PERFORMANCE COMPARISON OF NEIGHBOR DISCOVERY PROTOCOLS IN WIRELESS AD-HOC NETWORK

A Thesis

by

SANAT KUMAR PANDEY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jennifer Welch
Committee Members,	Alex Sprintson
	Riccardo Bettati
Head of Department,	Dilma Da Silva

August 2015

Major Subject: Computer Engineering

ABSTRACT

In this thesis we consider the problem of neighbor discovery in synchronous single hop wireless ad-hoc networks. The central problem is to establish a broadcasting sequence such that only one transmitter broadcasts at a time while all others listen and every transmitter in the network gets at least one chance to broadcast. We consider the question: how fast we can achieve neighbor discovery with k nodes in the network, each having a unique id assigned from an id space much larger than k in radio communication models with and without collision detection. We take the simulation route to answer this question. We implemented one randomized and two deterministic algorithms for neighbor discovery and compared their performance in terms of number of rounds required as a function of the number of nodes in the network and the size of the space from which ids are chosen. Our simulation results show that the randomized algorithm is most efficient and is easiest to implement. The deterministic algorithm for the no collision detection model has round complexity comparable to the size of the id space and is orders of magnitude less efficient than the randomized algorithm. The deterministic algorithm for the collision detection model is slower than the randomized algorithm by a factor of $\log(n)$, where n is the size of the id space. Our analysis would be useful for choosing optimal algorithms for field applications depending on the radio communication model and network topology. It will reveal any large constants or second order terms discarded in the asymptotic analysis of the algorithms, which reduces effectiveness of some algorithms in applications.

ACKNOWLEDGEMENTS

I would like to thank my advisor Professor Jennifer Welch for introducing me to the intriguing and wonderful world of distributed computing. I thank her for giving me the freedom and opportunity to explore exciting topics in distributed computing like memory consistency conditions, distributed shared registers and radio networks. The numerous discussions and feedbacks gave me new insights and understanding of the subject matter, while her kindness and patience made it a cherishable experience.

A special thanks to Dr. Alex Sprintson and Dr. Riccardo Bettati for agreeing to be committee members on my thesis defense panel.

I would also like to thank my colleagues and friends Dr. Hyun Chul Chung, Saptaparni Kumar and Edward Talmage for discussions, feedback and advice.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

A wireless add-hoc network consists of a set of mobile autonomous computing nodes which communicate through wireless transmission. In contrast to traditional cellular network models that rely on static wired base stations for communication, ad-hoc networks are based on node-to-node wireless transmission and message forwarding for communication. The absence of centralized co-ordination routers for communication makes them specially attractive for military usage, as they are robust and easy to set up. In general ad-hoc wireless networks are employed when setting up static network infrastructure is difficult or impractical like in defense, natural disasters or search and rescue. Other interesting applications include wireless sensor networks, swarm based robotics and wearable computing devices.

In wireless networks the communication medium is shared and only one node can transmit at any given time. The message becomes corrupted when more than one node transmits simultaneously. This is termed as collision. Radio networks are further classified into the *collision detection model* when nodes can distinguish between interference noise and background noise or silence, and the *no collision detection model* when nodes cannot distinguish between silence and interference noise. Nodes in wireless ad-hoc networks are generally resource-constrained in terms of communication bandwidth, computational power and memory on board. So developing resource-efficient algorithms is of paramount importance.

In wireless ad-hoc network it is not feasible to have all nodes within range of each other due to transmission power requirement and battery life concerns. Moreover in a mobile ad-hoc networks, nodes are continuously moving in arbitrary directions and even maintaining neighbor knowledge and connectivity information can be a

non-trivial exercise. A reliable communication protocol for a mobile ad-hoc network consists of two parts, i) an initial neighbor discovery algorithm and ii) keeping track of local neighborhood as topology of the network changes[1][2][3].

The initial neighbor discovery problem has been extensively studied for static radio networks[4][5][6] and the proposed solutions can be broadly categorized as deterministic and probabilistic. The deterministic algorithms terminate in a predetermined fixed number of communication attempts, while probabilistic algorithms guarantee that with high probability, termination will occur in a certain number of transmission rounds.

This work is motivated by work done in deterministic algorithms for reliable and efficient communication in mobile ad hoc networks by Viqar [1][2] as part of her PhD thesis. Viqar considered the more general problem of keeping track of neighborhood topology in a synchronous mobile ad-hoc network with no collision detection capability and an upper bound on the maximum speed, as nodes continuously move in and out of each other's transmission and interference range. She proposed a modular solution in which a mobile neighborhood discovery layer is built on top of a medium access layer, which handles broadcast collision. The mobile neighbor discovery layer handles changing network topology due to mobility of nodes. She proposed a region-based neighbor discovery protocol to keep track of changing neighborhood information due to mobility of nodes.

The area of interest is tiled into disjoint hexagons and each hexagon[1, p. 58], is assigned a unique color from the set $\{1, ..., m\}$, where m is a constant determined by the transmission and interference radius of radio broadcast. Partitions which are assigned same color are sufficiently separated in space so that they can transmit simultaneously without interference(collision). Each color is then assigned a round number by a scheduling algorithm which makes sure that no two colors are assigned

the same round number. This ensures that nodes in partitions having different colors never broadcast simultaneously and hence, never collide. Each round has a fixed number of broadcast slots and nodes within a partition broadcast in the slot determined by the relative rank of its id with respect to all the nodes present in that partition.

All nodes maintain knowledge about the trajectory information of every node in its own partition and its adjacent partition. Nodes assist each other in maintaining neighborhood knowledge by announcing their trajectory, whenever they are about to enter or leave any partition[3, p. 511]. The trajectory information is then used by other nodes to add or remove nodes in its immediate neighborhood.

The mobile neighbor discovery protocol relies on an initial neighborhood discovery protocol for initialization of neighborhood information. Deterministic initial neighbor discovery in the no collision detection radio model is difficult due to the inability of nodes to differentiate between interference noise due to multiple broadcasts and background noise when no node is broadcasting. This symmetry is broken by constructing a family of subsets (S) of the id space (U) which, when used as a broadcast schedule ensures that all nodes gets to broadcast alone at least once.

Viqar presented a deterministic algorithm for neighbor-discovery which relied on deterministic construction of combinatorial objects known as *k-selectors.*[1, p. 74]. Indyk presented an algorithm for deterministic construction of k-selectors of size $O(k \log^3 n)$[7], where k is the number of nodes in the network and n is the size of the id space. Indyk's algorithm for k-selector construction used a bipartite graph having random like properties called disperser. Ta-Shma presented an algorithm for constructing dispersers for every $k \geq poly(\log \frac{n}{\epsilon})$[8], where $\epsilon \in (0, 1)$. The disperser used in [7] uses $\epsilon = \frac{3}{4}$.

In this thesis we compare the performance of Viqar's deterministic algorithm

based on k-selectors against a simple randomized algorithm in terms of number of rounds, number of nodes in the network and the size of the id space. Also we present a deterministic algorithm which can outperform the k-selector-based algorithm in the collision detection radio model. Note that the k-selector-based algorithm works for both the no collision detection and collision detection models.

As part of our work comparing neighbor discovery algorithms, we study the computational complexity involved in deterministic construction of combinatorial structures like *k-selectors, dispersers and p-colliding family* of functions used as building blocks. These results may be of independent interest.

The deterministic neighbor discovery algorithm based on k-selectors for the no-collision-detection radio model looks asymptotically efficient when expressed in *big oh* notation $O(k \log^3 n)$ but it hides a large multiplicative factor of $(48^2 d^3)$, where d is the degree of the disperser graph used, which significantly degrades its performance when the size of the id space is less than 10^6. The big oh notation used to specify asymptotic behavior of algorithms is helpful in predicting the scalability of algorithms as the size of the input grows but can hide large constant factors that hinders absolute comparison in terms of computation cycles with other algorithms. Sometimes simple algorithms with bad asymptotic behavior can beat sophisticated asymptotically efficient algorithms with large constant multiplicative factors in the input range of interest. Since mobile ad-hoc network algorithms are used in a wide array of devices, from hand-held radio devices with very little memory and processing capability to vehicular ad hoc networks, our analysis will be useful for selecting appropriate algorithm in practice.

We also present a simple deterministic algorithm for the collision detection radio model which terminates in $2k \log(n)$ rounds, where k is the number of nodes in the network and n is the size of the id space, and a randomized algorithm that terminates

with $1 - \epsilon$ probability ($\epsilon < 10^{-4}$) in $20k$ rounds for $k < 1000$. The randomized algorithm is similar to the algorithm Rand-try presented in [9, p. 212] for the wake up problem. The deterministic algorithm for the collision detection model uses ideas similar to one used in [10] for consensus in radio networks and those used in [11] for conflict resolution in multiple-access channels.

2. RELATED WORK

Reliable communication in radio networks has been widely studied in the context of distributed computing[9][12][13][14]. Widely studied aspects of radio communication include the wake-up problem, reliable broadcast and the neighbor discovery problem.

In the wake-up problem some process wakes up spontaneously while others have to be woken up[9]. Only awake processes can send message. Sleeping processes wake up on receipt of the first messages. The objective is to wake up all processes in the minimum number of rounds. The complexity of the wake-up problem is measured as the number of rounds elapsed from the spontaneous wake-up of the first node to the round when all nodes are awake. Jurdzinski and Stachowiak presented probabilistic algorithms for the wake-up problem in single hop synchronous radio networks in $O(\log(n)\log\frac{1}{\epsilon})$ rounds [15], where n is the number of nodes in the network and ϵ is the probability of error.

