An Algorithm for Gossiping and Broadcasting

Guo Tai Chen

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirement
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

January 2006

Your file Votre référence
ISBN: 0-494-14319-3
Our file Notre référence
ISBN: 0-494-14319-3

Abstract

An Algorithm for Gossiping and Broadcasting

Guo Tai Chen

Effectively disseminating the information among processors is an important feature for an interconnection network. Among all information dissemination models, broadcasting and gossiping are two of the most popular research topics. Finding the optimal broadcasting and gossiping schedule in an arbitrary net is an NP-Hard complete problem. This thesis presents an algorithm for both broadcasting and gossiping in arbitrary networks. The time complexity of this algorithm is $O(Rn^3)$, where R is the rounds used for the broadcasting or gossiping, and n is the total number of nodes in the network. The test has been implemented in both regularly used graphs and four of the *NS-2* network design models. The regularly used graph models include *Butterfly*, CCC_d, *de Bruijn* and *Shuffle Exchange*. And the four NS-2 internet models include GT-ITM random, GT-ITM Transit-Stub, Tiers and Inet.

Acknowledgements

I would like to express my sincere appreciation to my supervisor, Dr. H. Harutyunyan, for his insightful advice and invaluable encouragement which have helped me in my studies at Concordia. I would also like to thank Mr. Bin Shao who has helped me a great deal during the implementation of the algorithm with advice and discussion about algorithm itself.

Table of Content

Table of Figures

Table of Tables

1. Introduction

In order to find the best communication structure for parallel and distributed computing, a lot of work has been done in the study of the properties of interconnection networks. The ability to effectively disseminate the information among the processors is an important feature for an interconnection network. There are four main problems regarding information dissemination: broadcasting, accumulating, multicasting and gossiping. The main focus of this thesis is broadcasting and gossiping.

1.1 Problem Statement

In order to formalize the gossip problem, let us assume that each node in a graph has a token that needs to communicate to all of the other nodes in the graph. Tokens can be combined so that all communications involve constant time. The time needed for combining is irrelevant and treated as zero. Thus a formal definition can be stated as follows:

Initialization: Let $G = (V, E)$ be a graph (interconnection network). Each node, v, is associated to an initial singleton set $S(v)$ *which is* the initial data. These initial singleton sets are disjoint.

Allowable Steps: Each node can send its set to a neighbor or neighbors or receive a set from a neighbor or neighbors depending on the model of communication used. After receiving some sets, a node unites its existing set with all sets received at that step thus forming a new set for the next step.

Final state: All nodes must have the same set locally, containing all tokens in the initial singleton set. [8]

We have named the action of exchanging tokens between two nodes as a call. Then, for gossiping, we have the following constraints in the model considered in this thesis:

1) A node can only call one adjacent node per unit of time.
2) A node can participate in only one call per unit of time.
3) Two-way mode is used, that is, a node sends its set and receives a set from its neighbor at the same time.
4) Each call requires one unit of time.
5) Many calls can be performed in parallel.

In order to measure the gossip time, we employ the term *round*. In gossiping, a round is a set of parallel calls in the same time unit. A solution to a gossip problem is a sequence of feasible rounds that finish the communication.

In broadcasting we assume that a source node in a graph has a token that needs to be sent to all the other nodes in the graphs. The formal definition for broadcasting is simply stated:

Initialization: Let $G = (V, E)$ be a graph (interconnection network). A source node, v, associate an initial unique token (the initial data). None of other nodes has a token.

Allowable Steps: Depending on the model of communication used, the nodes that have received this token can send it to a neighbor or neighbors who have not yet received this token.

Final state: All nodes must have this token locally.

For broadcasting, we have the following constraints in the model considered in this thesis:

1) A node can only call one adjacent node per unit of time.
2) A node can participate in only one call per unit of time.
3) Each call requires one unit of time.

4) Many calls can be performed in parallel.

For broadcasting, a *round* is a set of parallel calls in the same time unit. The number of rounds is used to measure the broadcast time. Since one round spends one unit of time, the number of rounds is equal to the total time-steps needed for broadcasting. Given a graph G, the broadcast time $b(G, u)$, or simply $b(u)$, is the minimum broadcast time of graph G originated at node u.

I will study both of these two problems on common topologies and four NS-2 models that will be introduced in Chapter 2 and Chapter 5 respectively.

1.2 Previously-Known Heuristics

There are several previously known heuristics. Among all of them, Round_Heuristic introduced in [1] is the best existing heuristic. This heuristic is designed for both gossiping and broadcasting. Its performance in broadcasting and gossiping is considered in this thesis. In [10], the testing results from this heuristic in several commonly used graphs are presented. Most of the results are equal to the optimal broadcast time.

In [1], two heuristics are given. One is matching heuristic, while another is coloring heuristic. The Matching heuristic computes the weight for all edges, constructs a maximum weighted matching to active the edged matched for token distribution and recalculates the token distribution. This procedure will continue until every node has sent the same token to every other node. The critical step is how to set the edge weights. The weight is the sum of the contributions from all tokens.

In each round of broadcasting or gossiping, weights are assigned to all edges in the graph. Then, a *maximum weighted matching* is constructed in the graph. The matched edges are active in this round. When an edge is active in a round, it means that a message is passed through this edge in this round.

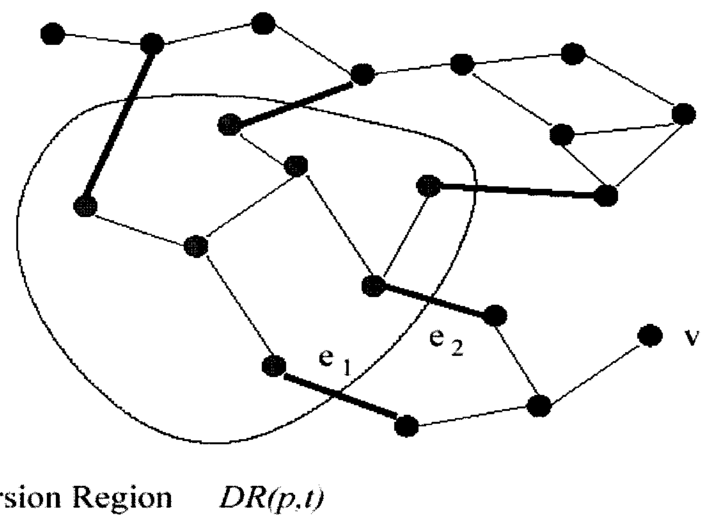


Figure 1: Definitions in Round_Heuristic

As stated in [1], setting the weight is the most important part for most other broadcasting and gossiping algorithms including Round_Heuristic. There were two approaches used in the Round_Heuristic. One is the *Potential Approach*, where the weight of an edge *(v, w)* is set to equal its *potential*, defined as the number of messages known by either *v* or *w*, but not by both of them. Although this approach is simple, requires little storage and is very fast, it does not perform as well as the Breadth-First-Search (BFS) approach.

Several definitions are needed in order to introduce the BFS approach. These definitionswill be used to introduce the new algorithm in Chapter 3. The dispersion region *DR (p,t)* of a message *p* refers to the set of nodes that know *p* at the beginning of round *t* (this is a connected sub graph). For a node *v*, $dist_v(p,t)$ denotes the shortest distance in the graph from *v* to a node *w* in *DR (p,t)*.

The set of border-crossing edges $bce(p, t) = \{(v, w) \in E \mid v \in DR(p, t) \text{ and } w \notin DR(p, t)\}$. For a node v not in $DR(p, t)$, $bce_v(p, t)$ consists of all edges in $bce(p, t)$ that lies on the shortest path from $DR(p, t)$ to v. See Figure 1. [1]

The question that decides the direction of improving how to calculate the contribution of edge e in $bce(p,t)$: (How useful is e for the rapid dissemination of p?) is a very important one. Message p should be routed on the shortest paths from $DR(p,t)$ to all other nodes. It is better for e to lie on more of these shortest paths. The larger $dist_v(p,t)$ is, the more priority should be given to forwarding p towards v. Based on this idea, two parameters, *Dist_Exp* and *Num_Exp* that could be used to highlight the importance of $dist_v(p,t)$ and the number of edges in $bce(p,t)$ were introduced in order to calculate the weight. These parameters play a very important role in the following formula to calculate weight. [1]

$$weight(v, p, t) = \frac{dist_v(p,t)^{Dist\_Exp}}{|bce_v(p,t)|^{Num\_Exp}}$$

In [1], a modified breadth first search algorithm is used to calculate the weight. Because it is a breadth first search algorithm, the nodes are considered in an order of increasing $dist_v(p, t)$. For nodes v with $dist_v(p, t) = 1$, $bce(p, t)$ consists of all adjacent edges that connect v to a node in $DR(p,t)$.

The set $bce_v(p, t)$ is the union of a maximum of n sets with a maximum of m elements each for a node v,. It takes $O(nm)$ time to finish this calculation. The $bce_v(p, t)$ is computed for all uninformed nodes. Thus, calculating the weights takes $O(n^2m)$ in total. Calculating a maximum weighted matching is viewed as an external routine in [1].

Many of the testing results in several commonly used topologies are equal to the real broadcast time. The paper [1] presents the test results in the CCC_k graph, the *Shuffle Exchange* graph, the *Butterly* graph and the *De Bruijn* graphs.

The choice of parameters has a great impact on the quality of the heuristic.. *Dist_Exp* is the parameter of particular importance. It determines the influence of the distance between nodes and dispersion regions. Values in the range from 0.25 to 60 are used. The precise choice of two topologies is mentioned in [1]: for the mesh, *Dist_Exp* = 4, and for *butterfly* graph, *Dist_Exp*=2.

In [7], a new algorithm was introduced. This thesis has replaced the way of finding destination nodes for dissemination by using matching, and the main idea is used in gossiping also. In [7], a heuristic for broadcasting in arbitrary networks is given. In this heuristic, at the beginning of a round, all the nodes in a graph are separated into two parts: a dark region which is composed of all uninformed nodes and a bright region which is composed of all informed nodes. The main purpose of this exercise is to find all the matched pairs among the nodes located at a bright border called bb(t) and the nodes located in the dark border called db(t) to spread the unique message from the former to the latter. The weight is assigned to the nodes db(t) only by calculating the estimated distance from any node in a dark region using the BFS method and identifying the descendant relationship between nodes using distance values and neighbor ordered by their estimated distance. After this has been accomplished, the maximum-number-weight matching between bb(t) and db(t) is used to decide which node in db(t) should be informed by checking the maximum weight assigned to them. This procedure continues until bb(t+1) becomes empty.

1.3 Thesis Outline

The structure of the remainder of this thesis is following. The related background knowledge is introduced in Chapter 2. This includes commonly used topologies, NP-Completeness of the broadcast and gossip problem, and the matching problem. The new algorithm for both broadcasting and gossiping is formally presented in Chapter 3. Chapter 4 introduces the implementation of the algorithm. The experimental results concerning the heuristic are introduced in Chapter 5. The performance of the new algorithm is compared with those from previously known heuristics in Chapter 5 as well. Finally, this thesis is concluded in Chapter 6.

2. Background

2.1 Terminologies and Symbols

In the following sections, I will use a graph to model a network. For the purpose of message dissemination, it is natural to assume that the network is represented by a connected graph. A graph can be represented as $G = (V, E)$, where V is the set of vertices and E is the set of edges. If an edge $e \in E$ and has two end vertices $u \in V$ and $v \in V$, we can say that $e = (u, v)$, and these two vertices u and v are *adjacent,* or vertex u (or v) is a *neighbor* of the other vertex. The *degree* of a vertex is the number of its neighbors. The degree of a graph G is the maximum degree among all vertices in this graph. We use Δ to denote the degree of a graph. The distance between a vertex u and a vertex v is the length of the shortest path between vertex u and vertex v, denoted by $dist(u,v)$. The *diameter* of a graph G is the maximal distance between any pair of vertices in the graph G. We denote it with D.

In order to measure the broadcast time and gossiping time, we use *round* that is the set of parallel calls in the same time unit. The number of rounds is the time used for broadcasting and gossiping. A broadcast scheme or broadcasting schedule is a series of round that perform broadcast. While a gossiping scheme or gossiping schedule is a series of rounds that perform gossiping. For a given graph G and u as any one of its nodes, the broadcast time of u is the minimum broadcast time of graph G originated at vertex u, this can be denoted by $b(u, G)$ or $b(u)$. The broadcast time of graph G is defined as follows: $b(G)= max \{b(u,G) \mid u \in V \}$. Similarly, $g(G)$ stands for the gossiping time of graph G.

2.2 Commonly Used Topologies

Both algorithms of Broadcasting and Gossiping will be tested based on some topologies or models. As presented in this section, there are many commonly used network topologies. *NS-2* models that simulate the real-life Internet structure and pure random networks are introduced in Chapter 5.

