Dissertation
Towards a Framework for DHT Distributed Computing

Andrew Rosen

Georgia State University

May 13th, 2016

Table of Contents

Georgia State University

Objective

- Our objective is to create a generalized framework for distributed computing using Distributed Hash Tables.

Objective

- Our objective is to create a generalized framework for distributed computing using Distributed Hash Tables.

Or

Objective

- Our objective is to create a generalized framework for distributed computing using Distributed Hash Tables.

Or

We want to build a completely decentralized distributed computing framework.

Challenges of Distributed Computing

Distributed Computing platforms experience these challenges:

Scalability As the network grows, more resources are spent on maintaining and organizing the network.

Challenges of Distributed Computing

Distributed Computing platforms experience these challenges:

Scalability As the network grows, more resources are spent on maintaining and organizing the network.

Fault-Tolerance As more machines join the network, there is an increased risk of failure.

Challenges of Distributed Computing

Distributed Computing platforms experience these challenges:

Scalability As the network grows, more resources are spent on maintaining and organizing the network.

Fault-Tolerance As more machines join the network, there is an increased risk of failure.

Load-Balancing Tasks need to be evenly distributed among all the workers.

Distributed Key/Value Stores

Distributed Hash Tables are mechanisms for storing values associated with certain keys.

- Values, such as filenames, data, or IP/port combinations are associated with keys.
- These keys are generated by taking the hash of the value.
- We can get the value for a certain key by asking any node in the network.

Strengths of DHTs

DHTs are designed for large P2P applications, which means they need to be (and are):

- Scalable
- Fault-Tolerant
- Load-Balancing

Table of Contents

Georgia State University

Required Attributes of DHT

One of our contributions is that we reduced a DHT down into its core components. A DHT needs:

- A distance function.
- A closeness or ownership definition.
- A Peer management strategy.

Terms and Variables

- Network size is n nodes.
- Keys and IDs are m bit hashes, usually SHA1 with 160 bits.
- Peerlists are made up of:

  Short Peers The neighboring nodes that define the network's topology.

  Long Peers Routing shortcuts.

Chord

- Ring Topology
- Short Peers: predecessor and successor in the ring.
- Responsible for keys between their predecessor and themselves.
- Long Peers: $\log n$ nodes, where the node at index i in the peerlist is

$$root(r + 2^{i-1} \mod m), 1 < i < m$$

A Chord Network

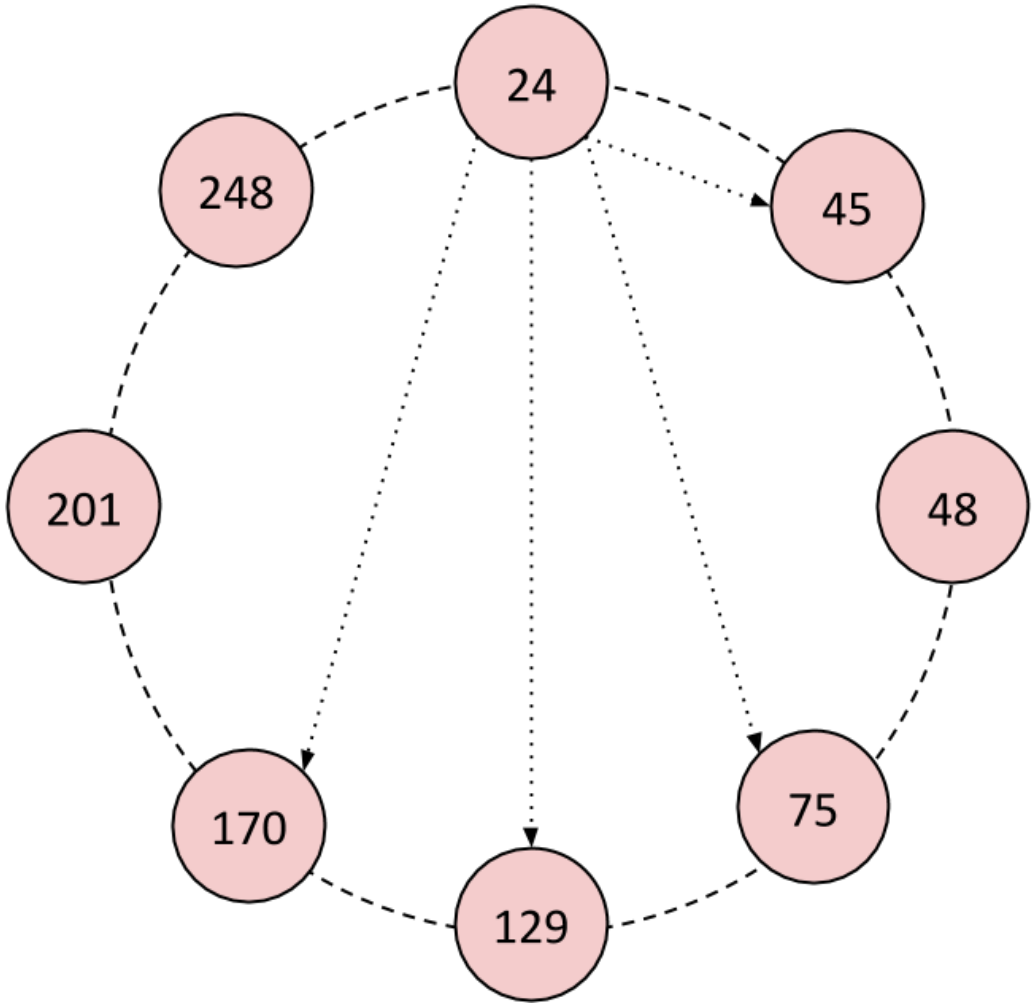


Figure: An 8-node Chord ring where $m = 8$. Node 24's long peers are shown.

Table of Contents

Overarching Theme

My research has been focused on:

- Abstracting out DHTs.
- Distributed computation using DHTs.

ChordReduce

Objective:

- Create an abstract MapReduce framework.
- Create a completely decentralized, fault-tolerant, distributed computing framework.

Results:

- We succeeded.
- Our network could handle nodes leaving, as well as assign joining nodes work.
- We found anomalous results with Churn.

VHash and DGVH

What was it:

- Made a Voronoi-based DHT to operate in multiple dimensions.
- Wanted to embed information, such as inter-node latency, into the position in the overlay.
- Voronoi computations required a greedy heuristic.
- More detail later.

Results:

- New concept: nodes that move their position.
- DGVH.
- We started to think how about how can we completely abstract any DHT topology into a Voronoi/Delaunay relationship.

Georgia State University

D^3NS

- Create a completely decentralized, distributed, DNS.
- Backwards compatible and completely reliable to the user.
- Used Blockchains and UrDHT.

Sybil Attack Analysis

- Studied the amount of resources needed to perform a Sybil attack.
- Examined how difficult it was for nodes to inject a Sybil into a specific location.

Table of Contents

Introduction

- Built on our DGVH and VHash projects.
- Create an abstract model of a DHT based on Voronoi/Delaunay relationship.
- Can be used as a bootstrapping network for other distributed systems.
- Can emulate the topology of other DHTs.

Goals

VHash and DGVH sprung from two related ideas:

- We wanted a way be able optimize latency by embedding it into the routing overlay.

Goals

VHash and DGVH sprung from two related ideas:

- We wanted a way be able optimize latency by embedding it into the routing overlay.
- We wanted to create a DHT based off of Voronoi tessellations. Unfortunately:

Goals

VHash and DGVH sprung from two related ideas:

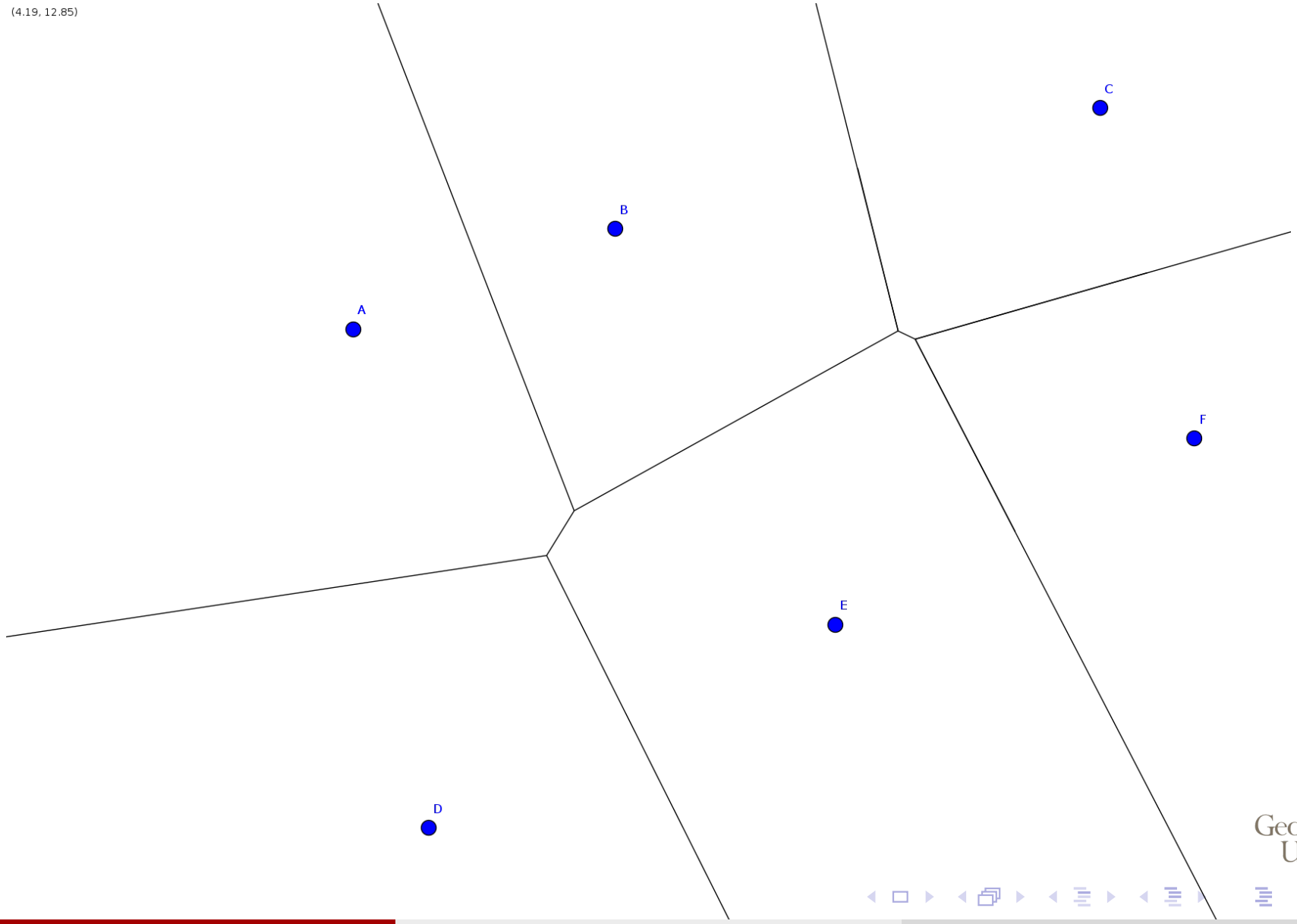
- We wanted a way be able optimize latency by embedding it into the routing overlay.
- We wanted to create a DHT based off of Voronoi tessellations. Unfortunately:
  - Distributed algorithms for this problem don't really exist.

