Choosing a Random Peer

Jared Saia
University of New Mexico

Joint Work with
Valerie King
University of Victoria
and
Scott Lewis
University of New Mexico

P2P problems

Easy problems on small networks become hard on large networks:

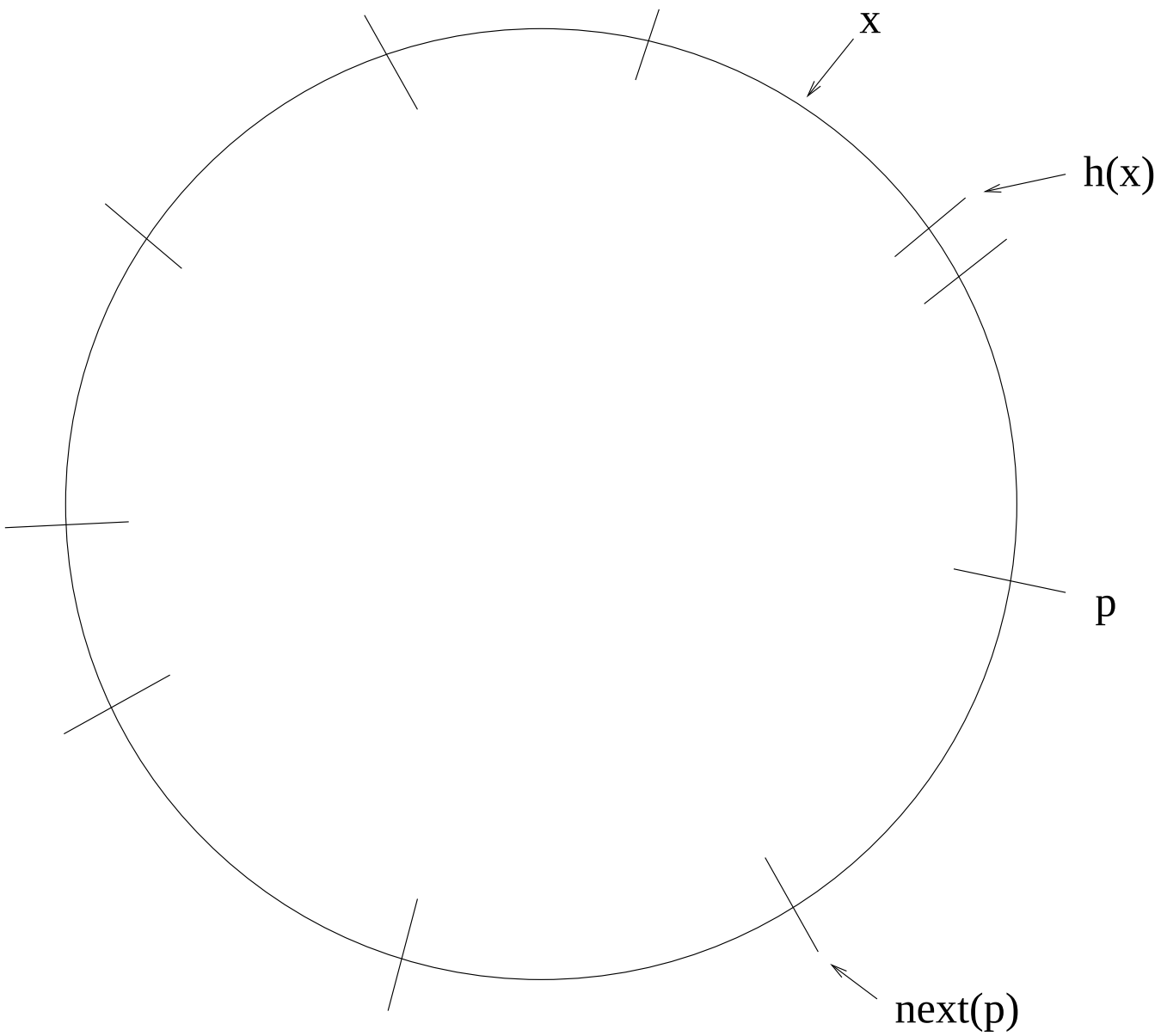
- Storing and finding data items
- Getting the number of nodes in the network
- Getting a random node in the network