Before introducing all of the commonly used network topologies, here are some preliminaries. For any graph that has n vertices, we have $\lceil \log_2 n \rceil \leq b(G) \leq n - 1$. Since any vertex holding a message can only send it to one of its adjacent vertices, the number of informed vertices could at most be doubled in each round. Thus, at least $\lceil \log_2 n \rceil$ rounds are needed for finishing broadcasting. On the other hand, in broadcasting, at least one vertex must be informed in each round. A situation in which no new vertex is informed means that the broadcasting has been finished. Therefore, broadcasting takes at most $n-1$ rounds. For any graph of maximum degree Δ, and diameter D, where $D \leq b(G) \leq \Delta D$, it is possible to broadcast in any shortest path spanning tree of G of height D and maximum degree Δ in at most ΔD rounds [5].

Lemma 1. *In any graph of diameter D, if three different vertices u, v_1 and v_2, with both v_1 and v_2 at a distance D from u, exist, then $b(G) \geq D+1$.*

In [11], some bounds of the gossiping problem are given as follows: since a gossiping scheme can be used as a 1-broadcast scheme, we have $b(G) \leq g(G)$. Again, gossiping can be performed in two phases. One must first collect all of the messages in one vertex through the use of a gather operation, and then one must broadcast the full information to all of the vertices, so we have $g(G) \leq 2b(G) - 1$. Therefore we have:

$$\lceil \text{[illegible]} \rceil \le b(G) \le g(G) \le 2b(G)-1 \le 2n - 3$$

In the rest of this section I will present the commonly used network topologies, and also indicate their broadcast and gossiping time. Most of the results presented in this section were surveyed in [5], [11], [12] and [13]. All of the figures in this section were presented in [5], [11] and [13].

The Complete Graph: In a complete graph, any vertex is linked to all other vertices. K_n denotes a complete graph [11]. Figure 2 (a) present an example with node number $n = 4$ and Figure 2 (b) shows another example with $n = 6$. The degree of all vertices is $n - 1$, while the diameter is 1 and the number of edges is $n(n - 1) / 2$ [11]. In [11], we can see that $b(K_n) = \lceil \log_2 n \rceil$, because any informed vertex can send the message to any of its k uninformed neighbors in each round. The following results are shown in [14]: $g(K_n) = \lceil \log_2 n \rceil$, if n is even, and $g(K_n) = \lceil \log_2 n \rceil + 1$, if n is odd.

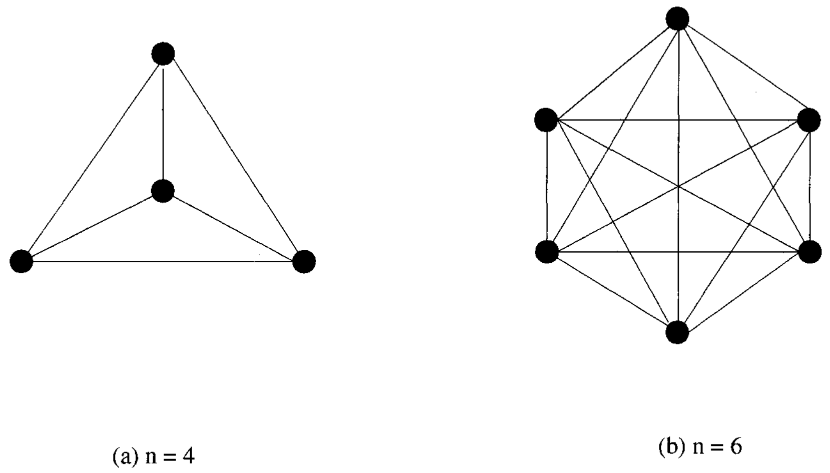

(a) n = 4 (b) n = 6

Figure 2: Complete Graphs with n = 4 and n = 6

The Path: In the graph of Path with the length of n, the vertices are all labeled by integers from 1 to n, and all edges connect a vertex labeled by integer $i(1 \le i \le n)$ to the vertex labeled by i + 1, we denote the path by P_n [5]. With n vertices, P_n has a diameter of n - 1 and a maximum degree of 2. Figure 3 gives an example of *P6* [5]. $b(u, P_n)$ is at the maximum, when the originator u is at either end of P_n. In such a case $b(u, P_n) = b(P_n) = n-1$.

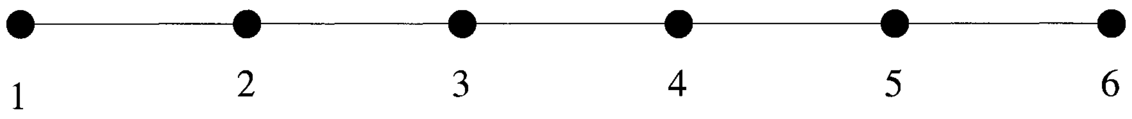
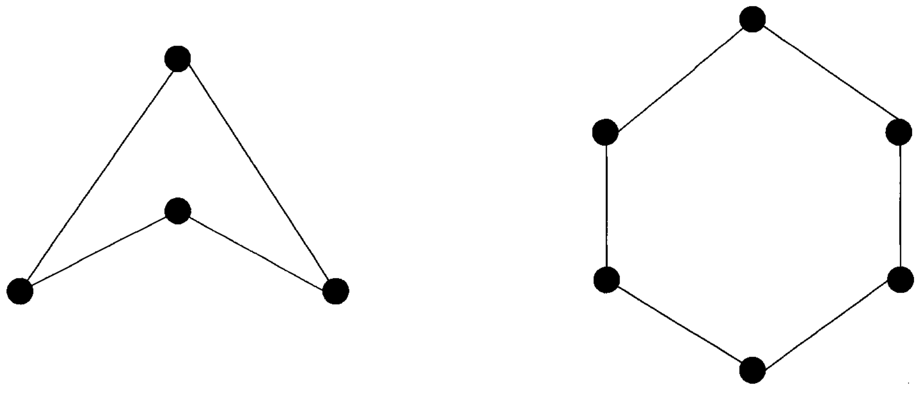


Figure 3: The Path with n=6

Cycle Graph: In a graph of cycle with n vertices, each vertex is linked to only two neighbors, it can be created from path P_n connecting the two end nodes. A Cycle graph with n vertices can be denoted by C_n. The degree of Cycle graph C_n is two. Figure 4 (a) shows the example of C_4, and Figure 4 (b) shows another example C_6 [11]. In the *cycle* graph C_n, the broadcast time is $\left\lceil \frac{n}{2} \right\rceil$ [11]. The result for the gossiping problem in a *cycle* graph is:

$g(C_n) = n/2 = D$, if n is even.

$g(C_n) = (n-1)/2 + 2 = D + 2$, if n is odd.



(a) n= 4 (b) n = 6

Figure 4: The Ring Graph with n=4 and n=6

The d-grid graph: The d-grid graph is a graph with d path graph parallelly connected. GD_N= P_{n_1} ... P_{n_d} for $1 \leq i \leq d$, where P_{n_i} is a path on n_i vertices [11]. Figure 5 gives an example of *2-grid.* We have the bound from [15], $b(P_{n_1}$ ... $P_{n_d}) = \sum_{i=1}^{d} n_i - d$ = D.

We use $G_{m,n}$ to denote a 2-grid with m columns and n rows. When the originator is on a corner of the graph, broadcast will have the worst case and broadcast time is $m + n - 2$ [15]. In [16], we can see the following results on gossiping time:

$b(P_{n_1}$ ... $P_{n_d})$ = D + 1, if d is odd;

$b(P_{n_1}$ ... $P_{n_d})$ = D, if d is even.

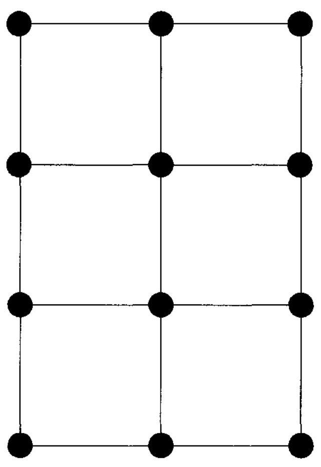


Figure 5: The 2-grid Graph of 12 Nodes

The d-Torus graph: A d-Totus graph is a d-grid graph with both ends of path of row and column connected as circle. If we use T_d to denote d-Torus graph, similarly with d-grid graph, we have T_d = C_{p_1} ... C_{p_d} for $1 \leq i \leq d$, where C_{p_i} is a cycle on p_i vertices [11]. Figure 6 presents a 2-Torus graph with 12 nodes.

The optimal broadcast time of the 2-Torus graph is $\left\lceil \frac{m}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil$, when m or n is even; and it is $\left\lceil \frac{m}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil$-1 when both m and n are odd [15]. The only known result of broadcast on multidimensional torus is : $D \leq b(C_{p_1} \;\ldots\; C_{p_d}) \leq D + \max(0, m - 1)$, where m is the number of odd dimensions in $C_{p_1} \;\ldots\; C_{p_d}$ [11]. For gossiping, $D \leq g(T_d) \leq D + 2d$ [17].

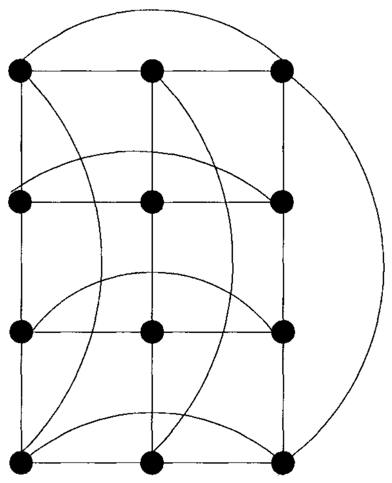
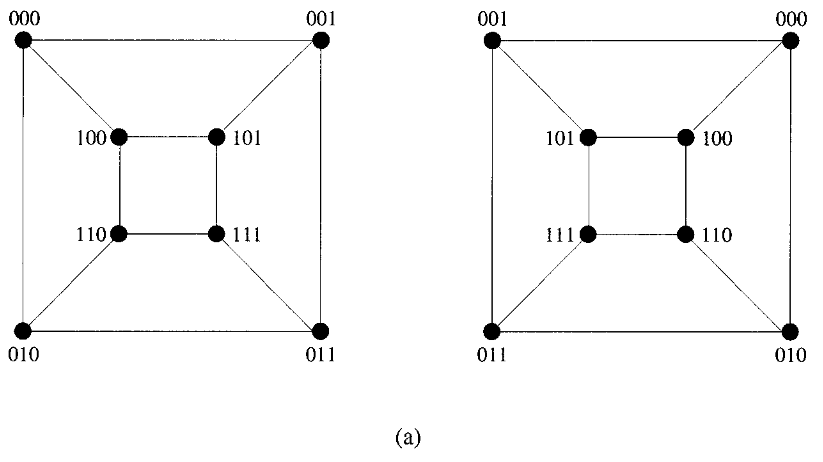


Figure 6: A 2-Tori Graph with 12 nodes

Hypercube Graph: A hypercube is a cube with d dimensions ($d \geq 1$), and each node in a d dimensional cube is directly connected to d other nodes. We can use H_d to denote hypercube graph. A d-dimensional hypercube has $n = 2^d$ vertices and $d2^{d-1}$ edges. It is beneficial to number each vertex with a corresponding d-bit binary string, because two vertices are linked with an edge if and only if their binary strings differ by precisely one bit [11]. As a consequence, each vertex is adjacent to d other vertices, one for each bit position [11]. The diameter of H_d is d.

From [13], we know that any d-dimensional hypercube could be derived from two (d – 1)-dimensional hypercube graphs. This can be seen from Figure 7 that gives an example of how a 4-dimensional hypercube Figure 7 *(b)* is constructed from two 3-dimensional hypercube (Figure 7 (*a*)).

Broadcasting in H_d can be done in *d* rounds as follows: at step *i*, each informed vertex sends the message in dimension *i* $(1 \leq i \leq d)$ [11]. Gossiping in H_d can also be done in *d* rounds [11].



