in progress
- Statistics

Strategy (1, Sandberg)



Strategy (2)

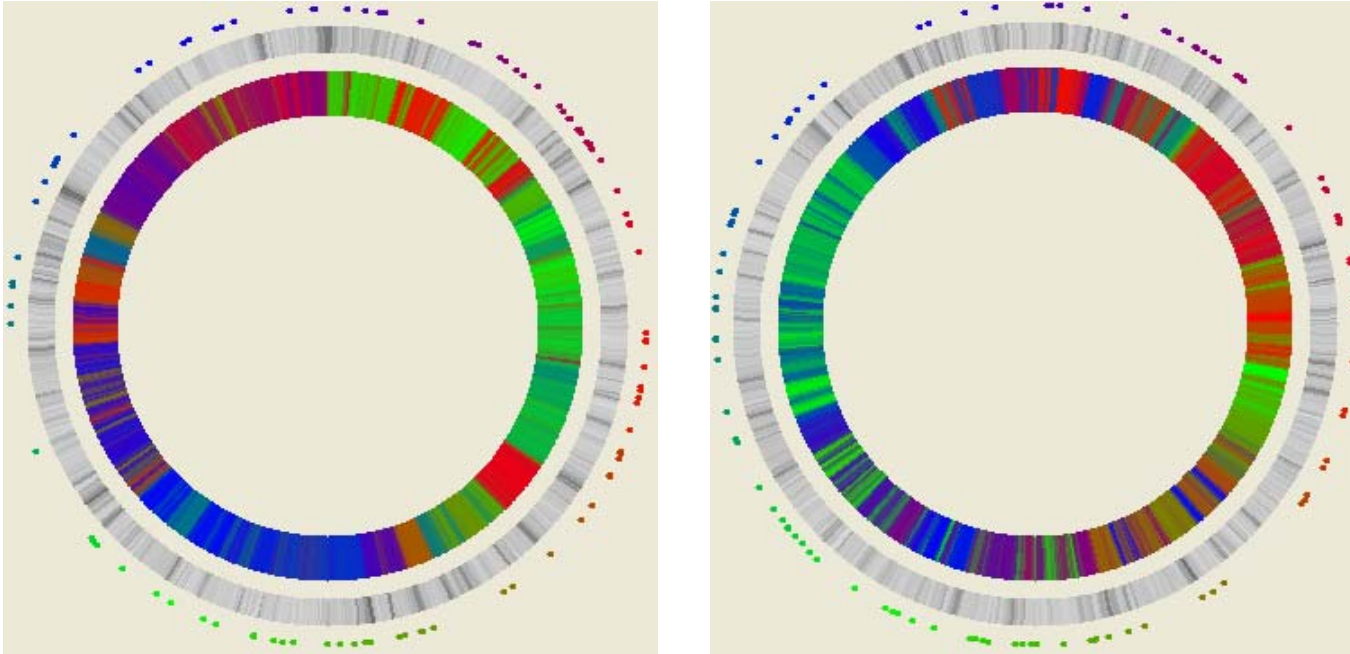


Convergence, Routing Length, and Success Rate

- The convergence of switching algorithm is fast
 - The first $100N$ iterations will give the network 90% success rate for routing and reach near stable state (below 3% switching percentage)
- The route length distribution can be seen in "Hops Frequency" histogram.
 - Many tests have short route length
 - Route failed when either exceed $\log^2(n)$ steps or stuck when all neighbors already visited
- Good up to 90% percentiles for the route length (even less than $\log(N)$):
 - For $N = 100,000$. Experiment results can go below $\log(N) = 16$ even for 90% percentiles.
 - With success rate = 99.6 % and
 - Average Hops = 8.595

Percentiles	Hops
10%	3.000
20%	4.000
30%	4.000
40%	5.000
50%	5.000
60%	6.000
70%	6.000
80%	8.000
90%	12.000