Storing and finding data items

A Distributed Hash Table (DHT) is an ADT to facilitate storing and finding data in p2p networks

- Key space of DHT is a unit circle
- Peers in DHT are the "buckets"
- Peers are distributed uniformly at random on unit circle
- A DHT provides the following functionality:
  - $h(x)$: for any point x on the unit circle returns the closest clockwise peer to x
  - $next(p)$: for any peer p returns the closest clockwise peer to p

Example



Implementing a DHT

- Possible to implement a DHT in a scalable way:
- If n nodes in network
  - Each node maintains $O(\log n)$ links
  - Calls to $h()$ take $O(\log n)$ messages and latency
  - Calls to $next()$ take $O(1)$ messages and latency
- Chord is an example implementation of a DHT [Stoica et al., '01] (extension of the "Akamai" paper)
- I'm not going to tell you how this works - we'll just treat the DHT as an ADT

Our Results

Assume peers are connected in a DHT. We give algorithms which allow any peer to:

- Get an approximation to the number of peers in the network
- Get a random peer in the network

Why a Random Peer?

Uses for a function which chooses a random peer:

- *Create Random Links*: Can maintain a network where every node has small number of links to other random nodes. Such a network will provably stay well-connected even in the face of a sudden, massive number of adversarial node deletions.
- *Support Randomized Algorithms*: Load-balancing and Scalable Byzantine Agreement algorithms require uniform sampling of peers
- *Collect Data*: Can sample peers to collect: peer opinions e.g. on popular content; physical properties of network nodes e.g. for measurement studies; environmental data e.g. for sensor networks

Difficulty of Choosing a Random Peer

- The number of nodes in the network, n, is typically very large (100K is possible)
- Thus, each peer usually knows only small number of other peers in the network e.g. $O(\log n)$
- Also, want any algorithm to have latency and bandwidth which is polylogarithmic in n

Idea 1: Random Walk

- One idea for obtaining a random peer is to take a random walk through the p2p network
- Require that: 1) the network graph is rapidly mixing and 2) the network graph is regular
- Then in about $O(\log n)$ steps, we would expect to be at any fixed peer with probability close to $1/n$.

Random Walk Problems

- To uniformly sample with a random walk, we need the network graph to be regular.
- A random walk over a regular network only *approximates* uniform sampling
- The closeness of the approximation is impossible to formally state without knowledge of the gap between the first and second eigenvalues of the adjacency matrix

Our Result

We give an algorithm which has the following properties with high probability (i.e. with probability of error polynomially small in n):