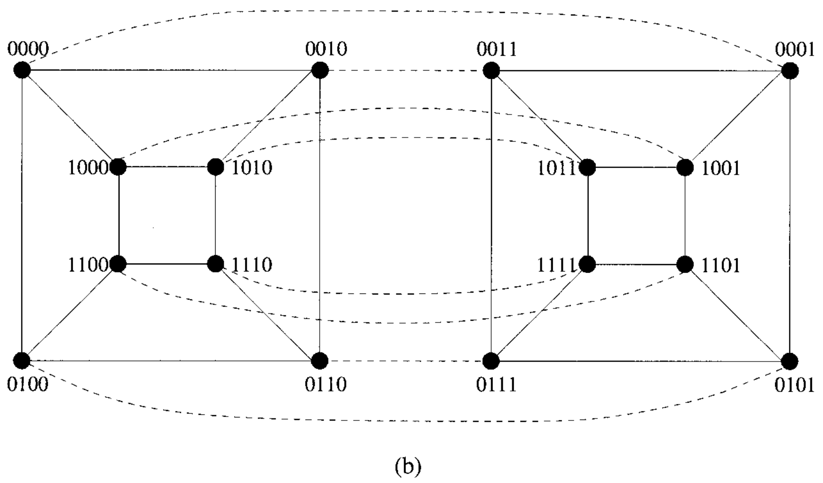


Figure 7: The Hypercube Graphs of 3 and 4 Dimensions

Cube Connected Cycles: A *d*-dimensional cube connected cycle can be constructed from a *d*-dimensional hypercube by replacing each vertex of the hypercube with a cycle of *d* vertices [11]. We use CCC_d to denote a d-dimensional cube connected cycle. The diameter of the CCC_d is

$2d + \left\lfloor \frac{d}{2} \right\rfloor - 2$, when d > 3 [4]. For broadcasting time, $D \leq b(CCC_d) \leq D + 1$ when d is even. But when d = 4, it is an exception, since $b(CCC_d) = D + 1$. And $D+1 \leq b(CCC_d) \leq D + 2$, when d is odd

In [6], we can see the result for gossiping in the CCC_d graph: $D \leq g(CCC_d) \leq 5d/2 = D + 2$, if d is even ; $D + 1 \leq g(CCC_d) \leq \left\lceil \frac{5d}{2} \right\rceil + 2 = D + 5$, if d is odd. Figure 8 shows an example of the CCC graph of 3 dimensions.

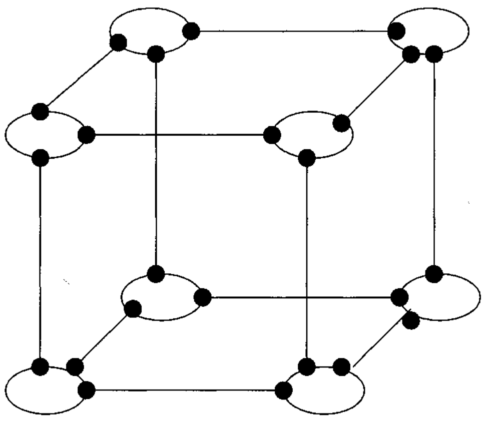


Figure 8: The CCC Graph with 3 Dimensions

The butterfly graph: The n-dimensional butterfly graph is a graph with vertices that are pairs (w, i), where w is a binary string of length n and i is an integer in the range 0 to n and with edges from vertex (w, i) to $(w^i, i + 1)$, if and only if w^i is identical to w in all bits with the possible exception of the (i + 1) bit counted from the left. We use BF_d to denote it. The d-dimensional butterfly BF_d has $d2^d$ vertices, Figure 9 showsan example of BF_3 [11]. According to the definition above we know that each vertex is labeled with a pair of numbers (l, x). l represents the level $(0 \leq l \leq d - 1)$, and $x = x_0 \dots x_{d-1}$ is a d-bits binary string called the *position-within-level* [11]. BF_d has degree four and diameter $\lfloor 3d/2 \rfloor$ [11].

From [18], we know that $1.7417 \le b(BF_d) \le 2d - 1$. For gossiping, we can conclude from the fact that CCC_d is a sub graph of BF_d, that $D \le g(BF_d) \le \lceil 5D/3 \rceil$ if d is even; $D \le g(BF_d) \le \lceil 5(D+10)/3 \rceil$, if d is odd.

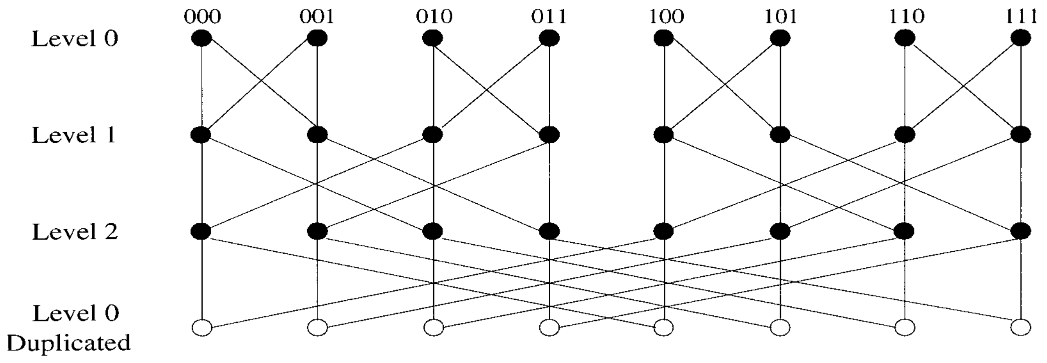


Figure 9: The Butterfly Graph with d = 3

The de Bruijn Graph: The de Bruijn digraph is a graph that has nodes that are sequences of symbols from the binary alphabet. Its edges indicate the sequence that might overlap. We use UB(*d*, D) to denote it. The de Bruijn digraph *B(d, D)* has in-degree and out-degree *d* and diameter *D*, and has $N = d^D$ vertices. The vertices are denoted by the words of length *D* on an alphabet of *d* letters [11]. There is a direct edge from each vertex $(x_0x_1...x_{D-1})$ to $(x_0x_1...x_{D-1}\gamma)$, where γ could be any letter in the alphabet [11]. Actually, we can get the de Bruijn graph *UB(d, D)* from *B(d, D)* by removing the edge orientation in *B(d,D)* [11]. Figure 10 is an example of *UB(2, D)*.

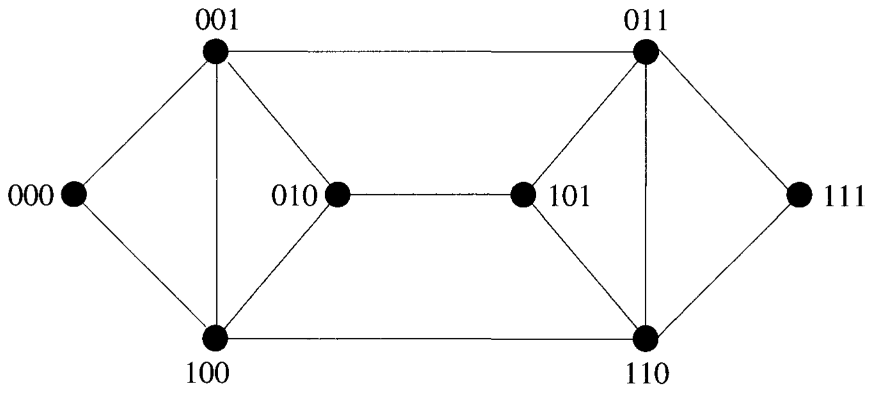


Figure 10: The de Bruijn Graph with D = 3, d = 2

The Shuffle-Exchange Graph: The shuffle-exchange graph consists of nodes that are all binary string of length *ma* and its edges connect each string αa, where α is a binary string of length m-1 and a is in {0,1}, with the string αa and with the string aα The *d*-dimensional shuffle-exchange graph has $n=2^d$ vertices. Figure 11 gives the example of SE_3 [11]. From this example we can see that each vertex corresponds to a unique *d*-bit binary number, and for two vertices *u* and *v,* they are linked by an edge, if either *u* and *v* differ in precisely the last bit, or *u* is a left or right cyclic shift of *v* [13].

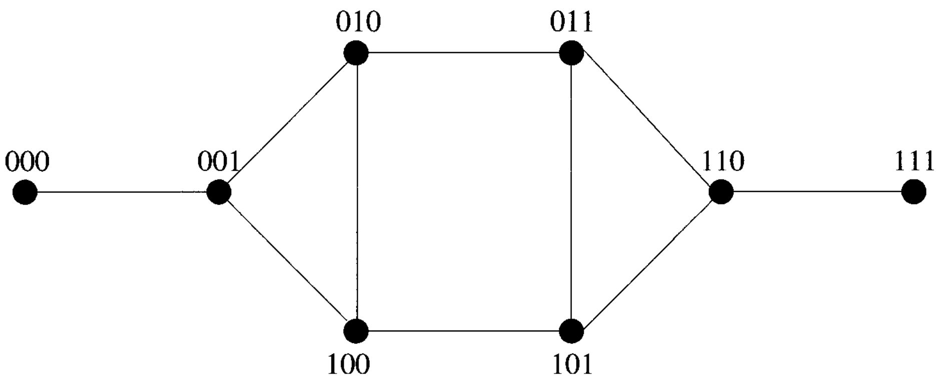


Figure 11: The Shuffle-Exchange Graph with d = 3

2.3 NP-Completeness

2.3.1 Definition

NP(Non-deterministic Polynomial time) is a set of decision problems that are solvable in polynomial time on a non-deterministic Turing machine [22]. Equivalently, it is the set of problems that can be "verified" by a deterministic Turing machine in polynomial time.

NP-complete problems are the most difficult problems in NP, in that they are the ones least likely to be in Polynomial time. This is due to the fact that if one were able to solve one NP-complete problem quickly, then one could use that algorithm to solve all NP problems quickly. The complexity class consists of all NP-complete problems.

In [22], A decision problem C is NP-complete if

1. it is in NP and
2. it is NP-hard, i.e. every other problem in NP is reducible to it

"Reducible" here means that for every problem *L*, there is a polynomial-time many-one reduction, a deterministic algorithm which transforms instances $l \in L$ into instances $c \in C$, such that the answer to *c* is YES if and only if the answer to *l* is YES. To prove that a NP problem *A* is in fact a NP-complete problem one must demonstrate that an already known NP-complete problem reduces to *A*. Thousands of other problems have been shown to be NP-complete by reductions from other problems previously shown to be NP-complete; many of these problems are collected in Garey and Johnson's 1979 book *Computers and Intractability: A Guide to NP-Completeness*.[22]

2.3.2 Broadcasting and Gossiping in arbitrary network is NP-complete

From [19], we know that the broadcast problem on an arbitrary graph is NP-complete. This can be proved because the NP-complete problem, the three-dimension matching (3DM) problem, reduces to the broadcast problem on an arbitrary graph.

In [20], we also learn that the gossiping problem on an arbitrary graph is also NP-complete. This can be proved under either full-duplex or half-duplex pair wise model of communications. The reduction uses the Minimum Broadcast Time Problem as described by Garey and Johnson [23] .

After giving the result for the half-duplex model, it can be modified to yield the same result for the full-duplex model. In addition the reversibility can be realized as well.

Through the observation that any single source broadcast algorithm becomes another algorithm when the sequence of communications is reversed, one can conclude that by running a broadcast algorithm backwards, one can collect tokens from all of the nodes. Given an instance of the Minimum Broadcast Time Problem involving a graph $G = (V, E)$ and a specified node v, one constructs another graph, $G` = (V`, E`)$ that depends on the choice of v with the property that the Minimum Broadcast Time Problem for $G$ and v has a solution using $k$ or fewer steps if and only if the gossiping problem for G` has a solution using $2k + 2$ or fewer steps. [20] demonstrates proof that a solution to the Minimum Broadcast Time Problem for G and v in k steps exists if and only if there is a solution to the gossiping problem for $G`$ in $2k+2$ steps. Since the construction of $G`$ uses only polynomial resources, the minimum-time gossiping problem is NP-complete.

Since the broadcasting process is always achieved through the use of a single source- node, the full duplex capability is not a possibility. In other words, there can be no case where the replacement of a half-duplex transmission by a full duplex one would improve any of the above algorithms. So, the entire above proof can be repeated for the full duplex model, with $2k + 1$ being used in place of $2k + 2$ because the exchange can be done in one step.

2.4 Matching

2.4.1 General Matching Problem

A matching in a graph is a set of edges without common vertices [24]. Given a graph *G=(V, E)* a matching M in G is a set of non-adjacent edges. The maximum matching makes the number of matched pairs as great as possible. There may be many maximum matches. A perfect matching is a matching that covers all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching.

Finding a maximum matching on a graph is a very useful procedure in that it can facilitate operation research and integer programming. Let us take the example of a university instructor who wishes to divide his entire class into teams of two. We will assume that certain students are prohibited from pairing up with each other because they are considered incompatible. In this case, the finding a maximum matching concept can be employed as a guide in order to choose the greatest possible number of teams of compatible students.

Augmenting path is a very important concept to understand the following implementation. An augmenting path with respect to a matching *M* is a simple path abc...yz of nodes connected by edges in E such that nodes *a* and *z* are exposed, and edges bc, de, ... xy are in M and the other edges of the path are not in *M*.

2.4.2 Matching Used in this Thesis

In this thesis, I will employ the implementation of Edmond's algorithm for maximum matching by Harold N. Gabow [25]. This implementation is based on a system of labels that encodes the structure of alternative paths. It constructs a maximum matching on a graph. It starts with empty matching, and continues to improve it with augmenting paths until it stops. Using an array called MATE to store the matching and array called LABEL for the every outer vertex, one can write the algorithm as follows.