The broadcast problem is how to propagate a message from a source to all nodes in a multi-hop radio network[13]. It has been formulated in two flavors, spontaneous protocols in which the starting time is known to all nodes and nonspontaneous, in which there is a single source initiating the process. Some solutions to the broadcast problem use a solution to the wake-up problem as a building block.

The neighbor discovery problem is to identify all the neighbors in a mobile device's transmission range[5]. It is an important first step in the initialization of a wireless ad hoc network. Neighbor discovery differs from the wake-up problem in that all nodes have woken up spontaneously but they don't know each others' ids, which have to be communicated through radio broadcasts. Neighbor discovery in single hop wireless

ad hoc networks is studied in [6], in heterogeneous cognitive radio networks in [4] and in multi-channel radio networks in [5].

A common sub-problem found in the wake up problem, reliable broadcast and the neighbor discovery problem is to establish a broadcast schedule such that all nodes in the network gets to transmit alone at least once, without interference due to simultaneous broadcast by any other node in its transmission or interference radius.

Randomized algorithms for discovering such schedules have been studied in [6][16]. Randomized algorithms for neighbor discovery based on birthday paradox is studied in [17]. In [6] an asynchronous Aloha [17] like algorithm is presented which is slower than the synchronous counterpart by a factor of two. Randomized algorithms for neighbor discovery in the collision detection model is presented in [16].

Deterministic algorithms for finding such a schedule are based on id-based arbitration in the collision detection model[10]. Radio broadcast in the collision detection model is discussed in [18]. A tree based collision detection algorithm in which only a subset of the nodes present in the network want to broadcast and the broadcast probability is upper bounded is presented in [19] .

Deterministic algorithms for the no collision detection model is based on the deterministic construction of combinatorial objects known as *selective family of subsets.*[20]. Selective families were used for deterministic distributed broadcast in [13]. A selective family S of a set U is a family of subsets such that for any subset K of U of size $\leq k$, there exists some subset in the family ($s_i \subset U$) which intersects subset K at exactly one element. This property guarantees that at least one node in the network will broadcast successfully, every $|S|$ rounds.

Interestingly in [20] the authors use a deterministic distributed broadcast algorithm for special networks, to prove lower bounds on the size of the selective family of any set U.

3. INITIAL NEIGHBOR DISCOVERY

A mobile ad-hoc radio network is modeled by a graph $G = (V, E)$. The vertices in V represent the transmitters and receivers in the network, also called processes or nodes, and E represents the connection or links between the the transmitters/receivers. We will use $|V| = k$ to denote the number of nodes and n to denote the size of the id space. Each node in the network has a unique id assigned from the id space. The ad-hoc nature of the network implies that initially the nodes (vertices) don't know about the topology of the network. Additionally in a mobile ad-hoc network the communication graph keeps changing over time as nodes move in and out of each other's transmission radius.

The deterministic reliable neighbor discovery protocol presented by Viqar [1, p. 58] assumes the initial neighbor knowledge before the nodes start moving. The algorithm presented by her then keeps track of the changing neighborhood configuration as the nodes start moving. To justify this assumption she presented a deterministic initial neighbor discovery algorithm [1, p. 74] which works for both the collision-detection and the no-collision-detection models.

To minimize collisions due to simultaneous broadcast, the overall algorithm divides the geographical/physical space into hexagons, whose size depends on transmission and interference radius of radio transmitters/receivers. Then each hexagon is assigned a time slot, which dictates the time window in which nodes belonging to those regions can broadcast. This ensures that nodes belonging to different hexagonal partition never collide. This reduces the problem of deterministic initial neighbor discovery of all nodes in the network, to deterministic neighbor discovery of nodes in each hexagon. All the nodes belonging to same hexagonal partition are within

each others' broadcast and interference radius. Furthermore, nodes do not cross their respective hexagonal partition during initial neighbor discovery phase.

The initial neighbor discovery algorithm assumed restricted motion during the initial neighbor discovery phase, which allowed the connectivity graph for each local neighborhood to be modeled by a complete graph. This implies that when a transmitter node broadcasts alone, all receiver nodes within the transmission radius will receive the message.

3.1 Problem Definition

Neighbor discovery problem : given a set of nodes each having a unique id and sharing a common communication medium, find an algorithm which will lead to all nodes successfully gaining the knowledge about every other node in the network.

In this thesis we follow Viqar's lead and assume a complete communication graph for the initial neighbor discovery problem. This complete graph based communication model is also known as a single-hop radio network. When the communication graph is not complete, it is termed as a multi-hop radio network.

The nodes in the network communicate in synchronous steps called rounds. In each round a node may choose to transmit or receive. When a node is not transmitting it is in listening mode (acting as a radio receiver). When a transmitter node transmits alone, all the receiver nodes within transmission radius r_{trans} receive the message. Radio network models are further classified into the *collision detection* model and the *no collision detection* model. In the *collision detection* model nodes can differentiate between background noise of silence and interference noise due to two or more nodes broadcasting simultaneously(collision) whereas in the no-collision-detection model nodes cannot differentiate between interference noise of collision and background noise of silence. Algorithm 2 shows that collision detection capability

greatly simplifies the deterministic neighbor discovery problem. The round complexity for successful neighbor discovery in this model is significantly lower than in the no collision detection model.

The neighbor discovery problem can be abstracted out and formulated in a number of ways depending on the radio network model used and the quality of the solution desired. It can be viewed as finding a $k \times l$ binary matrix such that for each row r_i there exists a column c_j such that $c_j[r_i] = 1$ and $c_j[r_{k \neq i}] = 0$. It is easy to verify that given such a matrix it can be transformed into a correct neighbor discovery protocol. Just assign each node in the radio network a distinct row of the given matrix and ask it to broadcast in rounds corresponding to row indices having entry 1. Since each row has an entry 1 at some index such that it is the only row with 1 in the corresponding column of the binary matrix, it broadcasts alone.

Another way of formulating the problem is in terms of designing hash functions: Find a hash function $f : \{1..n\} \mapsto \{1..k\}$, where n is the size of the id space and k is the number of nodes in the network, with some minimum progress guarantee, that is given any set $S \subset \{1, .., n\}, 2 \leq |S| \leq k$ of k ids belonging to the id space, the hash function should map them to some element in the set $\{1, .., k\}$ such that with high probability at least two among the $2 \leq |S| \leq k$ items being hashed does not collide. That is $\exists x_1, x_2 \in S$ such that $f(x_i) \neq f(y), \forall y \in S \setminus \{x_i\}, i \in 1, 2$. This ensures that in every k rounds at least two nodes broadcasts successfully. Now if the broadcasting nodes include ids of all nodes they have received message from, then at least one node learns about its successful broadcast every k rounds and stops broadcasting in future rounds. This can be used to implement a randomized neighbor discovery protocol that terminates with high probability in k^2 rounds. A neighbor discovery protocol terminates when each node in the network knows about all its neighbors.

The difficulty in designing an initial neighbor discovery protocol for ad-hoc radio

networks is the symmetry arising when more than one node within a transmission radius try to transmit in the same round. This symmetry could lead to perpetual collision. The deterministic algorithm has to terminate in a pre-determined number of rounds and each node must know the *id* of every node in the network when the algorithm terminates. The randomized algorithm has to terminate with probability $1 - \epsilon$ after n rounds where ϵ is a function of n and $\epsilon(n) \to 0$ as $n \to \infty$.

We abstract out the problem definition into deterministic neighbor discovery and randomized neighbor discovery to make it more amenable to solution algorithmically. The deterministic version of the neighbor discovery problem can be formulated as:

Deterministic neighbor discovery: Given a set $U = \{1, 2, 3, ..., N\}$, and a subset K of U, find a sequence of subsets of U, $S = \{s_1, s_2, ..., s_m;\ s_i \subset U\}$, such that for all elements x in K there exists a subset in S such that $s_i \cap K = \{x\}$

The deterministic solution tries to find a family of subsets (S) of the id space such that for any arbitrary subset of size less than or equal to k(the number of nodes in network) there always exists a subset in the family which intersects with the given arbitrary set at only one element. Hence if we use this family of subsets as the broadcast schedule, every node in the network gets to broadcast alone. Algorithm 2 for the collision detection model uses the information in K to construct the trivial family consisting of singleton elements of K. The algorithm for the no-collision-detection model constructs a general family of subsets valid for all possible combinations of k-element subsets of the id space.

Randomized neighbor discovery : Given an integer k find a function $f : \mathbb{N} \times \{1, .., k\} \mapsto [0, 1]$ such that if each node broadcasts with probability $f(i, j)$ in the ith round, with j neighbors known to it before the beginning of round i, then after n rounds of transmit/receive, the probability of successful neighbor discovery is $1-\epsilon(n)$, where $\epsilon(n) \to 0$ as $n \to \infty$. This corresponds to a random $M = k \times l$ matrix where

$M[r, j] = 1$ with probability $f(i, j)$. Each round can be mapped to a column of the matrix and each process corresponds to a row in the matrix.

The randomized solution tries to specify the probability of broadcast in each round as a function of the round number and the number of successful broadcasts so far. For example, initially if there are k nodes in the network, all nodes may broadcasts with probability $\frac{1}{k}$ and after successful broadcast by one node, the remaining may broadcast with probability $\frac{1}{k-1}$ and so on. In the case when the number of nodes in network is not known, all nodes may decide to broadcast in the first round with probability $\frac{1}{2}$, in the second round with probability $\frac{1}{4}$ and so on, until they hear a successful broadcast or hit some lower bound based on the knowledge about the maximum possible number of nodes in the network.