Initial Super Nodes Experiments



- Super nodes (more edges)
 1. The edges follow the power law distribution (left pic)
 2. The edges are randomized (right pic)
- The restored network is better looking with (1), not much effect on routing statistics

Lessons from Sandberg model

- Only local information – very minimal requirements
- Routing works! (only probabilistic guarantees ... but other P2P may not be better under dynamism)

Self-protection Issues

- Topology relatively static
 - Immune against churn (supposed to be the case)
 - not tested much yet
- But basic network has no protection against malicious nodes
 - Even 1% malicious nodes, the infection (position poisoning) can reach up to 60% of the network in small number of iterations.
 - The more neighbors, the more nodes malicious nodes can poison

General Small World Network Attacks

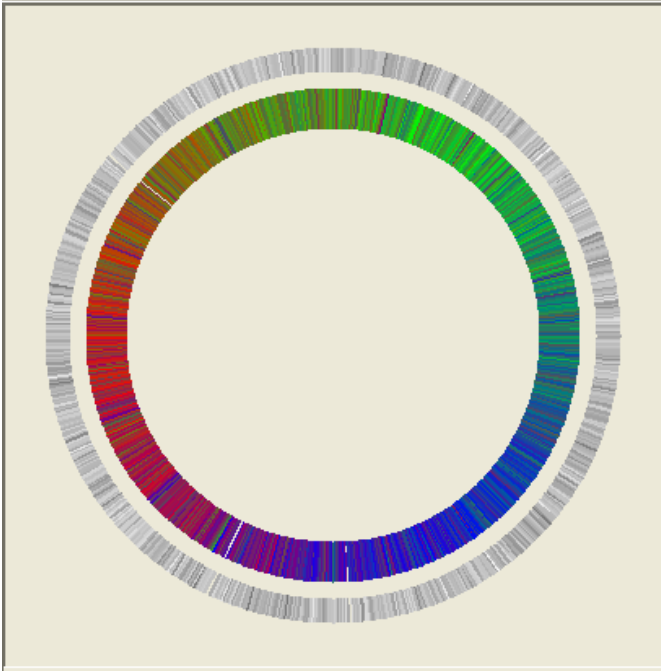
- Frequent add/remove friend
 - Effect: churn in social network
- Frequent online/offline
 - Effect: data loss, routing failure
- Incorrect swap decision
 - How: By providing incorrect neighbors, the attacker can force a swap or no-swap.
 - Effects: no data loss, minor effect to routing performance.
- Delete position
 - How: Reset to a fixed position X after each swap
 - Effects: number of distinct positions is reduced by 1 for each reset. All data associated to the position are lost. After $O(n)$ time, most nodes have the same position X. Routing becomes random routing.
- Concentrate position
 - How: Reset to a random position near X after each swap.
 - Effects: After $O(n)$ time, most nodes will have position near X. (concentrated at X) Newly added data will be concentrated in a few nodes not near X. Huge performance impact on routing also.

Routing attacks in Sandberg model

- The routing attack is achieved indirectly by attacking the embedding recovery algorithm
- Other than that any general routing attacks can be applied by:
 - Dropping packet
 - Effect: routing failure
 - Incorrect route
 - Effect: longer routing length