1. Read the graph into adjacency lists, number the vertices 1 to n and the edges $n + 1$ and $n + 2W$. Set all vertices as non router and unmatched. This step is for initialization.
2. Find unmatched vertex: Increase u by 1. If $u > V$, stop. U is unmatched, so assign a start label and begin a new search.
3. Choose an edge: choose an edge xy, where x is an outer vertex. If no such edge exists go to step 8. Edge xy can be chosen in an arbitrary order. A possible choosing method is "breadth-first".
4. Augment the matching: if y is unmatched and y is not u, set x as mate to y, rematch x and y, then go to step 8.
5. Assign edge labels: if y is outer, call label function, then go to step 3.
6. Assign a vertex label: set v as MATE(y), go to step 3.
7. Get next edge: y is nonouter and MATE(y) is outer, so add nothing. Go to step 3.
8. Sstop the search. Set LABEL(0) as -1, now all vertices are nonouter for the next search [25].

In order to view the algorithm in detail, please refer to [25], The computation time of this implementation is proportional to n^3, where n is the number of vertices. This algorithm can be generalized to find maximum matching on weighted graphs. In a weighted graph, each edge has a weight that is a real number. The main problem is to find a matching with maximum weight.

3. Algorithm

3.1 Broadcasting

3.1.1 Algorithm Description

For the purpose of simplifying the description, we will first define several terms. There are three kinds of definitions that are important for this algorithm: region, descent graph and children. Following the definitions, we will present both the algorithm and an example.

3.1.1.1 Definitions

Definition 1. *For a given graph G at round t, there are two regions according to the situation of the message distribution, the Dark Region and the Bright Region. The Dark Region, denoted by DR(t), is a subset of nodes in G that is composed of all uninformed nodes at the beginning of round t. Those nodes in DR(t) that have informed neighbors, compose the dark border, denoted by db(t). The bright border bb(t) is composed of those informed nodes that have uninformed neighbors. The edges that cross between Dark Region and Bright Region are called cross board edges that are denoted by cbe(t).*

Figure 12 illustrates how these concepts are defined. The dark region *DR(t)* is represented by the shadowed area. The nodes in *DR(t)* with the black backgrounds belong to *db(t)*, for example *k, i, j, h, m* and *l,* and the nodes not in *DR(t)* with shadowed backgrounds belong to *bb(t)*. The edges *(f, k), (f, i), (f, j), (d, b)* and *(g, m)* belong to *cbe(t)*.

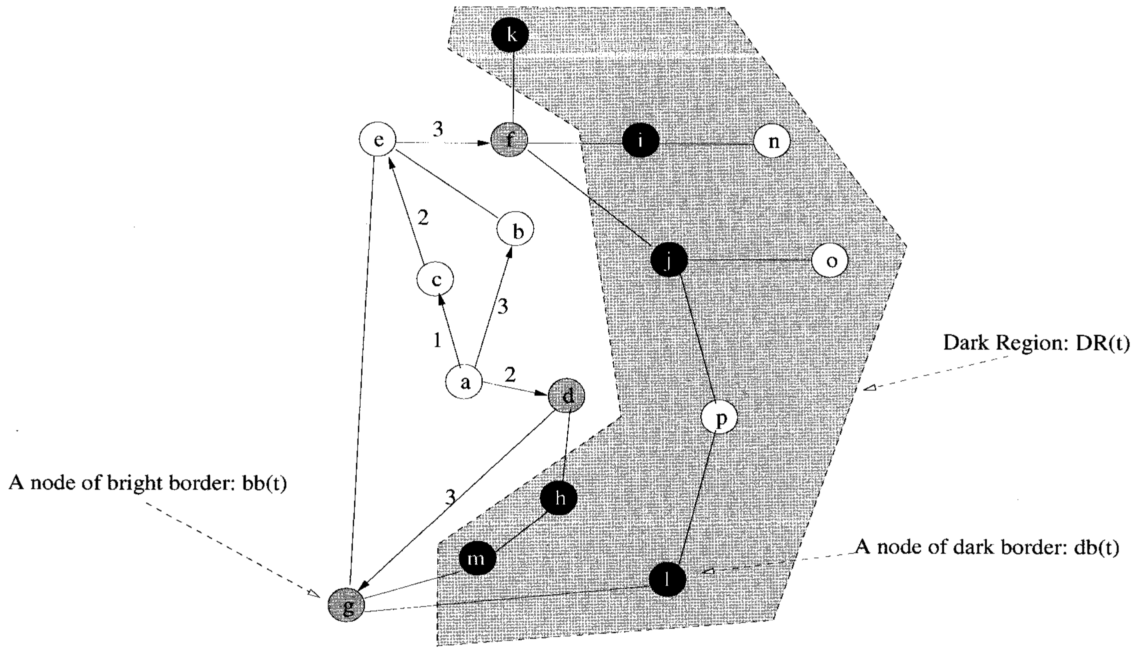


Figure 12 Definitions of Graph Parts

Definition 2. *For a graph and an uninformed node v at round t, There is a shortest distance from node v to a node in bb(t). The shortest distance is denoted as D(v,t).*

The shortest distance can be used to define a child as follows:

Definition 3 *child, parent and descendants: Given an uninformed vertex u and its uninformed neighbor v, if D(u, t) = D(v, t) + 1, we say u is a child of v, and v is the parent of u.* The node *u, its children and its children's children are all called v's descendants.*

Now it is possible to define the descendant graph as follows:

Definition 4. *For a graph and an uninformed node v at round t, one can find a descendant graph for v. This descendant graph consists of node v and all its descendants. This is named as the descendant graph of v, which is denoted by DG(V, E, v), or rather DG(v).*

Definition 5. *Estimated time: in order to estimate the broadcast time of $DG(v)$ in round t, we use $EB(v, t)$. $EB(v, t)$ is defined recursively as follows:*

1. *$EB(v, t) = 0$, if node v has no children.*
2. *If v has k children, $c_1, c_2, \ldots, c_k$, and all these k children are listed in order of $EB(c_i, t) \geq EB(c_{i+1},t)$, then $EB(v, t) = \max\{EB(c_i, t) + i\}$, for $1 \leq i \leq k$.*

The following is the algorithm to calculate $EB(v, t)$, given EB of all children of node v.

Algorithm of Calculating $EB(v, t)$.

1. Find $\max\{EB(c_i,t)\}$, and denote it by MAX.
2. Create k buckets, and number them from 0 to $k-1$.
3. Consider any child c, if $\text{MAX} - i \geq EB(c,t) \geq \text{MAX} - i - 1$, put c into the *ith* bucket. Here, we only record the minimum value and the number of elements. $SUM(i)$ denotes the number of elements in the first *ith* buckets and $MIN(i)$ denotes the minimum value in the *ith* bucket.
4. Get $EB(v, t)= \max\{EB(c_i, t) + i\}$.

For step 4 above, we have a lemma.

Lemma 1. $EB(v, t) = \max\{SUM(i) + MIN(i)\}$, *for* $0 \leq i < k$.

Proof: If a node v has k children, $c_1, c_2, \ldots, c_k$. And these children of v are ordered so that $EB(c_i, t) \geq EB(c_{i+1},t)$, then according to definition 5, we have $EB(v, t) = \max\{EB(c_i, t) + i\}$, *for* $1 \leq i \leq k$. $EB(c_i, t) + i$ is *order-weight* of c_i. Because $\text{MAX} - i \geq EB(c,t) \geq \text{MAX} - i - 1$ for any child c in the ith bucket, the maximum difference among EB of the children in this bucket is less than 1. Therefore, in the ith bucket, the child with the minimum EB has the maximum *order-weight*, which is equal to *SUM(i)* + *MIN(i)*. Thus, *max{SUM(i)* + *MIN(i)}, for* $0 \leq i < k$ is the maximum *order-weight* of all the children, which is *EB(v, t)*.

3.1.1.2 Algorithm

According to the above definitions, the algorithm can be defined as follows:

1. Initialize the graph to make all control information ready, Set up bright region BR(t) to contain one node only as the originator, t = 0. This node is the unique node in bb(t) also.
2. Calculate *EB(v, t)* for all nodes in DR*(t)*, and assign this as weight value to the corresponding node.
3. Assign weight to cbe(t) according to the weight value for adjacent nodes in bb(t)
4. Get the maximum weight matching between *bb(t)* and *db(t)*.
5. Disseminate the message to inform the matched node in db(t) and be moved them to bb(t), increase t by 1.
6. Repeat 2 to 5 until db*(t+1)* becomes empty.

Then the *t* would be the number of total rounds used for the broadcasting.

3.1.1.3 An Example

Here is an example to illustrate the process of broadcasting with this algorithm.

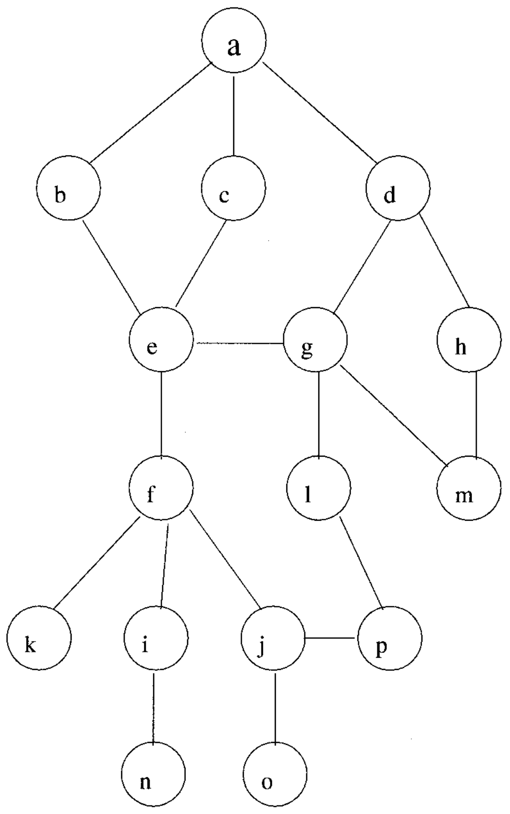
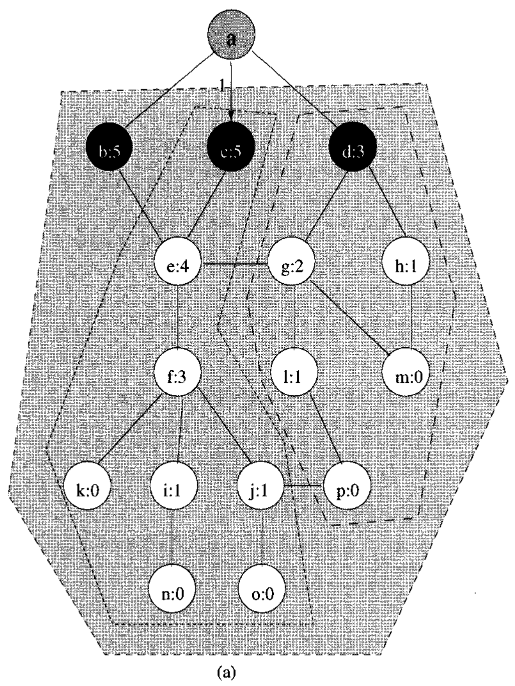


Figure 13: The Original Example Graph

Figure 13 shows the graph I will employ to do the broadcast. Vertex *a* is the originator in that *a* possesses the unique message that will be disseminated to all of the other nodes.



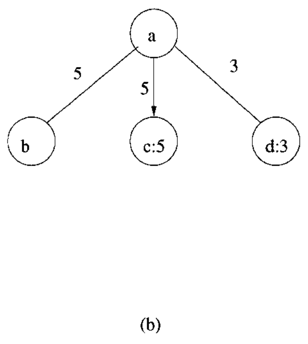


Figure 14: The Example Graph at First Round

In Figure 14 one can see how the first round is defined: only the originator *a* is informed and it is the only element in *bb(0)*. All of the other nodes have their weights as shown in the figure. Node *a* has three neighbors, *b, c* and *d* which consists of db(t). Node *b* and node *c* have the same weight 5 while node *d* has weight 3. The area encompassed by the shorter dashed lines, which is located on the left, is *DG(c)*, and the area defined by the longer dashed lines , which is located on the right, is *DG(d)*. The weight of node *c* is 5. This means that broadcasting in *DG(c)* that originated at node *c* takes 5 rounds. After assigning weight to cbe(0) which includes edge(a,b), edge(a,c) and edge (a,d), one can get a sub graph as shown in Figure 14(b). As a result of the algorithm of maximum weight matching, in the first round we get *a* and *c* as the matched pair. .