Goals

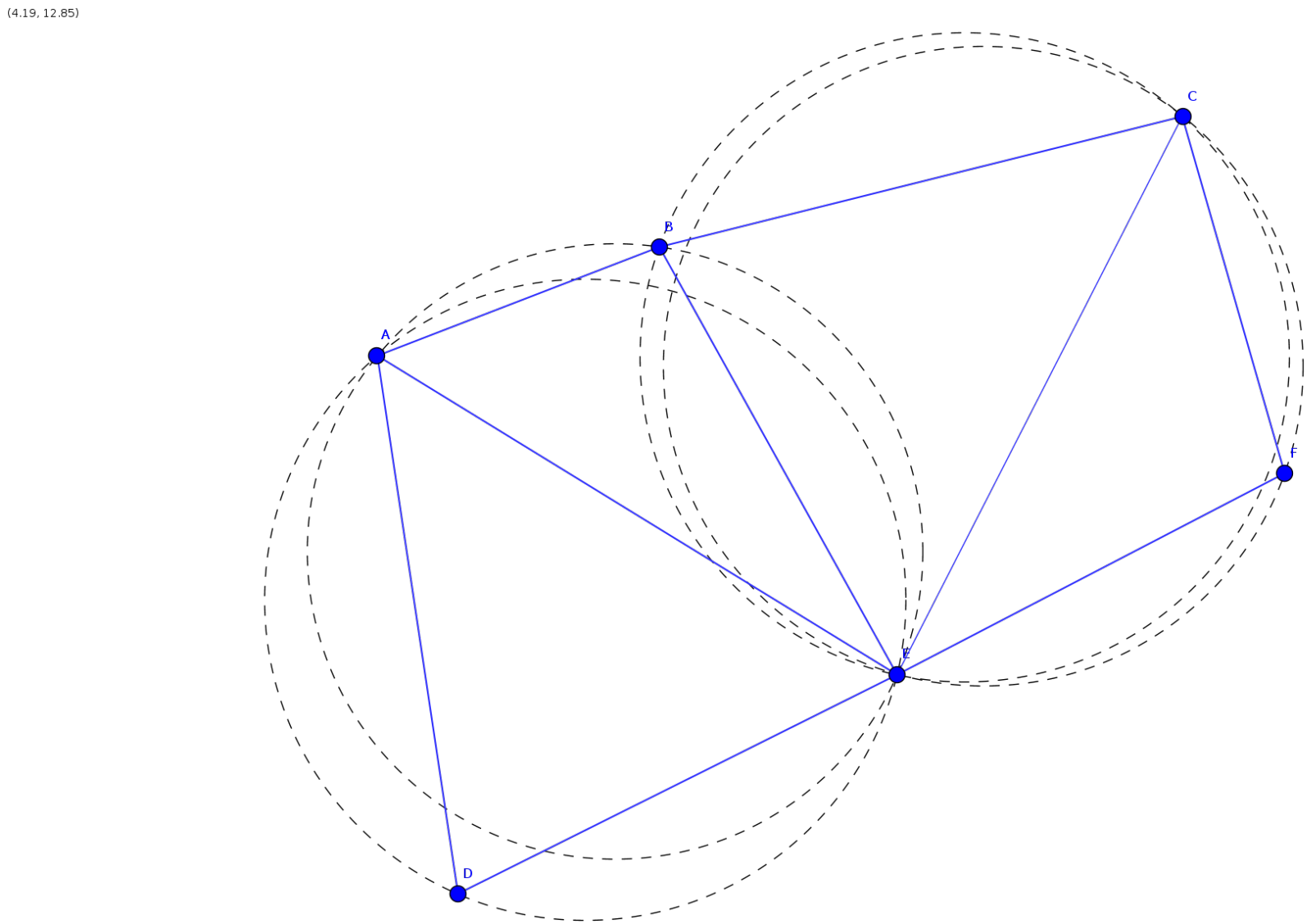
VHash and DGVH sprung from two related ideas:

- We wanted a way be able optimize latency by embedding it into the routing overlay.
- We wanted to create a DHT based off of Voronoi tessellations. Unfortunately:
  - Distributed algorithms for this problem don't really exist.
  - Existing approximation algorithms were unsuitable.

Voronoi Tesselation



Delaunay Triangulation



DHT and Voronoi Relationship

- We can view DHTs in terms of Voronoi tessellation and Delaunay triangulation.
  - The set of keys the node is responsible for is its Voronoi region.
  - The nodes neighbors are its Delaunay neighbors.

Distributed Greedy Voronoi Heuristic

- Assumption: The majority of Delaunay links cross the corresponding Voronoi edges.
- We can test if the midpoint between two potentially connecting nodes is on the edge of the Voronoi region.
- This intuition fails if the midpoint between two nodes does not fall on their Voronoi edge.

DGVH Heuristic

1: Given node n and its list of *candidates*.
2: *peers* $\leftarrow$ empty set that will contain n's one-hop peers
3: Sort *candidates* in ascending order by each node's distance to n
4: Remove the first member of *candidates* and add it to *peers*
5: **for all** c in *candidates* **do**
6: **if** Any node in *peers* is closer to c than n **then**
7: Reject c as a peer
8: **else**
9: Remove c from *candidates*
10: Add c to *peers*
11: **end if**
12: **end for**

DGVH Time Complexity

For k candidates, the cost is:

$$k \cdot \lg(k) + k^2 \text{ distances}$$

However, the expected maximum for k is $\Theta(\frac{\log n}{\log \log n})$, which gives an expected maximum cost of

$$O(\frac{\log^2 n}{\log^2 \log n})$$

or

$$O(\frac{\log^3 n}{\log^3 \log n})$$

Depending on whether we gossip with a single neighbor or all neighbors.

Summary

- DGVH is simple approximation for Delaunay Triangulation that guarantees a fully connected graph.
- Creates a fully connected subset of the Delaunay Triangulation.
- A DHT using DGVH can optimize over a metric such as latency and achieve superior routing speeds as a result.
- We built VHash to test this.
- UrDHT fully implements this and use

Georgia State University

What is UrDHT

- Abstract framework for implementing DHTs or various topologies
- Three Components
  - Storage
  - Networking
  - Logic
    - Protocol
    - Space Math

The Protocol

- Consists of
  - Node Information
  - Short peers
  - Long Peers
  - The functions we use
- Replaced `lookup` with `seek`
- Maintenance is gossip based, using functions provided by the Space Math
- Short peer selection is done by DGVH by default
- Once short peers are selected, `handleLongPeers` is called

Space Math

- Defines the DHT topology
- Requires a way to generate short peers and choose long peers

Space Functions

- `idToPoint` takes key, maps it to a point in space.
- `distance` outputs the shortest distance between a and b
- `getDelaunayPeers` which is DGVH.
- `getClosest`
- `handleLongPeers`

DHTs To Implement

We demonstrated how to implement:

- Chord / Symphony
- Kademlia
- ZHT

Setup

- Tested four different topologies
  - Chord
  - Kademlia
  - Euclidean
  - Hyperbolic
- We create a 500 node network, adding one node at a time and completing a maintenance cycle.

Data Collected

- Reachability.
- The average degree of the network.
- The worst case degree of the network.
- The average number of hops between nodes using greedy routing.
- The diameter of the network.

Chord Degree

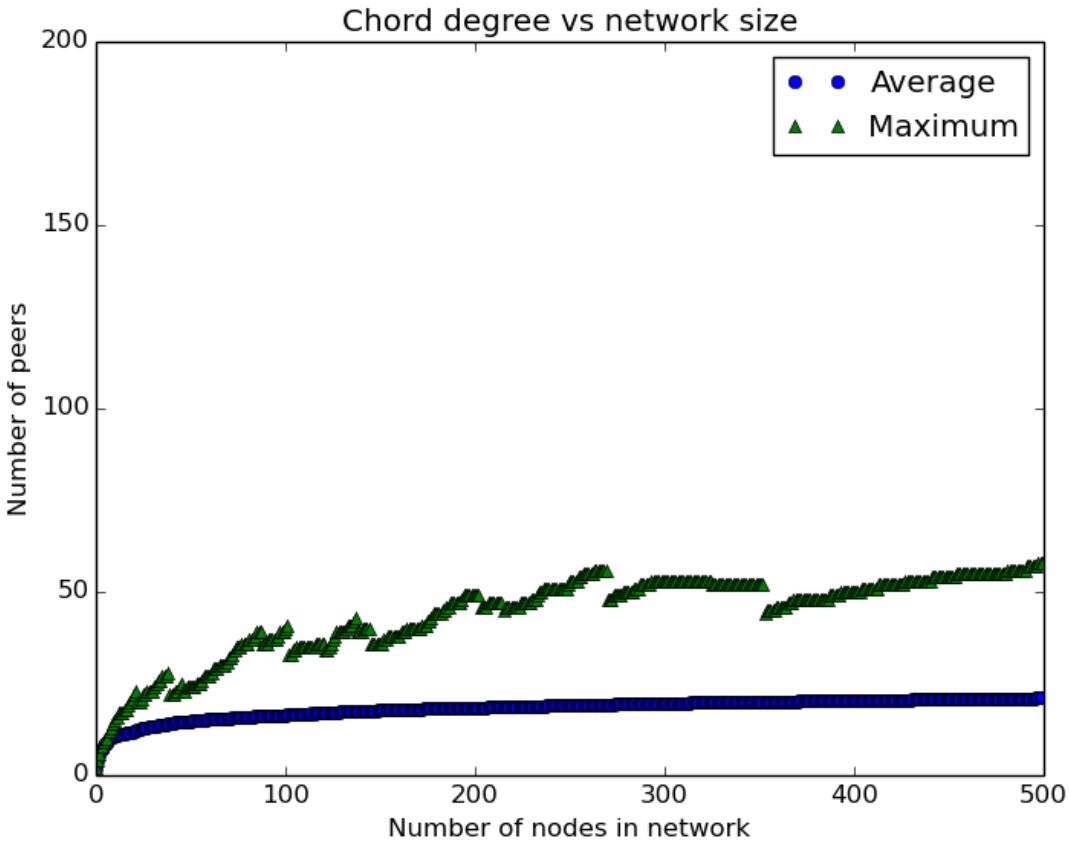


Figure: Average and max degree of nodes in Chord. This Chord network utilized a 120 bit hash and thus degree is bound at 122 (full fingers, predecessor and successor).

Chord Distance

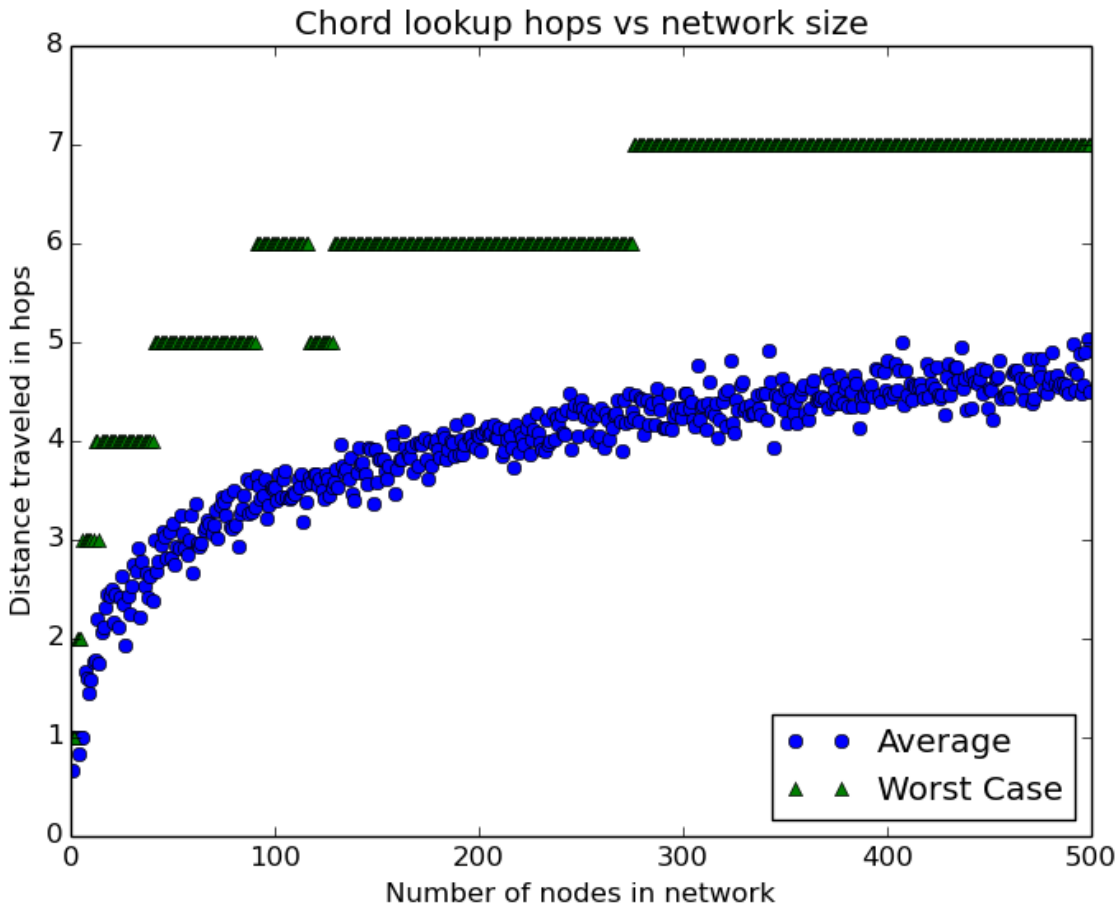


Figure: This is the number hops required for a greedy routed lookup in Chord. The average lookup between two nodes follows the expected logarithmic curve.