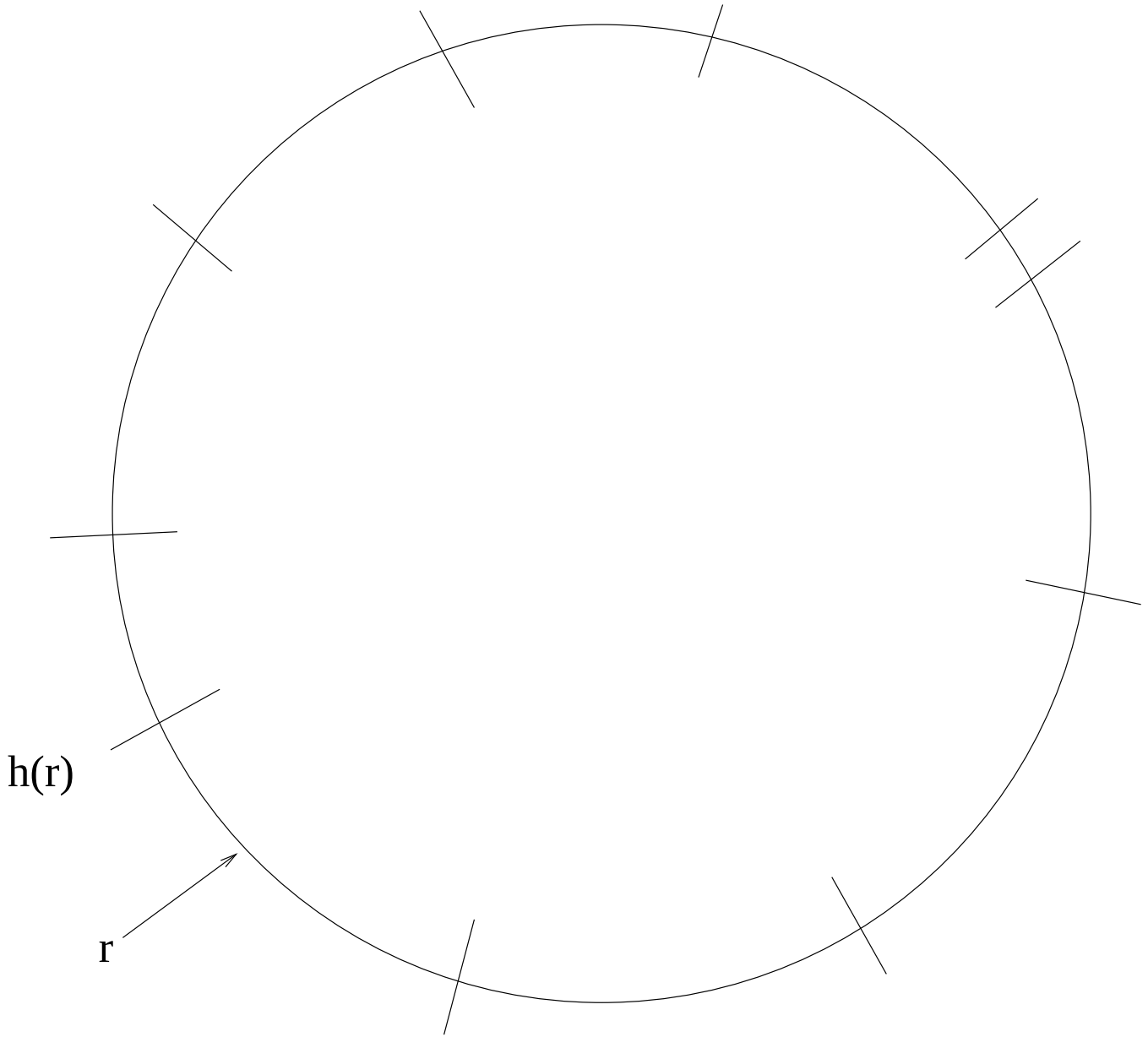
- Always chooses any given peer with probability *exactly* $1/n$
- Sends $O(\log n)$ messages and has $O(\log n)$ latency in expectation

(We use a DHT to achieve these properties)

Idea 2: DHT

- Given a DHT, a (naive) heuristic is to choose a random point r on the unit circle. Then return $h(r)$.
- Problem: Some peers have long arc lengths and some peers have short arc lengths.
- Each arc length is $1/n$ in expectation. However the minimum and maximum arc lengths are not $\Theta(1/n)$.

Example



Problem with Heuristic

W.h.p. the following is true:

- Maximum arc length is $\Theta(\log n/n)$ (easy Chernoff bound)
- Minimum arc length is $\Theta(1/n^2)$ (Birthday Paradox)

Arc Lengths

- The difference in the min and max arc lengths shows that the naive heuristic has significant bias
- Max arc length peer is chosen $n \log n$ times more frequently than min arc length peer