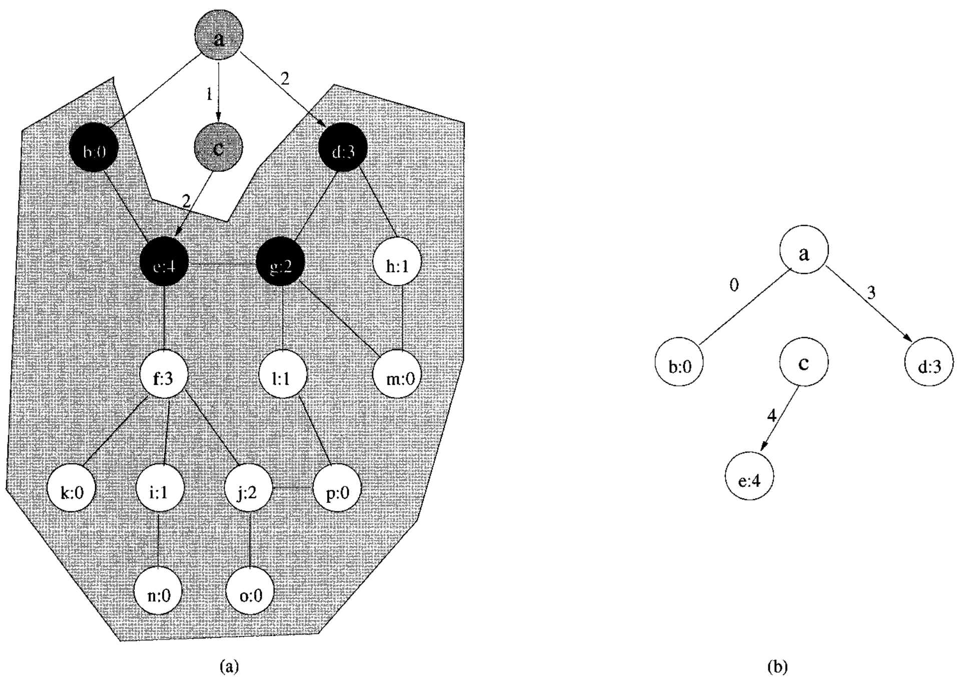


Figure 15: The Example Graph at the Second Round

One can see the second round from Figure 15. As shown in Figure 15(a), nodes *a* and *c* are informed nodes. Node *a* has two uninformed neighbors node *b* and node *d*. The weight of node *b* is 0 because *DG(b)* has only one node *b*. After assigning weight to corresponding cbe(1), one can get the graph shown as Figure 15(b). Based on this graph, one can obtain the match (a, d) and (c, e). This is the second round.

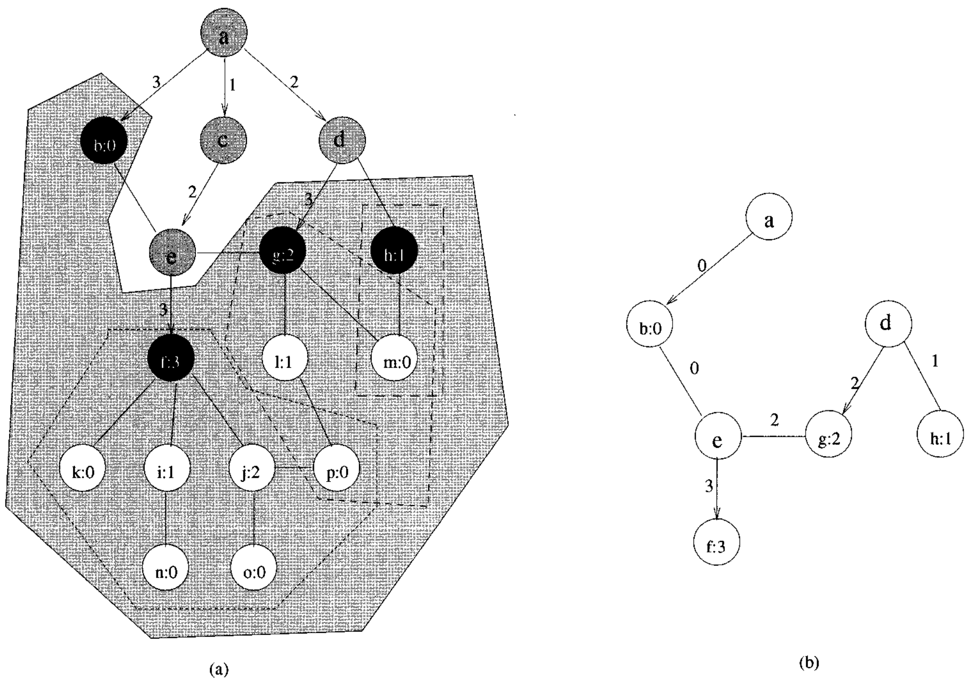


Figure 16: The Example Graph at the Third Round

For the third step (as shown in Figure 16) one is capable of observing that several descendant trees share nodes. The areas representing *DG(f), DG(g)* and *DG(h)* overlap. One can see this in Figure 16 (a), in similar steps to those presented in 1 and 2, one can get the match from the graph as shown in Figure 16(b).

The broadcasting continues in a similar fashion until all nodes are informed or have obtained the message originated from node *a*. Finally, one can get the schedule created by the new heuristic as shown in Figure 17. The numbers on the edges indicate the round in which the edge is used during broadcasting, and the arrows demonstrate the direction in which the message has been sent. One can see that in this particular graph the algorithm provides the optimal broadcast schedule.

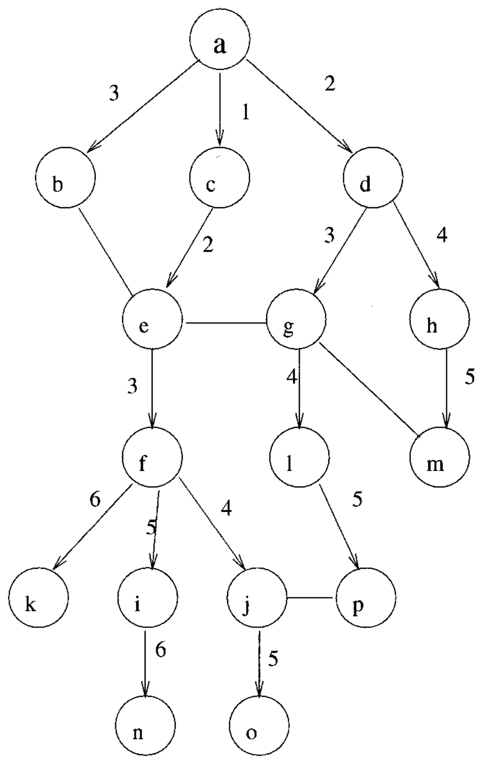


Figure 17: The Final Schedule in the Example Graph

3.1.2 Refinement

3.1.2.1 Motivation

One of the critical parts in regards to the successful functioning of this algorithm is to calculate *EB*(*v*, *t*) (which has played an important role for the match) and then to decide which one should be the next informed node accordingly. One can notice that a child may have more than one parent, and thus the weight of such a child could potentially be counted several times when one computes the weight of the parents. If this occurs, the effect of this child during the process of broadcasting is overestimated. In order to make the results to reflect the above theory, one should depend on the refinement solution as given in the following section.

3.1.2.2 Refinement Description

The refinement and the original algorithm are subject to the same procedure for the calculation of EB. To simplify, one must employ the same logic that I have referred to above: provide a definition and then describe the algorithm.

Definition 7. *Refined Estimated time: in order to make the estimation for the broadcast time of DG(v) in round t more accurately, we use REB(v, t). REB(v, t) is defined recursively as follows:*

1. *REB(v, t) = 1, if node v has no children.*
2. *If v has k children, c_1, c_2, ..., c_k, and all these k children are listed in order of REB(c_i, t) ≥ REB(c_{i+1},t), then*

$$REB(v,t) = \max\left\{\frac{EB(\mathrm{c_i},t)p}{n} + i\right\}.$$

n is the number of parents of c_i, p is a parameter.

With this new definition the refined algorithm can be defined as the follows:

1. Initialize the graph to make all control information ready. Set up the bright region BR(t) to contain one node only as the originator, *t* = *0*. This node is also the unique node in *bb(t)* also.
2. Calculate *REB(v, t)* for any node in DR*(t)*, and assign this as weight value to the corresponding node.
3. Assign weight to cbe(t) according to the weight value for adjacent nodes in bb(t)
4. Get the maximum weight matching between *bb(t)* and *db(t)*.
5. Disseminate message to inform the matched nodes in db(t) and move it to bb(t), t = t + 1.
6. Repeat 2 to 5 until db*(t+1)* become empty.

From the description above, we can see that the difference is only at step 2, and the improvement depends on the way p is selected and on the topologies. For some topologies, it works well, for example *Cube-Connected*, *Cycle*, the *ButterFly* and the *ShuffleExchange.* With other topologies, it does not work very well. The parameter p also has some impact. We can simply assign $p = 1$ to all kinds of topologies. In such a situation, the performance is improved in some topologies, but, the improvement is not significant. To make the results better, we have to adjust p accordingly to different topologies.

3.1.2.3 An Example that Makes a Difference

Let us use the following graph shown in Figure 18 as the topology. I will illustrate the difference that the refinement algorithm will make. In this graph, node *a* is the originator. The difference will be demonstrated from round 2. The first round is (a, b) no matter which algorithm one follows. So we start from second round, that is, after node *a* and node *b* are informed.

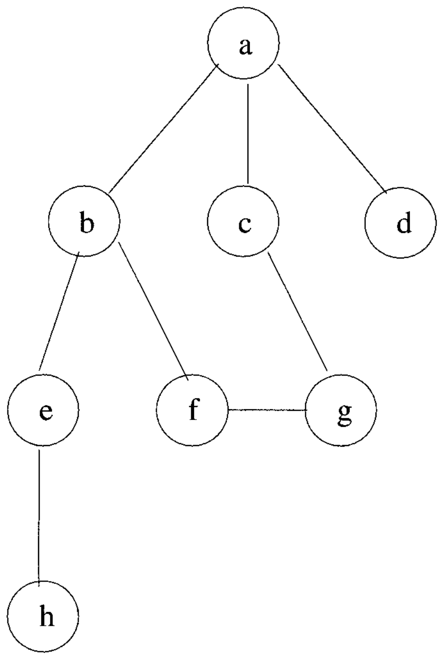


Figure 18: The Example that Refinement Makes Difference

With the original algorithm as shown in Figure 19 (a), the weights of each uninformed node are presented in round 2. The nodes with shadowed backgrounds are informed nodes. As mentioned before, the weight of a node is the approximate time before broadcasting is finished after the node is informed. By using the new heuristic, in the second round, the weight of node *f* and *c* are both 1 because node *g* is a child of theirs. Based on the calculation following the original algorithm, we can ultimately get the final schedule as seen in Figure 19 (b).

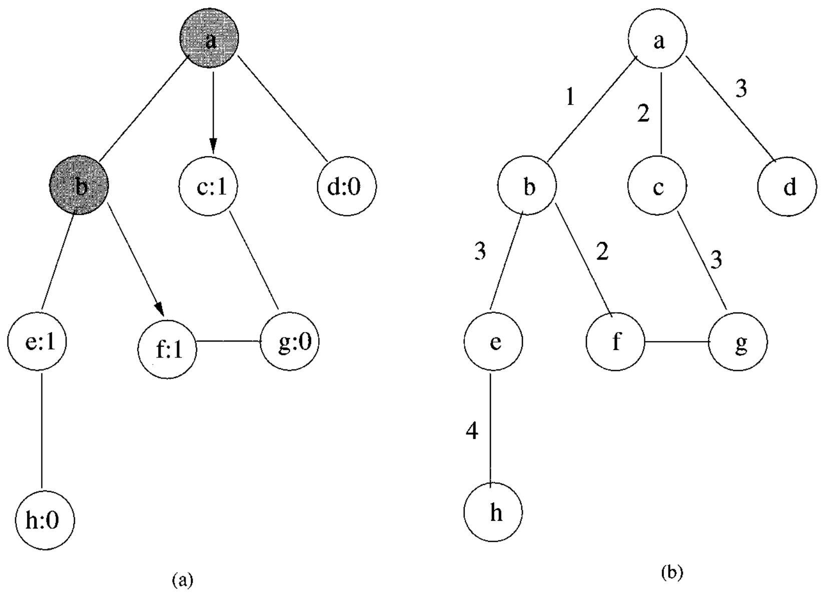


Figure 19: The Example with Schedule from Original Algorithm

With the refined algorithm as shown in Figure 20, the weights of each uninformed node are also presented in round 2. By using the new heuristic, in the second round, the weight of node *f* and *c* are both 1.5, because node *g* is a child of theirs and g is divided into two and assigned to its parents accordingly. The weights of each uninformed node are presented in round 2 according to definition 7. Based on the calculation following the refined algorithm, one can get the final schedule as show in Figure 20(b). And one can see that with the refined algorithm, one can reduce one round for this topology