3.2 Trivial Solution

A trivial solution to the neighbor discovery problem is to use a schedule of length n, where n is the size of the id space. S = {{1},{2},{3},.....,{n}}. The node with the id i broadcasts in round i. Since each node in the network has a unique id, at most one node broadcasts in any round. The drawback of the trivial algorithm is that it is a wasteful strategy as the number of nodes in the network is much smaller than the size of the id space. Nevertheless, it serves as a good benchmark to measure the efficiency of other solutions.

3.3 Randomized Algorithm

We attempt to find a randomized algorithm for the neighbor discovery problem by using a constant probability function $f(n) = p$; that is, in each round all nodes broadcast independently with probability p and listen with probability $(1 - p)$. Since there are k nodes, the resulting communication pattern can be seen as a *binomial distribution* over $\{0, ..., k\}$. The probability of m nodes broadcasting simultaneously

is given by the $(m+1)^{th}$ term in the binomial expansion of $((1-p)+p)^k = 1$.

$$((1-p)+p)^k = (1-p)^k + k(1-p)^{k-1}p + \frac{k(k-1)}{2!}(1-p)^{k-2}p^2 + ... + p^k$$

The probability of successful broadcast, that is, only one node broadcasting is given by the second term on the right hand side of the above equation, which is $k(1-p)^{k-1}p$. We would like to maximize the probability of successful broadcast to minimize the round complexity of the algorithm. Hence differentiating $k(1-p)^{k-1}p$ with respect to p and setting it to 0 gives

$$\begin{aligned} \frac{d}{dp}((1-p)^{k-1}p) &= 0 \\ (1-p)^{k-2}((1-p)-(k-1)p) &= 0 \\ p &= \frac{1}{k} \end{aligned}$$

Hence in each round we broadcast with probability $\frac{1}{k}$ and listen with probability $(1-\frac{1}{k})$. The expected number of rounds(r) for successful broadcast will be

$$E(r) = \frac{k^{k-1}}{(k-1)^{k-1}} \leq e \; \forall k \geq 3$$

Since there are k nodes, the expected number of rounds for all nodes to finish broadcasting is less than ek rounds. To find the probability of successful broadcast after m rounds we use the lower bound of $P(success) \geq \frac{1}{e}$ in each round. The probability of failure (ϵ) after m rounds of broadcasts, that is, no node is able to broadcast alone

in first m rounds is $(\epsilon = (1 - \frac{1}{e})^m)$. Hence the probability of success of at least one node after m rounds is $P(|success| \geq 1) \geq 1 - (1 - \frac{1}{e})^m$.

$$1 - \epsilon = 1 - (1 - \frac{1}{e})^m$$
$$m = \frac{\log(\epsilon)}{\log(e-1) - 1}$$

Substituting $\epsilon = .0001$ for 99.99% success probability, we get $m \approx 20$. Hence in $20k$ rounds, the randomized algorithm will succeed with probability 0.9999.

Using typical values for broadcast radius and maximum speed of nodes, in a vehicular wireless ad-hoc network from [1, p. 68], we approximate the number of nodes in each hexagonal partition to be ≈ 20. Substituting $k = 20$ for number of nodes in one hexagonal cell in a vehicular ad-hoc network, in 99.99% of the trials it will succeed in at most 400 rounds. So when the number of nodes in the network is less than 5% of the size of the id space, this could be a good strategy.

Algorithm 1 gives the pseudo code for the randomized algorithm. Each node runs an instance of the algorithm and broadcasts with probability $\frac{1}{k}$ in each round. Note that the randomized algorithm does not require any knowledge about the size of the id space. The broadcasting node includes the ids of all the nodes it has heard from in its broadcast message. When a listening node finds its id in the received message, it learns about its previous broadcast success and stops broadcasting in further rounds. The number of successful broadcasts information is used to increase the broadcast probability of the remaining nodes for faster termination.

Table 3.1 presents the simulation data for one hundred runs of Algorithm 1. We observe that the average number of rounds for termination for one hundred simulations is $\approx ek$ and all simulations terminate within $5k$ rounds. This is expected

as the probability of success after $5k$ rounds is ≈ 0.9. Also we observe that Algorithm 1 performs better than expected. This is due to disabling of self broadcast by nodes once they broadcast successfully and learn about their success in subsequent rounds.

Algorithm 1 Randomized neighbor discovery algorithm

```
1:  procedure RandomizedDiscovery (k, id)
2:      ▷ k is number of nodes in network, id is unique process identifier
3:      ▷ Broadcast(id) and Listen() are primitives used for communication
4:      neighborSet ← {∅}
5:      transmitSuccess ← False                    ▷ store ids of neighbors discovered
6:      while (|neighborSet| ≤ k) do
7:          randint ← Random(1, k^2)          ▷ From uniform distribution on 1 ... k^2
8:          if randint ≤ k and transmitSuccess = False then
9:              Broadcast({id ∪ neighborSet})
10:         else
11:             message ← Listen()
12:             if message.status = success then
13:                 insert(neighborSet, message.data)
14:                 if (id in neighborSet) then
15:                     transmitSuccess ← True
16:                 end if
17:             end if
18:         end if
19:     end while
20:     return neighborSet
21: end procedure
```

k	*min rounds*	*max rounds*	*mean*	*standard deviation*
20	36	94	55	9
60	123	216	164	17
100	228	338	271	21
140	327	439	378	27
180	428	562	490	30
220	497	704	600	36

Table 3.1: Simulation data for randomized algorithm

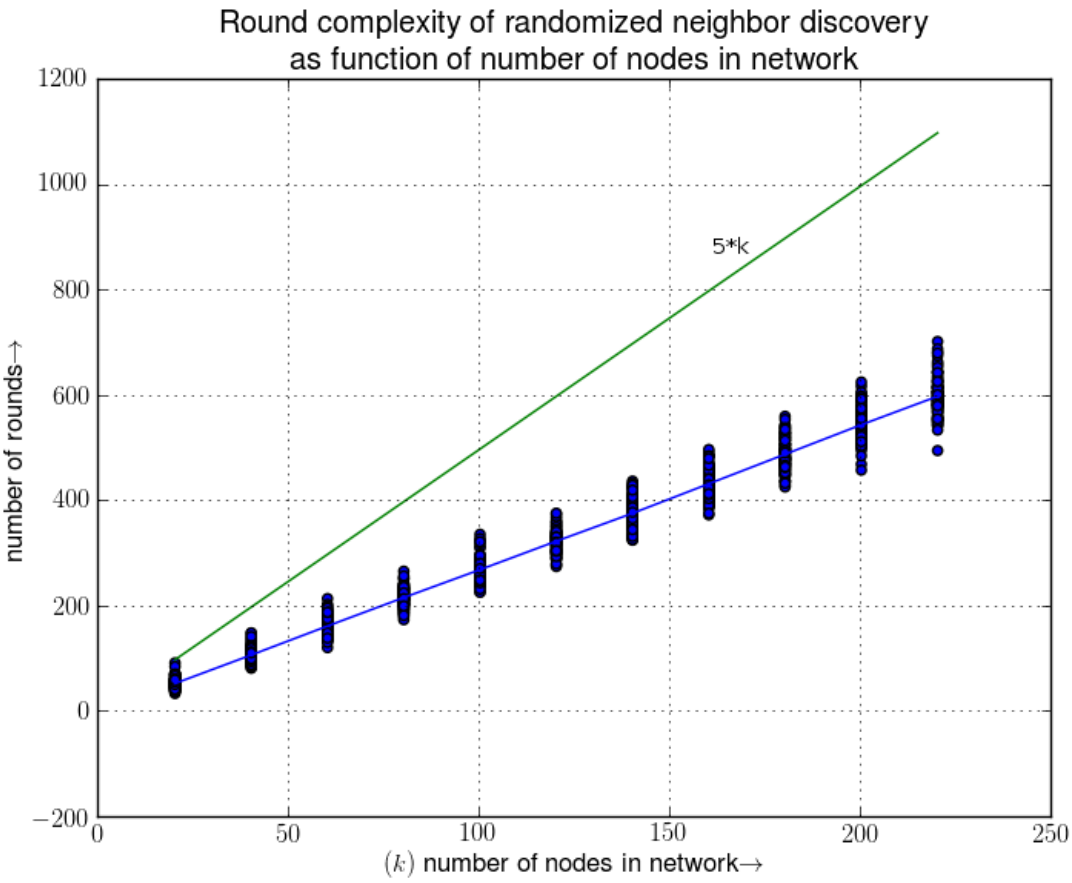

Figure 3.1: Scatter plot of simulation data for 100 iterations

The drawback of this approach is that it requires knowledge of the number of

nodes k in the network. This can be ameliorated by using an exponential back-off based network size estimator. It can start with an estimate of $k = 2$ and double the estimate of k every $5k$ rounds until it hears a successful broadcast or some upper bound on estimate of k is reached. We suggest $5k$ rounds of broadcast before doubling the estimate as the probability of successful broadcast is approximately 0.90 when the correct value of k is reached. If we go for higher accuracy the required number of rounds for each guess will increase significantly with marginal gain in accuracy as $m \propto \log(\epsilon)$ and increase in accuracy by one significant digit will lead to doubling of the number of rounds.

3.4 Deterministic Algorithm With Collision Detection

The deterministic algorithm for neighbor discovery using the collision detection radio model uses unique ids of the nodes to dynamically discover the correct schedule for the broadcast. The idea is similar to searching in a balanced binary search tree as shown in figure 3.2. A similar strategy is used for reaching consensus in radio networks in [10] and controller area network protocol arbitration[21]. The nodes use the binary representation of their ids as initial input and based on the broadcast status history of previous rounds take the broadcast decision in the next round to minimize collision chances. All nodes use binary strings of the same length to represent their ids, padding most significant bits with 0 when required.