Experimental Attack Results

- The statistics in the next page shows the most destructive attack
 - By concentrating the positions
- Parameters: $N = 10,000$ and 1% malicious nodes
 - Each N iterations, malicious nodes can infect 8 nodes if they are not already infected
- Routing performance:
 - The success rate drops to below 40%
 - The switching stays constant above 7%
 - 60% of the nodes are poisoned



```

drvis.network.generator.SandbergNetwork
infection_rate = 8
malicious_nodes = 100
network_edges_size = -1
network_size = 10000
network_type = undirected
sparse_positions = false
super_peer_added_links = 100
super_peer_random_links = true
super_peer_size = 0
    
```

```

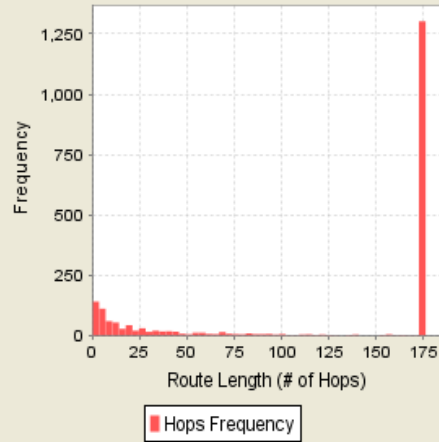
drvis.network.optimizer.Switching
aggressive_malicious = false
end_nodes_switch_only = true
fix_position_on_dup = true
opt_iterations = 500
opt_test_iterations = 3
switching_power = 1.00
switching_random_walk = 8
    
```

```

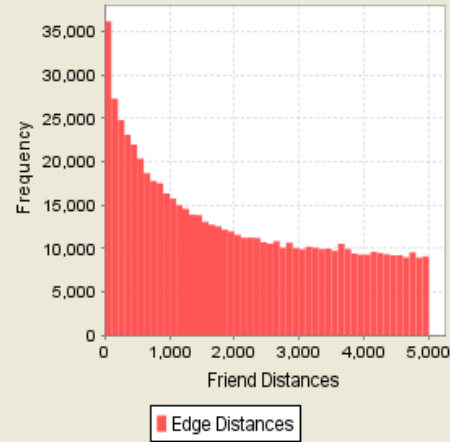
drvis.network.router.Sandberg
fix_position_on_dup = true
routing_failure = -1
routing_tests = 2000
use_base_connections = false
    
```