Kademlia Degree

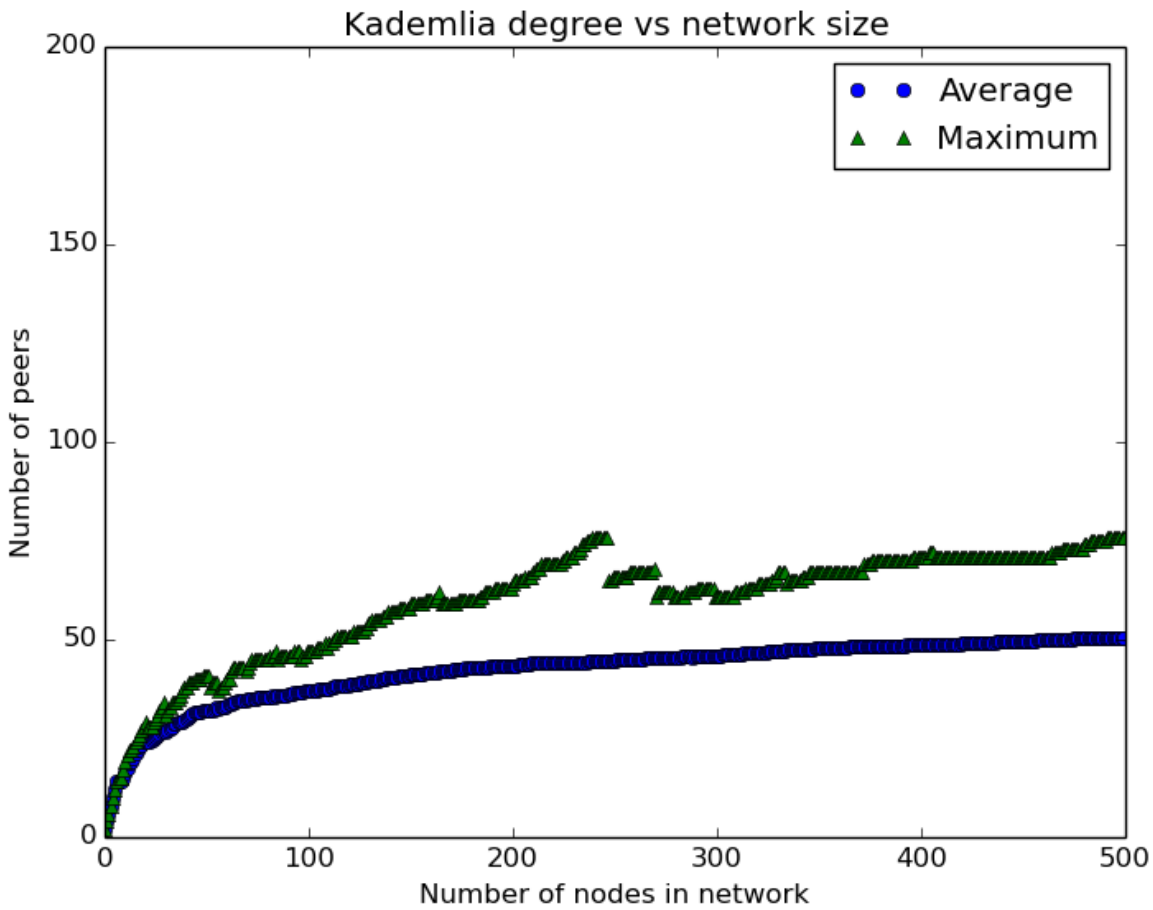


Figure: This is the average and maximum degree of nodes in Kademlia. Both the maximum degree and average degree are $O(\log n)$.

Kademlia Distance

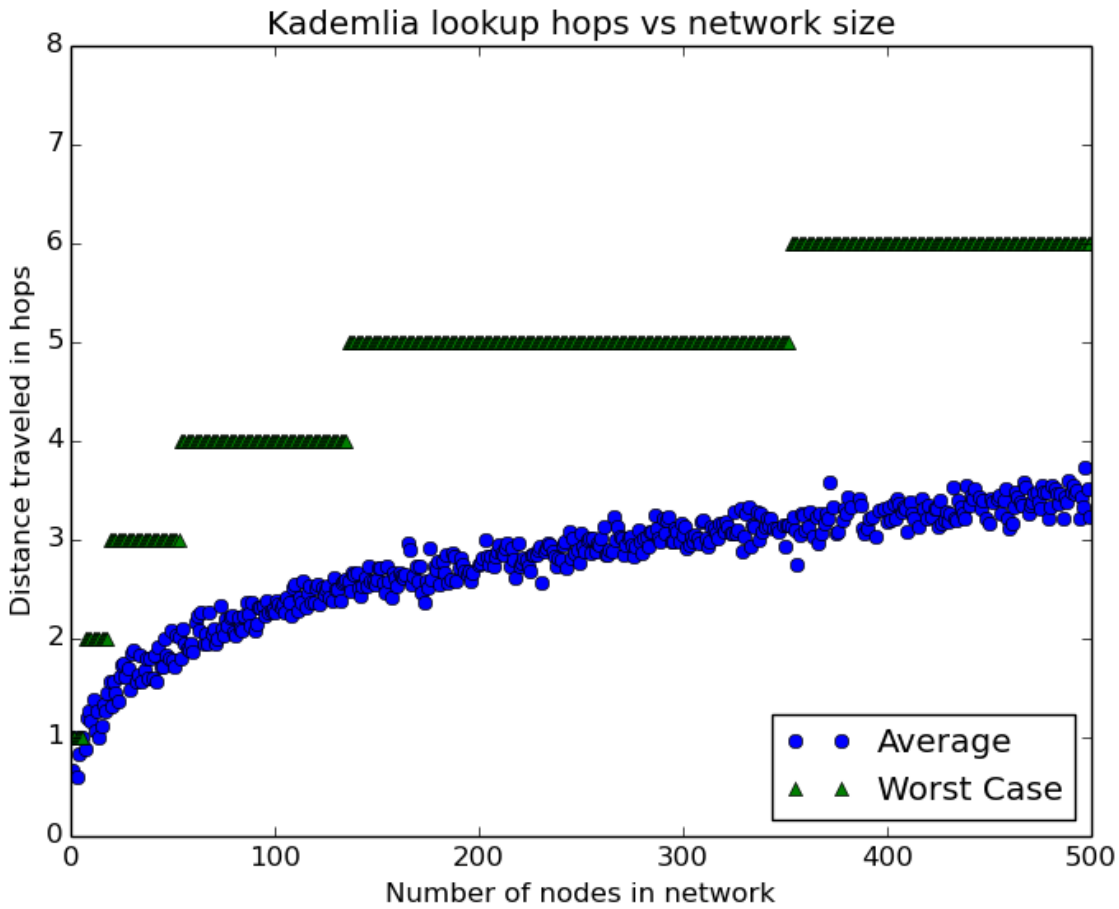


Figure: Much like Chord, the average degree follows a logarithmic curve, reaching an average distance of about 3 hops with 500 nodes in the network.

Euclidean Degree

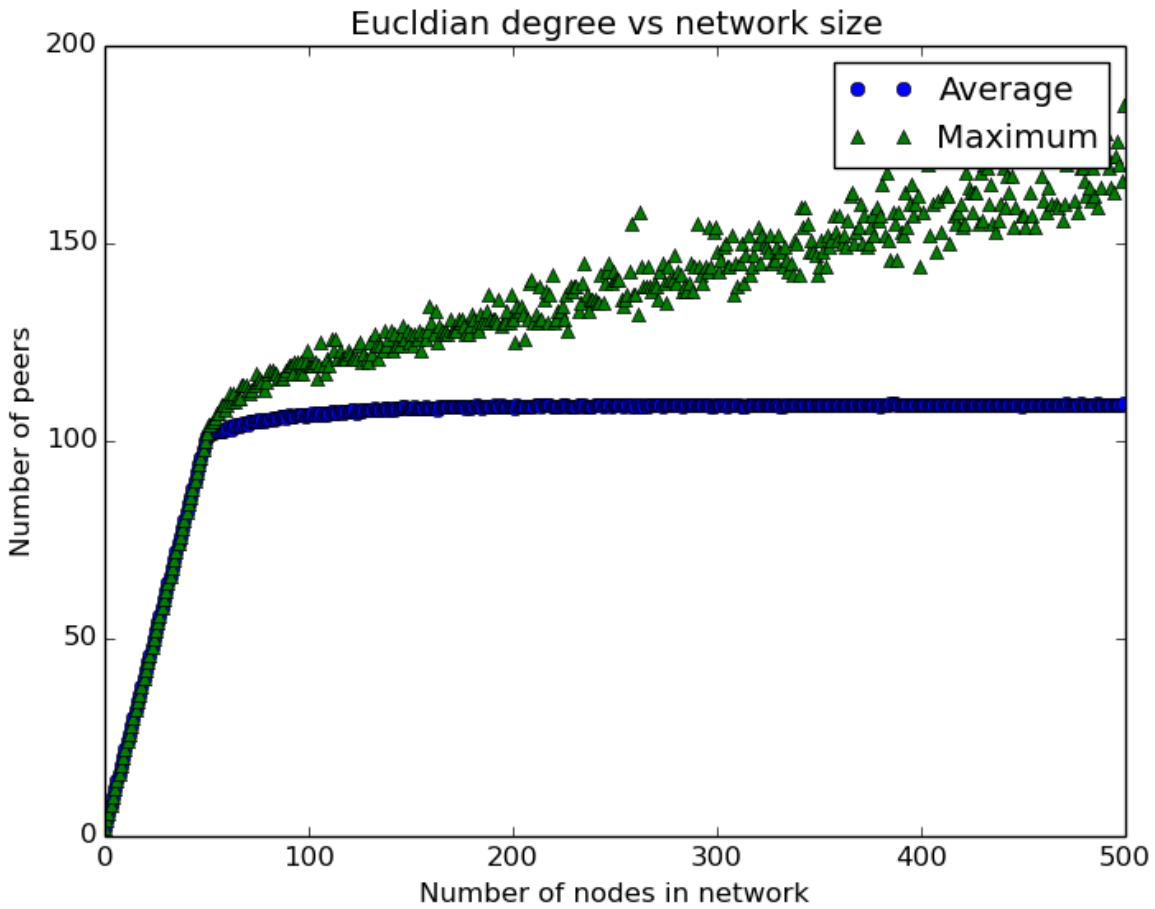


Figure: Because the long peers increase linearly to the maximum value (49), degree initially rises quickly and then grows more slowly as the number of long peers ceases to grow and the size short peers increases with network size.

Euclidean Distance

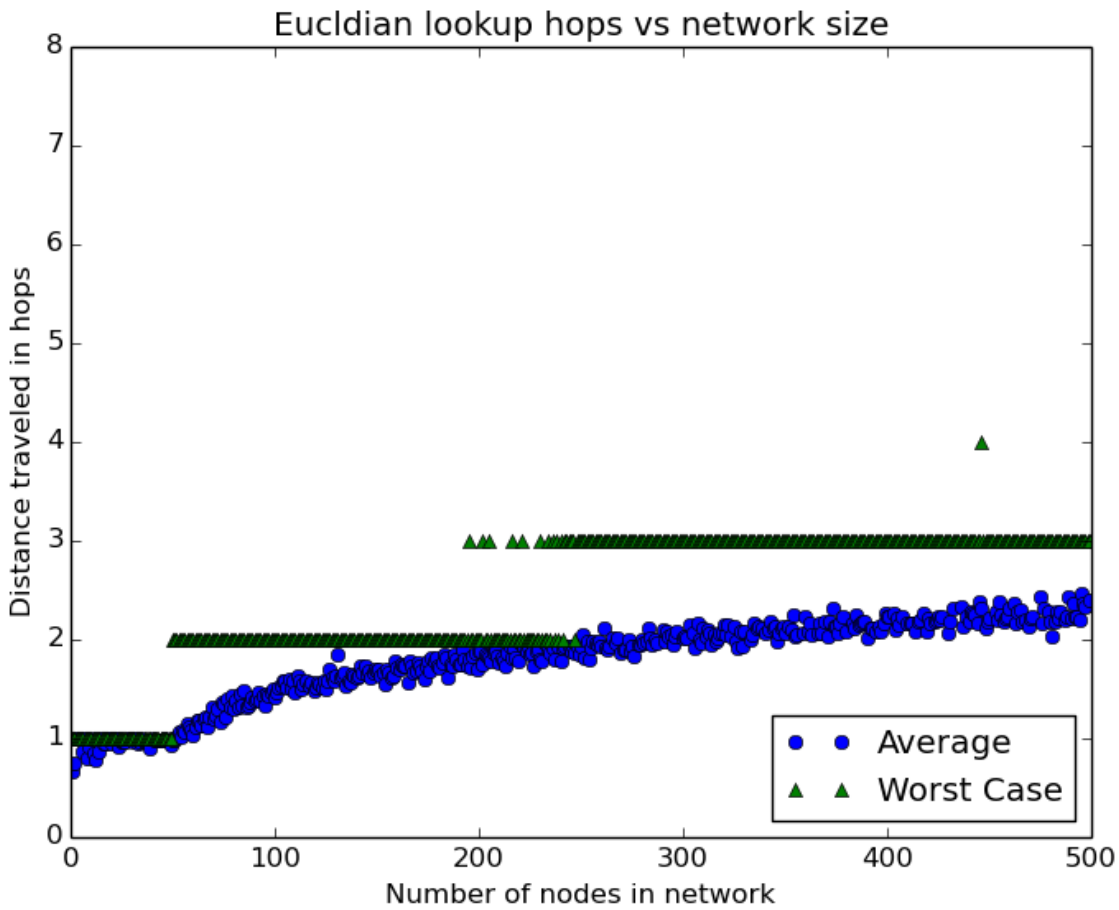


Figure: The inter-node distance stays constant at 1 until long peers are filled, then rises at the rate of a randomly connected network due to the distribution of long peers selected.