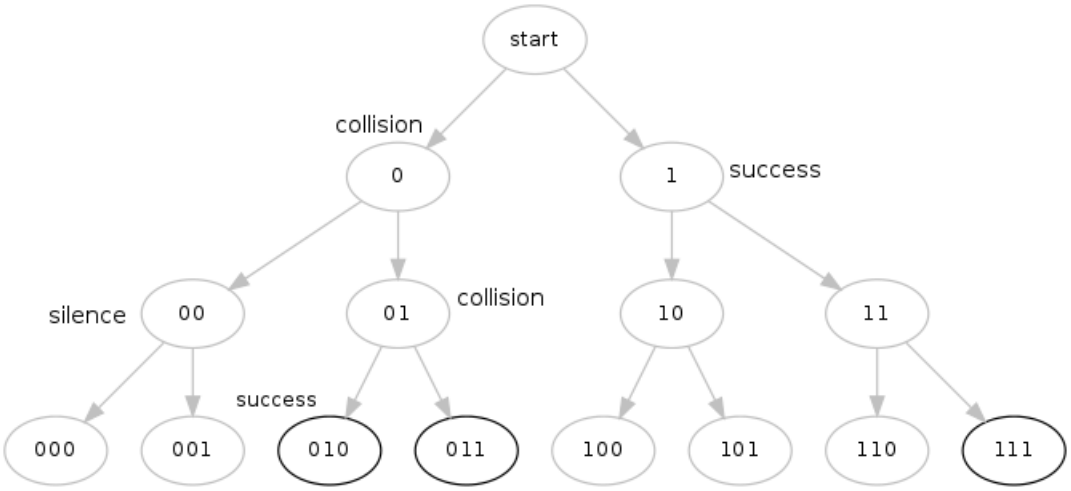

Figure 3.2: Broadcast progress with 3 bit id. Initially all nodes with 0 as most significant bit broadcast. They observe collision as there are two nodes in the network with 0 as msb (010,011). In the next round nodes with msb 00 broadcast and observe silence, hence ids with prefix 00 are not present. Then nodes with msb 01 broadcast and observe collision. Finally nodes with ids 010 and 011 broadcast alone.

Each node locally simulates the search operation on the binary search tree whose leaf nodes have ids $\{1, ..., maxId\}$, and agree on the subtree being searched in each round. Initially all nodes having zero as the most significant bit(msb) broadcast while those having 1 as msb listen. When they observe silence, it means no node having id in the left subtree is present and they start searching in the right subtree. If they observe a collision then it is interpreted as more than one node with id in the left subtree is present and it is further divided into left and right subtree and recursively searched. Observing a successful broadcast is inferred as only one node was present in the active subtree(the subtree currently being searched) and that subtree does not require further investigation.

The algorithm gurantees that at least one node is able to broadcast successfully every $\lceil 2\log_2(n) \rceil$ rounds where n is size of id space. The worst case occurs when nodes

with maximum consecutive ids are present in the network. In that case, starting from the msb, both 0 and 1 are tried at each bit position from the most significant to the least significant bit and the conflict is finally resolved at the least significant bit in the $\lceil 2\log_2(n) \rceil^{th}$ round. Hence the algorithm gurantees that it will terminate in at most $\lceil 2k\log_2(n) \rceil$ rounds, where k is number of nodes in the network.

Algorithm 2 gives the pseudo code for the deterministic algorithm for the collision detection model. Each node in the network executes a copy of the algorithm locally. We arbitrarily choose to traverse the left subtree before the right subtree. The algorithm uses a stack to track the most significant bit prefix used to decide the broadcast status in the current round, conveniently called current round prefix here. When silence is heard or when a node broadcasts alone, all nodes pop the bit on the stack top; this corresponds to discarding the subtree with prefix represented by the current state of stack. When collision is detected all nodes push a one followed by zero on the stack top. This corresponds to recursively exploring the left subtree followed by the right subtree. The stack is initialized with $\{1, 0\}$ to explore the left subtree followed by the right subtree. For exploring the right subtree first followed by the left subtree initialize the stack with $\{0, 1\}$ and push zero followed by one on collision detection. When a node broadcasts in a round, it always assumes collision has occurred and explores ids with most significant bit prefix corresponding to the left and the right subtree of the current round prefix. If it is the only node broadcasting in a round then it will continue broadcasting once every two consecutive rounds until it reaches its least significant bit in at most $\lceil 2\log_2(n) \rceil$ rounds. When a node broadcasts the bit prefix corresponding to its id then it marks itself successful and disables broadcasting. When a node listens to a single broadcast it waits until it hears silence in two consecutive rounds. When a node is in the wait mode, it only listens.

Algorithm 2 Deterministic neighbor discovery with collision detection

```
1: ▷ Broadcast(id) and Listen() are primitives used for communication
2: ▷ Variables used
3: IdSize, flagBit, globalIndex, localIndex ← 0
4: enableTransmit ← True
5: success ← False
6: bitStack ← {1, 0}
7: procedure DeterministicWithCD (k, id, maxID)                ▷ main function
8:     ▷ k is number of nodes in network, id is unique process identifier
9:     ▷ maxID is maximum id in id-space
10:    IdSize ←Binary(maxID)
11:    idbits ←Binary(id)                          ▷ pad to make it same as IdSize
12:    neighborList = ∅                            ▷ store ids of neighbors discovered
13:    while (|neighborList| ≤ k − 1) do
14:        myChoice ←BroadcastStatus()
15:        if myChoice = True then
16:            Broadcast(id)
17:            localIndex ← localIndex + 1
18:            bitStack.push([1, 0])          ▷ explore left subtree before right subtree
19:            globalIndex ← globalIndex + 1                  ▷ descend into sub tree
20:        else
21:            message ← Listen()
22:            if message.status = success then
23:                neighborList.append(message.data)
24:                Wait for IdSize − globalIndex rounds
25:                ▷ wait until broadcaster disables itself
26:                if bitStack.peek() = 1 then
27:                    globalIndex ← globalIndex − 1
28:                end if
29:                bitStack.pop()  ▷ successful broadcast, hence no node in same sub
   tree
30:            else if message.status = collision then
31:                bitStack.push([1, 0])                  ▷ descend into sub tree
32:                globalIndex ← globalIndex + 1
33:            end if
```

```
34:         end if
35:     end while
36:     return neighborList
37: end procedure
38: procedure BroadcastStatus( )
39:     if success = True then
40:         enableTransmit ← False        ▷ Always listen after successful broadcast
41:     else
42:         flagBit ← bitStack.peek()
43:         if localIndex < globalIndex then              ▷ id not in active sub tree
44:             enableTransmit ← False
45:         else if localIndex = globalIndex then
46:             mychoice ← idbits[localIndex]
47:             if mychoice = flagBit then
48:                 enableTransmit ← True                 ▷ id in active sub tree
49:             else
50:                 enableTransmit ← False
51:             end if
52:         end if
53:         if localIndex = IdSize then
54:             enableTransmit ← False            ▷ reached lsb, broadcast success
55:             success ← True
56:         end if
57:     end if
58: end procedure
```

k	*min rounds*	*max rounds*	*mean*	*standard deviation*	$k\log(n)$
20	902	959	927	11	640
60	2599	2708	2656	20	1920
100	4269	4389	4325	28	3200
140	5872	6037	5956	30	4480
180	7483	7646	7562	35	5760
220	9083	9275	9156	40	7040

Table 3.2: Simulation data for the deterministic algorithm with collision detection

Table 3.2 presents the simulation data for this deterministic algorithm. Figure 3.3 presents the plot of rounds required for completion as a function of the number of nodes in the network, with ids picked from an id space of size 32 bits. For simulation ids were chosen randomly from the uniform distribution on $\{1, ..., 2^{32}\}$. As expected the round complexity in each run is between $k\log(n)$ and $2k\log(n)$. The average round complexity for 100 runs of the algorithm was observed to be $\approx 1.35k\log(n)$. For id space greater than 5 bits long(max id greater than 31), the deterministic algorithm is expected to be slower than the randomized algorithm presented in the previous section(Algorithm 1), since $k\log_2(n) > 5k$ for $n > 32$.