Round: Tailing

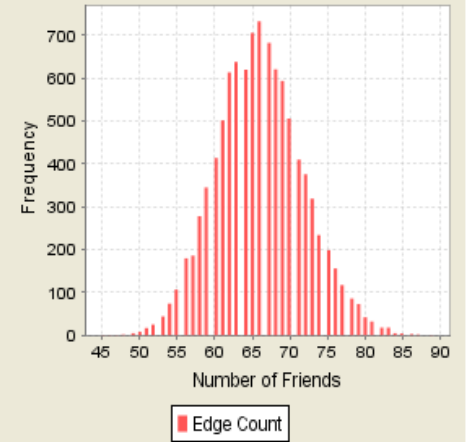
Hops Frequency



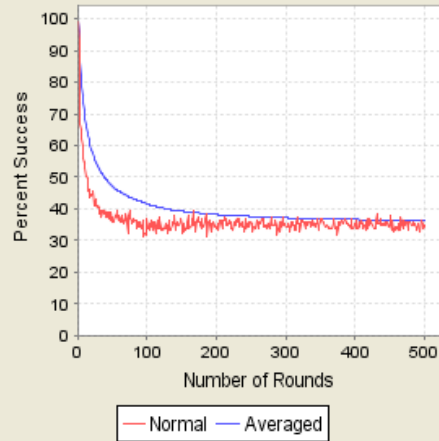
Edge Distance Frequency



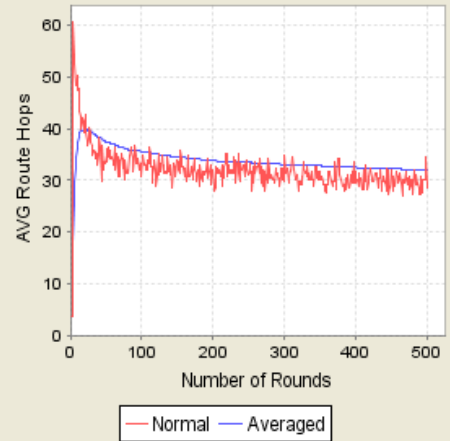
Edge Count Frequency



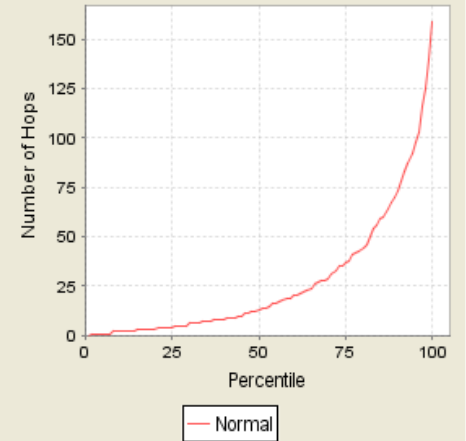
Success Rate