Hyperbolic Degree

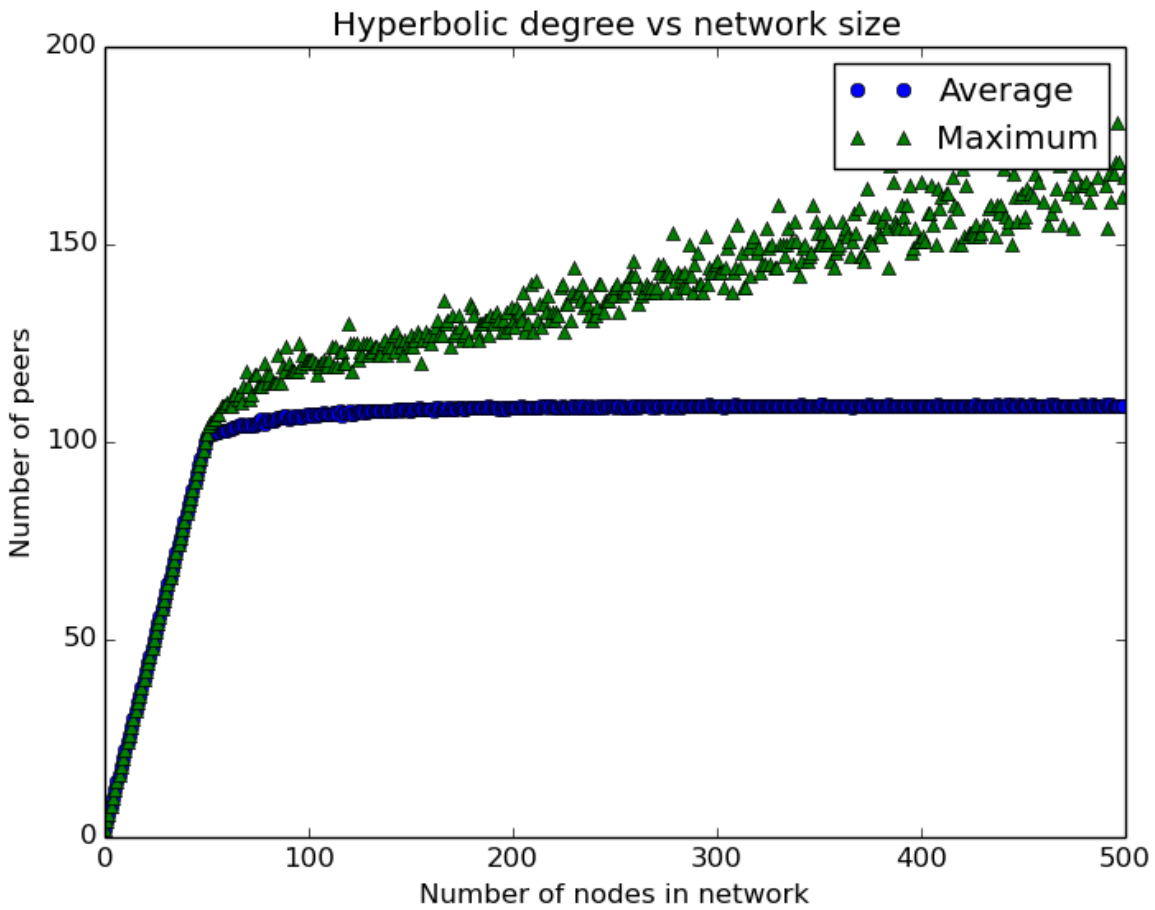


Figure: The Hyperbolic network uses the same long and short peer strategies to the Euclidean network, and thus shows similar results.

Hyperbolic Distance

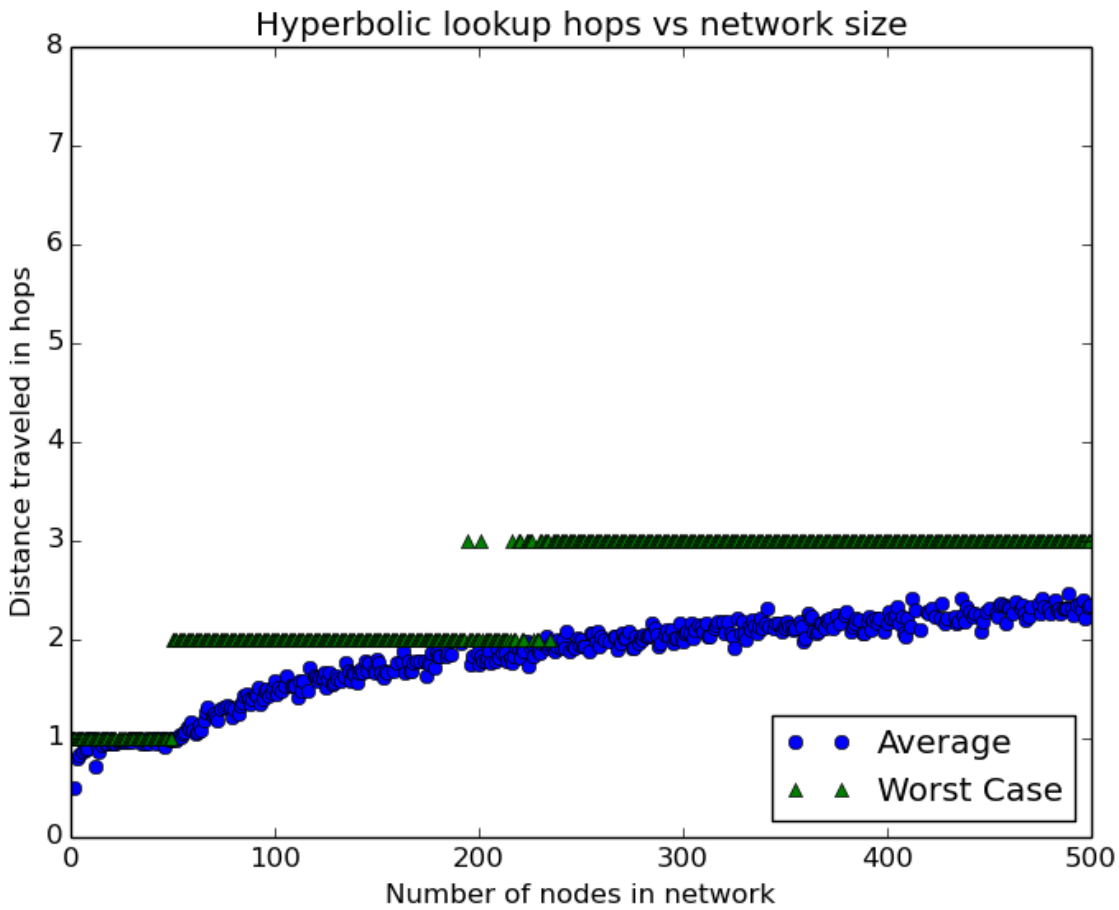


Figure: Like the Euclidean Geometry, our Poincarè disc based topology has much shorter maximum and average distances.

Conclusion

- UrDHT abstracts DHTs.
- UrDHT can emulate the topology and performance of various DHTs.
- The maintenance algorithm can handle nodes moving in the overlay.
- Our abstract implementation allows for both the easy creation of new DHTs as well as the simulation of various topologies.

Table of Contents

Georgia State University

Introduction

- In this project, we set out to confirm the results of ChordReduce
- Objectives:
  - Confirm that high levels of churn can help a DHT based computing environment.
  - Develop better strategies than randomness

Strategies

- Churn
- Random Injection
- Neighbor Injection
- Invitation

Distribution in Networks of Different Sizes

Table: The median distribution of tasks (or files) among nodes. We can see the standard deviation is fairly close to the expected mean workload ($\frac{tasks}{nodes}$). Each row is the average of 100 trials. Experiments show there is practically little deviation in the median load of the network.

Nodes	Tasks	Median Workload	σ
1000	100000	69.410	137.27
1000	500000	346.570	499.169
1000	1000000	692.300	996.982
5000	100000	13.810	20.477
5000	500000	69.280	100.344
5000	1000000	138.360	200.564
10000	100000	7.000	10.492
10000	500000	34.550	50.366
10000	1000000	69.180	100.319

Distribution of Work in A DHT

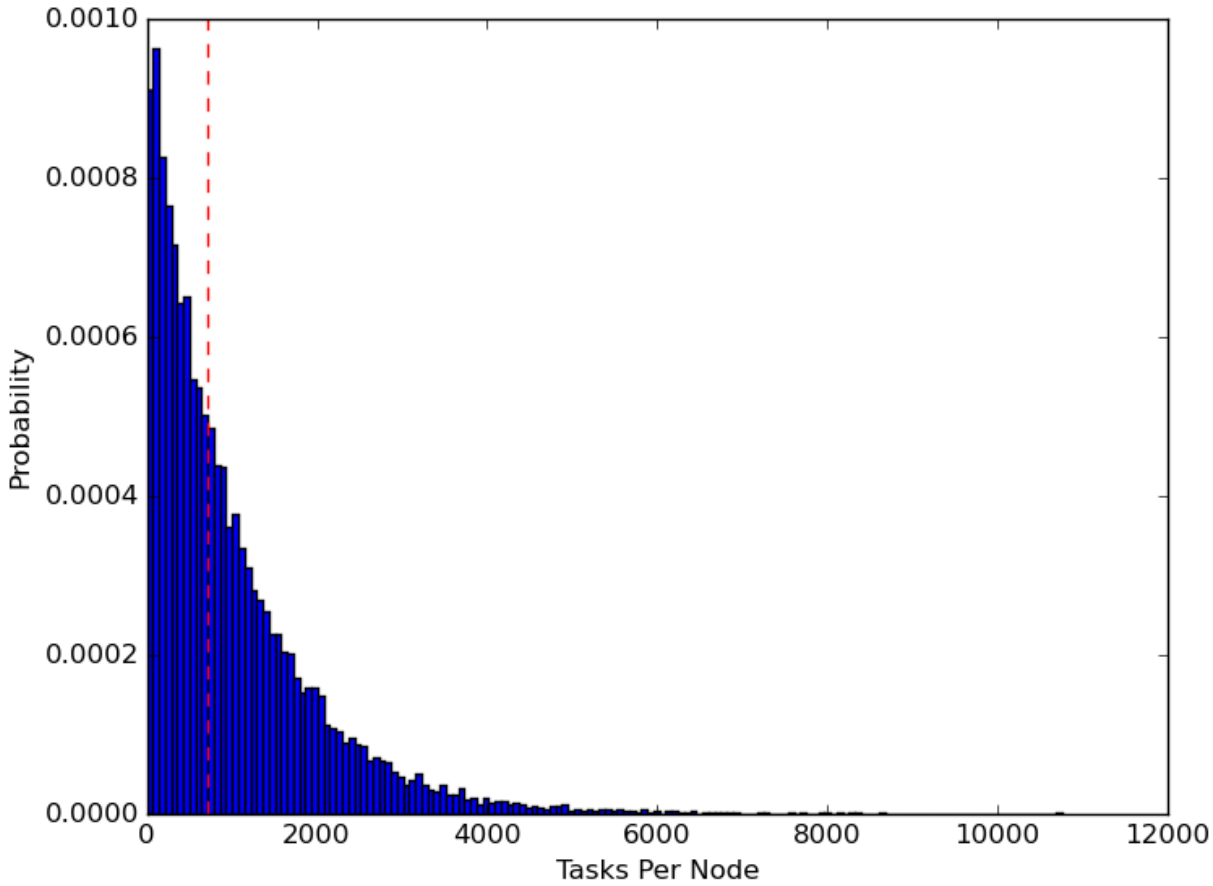


Figure: The probability distribution of workload in a DHT with 1000 nodes and 1,000,000 tasks or files. The vertical dashed line designates the median.