Estimating n

- The bounds on the min and max arc length allow us to get a pretty good estimate of n, the number of peers in the network
- Note that we already have a good estimate of $\log n$. [Mahlki et al].
- Fix a peer. Let q be the inverse of the distance between that peer and its nearest clockwise neighbor. Then:

$$n/\log n \leq q \leq n^2$$

- Which implies that:

$$\log n - \log\log n \leq \log q \leq 2\log n$$

Estimating n

- We can now use, q, which is $\Theta(\log n)$ to get a good estimate of n
- Let r be the distance between the fixed peer and the q-th closest clockwise neighbor
- r is $\Theta(\log n/n)$
- Thus q/r is $\Theta(n)$

Idea 3: Naive with restarts

- We can use this estimate of n to create a correct but slow algorithm
- *New algorithm: Let r be a random number between* 0 *and* 1. *If $h(r)$ is within distance $1/n^2$ of r, then return $h(r)$, else repeat*
- Since the min arc length is $\Theta(1/n^2)$, each peer will be chosen with probability exactly $1/n^2$
- Problem: This algorithm is *slow*: calls h $O(n)$ times in expectation

Our Algorithm

- Can think of this slow algorithm as mapping $\Theta(1/n^2)$ length of "real estate" to each peer.
- Need to ensure that each peer has the same amount of "real estate" mapping to it
- However, also want to have about $\Theta(1/n)$ length of real estate map to each peer (in order to get a fast algorithm)
- Idea: peers with short arc lengths get extra real estate from peers with long arc lengths

d_p and $tmax_p$

Our algorithm requires estimates d_p and $tmax_p$ with the following properties:

- $d_p = \Theta(\ln n/n)$ and $tmax_p = \Theta(\ln n)$;
- Any arc of length d_p has at most $tmax_p$ peers in it

A simple procedure will give us such estimates with high probability

Our Algorithm (Arc-Length)

- In each round, we choose a random number, r, in $(0, 1]$
- We put all peers within d_p of r into a bag
- We add enough "dummy peers" to that bag to ensure that the bag contains $tmax_p$ peers
- We choose a peer uniformly at random from all peers in the bag
  - If the chosen peer is a dummy peer, we go back to the first step
  - If the chosen peer is a real peer, we output that peer

Arc-Length

1. While TRUE do :
2. $r \leftarrow$ random number in $(0, 1]$;
3. $x \leftarrow$ random integer in $[1, tmax_p]$;
4. $i \leftarrow 1$; $p' \leftarrow h(r)$;
5. While $i < x$ and $d(r, p') \leq d_p$:
6. $i \leftarrow i + 1$; $p' \leftarrow next(p')$;
7. If $d(r, p') \leq d_p$ return p';