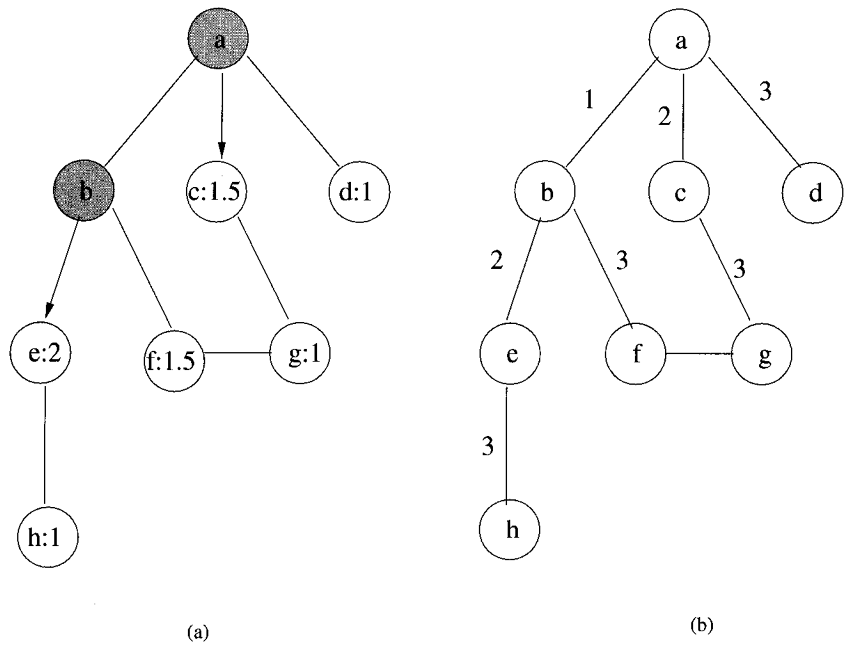


Figure 20: The Example with Schedule from Refinement Algorithm

3.1.3 Complexity

Steps	Time Complexity
Step 1	O(n)
Step 2	O(m)
Step 3	O(n)
Step 4	O(n^3)
Step 5	O(n)

Table 1: List of Steps' Complexity

The original and refinement algorithm have the same complexity since they are only different in the way the weight calculation is done. That does not make a difference for the complexity. From the algorithm one can see that steps 2 and 4 dominate the time complexity. Step 2 (the implementation of assigning weights) has two phases. In the first phase, the heuristic performs a Breadth First Search (BFS) of *DR(t)* from the *bb(t)* and label each node by *D(v, t)*. At the same time, a set is created to save nodes that have no children. This set is denoted by *rb(t)*, which stands for remote border where *m* denotes the number of edges of graph *G*. This step can be done in *O(m)* time. In the second phase, a recursive process is used to compute weight for each node in *DR(t)*. This process starts from *rb(t)* towards *db(t)*. In the worst case, one has to calculate *EB(v,t)* for every node of graph *G*. The degree of the *ith* node of graph *G* is denoted by d_i. By using the new heuristic to calculate *EB(v, t)*, the time needed for a node with degree *d* is *O(d)*. The time needed to calculate all the nodes is $\sum_{i=1}^{n} O(d_i)$. Since $\sum_{i=1}^{n} d_i$ = 2m, the complexity of this step is *O(m)*. Thus, the total complexity of assigning weights (step 2) is *O(m)*[7].

When using the maximum matching algorithm and Gabow's implementation of Edmond's algorithm for step 4, the time complexity in one round is $O(n^3)$ according to Chapter 2 This approach to implementation will dominate the complexity of the whole algorithm. As seen from Table 1, the complexities at step 1, 3 and 5 are trivial for the whole process. Assuming that the total round for the whole procedure is *R*, so we can see that the total time complexity of the new algorithm is $O(Rn^3)$.

3.2 Gossiping

3.2.1 Description

This section presents a heuristic for gossiping in arbitrary graphs. This heuristic is derived from the broadcasting algorithm as described above. The input of this algorithm is a graph *G=(V, E)* and its output is a gossiping schedule for this graph.

At each round *t*, a message *s* has an informed area (vertices holding *s*), an uninformed area (vertices not holding *s*), a bright border (vertices are holding *s* and having uninformed neighbors) and a dark border (a set of vertices are not holding *s* and have informed neighbors). For a message *s*, this algorithm performs a variant of BFS from the bright border to the uninformed border and labels each visited vertex *u* with the shortest distance from *u* to the bright border of *s*. Then, this algorithm calculates the weights for all uninformed vertices to message *s*. Given an edge (*u, v*), the weight of (*u, v*) of message *s* is the weight of *v* if *u* is in the bright border of *s* and *v* is in the dark border of *s*, and is zero otherwise. In total, the final weight of an edge is the sum of its weights of all the $|V|$ messages. Then, the new algorithm finds the matched pairs of nodes for graph *G* based on the weights of all edges Maximum-Weighted Matching. Based on this idea one can describe the algorithm as follows:

1. Initial graph: All nodes have one unique message. Bright Region BR(*t*) contains one node only for any node.
2. Select the first node in the graph.
3. Calculate *EB* for any node in Dark Region DR(*t*) for round t, and set this node's weight as EB.
4. Extract the weight value for all nodes in bright border bb(*t*), and add this weight to adjacent edge in cbe(*t*).
5. Go to the next node, repeat step 3 and step 4, until all nodes have been selected to assign weight to their cbe(*t*).
6. Get the maximum-weight match for the whole graph.
7. Exchange messages between the matched nodes, so both of them have the same information. Recalculate BR(*t*) and DR(*t*) for all nodes. Increase *t* by 1.
8. Repeat 2 to 8 until all the nodes have all the messages from all the other nodes.

Actually, we have two solutions for this also: original and refinement. These solutions are based on the way we calculate the EB in step 3. For the original we set EB as *EB*(*v, t*) (by definition) and similarly we chose *REB*(v, t) (as shown in definition 7) for refinement. Again the refinement makes a difference only for some topologies.

3.2.2 Complexity

The gossiping problem on an arbitrary graph is also NP-complete [20]. Several heuristics for gossiping have been presented in [10] and [21]. Among them, the algorithm in [10] is the best existing heuristic in practice.

In regards to the new algorithm, one can derive the complexity in the same way as in section 3.1.3. The dominating step is step 6 which is the same as step 5 in the broadcasting process. Step 2 to step 5 is the procedure that involves assigning weight to edges. The complexity for them is $O(nm)$, since this is the repeating process for all nodes. Thus the complexity for gossiping is the same as that of broadcasting: $O(Rn^3)$.

3.2.3 Example

Suppose one has a graph with 4 nodes and 3 edges as shown in Figure 21. At the very beginning, each node has a unique message (identified by its ID) to be disseminated to all other nodes. At round t, every message P can contribute to the weight of all edges in its cbe(t). Thus, one can get weight for every edge that sums weight contributions from all nodes. The entire procedure will be described in the following section.

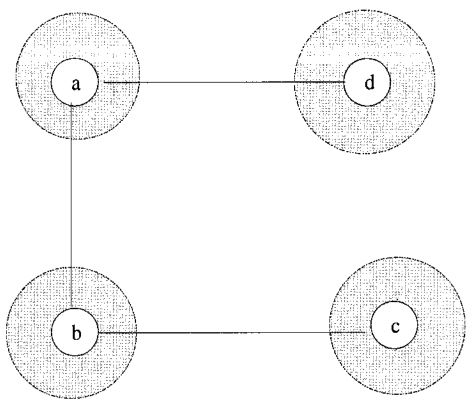


Figure 21 Gossip Example -- Original State

The gossiping procedure is finished with 3 rounds as shown in Figure 22. In the first round, all of the weights assigned to the three edges are listed in Table 2. N stands for nodes, while E refers to edge. These values are weights and the distribution situation for the edges from corresponding nodes obtained from the algorithm discussed above. So edge (b, c) and (a, d) are selected as matched edges for round 1.

N / E	B	a	c	d	Total	Matching
(b,a)	1	1			2	No
(b,c)	1		2		3	Yes
(a,d)		1		2	3	Yes

Table 2: Weight Calculation at Gossip example -- round 1

After round 1, every node changes its corresponding cross border edges for round 2 Specifically, node *a* and *d* share the same cross border edge (*a, b*) and have the same DR and BR. Node b and node c share the same cross border edge (*a, b*) as well and have the same DR and BR. In the end the only cross border edge all of these four nodes share are the same (*b, a*). It is obvious that only (*b, a*) is selected as the matched pair.

N \ E	B	A	C	d	Total	Matching
(b,a)	1	1	1	1	4	Yes
(b,c)						No
(a,d)						No

Table 3: Weight Calculation at Gossiping Example -- round 2

N \ E	B	A	c	d	Total	Matching
(b,a)						No
(b,c)				1	1	Yes
(a,d)			1		1	Yes

Table 4: Weight Calculation at Gossiping Example -- round 3

Similarly, for round 3 the matched pairs (*b, c*) and (*a, d*) are derived from the Table 4. Actually in this situation there are two BRs, for *a, d,* it is (*a, b, d*) and for *b, c* it is (*a, b, c*). Edge (*a, b*) is always in the BR, so it is impossible for it to be select as a matched pair. The result for round 3 is the same as round 1. After this round, all of the nodes have received all of the messages, thus, all nodes have become experts in regards to the information.

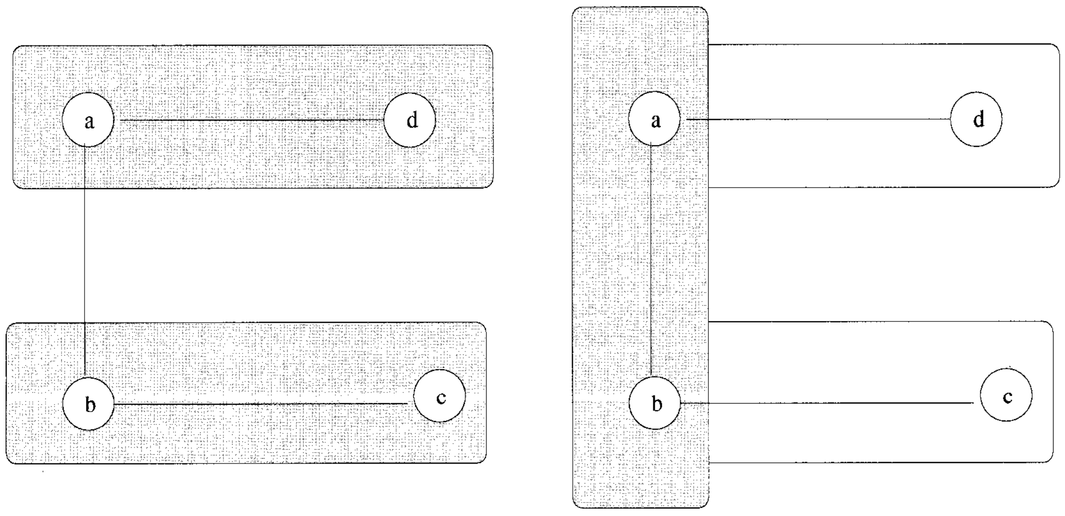


Figure 22 Gossiping Example -- Round 1 & 2

Finally, we can see that all of the messages are disseminated to all of the nodes as shown in Figure 23. All the nodes have received all of the messages, and all the nodes have the same DR, BR and the cross border edge set is empty.

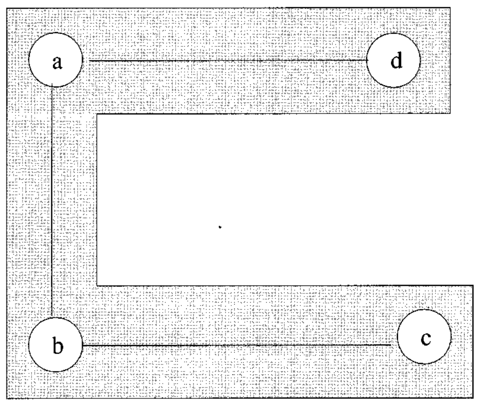


Figure 23: Gossiping Example -- Final State

4. Implementation

In order to test the algorithm and compare it with other heuristics, this implementation was done with a pure C++ solution. The objective of this implementation is to develop a research-grade, full C++ compliant and easy to use application.

To run this program, one only needs the exe file of graph.exe and a set of empty folders for specified graphs to hold the log file and other temporary files. These folders can be generated with an accompanying program called Makdir.exe. This application is coded with Visual C++ 6.0 and has been tested on Windows 2000 and Windows XP. The run program graph.exe is about 800K in size.

In this chapter, the high level classes are introduced in section two and the data structure used is presented in section three. These two sections illustrate the basics for the implementation in following sections. The flowchart and some important methods are introduced in section four and five.