Distribution of Work in Chord

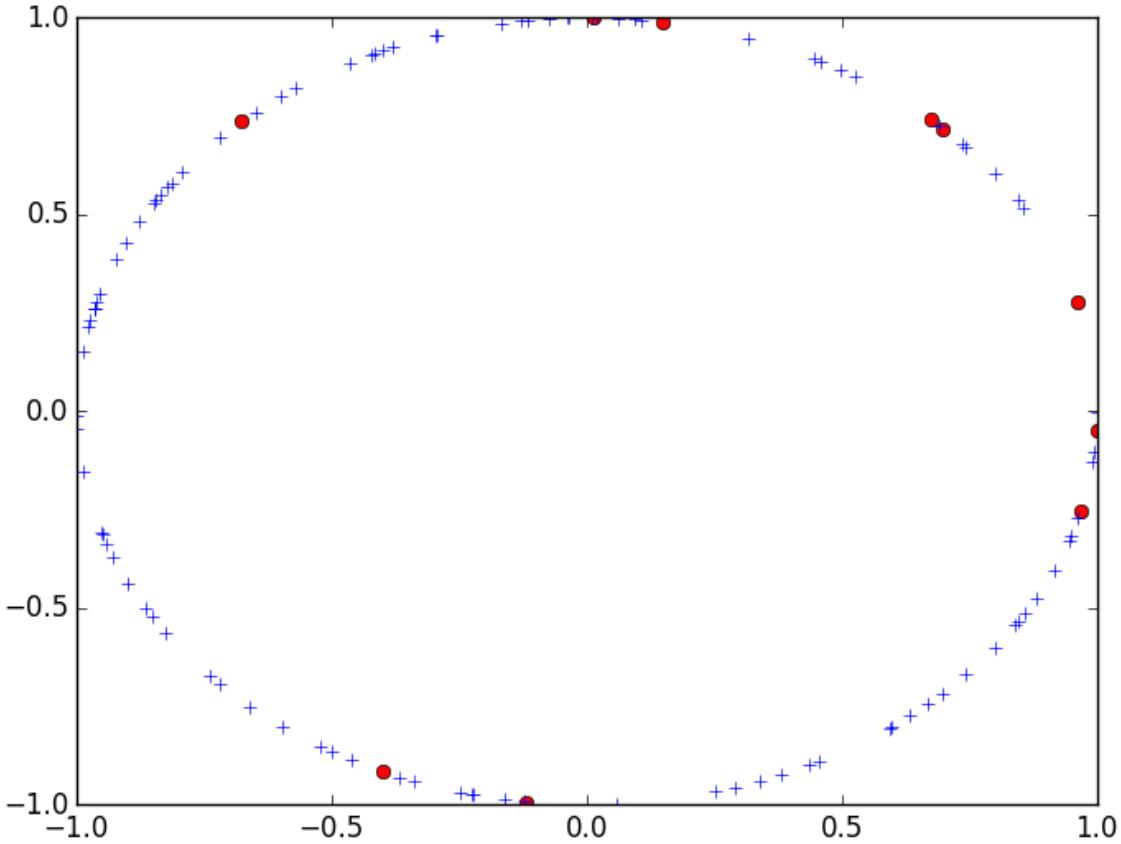


Figure: A visual example of data and nodes in a Chord DHT with 10 nodes (represented by red circles) and 100 tasks (blue pluses).

Distribution of Work in Chord with Evenly Distributed Nodes

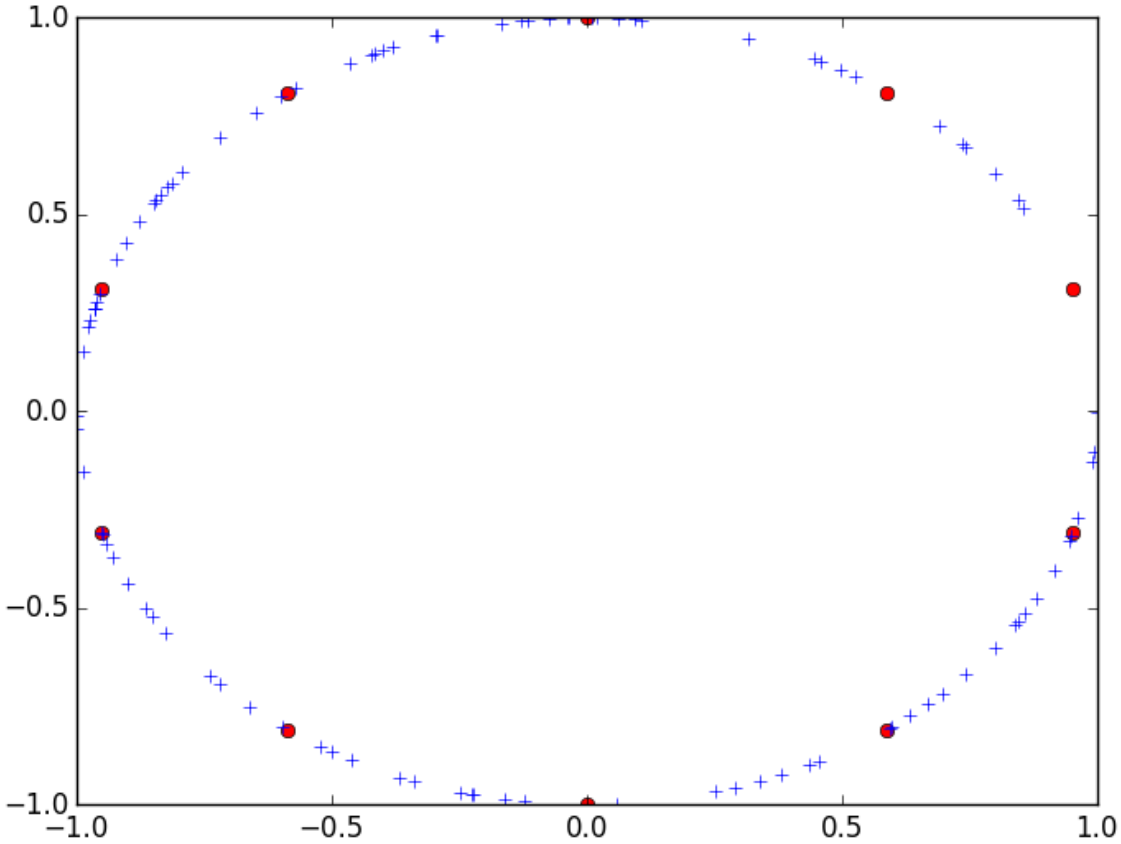


Figure: A visual example of data and nodes in a Chord DHT with 10 evenly distributed nodes (represented by red circles) and 100 tasks (blue pluses).

Terms and Assumptions

- Time is measured in ticks.
- A tick is enough time to perform aggressive, reactive maintenance[1]
- Jobs are measured in tasks; each task can correspond to a file or a piece of a file

[1] This has been implemented and tested.

Variables

- Strategy
- Homogeneity
- Work Measurement
- Number of Nodes
- Number of Tasks
- Churn Rate
- Max Sybils or Node Strength
- Sybil Threshold
- Number of Successors

Output

- Ideal Runtime
- Runtime
- Runtime Factor
- Task Distribution

Strategy

- The network load-balances using churn
- `churnRate` chance per tick for each node to leave network
- Pool of potentially joining joins at the same rate

Runtime

Table: Runtime factor of networks of varying sizes and number of tasks, each using the Churn strategy to load-balance. Each result is the average of 100 trials. The networks are homogeneous and each node consumes one task per tick. A runtime of 1 is the ideal and target.

Churn Rate	10^3 nodes, 10^5 tasks	10^3 nodes, 10^6 tasks	100 nodes, 10^4 tasks	100 nodes, 10^5 tasks	100 nodes, 10^6 tasks
0	7.476	7.467	5.043	5.022	5.016
0.0001	7.122	5.732	4.934	4.362	3.077
0.001	6.047	3.674	4.391	3.019	1.863
0.01	3.721	2.104	3.076	1.873	1.309

Churn vs Runtime factor

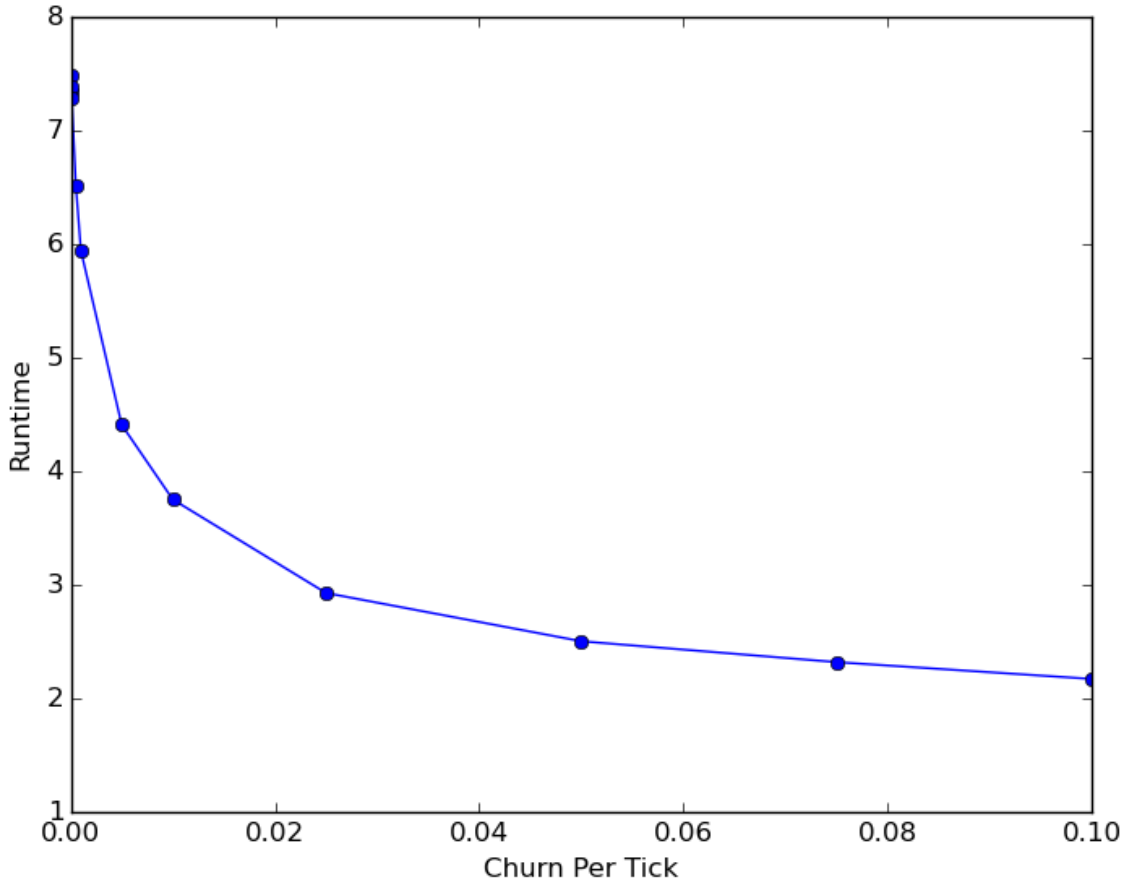


Figure: This graph shows the effect churn has on runtime in a distributed computation.