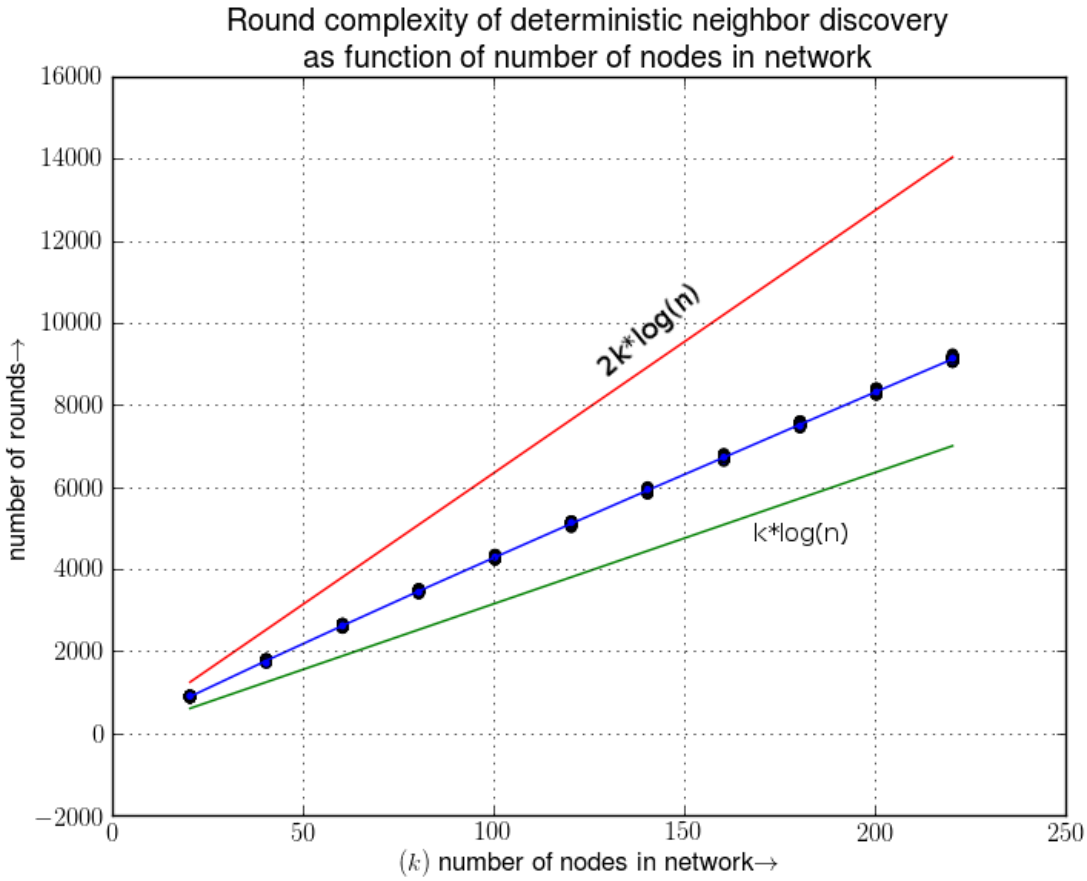

Figure 3.3: Simulation result for the deterministic algorithm. Ids picked randomly from the uniform distribution on $\{1, ..., 2^{32}\}$

3.5 Deterministic Algorithm For No Collision Detection Radio Model

The deterministic algorithm for neighbor discovery in the no collision detection model is based on a deterministic construction of combinatorial objects known as *k-selective* family of subsets[20] and *k-selector* family of subsets[7]. A k-selective family of subsets of any set U has the property that for any arbitrary subset $K \subset U, |K| \leq k$, there always exists a subset in the k-selective family that intersects with K at exactly one element. A k-selector family of subsets has the property that given any arbitrary

subsets X and Y of U, there always exists a subset in the k-selector family that intersects with set X at exactly one element but has no element in common with set Y. It is easy to show that k-selective and k-selector family of subsets always exist as the trivial family of subsets consisting of singleton elements of U is a k-selective as well as k-selector. An asymptotically efficient deterministic algorithm for neighbor discovery in the no-collision-detection model is based on constructing a k-selective or k-selector family of size polynomial in $\log(n)$, where $n = |U|$ is the size of the id space. In [14], the authors prove the existence of selectors of size $O(k \log(n))$ using probabilistic methods.

Viqar gave an $O(k \log(n))$ algorithm [1, p. 74] for deterministic neighbor discovery using a k-selector family of subsets, where k is the number of nodes in network and $n = |U|$ is the size of the id space.

Definition 1 A family $S = S_1, S_2, ..., S_l$ of subsets of a set U,$|U| = n$, is called *k-selective* if for any subset X of U, $|X| \leq k$, there exists $S_i \in S$ such that $|S_i \cap X| = 1$. [20, p. 711].

Definition 2 A family $S = S_1, S_2, ..., S_l$ of subsets of U, $|U| = n$, is called a *k-selector* if for any pair X, Y of disjoint subsets of U such that $\frac{k}{2} \leq |X| \leq k$ and $|Y| = k$, there exists S_i such that $|S_i \cap X| = 1$ and $|S_i \cap Y| = 0$. [7].

From the definition of k-selective and k-selector family of subsets, it follows that one can construct a k-selective family of subsets by taking the union of $(2^i, n)k$-selectors for $i = 1, ..., \log(k)$. Hence if we have a deterministic algorithm for constructing k-selectors for any value of k, we can use it for deterministic construction of a k-selective family of asymptotically the same size.

A k-selector family of subsets when used as the schedule for deterministic radio broadcast provides a progress guarantee for neighbor discovery. This was proved in [12, lemma 1].

Lemma 1 : Let S be a k-selector over $\{1, 2, ..., N\}$ and let Z be any subset of $\{1, 2, ..., N\}$ such that $k \leq Z \leq 2k$. Let Y be the set of all elements $y \in Z$ such that there exists a set s_i in S such that $|s_i \cap Z| = 1$. Then Y contains more than half of the elements of Z.

The above lemma provides a guarantee that at least half of the nodes present in the network are able to broadcast successfully when a k-selector family of subsets is used as the broadcast schedule. Given a k-selector family of subsets for $U = \{1, ..., IdMax\}$, we can use it to schedule deterministic broadcast in the no-collision-detection radio network. We order the k-selector family of subsets in any arbitrary order such that all nodes agree on the ordering and then the subset at index i decides which set of nodes broadcast in the ith round. When a node's id is present in the subset at index i, it broadcasts in round i. This ensures that at least $\frac{k}{2}$ nodes present in the network broadcast alone in $|S|$ rounds, where $|S|$ is the size of the selector. We then repeat this process with a $\frac{k}{2}$-selector and so on until all nodes broadcast successfully in less than $2k \log(n)$ rounds.

Indyk gave a deterministic algorithm[7] for the construction of k-selectors of size $O(kd^3 \log^3 n)$ using an (n, k, d) *disperser*, where d is the degree of the disperser graph, and a *p-colliding* family of functions. The dependency graph of the deterministic algorithm[1, p. 74] based on k-selectors is presented in figure 3.4.

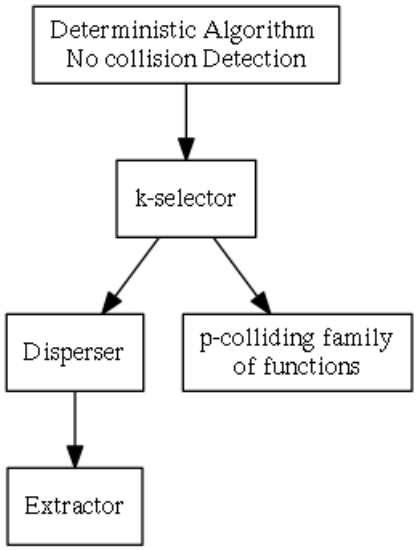

Figure 3.4: Dependency graph of the deterministic algorithm for neighbor discovery in the no-collision detection model based on deterministic construction of k-selectors.

A disperser is a bipartite graph with strong random like properties and a p-colliding family of functions is like a universal hash function and provides a guarantee that the probability of two elements in the domain getting mapped to the same element in the range is less than some constant $p \in (0, 1)$. Ta-Shma gave an algorithm[8] for the explicit construction of an (n, k, d) disperser for all $d \geq poly\log(\frac{n}{\epsilon})$ and $k \geq poly\log(\frac{n}{\epsilon})$, where k is the number of nodes in the network, n is the size of the id space and $\epsilon = \frac{3}{4}$.

The pseudocode for the construction of a k-selector is presented in Algorithm 3 and is based on the deterministic construction of a k-selector presented by Indyk in [7]. Figure 3.5 gives a plot of the expected size of the k-selector as a function of the number of nodes in the network(k) for id space $U = \{1, ..., 10000\}$. The algorithm uses the deterministic construction of a disperser presented in [8] and the p-colliding family of functions presented in [7, p. 3]. The expected size of the deterministic

Algorithm 3 *k-selector* Construction algorithm

procedure KSELELECTOR(n, k)
  ▷ $|A| = n, |B| = k$, common degree $= d$
  ▷ Return family of subsets of A
  $D(A, B, E) \leftarrow$Disperser(n, k)
  $d \leftarrow \delta(v)$, $v \in A$ ▷ all vertices in A have same degree
  $h_i : A \mapsto B$, $h_i(x) = \Gamma_D(x)^{i^{th}}$ ▷ order edges of each vertex in A arbitrarily
  $G \leftarrow$PcollidingFamily$(\frac{1}{48d}, n)$
  $hashMap \leftarrow \{\emptyset\}$ ▷ key = pair(int,int), Value = Set $\subset A$
  for (v in A) **do**
    $h_x \leftarrow h(v)$
    for (g_i in G) **do**
      $g_x \leftarrow v \bmod g_i$
      $hashMap[(h_x, g_x)].insert(v)$
    end for
  end for
  $S \leftarrow \{\emptyset\}$
  for $(key, value)$ in $hashMap$ **do**
    $S.append(value)$
  end for
  return S
end procedure4

k-selector is $48^2 k \log^6(n)$. The constant factor associated with this method is large and dominates the round complexity when the size of the id space is less than 10000. When the size of the id space is less than 10000, the deterministic Algorithm 3 yields the trivial selector consisting of the singletons $\{\{1\},\{2\},...,\{n\}\}$, $n \leq 10^5$. Figure 3.6 plots the expected size of the k-selector as a function of the size of the id space. The large expected size of the k-selector family of subsets of the id space U obtained by the deterministic construction makes us conclude that it is no better than the trivial algorithm in which the node with id i broadcasts in round i when the size of the id space is in the range (1,10^5).