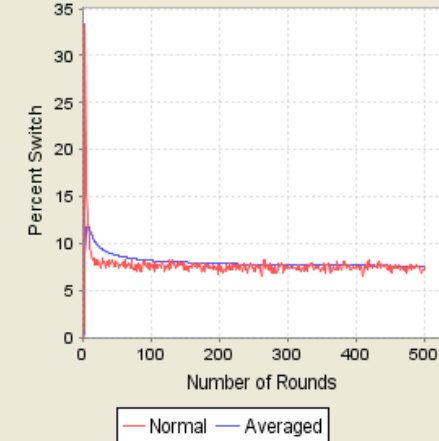
AVG Routing Hops



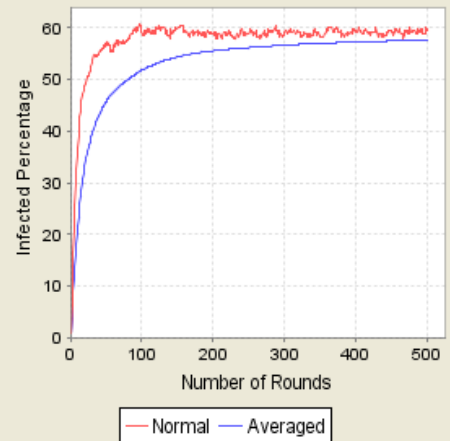
Routing Hops Percentiles



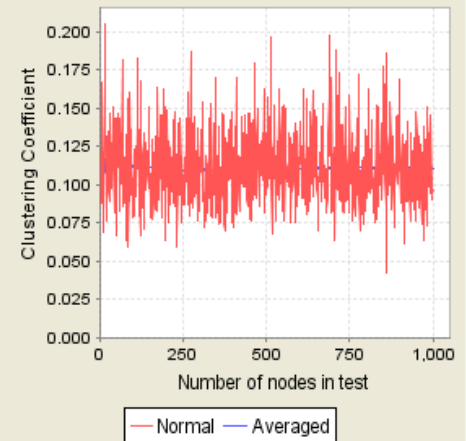
Switch Percentage

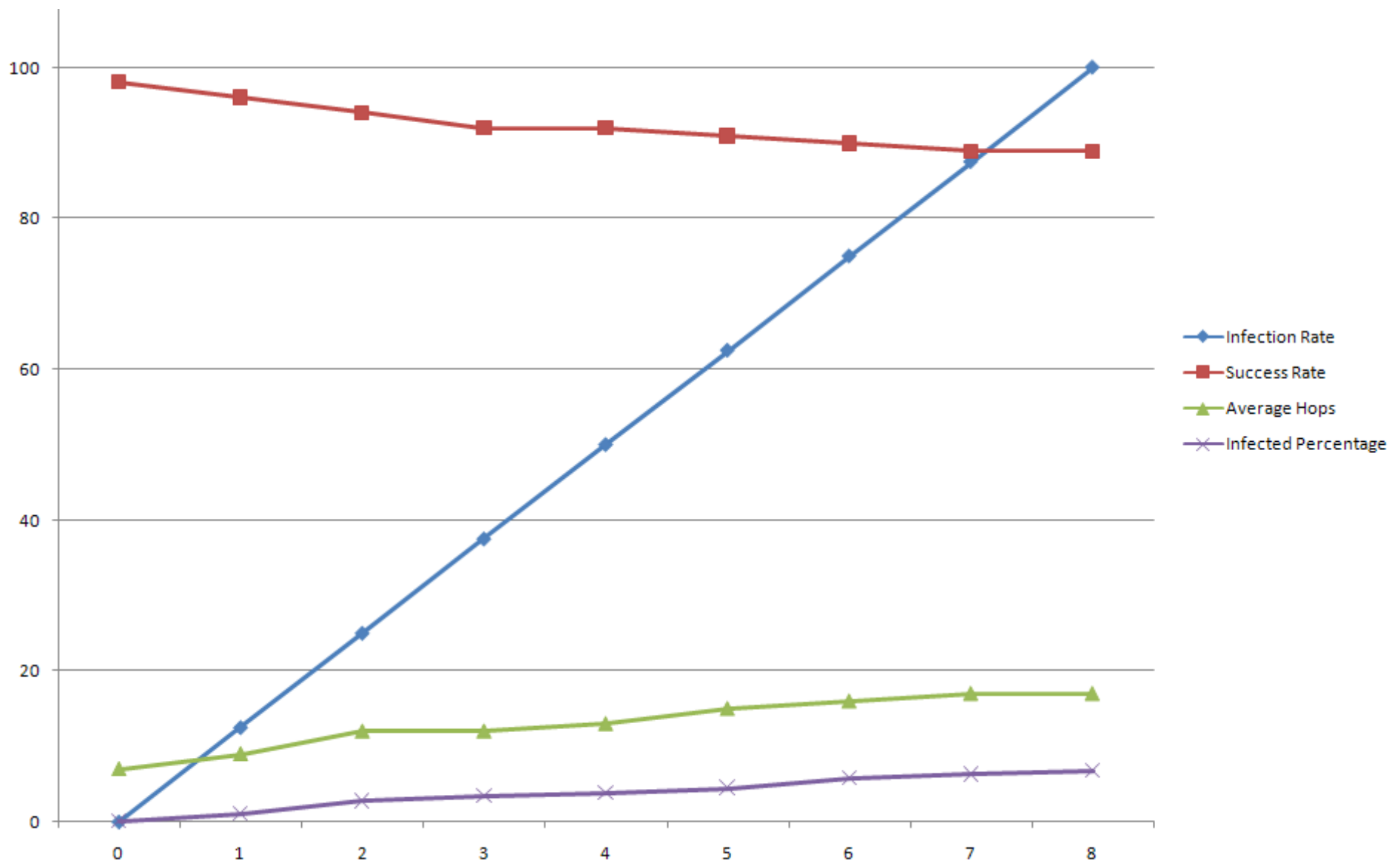


Infection Percentage