Base Work Distribution

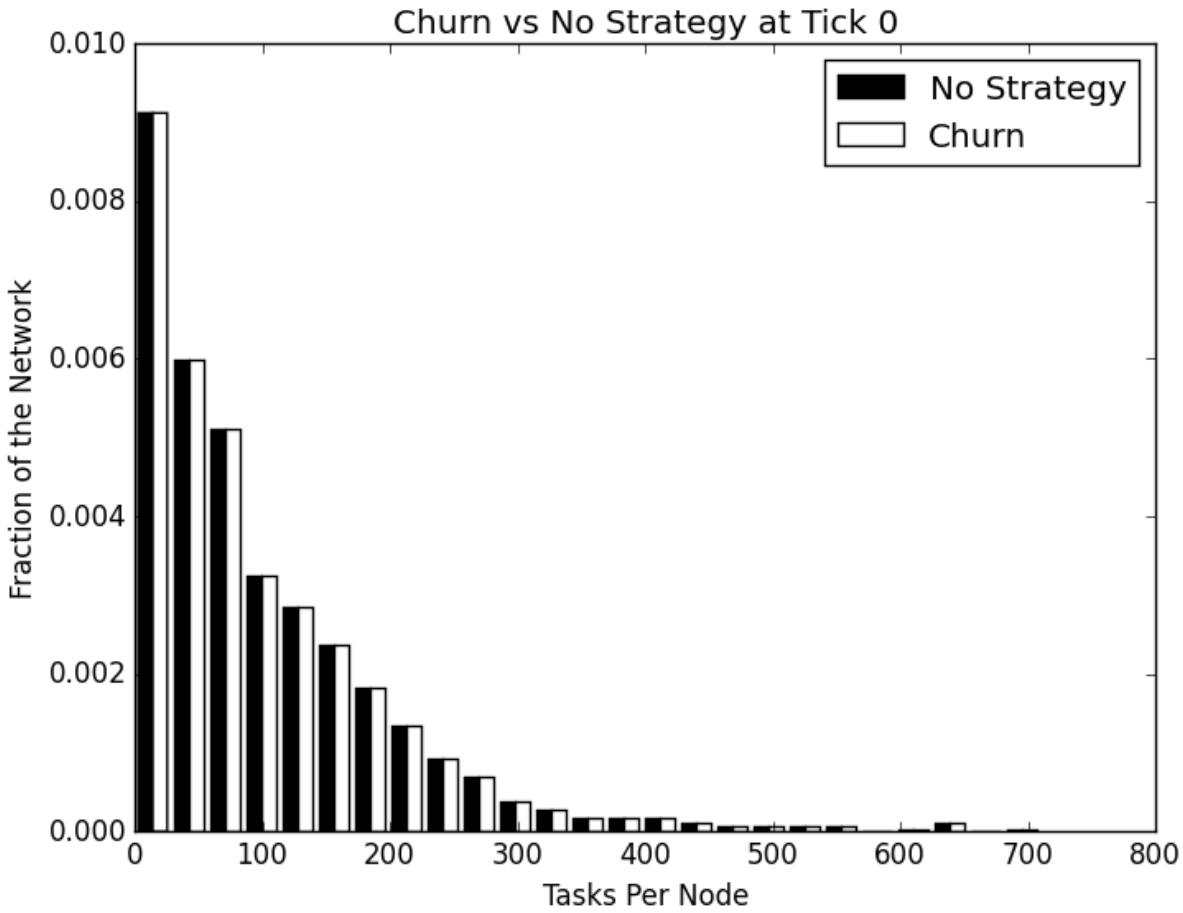


Figure: The initial distribution of the workload in both networks.

Churn Distribution after 35 Ticks

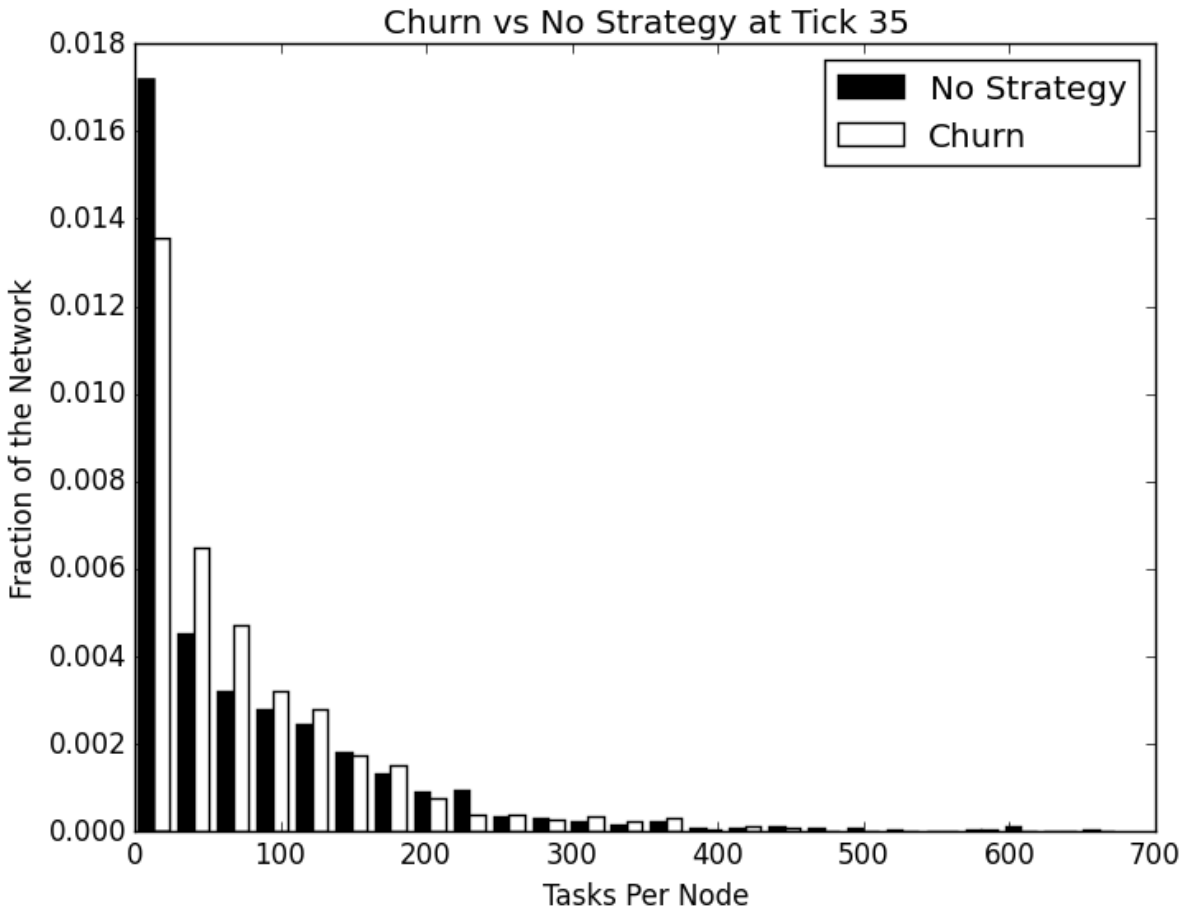


Figure: After 35 ticks

Remarks

- Diminishing returns.
- Maintenance costs can get excessive
- We don't actually have to kill nodes, most of the speedup is from joining.

Strategy

- Nodes with loads $\leq$ `sybilThreshold` create Sybils.
- This check occurs every 5 ticks before work is performed.
- These Sybils are randomly placed.
- Act as a virtual node so the same node essentially exists in multiple locations.
- Sybils are removed if the node that created it has no work.

Effects of Network Size

- A homogeneous, 1000 node/100,000 task network, never has an average runtime factor greater than 1.7
- Minimum was 1.36.
- In the same network with 1,000,000 tasks, these runtimes were 1.25 and 1.12 respectively.
- On average, the 1,000,000 task network had a runtime factor 0.82 less than the 100,000 task network.

Random Injection vs No Strategy After 35 ticks

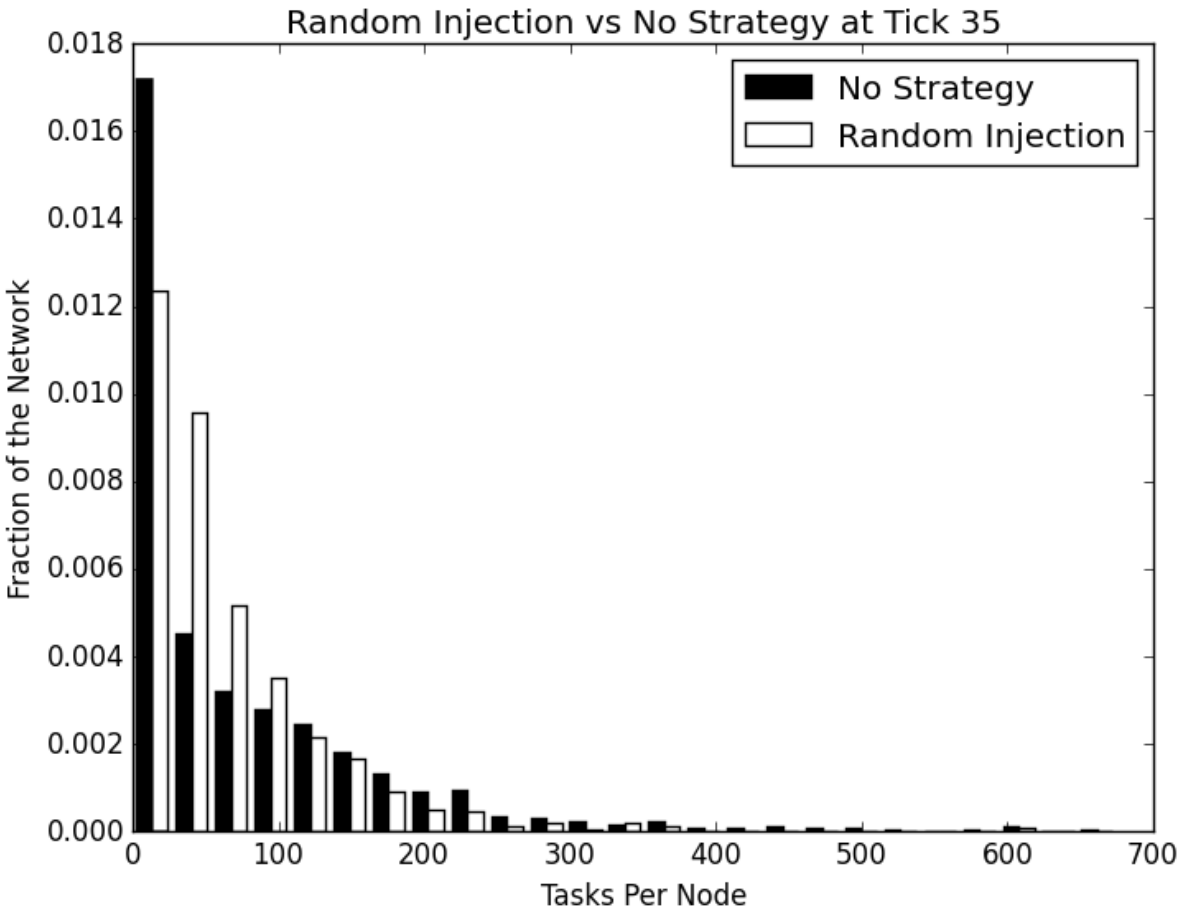


Figure: The networks after 35 ticks.