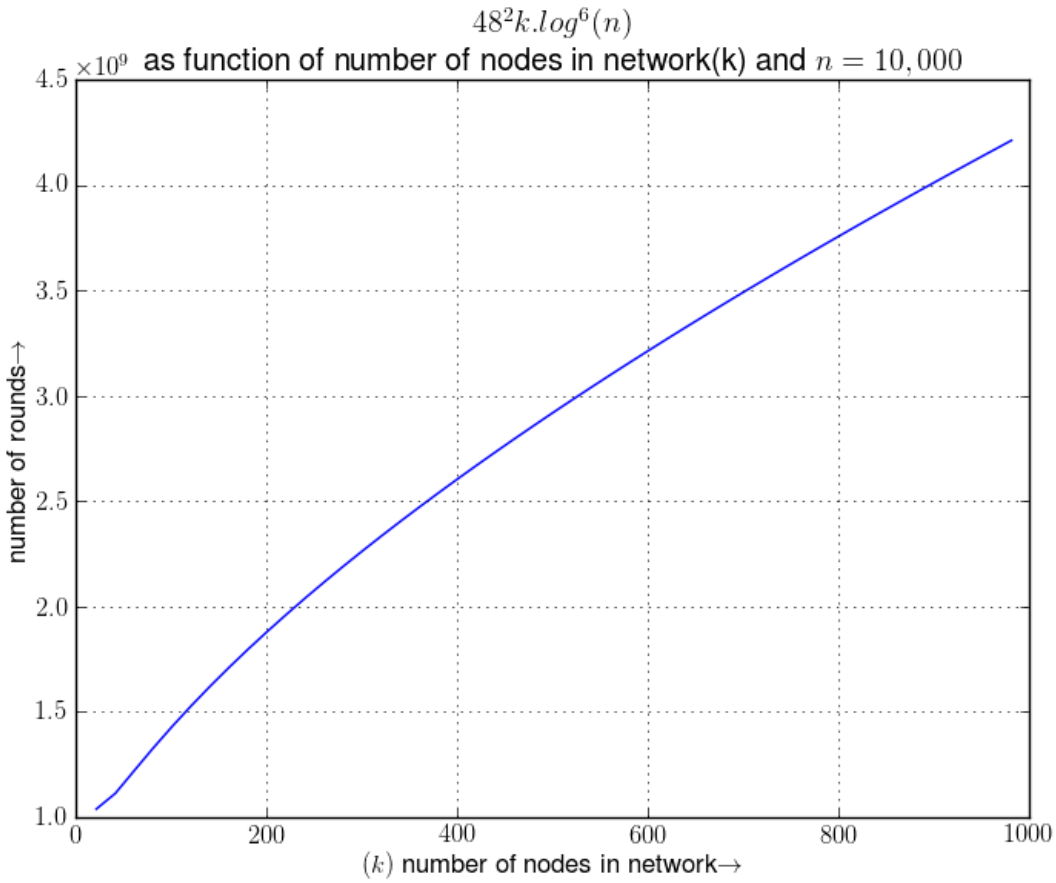

Figure 3.5: Expected k-selector size as function of k

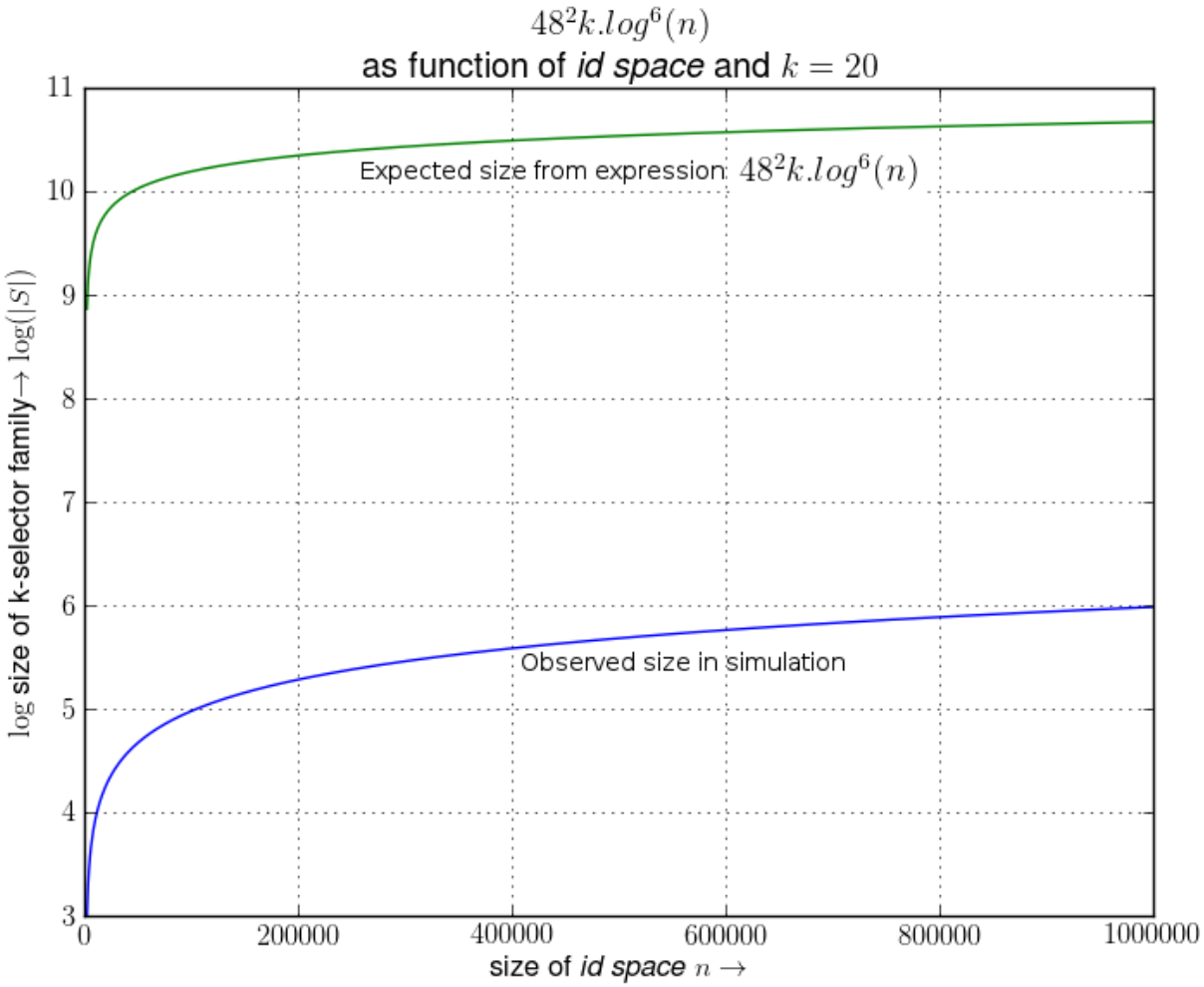

Figure 3.6: Expected k-selector size as function of *id space*

Since the size and computational complexity of the deterministic construction of k-selectors depends on the disperser and p-colliding family of functions constructions, we now analyze the computational complexity involved in their construction.

Definition 3. A (n, k, d)-*disperser* is a bipartite graph $G = (A, B, E)$ such that $|A| = n, |B| = k$ and for any $Z \subset A$ such that $|Z| = \frac{k}{2}$, the size of the set $\Gamma_G(Z)$ of neighbors of Z has size at least $\frac{k}{4}$. The degree of each node in A is d. [8]

Definition 4 . *p-colliding family* of functions is $G = \{g_0, g_1, ..., g_{r-1}\}, g_i : A \to [u]$ such that for any $x, y \in A$ and $x \neq y$, we have $Pr_{g \in G}[g(x) = g(y)] \leq p$. [7, p. 3]

A p-colliding family of functions (G) of size $O(\frac{1}{p}\log(n))$ is constructed by selecting $r = \lceil \frac{1}{p}\log(n) \rceil$ consecutive prime numbers $\{p_1, p_2, ..., p_r\}$ such that the smallest of them is greater than $\frac{1}{p}\log^2(n)$. We then define $g_i \in G$, $g_i = x \mod p_i$, $x \in U$.

The pseudocode for the construction of a p-colliding family is presented in Algorithm 4. The probability of collision used in Algorithm 4 is $p = \frac{1}{48\log(n)}$, where n is the size of the id space. It tests primality of $48\log^2(n)$ numbers of order $O(\log^3(n))$. Since primality testing of a number m has computational complexity $O(\log^6(m))$ using the AKS algorithm[22], p-colliding family of functions has computational complexity of order $O(\log^2(n)\log^6\log(n))$.

The p-colliding family of functions contribute a large constant multiplicative factor 48^2 and a poly-logarithmic factor $O(\log^5(n))$ to the size of the deterministic k-selector. Finding an optimal size p-colliding family of functions may lead to much smaller constant factor and reduced size of deterministic k-selector.