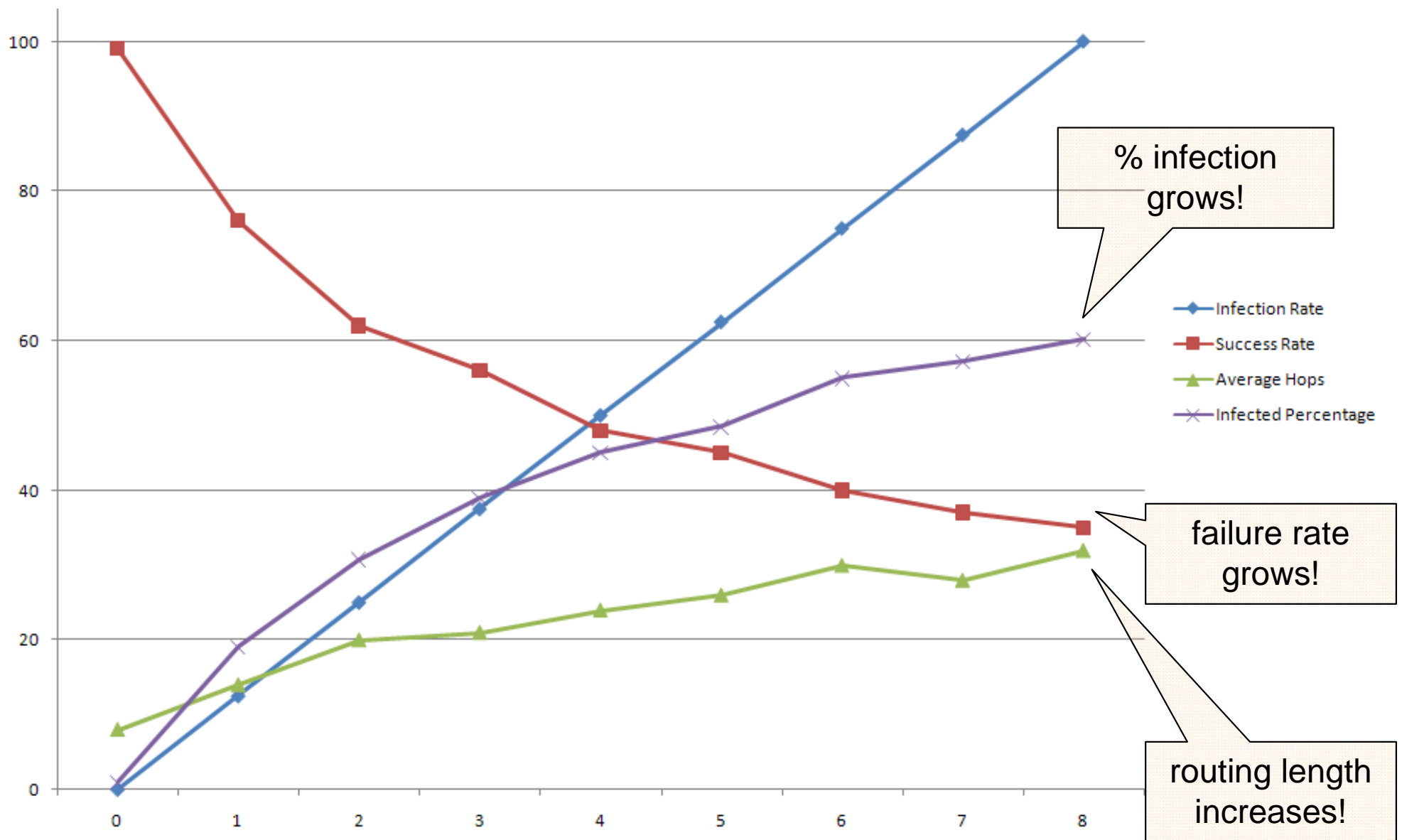


Clustering Coefficient





Infection percentage for 0.1 % malicious nodes.
 N = 10,000 and **10** malicious nodes.



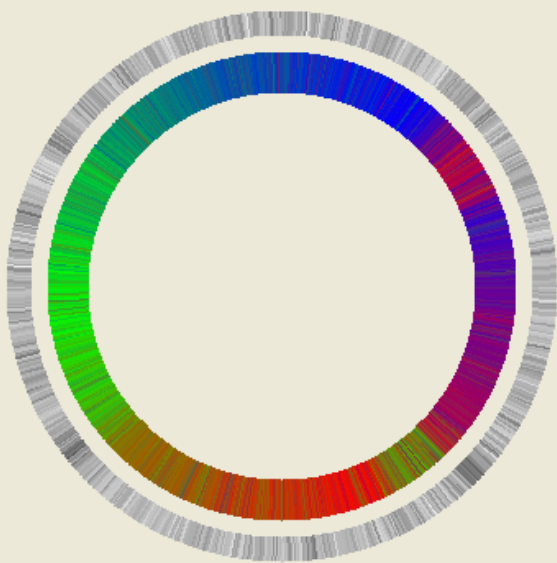
Infection percentage for 1 % malicious nodes.
 N = 10,000 and **100** malicious nodes.