Random Injection in a Heterogeneous Network

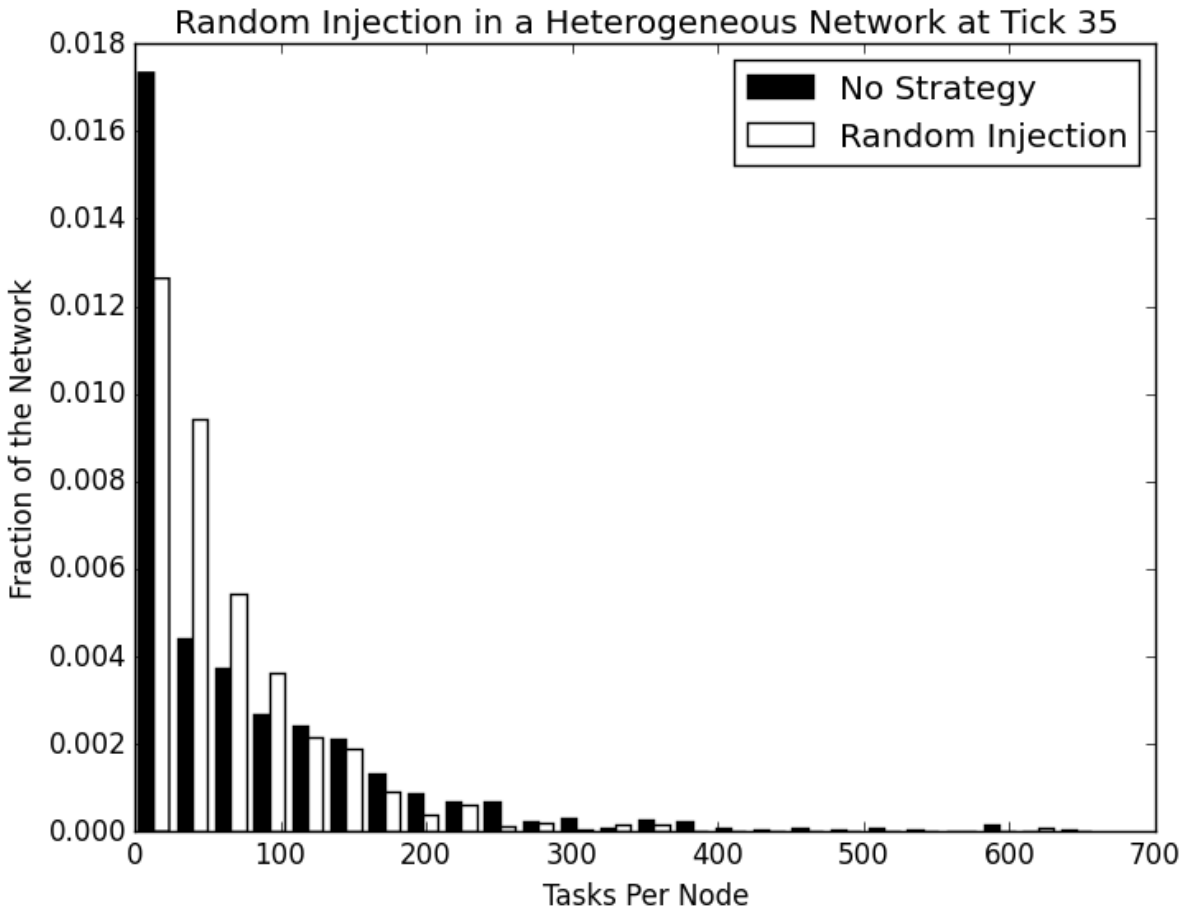


Figure: The workload distribution of heterogenous networks after 35 ticks.

Random Injection VS Churn

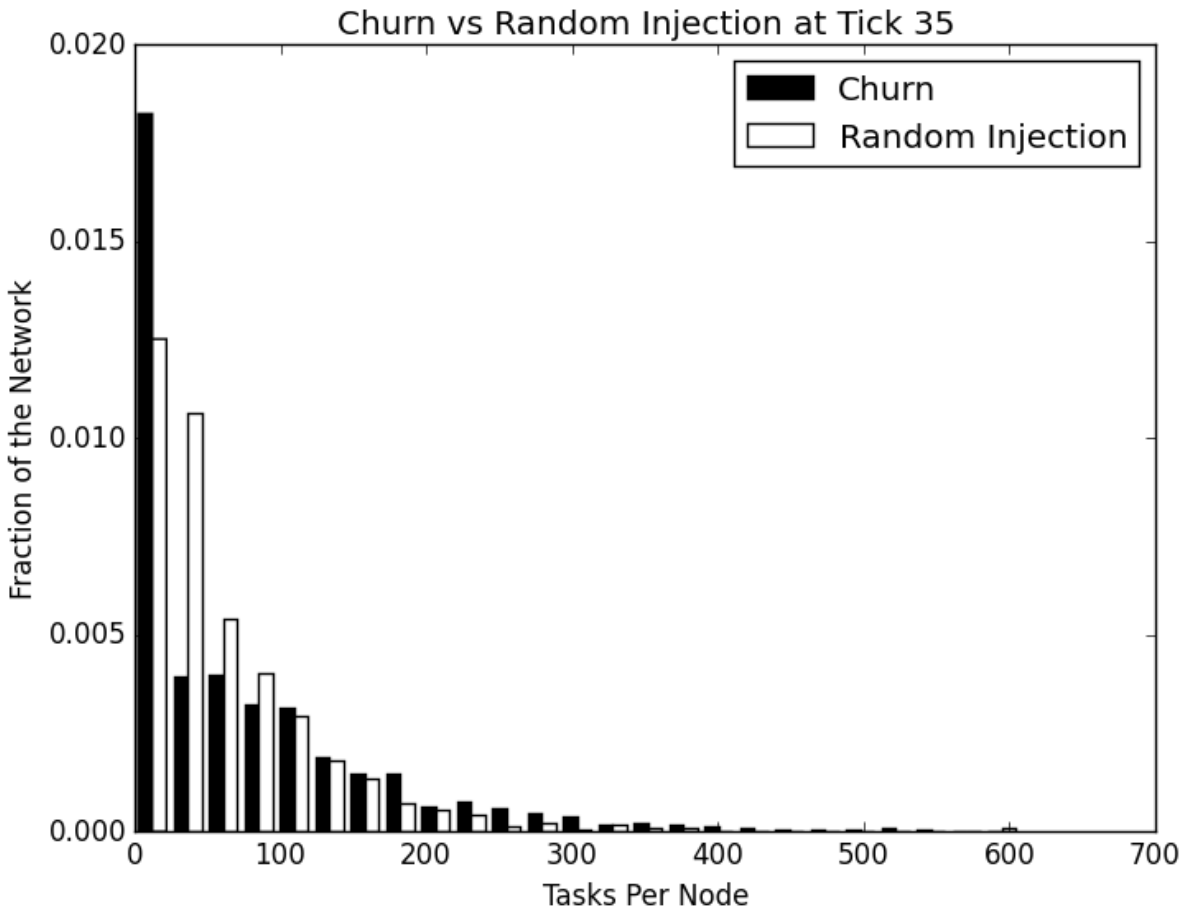


Figure: The networks after 35 ticks. The network using random injection load-balances significantly better than the network using Churn.

Impacts of Variables

- `sybilThreshold` would lower the runtime factor.
- Churn had no significant effect.
- `maxSybils` (node strength) had no effect in homogeneous networks.

Remarks

- Best runtime factor of our experiments.
- Could still incur high maintenance costs, especially with nodes being deleted as soon as they are made.

Strategy

- Rather than creating Sybils randomly, nodes create one in their successors
- Finding node id uses mashes
- Estimates which successor has most work.
- Tested estimation against smart method.

Base Runtime

- The base runtime in a 1000 node/100,000 task homogeneous network was 5.033.
- 2.4 lower than no strategy.
- Base runtime in a heterogeneous network was worse.
- `numSuccessors` improves the runtime factor (0.3 for base network)
- Other variables had no significant effect.

Neighbor Injection after 35 Ticks

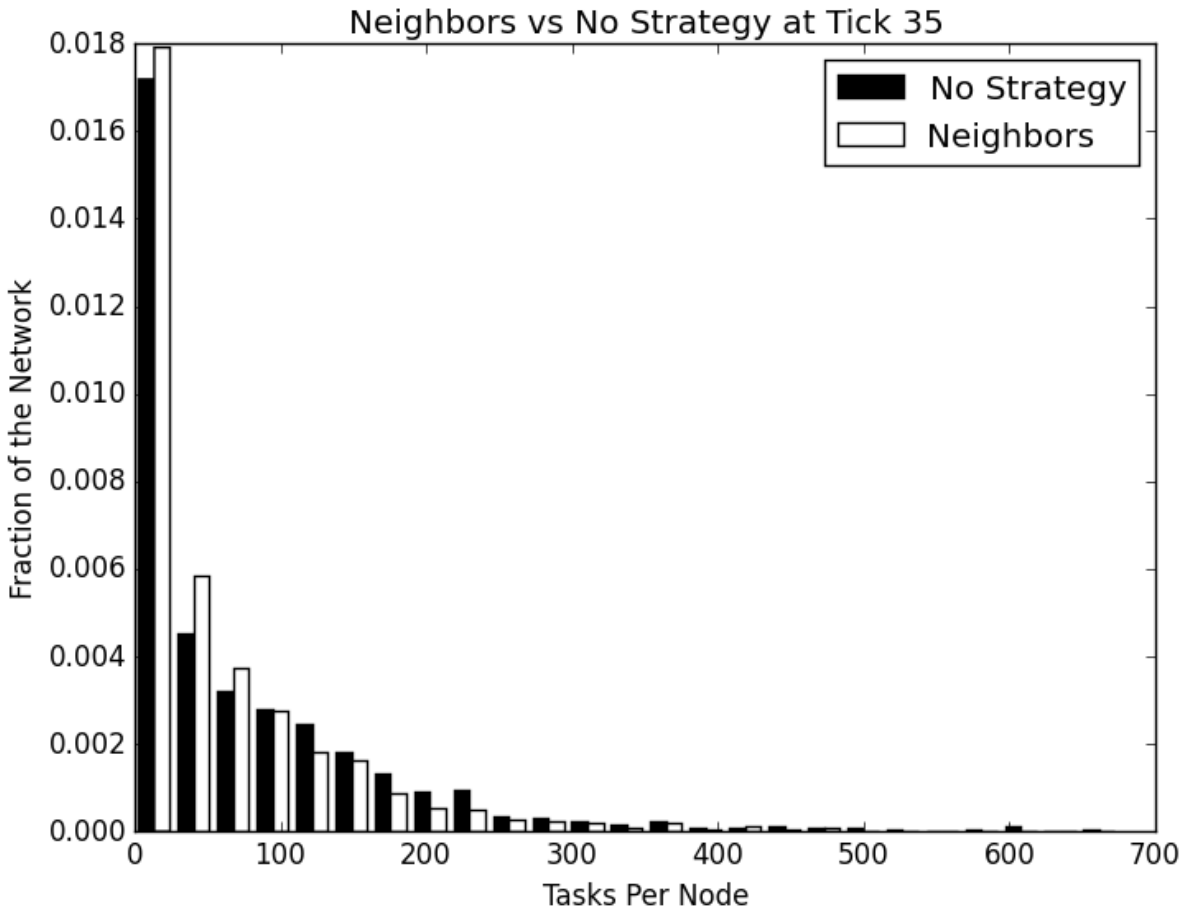


Figure: Neighbor injection vs no strategy after 35 ticks.

Smart Neighbor Injection after 35 Ticks

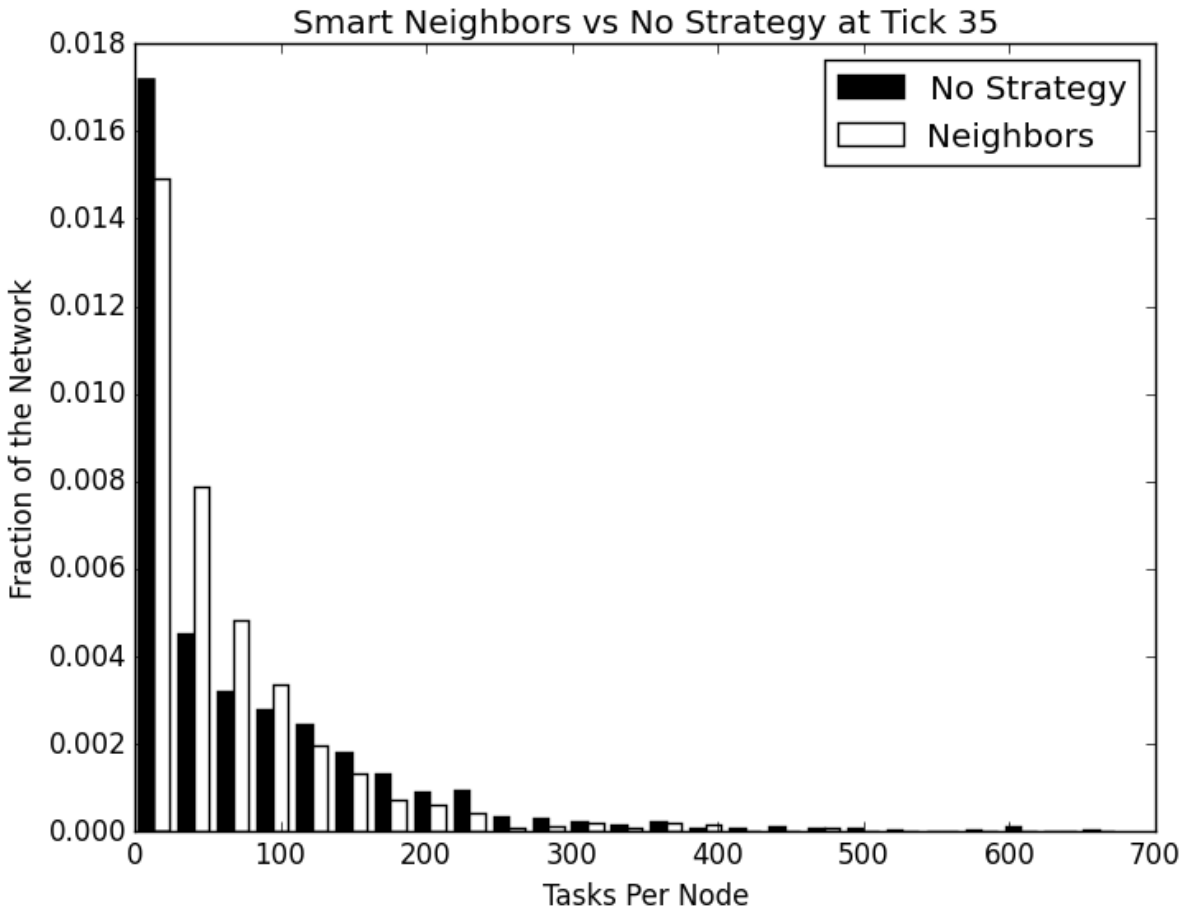


Figure: Smart Neighbor injection vs no strategy after 35 ticks.