The pseudocode for the construction of a disperser is presented in Algorithm 5. The simplest disperser is the complete bipartite graph. Efficient construction of dispersers requires the degree of the bipartite graph to be of order $O(\log(n))$. Constructing a sparse explicit disperser is a non-trivial exercise. The explicit construction of a disperser presented in [8] uses the construction of combinatorial objects known as extractors and block-wise extractors. Extractors take as input a source with low randomness and a uniform probability distribution function on $\{1, ..., \log(n)\}$ and outputs a uniform random distribution on $\{1, ..., m\}$, where $m > \log(n)$.The exact value of m is dependent on the amount of randomness present in the input random source. Hence extractors act as randomness-extracting devices. The disperser construction in Algorithm 5 uses a uniform random distribution on {1,...,k}, where

Algorithm 4 p-colliding family of functions

```
1: procedure PCOLLIDINGFAMILY(p, n)
2:     ▷ p is probability of collision and n is size of id space
3:     r ← (1/p) log (n)
4:     u ← (1/p) log^2 (n)
5:     listOfPrimes ← ∅
6:     if (u ≡ 1 mod 2) then
7:         x ← u
8:     else
9:         x ← u + 1
10:    end if
11:    while |listOfPrimes| < r do
12:        if IsPrime(x) then
13:            listOfPrimes.append(x)
14:        end if
15:        x ← x + 2
16:    end while
17:    return listOfPrimes
18: end procedure
```

k is the number of nodes in the network. The computational complexity for the construction of dispersers is of order $O(n)$, where n is the size of the id space.

3.5.1 An Interesting Observation

While trying to reduce the size of k-selectors, we observed that if we replace the p-colliding family of functions with the first $\log_2(n)$ primes, we get a family of subsets of the id space of size $\approx 2 * k \log(n) \log\log n$ with selector-like properties. Using such a family of sets as a broadcast schedule leads to successful broadcast more than 90% of the time. The plot of the number of rounds required for neighbor discovery using such a family of subsets is presented in figure 3.7.

Algorithm 5 Disperser construction using pseudorandom numbers

```
1:  procedure DISPERSER(A, B)
2:      ▷ |A| = n, |B| = k
3:      G ← BipartiteGraph(A, B)
4:      d ← min(k/2, log_k(n))
5:      for (v_A in G.A) do
6:          for i in {1, ..., d} do
7:              v_B ← Random(1, k)          ▷ from uniform distribution on 1..k
8:              Add edge (v_A, v_B) to G
9:          end for
10:     end for
11:     return G
12: end procedure
```

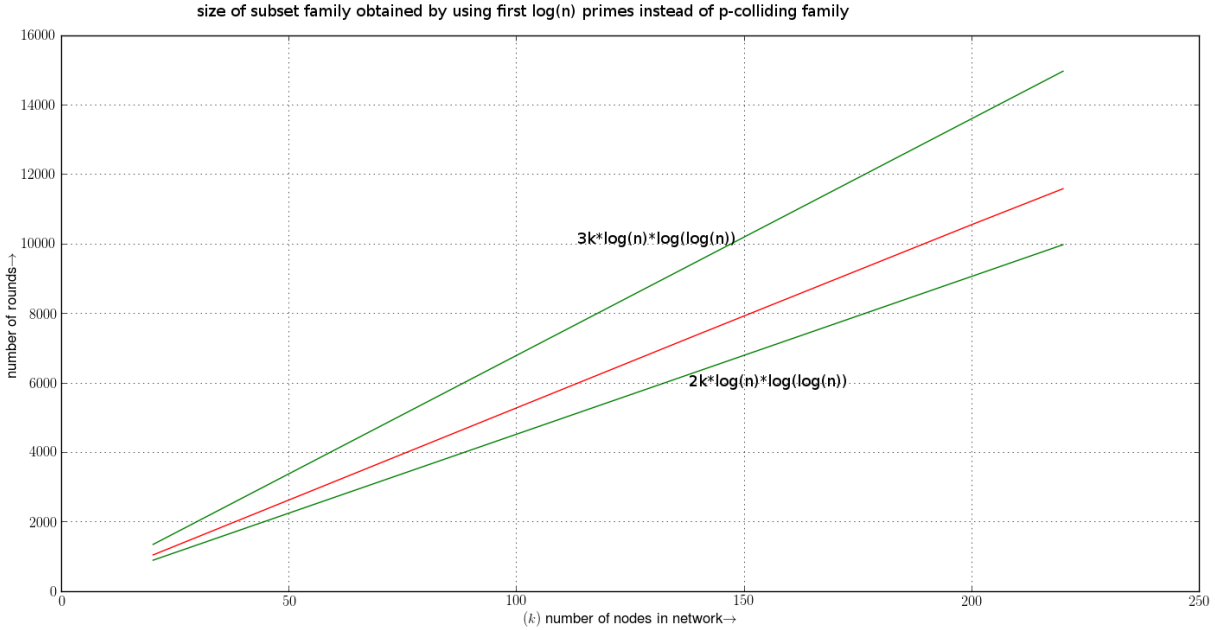

Figure 3.7: Size of subset family obtained by replacing p-colliding family by first $\log_2(n)$ primes

This idea is inspired by the algorithm Prime-Steps presented in [9] and the proof of the deterministic k-selector construction provided in [7]. The algorithm

Prime-Steps[9, page. 214] tried to solve the wake-up problem in the synchronous, no-collision-detection model. It exploited the property of sequences of the form $p_1, 2p_2, ...$ and $p_2, 2p_2, ...$, where p_1 and p_2 are prime numbers, that they collide once every p_1p_2 rounds. The algorithm finds a sequence of prime multiples of sufficient length such that all nodes wake up successfully.

The proof of the deterministic algorithm presented in [7, p. 700] used the disperser property to prove that any arbitrary subset K of the id space will be separated into at least $\frac{k}{4d}$ distinct groups by the disperser, where d is the degree of the disperser. Then it used the p-colliding family of functions to further subdivide these $\frac{k}{4d}$ subsets into even smaller subsets such that at least one element ends up as a singleton set. This subset will then correspond to subset s_i such that $|s_i \cap K| = 1$.

Now if we replace the p-colliding family by the first $\log_2(n)$ primes, then for most of the random k-element subsets of the id space, we expect at least one prime(p_i) in first $\log_2(n)$ primes to map at least one element in the given k-element subset to a unique number modulo p_i.

Variations of this method may be tested for accuracy, like using the first $\log_2(n)$ primes greater than k, the number of nodes in network.

4. CONCLUSION

The simulation results make us conclude that the randomized algorithm is most efficient in terms of round complexity and should be the algorithm of choice unless some factor other than communication efficiency is of prime importance. The randomized algorithm is simple to implement, is light on local computation and outperforms the deterministic algorithms in both collision detection and no-collision detection models.

For deterministic neighbor discovery, the algorithm for the collision detection model is efficient in terms of local computation and communication rounds. It is slower than the randomized algorithm by a factor of $\log(n)$ but faster than the deterministic algorithm for the no-collision-detection model by a factor of $\log^5(n)$. If deterministic neighbor discovery is desired and the cost of communication is expected to be more than the cost of upgrading to the collision detection model, then it should be the algorithm of choice.

The deterministic algorithm for the no-collision-detection model has a large constant factor which makes it no better than the simplest strategy of serially trying all possible ids, when the size of the id space is less than 10^5. The large multiplicative constant factor in the deterministic construction of k-selectors comes from using a p-colliding family of functions which additionally contributes a multiplicative factor of order $\log^5(n)$. The combinatorial building blocks required for constructing k-selectors is computationally demanding, as the disperser construction takes cpu cycles in the order of the size of the id space. Computing and storing a disperser graph for a 32 bit id space could take up billions of cpu cycles and gigabytes of memory.

Efficient practical deterministic neighbor discovery protocol for the no-collision-detection model that works for all sizes of the id space is still an open problem. Efficient construction of explicit dispersers of low degree is another very important problem having application in construction of expander graphs, leader election and other interesting applications discussed in the survey paper [23].

REFERENCES

[1] Saira Viqar. "Communication algorithms for wireless ad hoc networks". PhD thesis. Texas AM University, Aug. 2012, pp. 1–125 (cit. on pp. 2, 3, 8, 14, 24, 25).

[2] Alejandro Cornejo, Saira Viqar, and Jennifer L. Welch. "Reliable neighbor discovery for mobile ad hoc networks". In: *Ad Hoc Networks* 12 (2014), pp. 259–277 (cit. on p. 2).

[3] Saira Viqar and Jennifer L. Welch. "Deterministic collision free communication despite continuous motion". In: *Ad Hoc Networks* 11.1 (2013), pp. 508–521 (cit. on pp. 2, 3).

[4] Lin Chen, Kaigui Bian, Lin Chen, Cong Liu, Jung-Min "Jerry" Park, and Xiaoming Li. "A group-theoretic framework for rendezvous in heterogeneous cognitive radio networks". In: *The Fifteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014*. 2014, pp. 165–174 (cit. on pp. 2, 7).

[5] Lin Chen, Kaigui Bian, and Meng Zheng. "Heterogeneous multi-channel neighbor discovery for mobile sensing applications: theoretical foundation and protocol design". In: *The Fifteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014*. 2014, pp. 307–316 (cit. on pp. 2, 6, 7).

[6] Sudarshan Vasudevan, Micah Adler, Dennis Goeckel, and Don Towsley. "Efficient algorithms for neighbor discovery in wireless networks". In: *IEEE/ACM Trans. Netw.* 21.1 (2013), pp. 69–83 (cit. on pp. 2, 7).

[7] Piotr Indyk. "Explicit constructions of selectors and related combinatorial structures, with applications". In: *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*. 2002, pp. 697–704 (cit. on pp. 3, 23–26, 30, 32, 33).

[8] Amnon Ta-Shma. "Almost optimal dispersers". In: *Combinatorica* 22.1 (2002), pp. 123–145 (cit. on pp. 3, 26, 29, 30).

[9] Leszek Gasieniec, Andrzej Pelc, and David Peleg. "The wakeup problem in synchronous broadcast systems." In: *SIAM Journal on Discrete Mathematics* 14.2 (2001), pp. 207 –222. ISSN: 08954801 (cit. on pp. 5, 6, 32, 33).