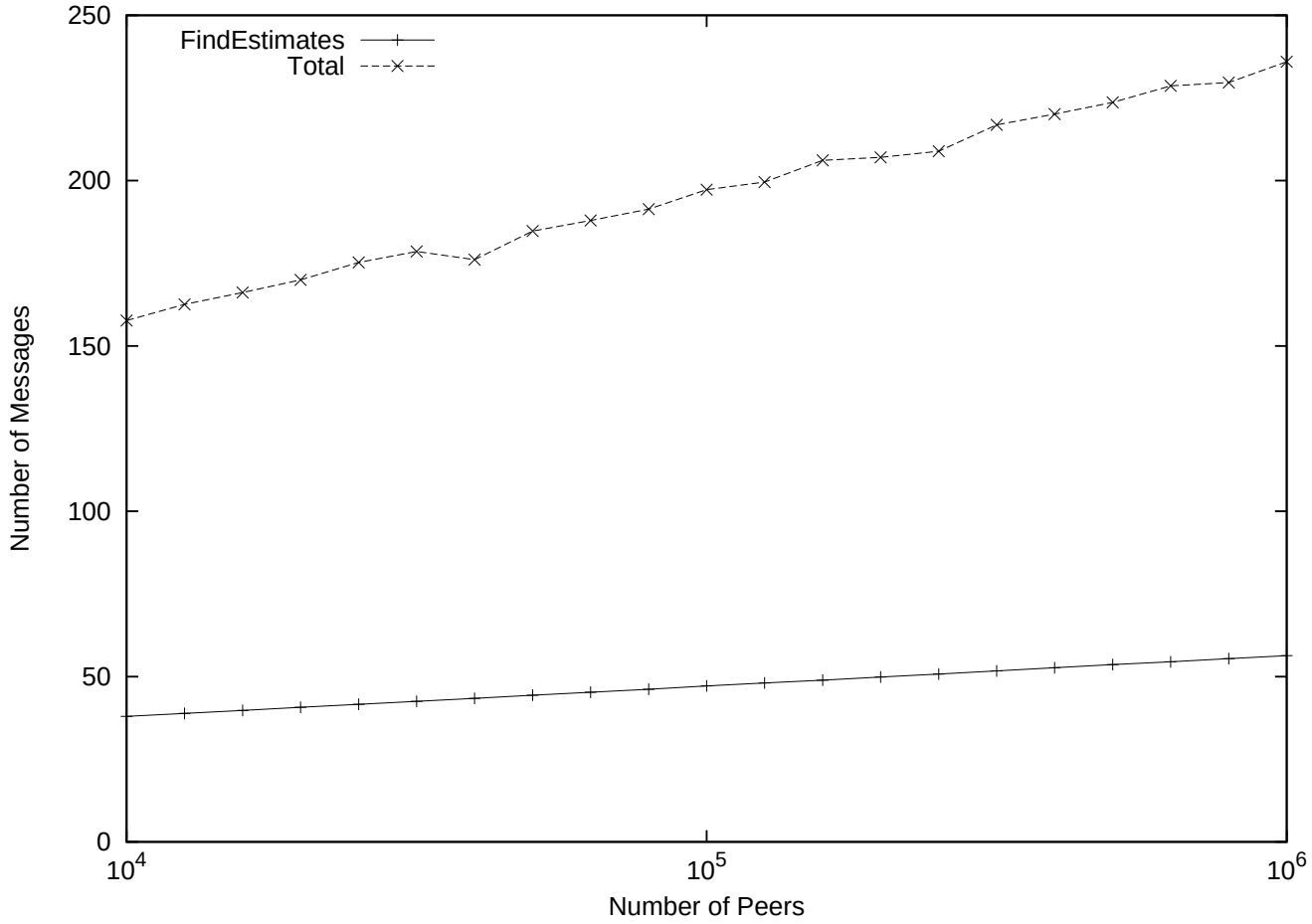
Correctness (Sketch)

- Each peer has probability d_p of being put in the bag
- Given that a peer is put in the bag, it has probability $1/tmax_p$ of being chosen from the bag
- Thus each peer has probability exactly $d_p/tmax_p$ of being chosen

Resource Costs

- $d_p/tmax_p = \Theta(1/n)$, so, every time we choose a new random seed, there is constant probability that the algorithm will map that seed to a peer.
- Thus expected number of rounds is $\Theta(1)$
- This implies that the expected latency and message cost is $O(\log n)$.

Empirical Results



Unstructured Networks

- Imagine we have a completely unstructured network (i.e. no DHT), that all nodes have random IDs between 0 and 1, and that the "choosing" node knows the approximate size of the network
- Can still use the Arc Length algorithm in the following way:
- The choosing node broadcasts a random seed to all other nodes
- The $\Theta(\log n)$ nodes which have IDs sufficiently close to this seed reply to the choosing node
- The choosing node node simulates the Arc Length algorithm on all nodes that replied to its messages
- This gives the choosing node a node selected uniformly at random from entire network

Open Problems

- Can we choose a random, or near random, peer efficiently in a less structured p2p network like gnutella? I.e. can we do better than a random walk?
- Empirical evaluation - Can this algorithm (or something like it) do well in practice?
- Can we choose a peer with a biased probability? For example, many gossiping protocols require the ability to choose a node with probability proportional to the inverse of the distance from us. Is there an efficient algorithm to choose a peer with this biased probability?