Remarks

- Smart method improved runtime factor by 1.2 on average.
- Smart would require querying, "dumb" estimation still would provide improvement.
- Less churn of joining nodes.

Strategy

- Nodes with too much work ask for help.
- Predecessor with smallest workload and below `sybilThreshold` creates a Sybil.
- Reactive vs Proactive.

Invitation at 35 Ticks

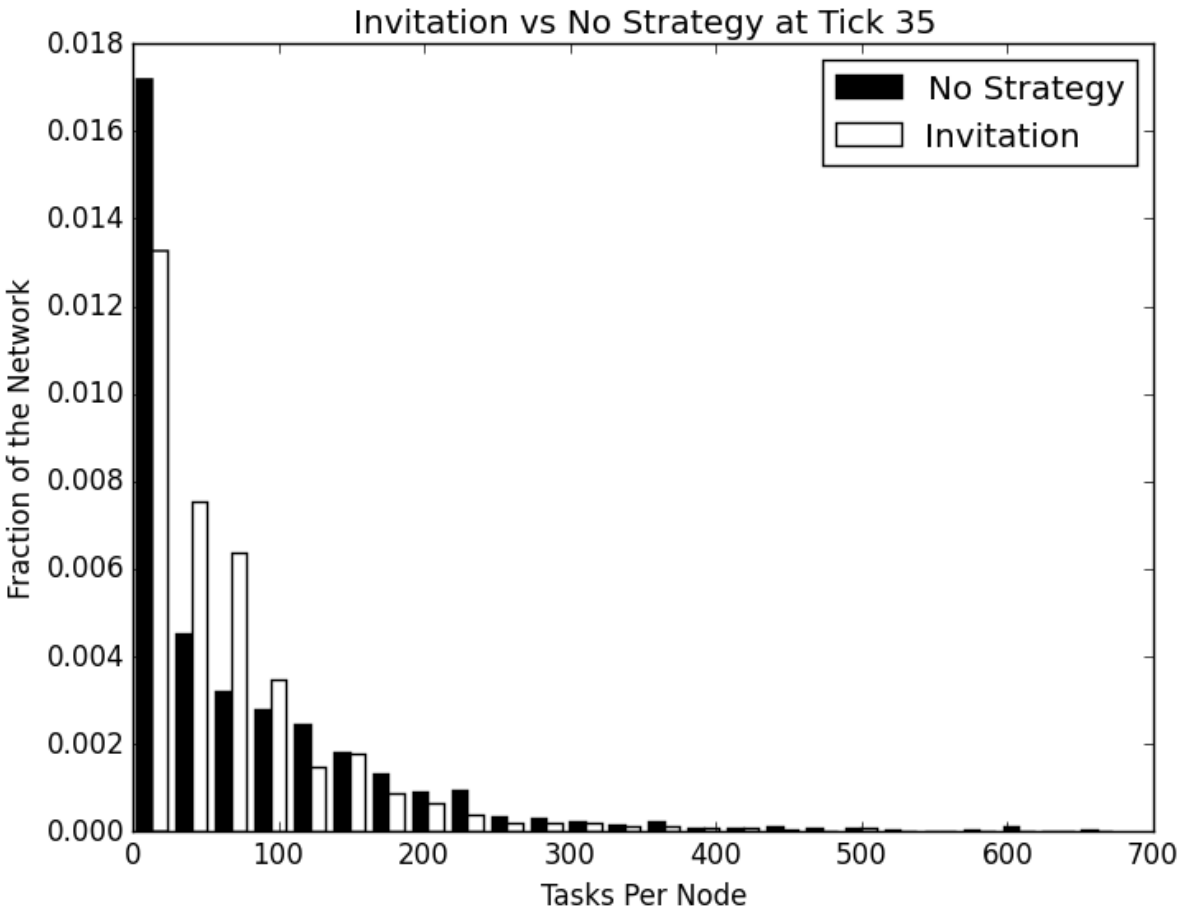


Figure: Invitiation vs no strategy after 35 ticks

Invitation vs Smart Neighbors at Tick 35

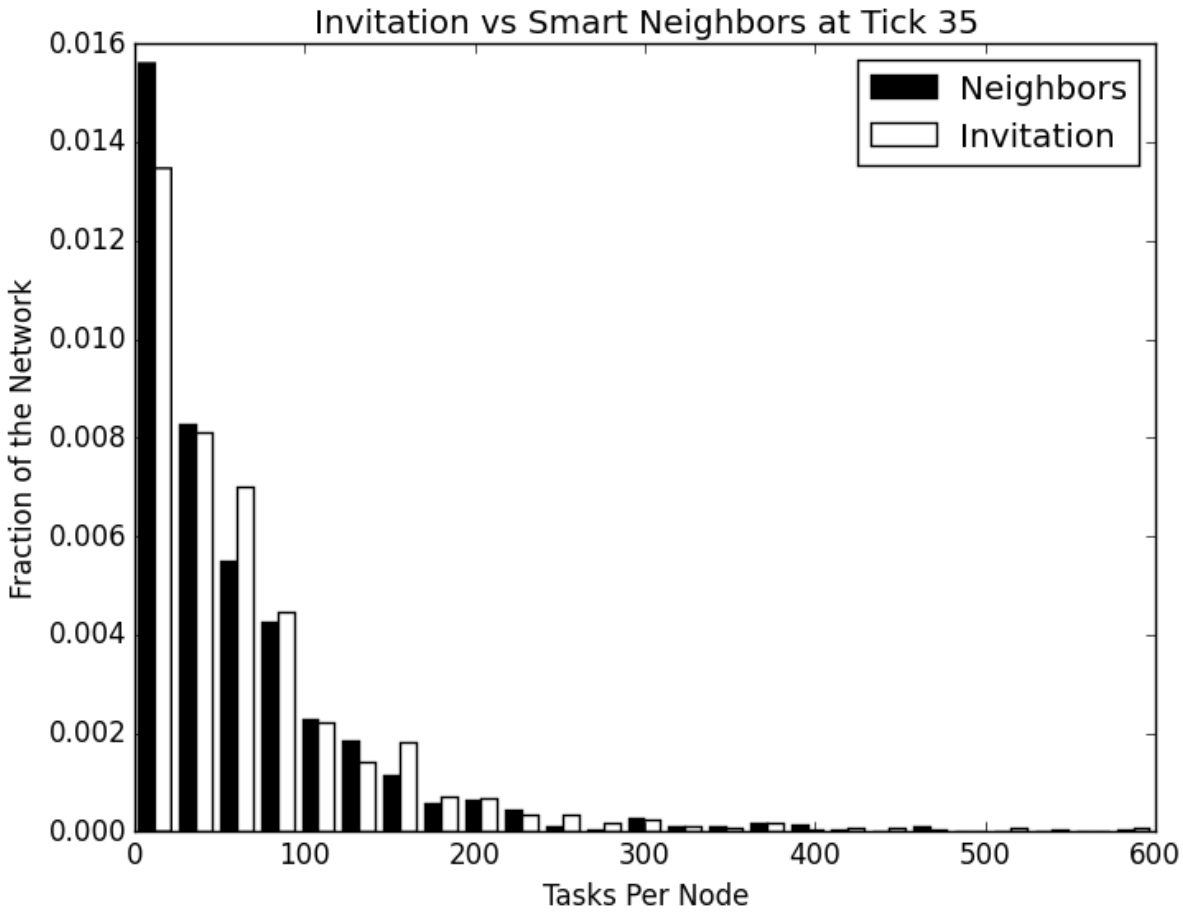


Figure: Invitation vs smart neighbor injection after 35 ticks.

Remarks

- Impact of Invitation was closely tied to the number of nodes in the network.
  - 100 node/ 100,000 task network (1000 tasks per node), the base average runtime factor was 3.749.
  - 1000 node/ 100,000 task network had a base average runtime of 5.673.
- Performed poorer in heterogeneous networks, but better than base on average (6.097 vs 7.5).
- Better than smart neighbors and uses less bandwidth.

Summary

- Reactive vs Proactive
- Heterogeneity was best handled by Churn or Random Injection.
- Random injection was best overall
- Load balanced did not mean faster since node strength was not taken into account.

Table of Contents

Conclusion

- UrDHT provides a platform for quickly creating DHTs.
- UrDHT shows that almost any DHT can be implemented by defining the Space Math.
- We showed that files are unevenly distributed in a DHT.
- We demonstrated strategies nodes can use to autonomously load balance the network and approach close to ideal times.

Published Work

- Andrew Rosen, Brendan Benshoof, Robert W. Harrison, Anu G. Bourgeois "MapReduce on a Chord Distributed Hash Table" Poster at IPDPS 2014 PhD Forum
- Andrew Rosen, Brendan Benshoof, Robert W. Harrison, Anu G. Bourgeois "MapReduce on a Chord Distributed Hash Table" Presentation ICA CON 2014
- Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, Robert W. Harrison "VHASH: Spatial DHT based on Voronoi Tessellation" Short Paper ICA CON 2014
- Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, Robert W. Harrison "VHASH: Spatial DHT based on Voronoi Tessellation" Poster ICA CON 2014
- Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, Robert W. Harrison "A Distributed Greedy Heuristic for Computing Voronoi Tessellations With Applications Towards Peer-to-Peer Networks" IEEE IPDPS 2015 - Workshop on Dependable Parallel, Distributed and Network-Centric Systems
- Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, Robert W. Harrison "Distributed Decentralized Domain Name Service" IEEE IPDPS 2016 - Workshop on Dependable Parallel, Distributed and Network-Centric Systems

Georgia State University

Submitting

- Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, Robert W. Harrison "Replication Strategies to Increase Storage Robustness in Decentralized P2P Architectures"
- Brendan Benshoof, Andrew Rosen, Anu G. Bourgeois, Robert W. Harrison "Online Hyperbolic Latency Graph Embedding as Synthetic Coordinates for Latency Reduction in DHTs"
- Andrew Rosen, Brendan Benshoof, Robert W. Harrison, Anu G. Bourgeois "Autonomous Load Balancing in a DHT"
- Andrew Rosen, Brendan Benshoof, Robert W. Harrison, Anu G. Bourgeois "UrDHT: A Generalized DHT"
- Andrew Rosen, Brendan Benshoof, Robert W. Harrison, Anu G. Bourgeois "The Sybil Attack on Peer-to-Peer Networks From the Attacker's Perspective"

Georgia State University

Other Publications

- Erin-Elizabeth A. Durham, Andrew Rosen, Robert W. Harrison "A Model Architecture for Big Data applications using Relational Databases" 2014 IEEE BigData - C4BD2014 - Workshop on Complexity for Big Data
- Chinua Umoja, J.T. Torrance, Erin-Elizabeth A. Durham, Andrew Rosen, Dr. Robert Harrison "A Novel Approach to Determine Docking Locations Using Fuzzy Logic and Shape Determination" 2014 IEEE BigData - Poster and Short Paper
- Erin-Elizabeth A. Durham, Andrew Rosen, Robert W. Harrison "Optimization of Relational Database Usage Involving Big Data" IEEE SSCI 2014 - CIDM 2014 - The IEEE Symposium Series on Computational Intelligence and Data Mining

Georgia State University