[10] Gregory Chockler, Murat Demirbas, Seth Gilbert, Nancy A. Lynch, Calvin C. Newport, and Tina Nolte. "Consensus and collision detectors in radio networks". In: *Distributed Computing* 21.1 (2008), pp. 55–84 (cit. on pp. 5, 7, 17).

[11] J. Komlos and Albert G. Greenberg. "An asymptotically fast nonadaptive algorithm for conflict resolution in multiple-access channels". In: *Information Theory, IEEE Transactions on* 31.2 (1985), pp. 302–306. ISSN: 0018-9448 (cit. on p. 5).

[12] Leszek Gasieniec, Aris Pagourtzis, Igor Potapov, and Tomasz Radzik. "Deterministic communication in radio networks with large labels". In: *Algorithmica* 47.1 (2007), pp. 97–117 (cit. on pp. 6, 24).

[13] Bogdan S. Chlebus, Leszek Gasieniec, Alan Gibbons, Andrzej Pelc, and Wojciech Rytter. "Deterministic broadcasting in unknown radio networks". In: *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, January 9-11, 2000, San Francisco, CA, USA*. 2000, pp. 861–870 (cit. on pp. 6, 7).

[14] Marek Chrobak, Leszek Gasieniec, and Wojciech Rytter. “Fast broadcasting and gossiping in radio networks”. In: *J. Algorithms* 43.2 (2002), pp. 177–189 (cit. on pp. 6, 24).

[15] Tomasz Jurdzinski and Grzegorz Stachowiak. “Probabilistic algorithms for the wake-up problem in single-hop radio networks”. In: *Theory Comput. Syst.* 38.3 (2005), pp. 347–367 (cit. on p. 6).

[16] K.R. Narayanan and H.D. Pfister. “Iterative collision resolution for slotted Aloha: an optimal uncoordinated transmission policy”. In: *Turbo Codes and Iterative Information Processing (ISTC), 2012 7th International Symposium on*. 2012, pp. 136–139 (cit. on p. 7).

[17] Michael J. McGlynn and Steven A. Borbash. “Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks”. In: *Proceedings of the 2Nd ACM International Symposium on Mobile Ad Hoc Networking & Computing*. MobiHoc ’01. Long Beach, CA, USA: ACM, 2001, pp. 137–145. ISBN: 1-58113-428-2 (cit. on p. 7).

[18] J. F. Hayes. “Adaptive technique for local distribution.” In: *IEEE Transactions on Communications* 26 (1978), pp. 1178 –1186. ISSN: 00906778 (cit. on p. 7).

[19] J. I. Capetanakis. “Generalized TDMA.” In: *IEEE Transactions on Communications* 27 (1979), pp. 1476 –1484. ISSN: 00906778 (cit. on p. 7).

[20] Andrea E. F. Clementi, Angelo Monti, and Riccardo Silvestri. “Selective families, superimposed codes, and broadcasting on unknown radio networks”. In: *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA*. 2001, pp. 709–718 (cit. on pp. 7, 23, 24).

[21] Bosch. *CAN specification 2.0*. Ed. by Robert Bosch Gmbh. 1991. URL: http://www.kvaser.com/software/7330130980914/V1/can2spec.pdf (cit. on p. 17).

[22] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. "Primes is in P." In: *Annals of Mathematics* 2 (2004), p. 781. ISSN: 0003486X (cit. on p. 30).

[23] Noam Nisan and Amnon Ta-Shma. "Extracting randomness: a survey and new constructions". In: *J. Comput. Syst. Sci.* 58.1 (1999), pp. 148–173 (cit. on p. 35).

APPENDIX

The python code for simulating deterministic neighbor discovery in the collision detection model.

```
import random
import sys
import os
import math

class node:
    IdSize = int(math.pow(2,32))
    choiceVector = list()
    broadcasterID = list()
    maxIdLength = len(str(bin(IdSize)))-2
    def __init__(self,id_):
        self.currentRoundBit = 1
        self.globalBitIndex = 0
        self.ID = id_
        self.binStrID = str(bin(id_))[2:]
        if(len(self.binStrID)<node.maxIdLength):
            padd = '0'*(node.maxIdLength-len(self.binStrID))
            self.binStrID = padd + self.binStrID
        self.knownID = [id_]
        self.localBitIndex = 0
        self.broadcastEnable = True
        self.broadcastSuccess = False
        self.detectCollision = False
        self.bitStack = [1,0]
        self.broadcastHist = []
        self.waitForWinner = False

    def prepareNextRound(self):
        if(self.broadcastSuccess):
            return
        if(self.waitForWinner):
            self.broadcastEnable = False
        else:
            try:
                self.currentRoundBit = self.bitStack[-1]
            except IndexError:
                self.printDebug()
```

```
            if(self.localBitIndex < self.globalBitIndex):
                self.broadcastEnable = False
            elif(self.localBitIndex == self.globalBitIndex):
                myChoice = int(self.binStrID[self.
                    localBitIndex])
                if(myChoice == self.currentRoundBit):
                    self.broadcastEnable = True
                else:
                    self.broadcastEnable = False
            else:
                print "Error local index greater than global
                    "
        if(self.localBitIndex == node.maxIdLength):
            self.broadcastEnable = False
            self.broadcastSuccess = True
            #print "Success ID :", self.ID

    def makeBroadcastChoice(self):
        if(self.broadcastSuccess):
            node.choiceVector.append(0)
            return False
        if(self.broadcastEnable):
            node.choiceVector.append(1)
            node.broadcasterID.append(self.ID)
            return True
        else:
            node.choiceVector.append(0)
            return False

    def processSilence(self):
        self.detectCollision = False
        self.broadcastHist.append(False)
        if(not self.waitForWinner):
            val = self.bitStack.pop()
            if(val == 1):
                self.globalBitIndex -= 1
        if(len(self.broadcastHist) > 2):
            if((self.broadcastHist[-1] == False) and
                (self.broadcastHist[-2] == False)):
                if(self.broadcastHist[-3] == True):
                    # two consecutive silence after
                        broadcast means winner of round x-2
```

```
                    self.broadcastSuccess = True
                    self.broadcastEnable = False
                    #print "Success ID :", self.ID
                else:
                    self.waitForWinner = False

    def decideWinner(self):
        if(not self.broadcastEnable): # Receiving Mode
            broadcastCount = sum(node.choiceVector)
            if(broadcastCount == 1): # success
                if(not self.waitForWinner):
                    val=self.bitStack.pop()
                    if(val == 1):
                        self.globalBitIndex -= 1
                self.knownID.append(sum(node.broadcasterID))
                self.detectCollision = False
                self.broadcastHist = []
                self.waitForWinner = True
            elif((broadcastCount > 1)): # collison
                self.detectCollision = True
                self.broadcastHist = []
                self.waitForWinner = False
            else: # silence
                self.processSilence()

        else : # Transmit Mode
            self.broadcastHist = [True]
            self.detectCollision = False
            if(self.localBitIndex == node.maxIdLength):
                self.broadcastSuccess = True
                #print "Success ID :", self.ID
                self.broadcastEnable = False

    def endRound(self):
        assert((self.broadcastEnable and self.
           detectCollision) == False)
        node.choiceVector = []
        node.broadcasterID = []
        if(self.broadcastEnable):
            self.localBitIndex += 1
            self.bitStack.append(1)
            self.bitStack.append(0)
```

```
            self.globalBitIndex += 1
        if(self.detectCollision):
            self.bitStack.append(1)
            self.bitStack.append(0)
            self.globalBitIndex += 1
        if(self.localBitIndex == node.maxIdLength):
            self.broadcastSuccess = True
            #print "Success ID :", self.ID

    def __call__(self):
        #self.printDebug()
        if self.broadcastSuccess:
            return (self.ID, True)
        return (0, False)

    def printDebug(self):
        print "currentRoundBit :", self.currentRoundBit
        print "globalBitIndex :", self.globalBitIndex
        print "ID :", self.ID
        print "binStrID :", self.binStrID
        print "knownID :", self.knownID
        print "localBitIndex :", self.localBitIndex
        print "broadcastEnable :", self.broadcastEnable
        print "broadcastSuccess :", self.broadcastSuccess
        print "detectCollision :", self.detectCollision
        print "bitStack :", self.bitStack
        print "broadcastHist :", self.broadcastHist
        print "waitForWinner :", self.waitForWinner



class Broadcast:
    def __init__(self, node_count=100, bitsize=32):
        node.IdSize = math.pow(2,bitsize)
        self.nodeCount = node_count
        self._nodes = list()
        self.discovered = []
        id_dict = dict()
        for i in range(1,node_count+1):
            id_ = random.randint(1,node.IdSize-1)
            if id_ in id_dict:
                id_dict[id_] += 1
```

```
            i -= 1
        else:
            id_dict[id_] = 1
            self._nodes.append(node(id_))
def simulate(self):
    rounds = 0
    res = dict()
    count = 0
    while(count != self.nodeCount):
        for node_i in self._nodes:
            node_i.prepareNextRound()
        for node_i in self._nodes:
            node_i.makeBroadcastChoice()
        for node_i in self._nodes:
            node_i.decideWinner()
        for node_i in self._nodes:
            node_i.endRound()
        for node_i in self._nodes:
            idS, isSucces = node_i()
            if(isSucces):
                self._nodes.remove(node_i)
                count +=1
                res[idS] = 1
                self.discovered.append(idS)
        rounds += 1
    return rounds
def printState(self):
    for node_i in self._nodes:
        node_i.printDebug()
```