Self-protection mechanisms

- Check swapping frequency?
 - Also lower the convergence
- Limit the number of swaps per node?
 - A malicious node can swap as many as the number of their neighbor per iterations
- Observing the distributions of positions among neighbors and decide something strange based on the size of the network?
 - Hard to precisely count the network size
- Periodic Reset of the positions?
 - Limit the success rate to 70-80% (never get to the maximum success rate)
 - The switching percentage is quite high all the time
- Verifying switches on every node before switching?
 - The malicious nodes can collude with other malicious nodes

Partial Restart Positions

- To avoid being concentrated to a particular positions, each node will randomize its position with probability 0.01 each round
 - The probability is derived from the fact that 100N already made the network converged
- Compared to without restart, with restart:
 - The success rate goes up from 36% to 86%
 - It will never reach the best success rate of 99% because of restart
 - The average hops goes down from 30 to 22
 - Constant switch percentage goes down from 8% to 4%
 - Infection percentage goes down from 60% to 5%



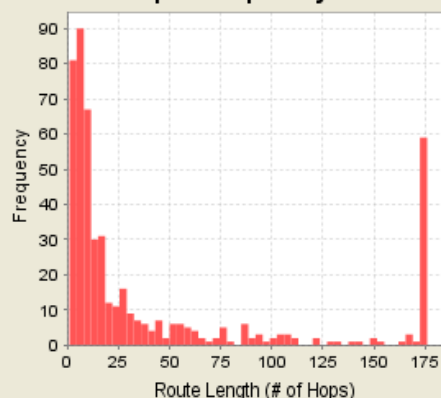
```
drvis.network.generator.SandbergNetwork
infection_rate = 8
malicious_nodes = 100
network_edges_size = -1
network_size = 10000
network_type = undirected
sparse_positions = true
super_peer_added_links = 100
super_peer_random_links = true
super_peer_size = 0
```

```
drvis.network.optimizer.Switching
aggressive_malicious = false
end_nodes_switch_only = true
fix_position_on_dup = false
opt_iterations = 6000
opt_test_iterations = 3
restart_probability = 0.01
switching_power = 1.00
switching_random_walk = 8
```

```
drvis.network.router.Sandberg
fix_position_on_dup = false
routing_failure = -1
routing_tests = 500
use_base_connections = false
```