4.2 High Level Class Structure Diagram:

4.2.1 The Class Diagram

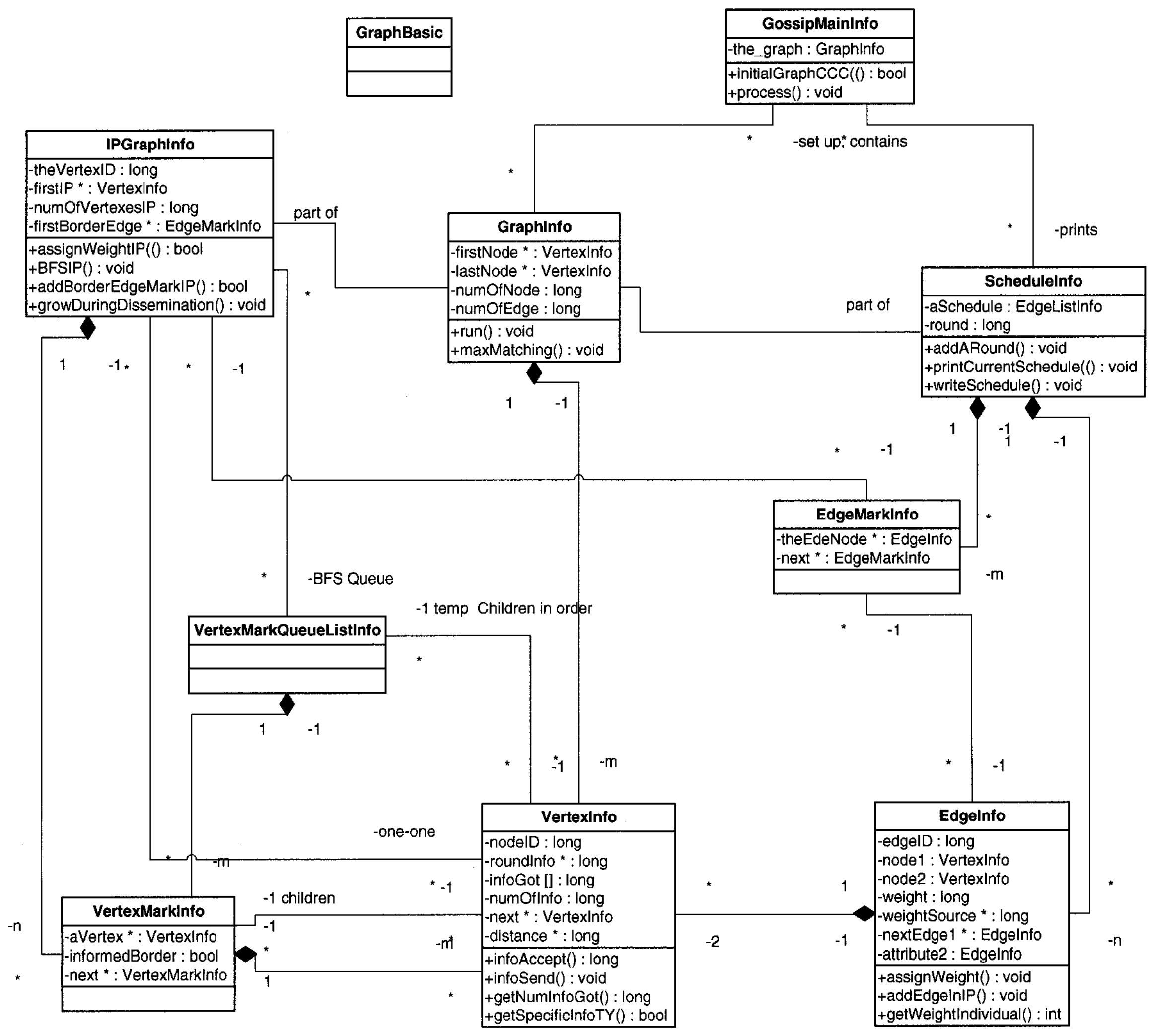


Figure 24: Static Class Diagram

Figure 24 presents the structure of the high level classes. In this diagram, we can see 9 classes. The names follow specific naming conventions for the purpose of consistency. The class ending with "Info" is the physical class. The corresponding name without "Info" is the pointer to this class. Among these classes as shown in Figure 24, the Class EdgeInfo and VertexInfo are the basic classes that contain the important info for the class Graph. The class IPGraphInfo is the heuristic class that is the main entity that is employed to implement the algorithm and control the dissemination process. In this case, IP stands for "Informed Points". The class of GraphBasicInfo contains all the constants and variables including those that may be used to adjust the interval distance and parameters to decide to run the original version of the algorithm or the refined one.

4.2.2 Classes Introduction

Classes	Descriptions
GossipMainInfo	This class initializes the graph given and offers the procedure's entrance and top level procedure control (initial a graph as input, print schedule as output)
VertexInfo	This is the main object used to store the information needed: message (represented by index ID) of its own or from others, it is the one that receives and sends messages to and from others. The main function of Vertex is to aid in determining the weight for the border edge by calculating the smallest distance from a given message area represented by IPGraph with BFS. Every vertex has a growing IPGraph
VertexMarkInfo	This class is the virtual map to the vertex used for IPGraph
VertexMarkQueueListInfo	This class is temporarily used for the process of BFS (used as queue) and sort the children (used as linked list)
EdgeInfo	This is the class for the connection between two vertices for messages to disseminate. It has weight that can be used as inference for maximum weight matching.
EdgeMarkInfo	This class is the virtual map of the edge for IPGraph's border cross edge and is an element of the class of Schedule in a round
GraphInfo	This class provides a container for overall vertices and edges for the dissemination of messages.
IPGraphInfo	This class represents a part of the graph with vertexMarks as a dispersion area for a given message. Like GraphInfo, it is a linked list based on vertex. However, while GraphInfo consists of real vertexes, IPGraphInfo only contains the pointer to the existing vertexes in the GraphInfo. Each vertex (message) has an IPGraph. It is the heuristic class.
ScheduleInfo	This class contains the final result of the run. It is a sequence of sets of edges, the result for the dissemination track. It contains an edge list that consists of pointers to existing edges of a given graph.

Table 5: Classes Introduction

Table 5 lists the main top level classes and their descriptions. These top classes were selected carefully based on the algorithm description. They have properties of independence, balance and interrelation. Any particular aspect of the solution fits into one and only one class. For example, the VertexInfo is the container for all of the information originating from a node, while corresponding IPGraphInfo is the control class for the entireinformation dissemination process.

4.2.3 Design Evaluation

The standard and the initial design were completed based on the principles of simplicity, coupling and cohesion, as well as information hiding.

Simplicity: Classes, methods, and the overall design are finished in as simple a way as possible. At class level, any operation or data element can no longer be removed. All the parameters and their return types are necessary and simplified. The code for any particular method fits on one page or one screen and each method need only be aware of its local environment and need not be aware of the overall class structure of the implementation of other classes, which could easily change.

Cohesion and Coupling: The proper set of classes has been chosen through cohesion and coupling. By cohesion, each class has only a single purpose. In addition, all classes do not depend on knowledge of the internal implementation of other classes. All of the relationships among classes are based on operations as opposed to data elements.

Information Hiding: The design has provided a high degree of abstraction and information hiding: it conceals implementation details within a particular class or method and separates those details from other classes. Data elements of each class are local to that class and are inaccessible elsewhere.

4.3 Data Structure

In this implementation, the main classes that hold information which need space are VertexInfo and Edgeinfo. VertexInfo has an ID that represents the unique message and contains the control information such as the number of messages received so far. Also, it has the message dissemination status info such as, its corresponding IPGraphInfo, weight value, its children and parents for other nodes during the weight calculation process. EdgeInfo contains information such as its weights, contribution distribution from the adjacent nodes. Other classes such as GraphInfo and IPGraphInfo contain the pointer to them with a linked list. There are other temporary classes which are mainly used at the middle of the weight calculation procedure such asVertexMarkInfo, VertexMarkQueueListInfo and EdgeMarkInfo. They are pure linked lists for VertexInfo or EdgeInfo. It should be noted that the most important part of datastructure in this implementation is the Multilist that was used in VertexInfo and EdgeInfo.

In order to introduce why one needs this data structure, let us start with a question that is asked quite often in school. Suppose one has 11,000 students in a university that offers 600 courses. And one wishes to record both every course that a student takes and every individual who attends a specific class. If one decided to select an array as the data structure, this would require an 11,000 x 600 array, which is huge. Furthermore, most of the entries in the array would be empty! Clearly this is a case for a Multilist. A Multilist is a data structure capable of providing several simultaneous linkages of data. The data points are regarded as belonging to several lists--each data point belongs to one list for each list type or entity. As shown in Figure 25, one can employ this data structure to make it more efficient. To see which students are in course C1, one simply checks the first column of the Multilist. To see which courses student S1 is taking, all one has to do is check the first row of the Multilist.

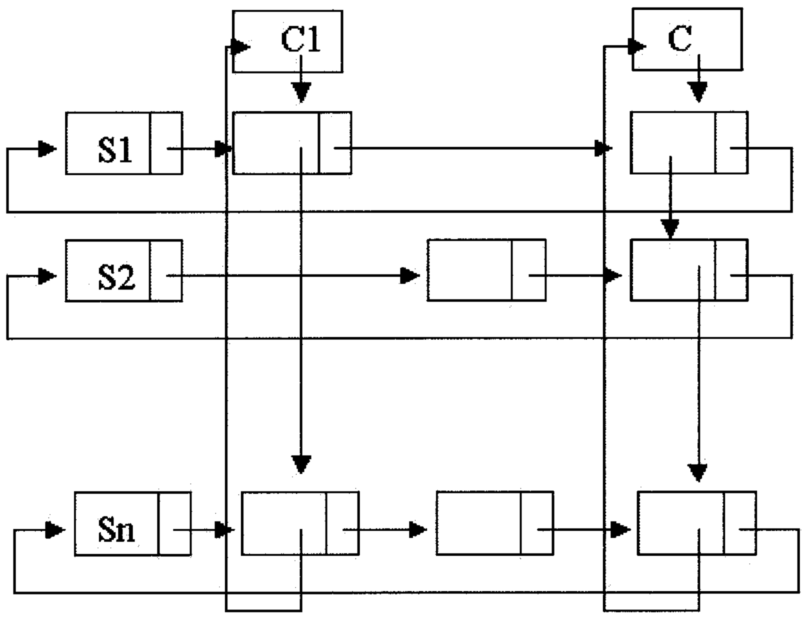


Figure 25: Multilist for Courses and Students

Similar to the case shown above, one can employ the same idea in the data structure for a graph. A node may have multiple adjacent edges, and an edge has two end nodes as well In this case the situation is less complex, since an edge has only two end nodes. So, one can simulate the situation as shown in Figure 26. *N* represents the nodes while *E* represents the edge. An edge is always located at a list with two nodes only.

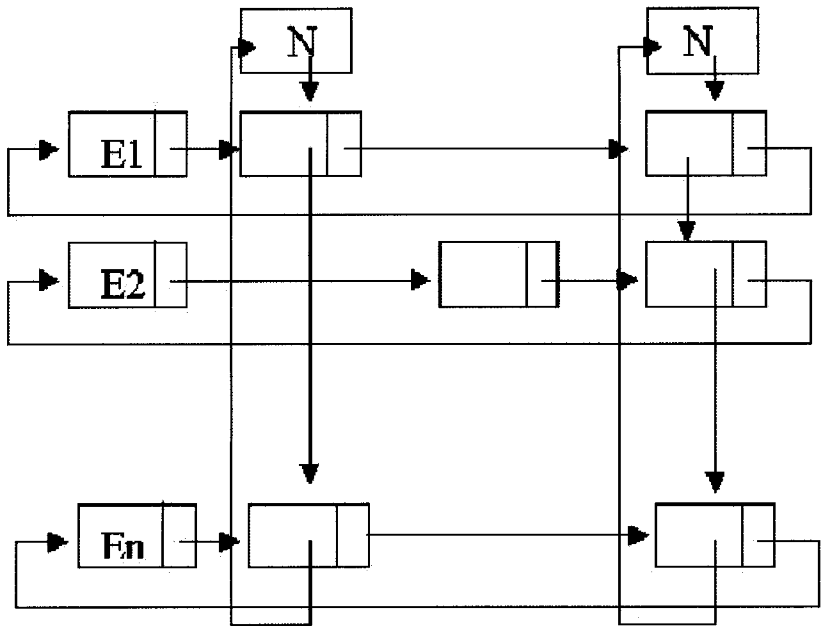


Figure 26: Multilist for Graph

The main purpose for creating the class of IPGraphInfo for each node is to make a visual net on a set of vertices corresponding to the way in which this node's message is disseminated. This virtual net expands based on the message dissemination procedure. This virtual net that covers the graph originates from its owner node. Whenever a round finishes, the net will spread in the direction of the neighbor nodes that have been chosen in the matched pairs.

4.4 Execution Flow Chart

As shown in Figure 27, one can see the process of the program. All of these operation methods are distributed to corresponding classes.