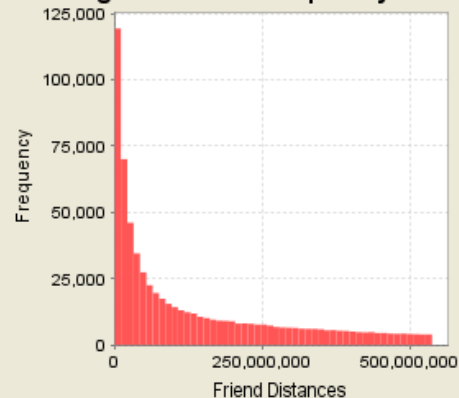
Round: Tailing

Hops Frequency



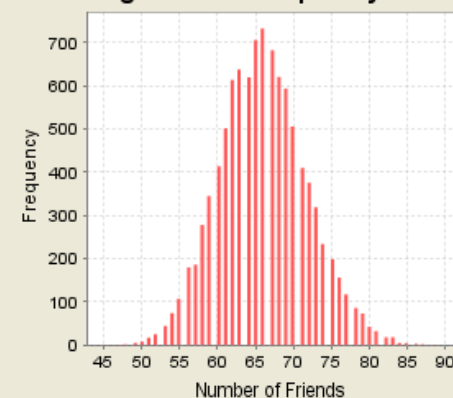
Hops Frequency

Edge Distance Frequency



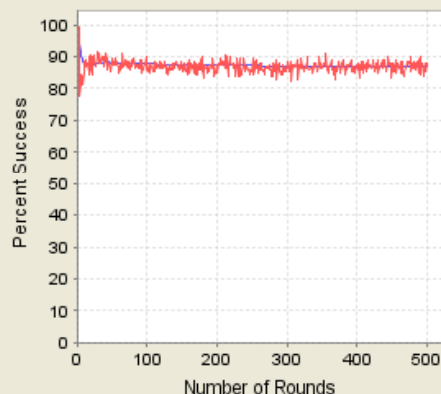
Edge Distances

Edge Count Frequency



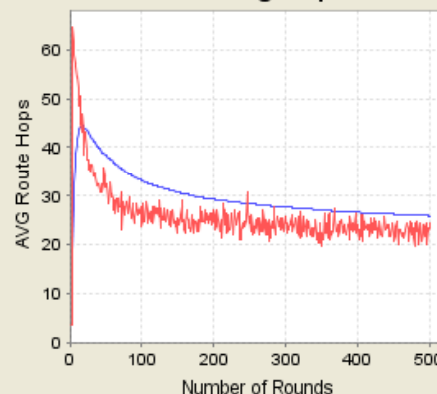
Edge Count

Success Rate



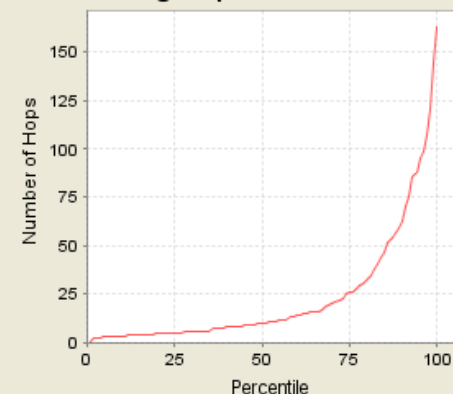
Normal Averaged

AVG Routing Hops



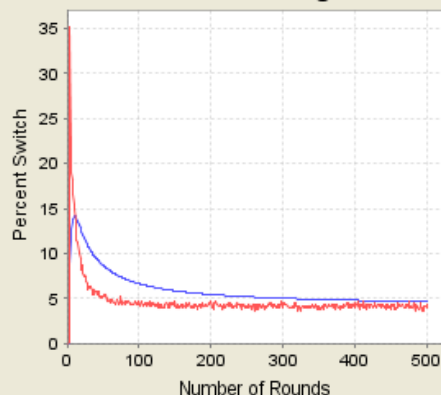
Normal Averaged

Routing Hops Percentiles



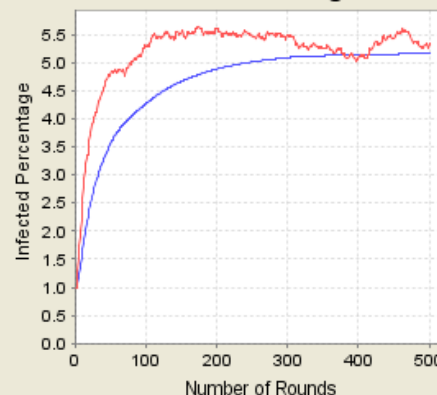
Normal

Switch Percentage



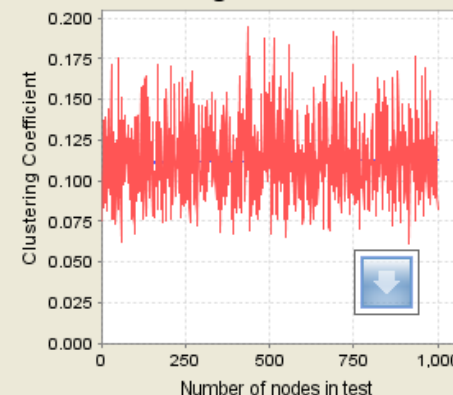
Normal Averaged

Infection Percentage



Normal Averaged

Clustering Coefficient



Normal Averaged

- Still work in progress
 - unclear how much self-protection can get under very weak assumptions
 - maybe stronger assumptions needed
 - data issues not investigated yet