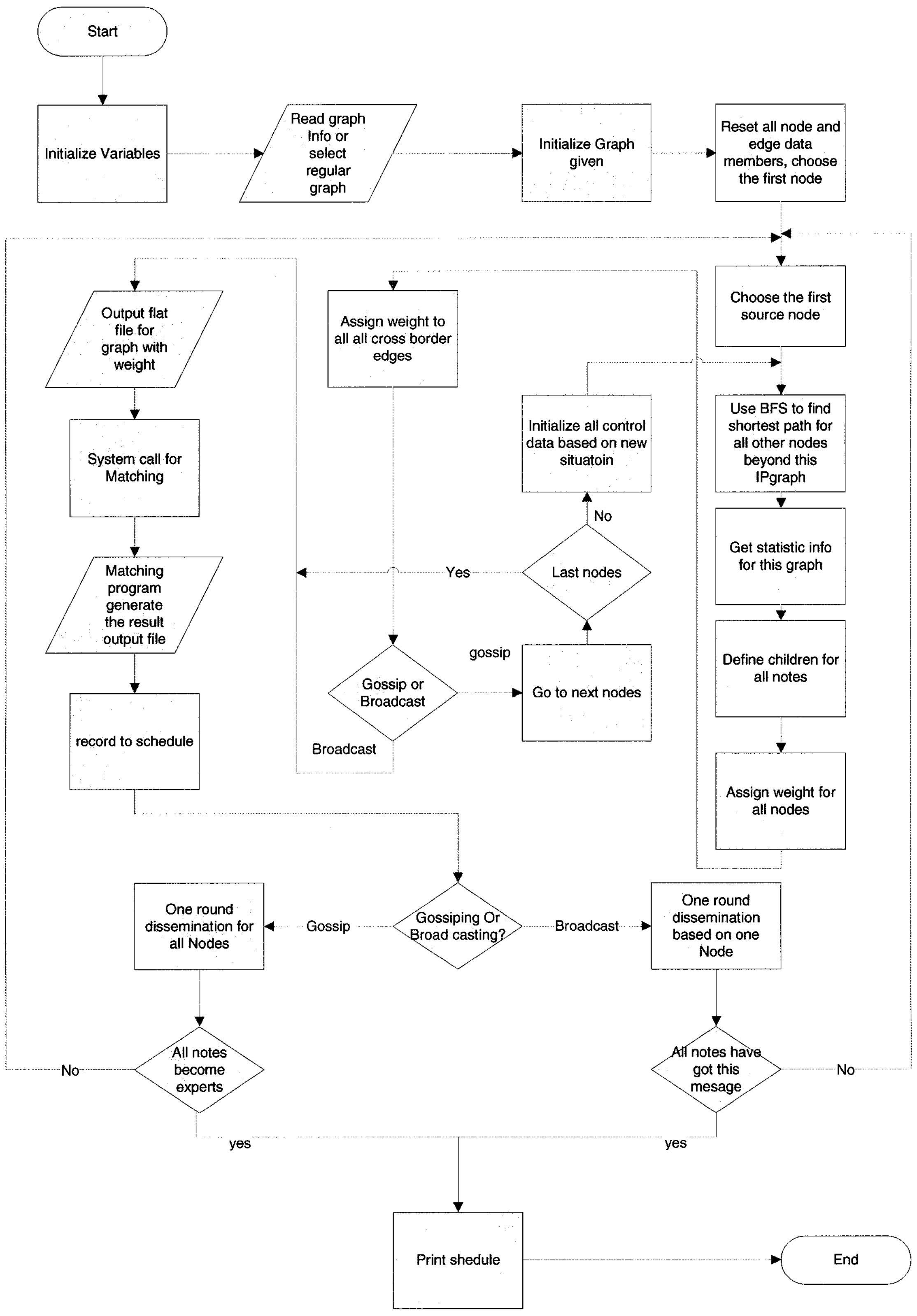

Start
Initialize Variables
Read graph Info or select regular graph
Initialize Graph given
Reset all node and edge data members, choose the first node
Choose the first source node
Use BFS to find shortest path for all other nodes beyond this IPgraph
Get statistic info for this graph
Define children for all notes
Assign weight for all nodes
Assign weight to all all cross border edges
Initialize all control data based on new situatoin
No
Last nodes
Yes
Gossip or Broadcast
gossip
Go to next nodes
Broadcast
Output flat file for graph with weight
System call for Matching
Matching program generate the result output file
record to schedule
Gossiping Or Broad casting?
Gossip
Broadcast
One round dissemination for all Nodes
One round dissemination based on one Node
All notes become experts
No
yes
All notes have got this mesage
No
yes
Print shedule
End

Figure 27: Flow Chart for the Implementation

4.5 Some of the Core Methods in this Solution.

Some of the methods that are critical for the solution are listed as follows.

- bool GossipMainInfo::initialGraphCCC(): This is the sample of the methods that initialize graph. There are groups of this kind of methods. Different graph ends with abbreviation.
- void GraphInfo::runGossip(): This is the main method that controls the entire process. It is the start point to go to all of the steps in detail or control the display of the message or the log file.
- void GraphInfo::runBroadcast(IntegerID aIDD): Similar with runGossip(), but for broadcasting process.
- void IPGraphInfo::BFSFindChildrenIP(): This is the one of the main parts of the algorithm that finds the children of all nodes according to the current message that is spreading.
- void IPGraphInfo::growDuringDissemination(EdgeMark aEM): This method is used to control the process of the virtual net growing.
- bool IPGraphInfo::assignWeightIP(): This is place to get the weight values contributed from a node.

5. Experimented Result and Comparison with Round_Heuristic

5.1 Introduction

This chapter presents the test results in several regular topologies shown in Chapter two and three NS-2 models for both gossiping and broadcasting. The test result for four regular topologies will be shown in section 2, while the results for three models are introduced in section 3. Also in section 3, the three models are presented for the reference. Finally in section 4, a summary will be given.

On the tables and figures, we use the following abbreviations:

- RH: The best result obtained from Round_Heuristics algorithm [1] ;
- RH81: The testing result from Round_Heuristic with parameters of Dis_Exp as 8 and Num_Exp as 1
- Original: test result of the original version of the new algorithm.
- Refinement: test result from the refinement version of the new algorithm.
- P: the parameter in the refinement version.
- LB: lower bound
- UP: Upper bound
- K: dimension number

5.2 Test Result of Four Regular Graphs and Comparison

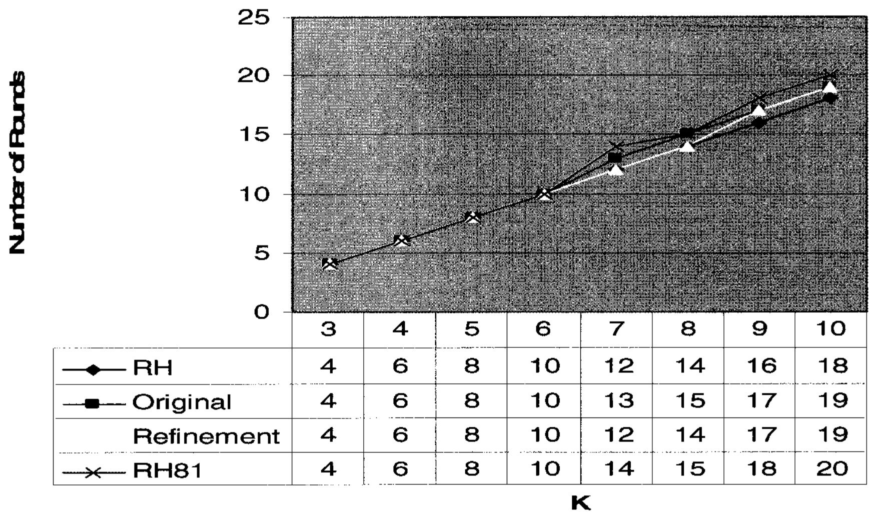
In this part, one can see the test result for four regular graphs, the *CCC_d* graph, the *deBruijn* graph, the *Shuffle Exchange* graph and the butterfly graph. The following algorithms were tested: Round_Heuristic, Round_Heuristics with parameter Dis_Exp as 8 and Num_Exp as 1, original version of the new algorithm, the refinement algorithm, and TBA [7] for the broadcasting part. These will be shown together both in the table and figures in order to clarify the comparison between them.

5.2.1 Testing Result of Gossiping

5.2.1.1 The de Bruin Graph

K	RH	Original	Refinement	P	RH81	LB	UB
3	4	4	4	2	4	4	11
4	6	6	6	1	6	6	14
5	8	8	8	1	8	7	17
6	10	10	10	1.6	10	8	20
7	12	13	12	2.8	14	10	23
8	14	15	14	3	15	11	26
9	16	17	17	0.5	18	12	29
10	18	19	19	1	20	13	32

Table 6: Testing Result of de Bruin Graph for Gossiping



K	3	4	5	6	7	8	9	10
RH	4	6	8	10	12	14	16	18
Original	4	6	8	10	13	15	17	19
Refinement	4	6	8	10	12	14	17	19
RH81	4	6	8	10	14	15	18	20

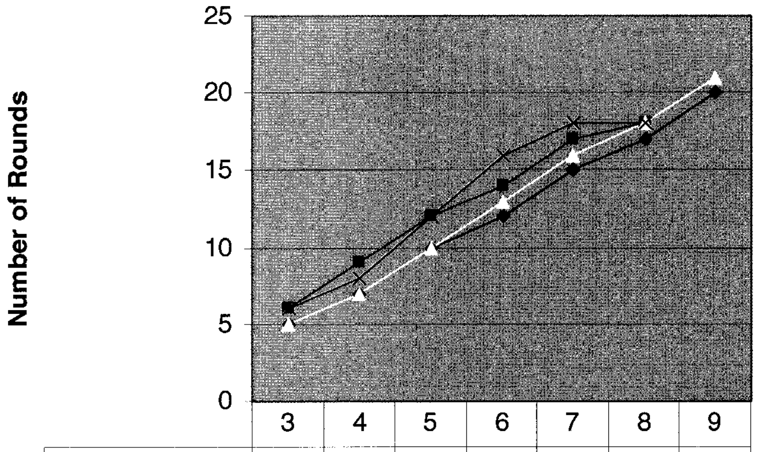
Figure 28: Testing Result of de Debruin for Gossiping

For the de Bruin Graph, we tested a variety of numbers (K) from 3 to 10 as shown in Table 6 and Figure 28. When K is small, all of the algorithms perform the same way. Then when K increases, the difference gradually becomes apparent. The best result from Round Heuristics has a stronger result. But the refinement of the new algorithm always generates a better result than RH81.

5.2.1.2 The Butterfly Graph

K	RH	Original	Refinement	P	RH81	LB	UB
3	5	6	5	0	6	5	9
4	7	9	7	1.6	8	7	13
5	10	12	10	1.2	12	9	17
6	12	14	13	1.6	16	11	21
7	15	17	16	0.8	18	13	25
8	17	18	18	2.4	18	15	29
9	20		21	1.6		17	33

Table 7: Testing Result of Butterfly for Gossiping



K	3	4	5	6	7	8	9
—◆— RH	5	7	10	12	15	17	20
—■— Original	6	9	12	14	17	18	
Refinement	5	7	10	13	16	18	21
—×— RH81	6	8	12	16	18	18	

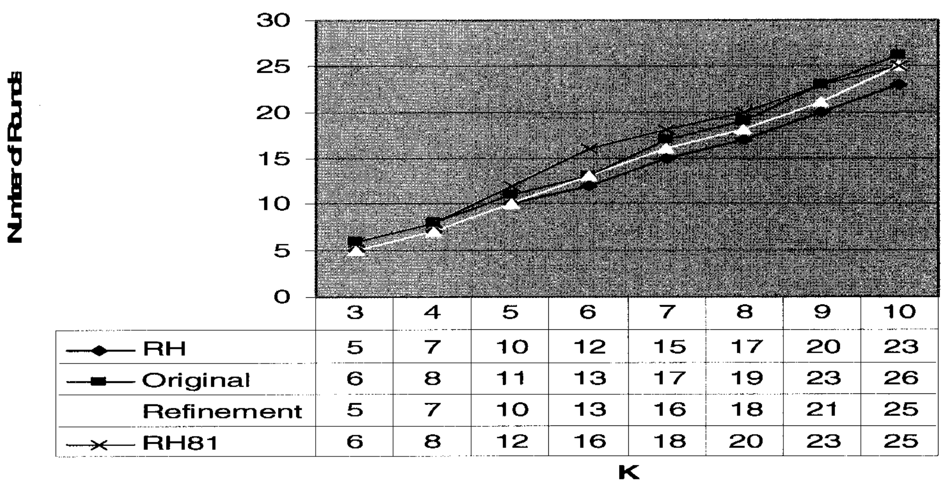
Figure 29: Testing Result of Butterfly for Gossiping

The difference for the Butterfly graph starts at the very beginning as evidenced in Table 7 and Figure 29. The results from the original algorithm and RH81 always get a worse result, while the result of the refinement keeps close to Round_Heuristics.

5.2.1.3 The Shuffle-Exchange Graph

K	RH	Original	Refinement	P	RH81	LB	UB
3	5	6	5	0	6	5	9
4	7	8	7	1.6	8	7	13
5	10	11	10	1.2	12	9	17
6	12	13	13	1.6	16	11	21
7	15	17	16	0.8	18	13	25
8	17	19	18	2.4	20	15	29
9	20	23	21	1.6	23	17	33
10	23	26	25	2	25	19	37

Table 8: Testing Result of Shuffle-Exchange for Gossiping



K	3	4	5	6	7	8	9	10
—◆— RH	5	7	10	12	15	17	20	23
—■— Original	6	8	11	13	17	19	23	26
Refinement	5	7	10	13	16	18	21	25
—×— RH81	6	8	12	16	18	20	23	25

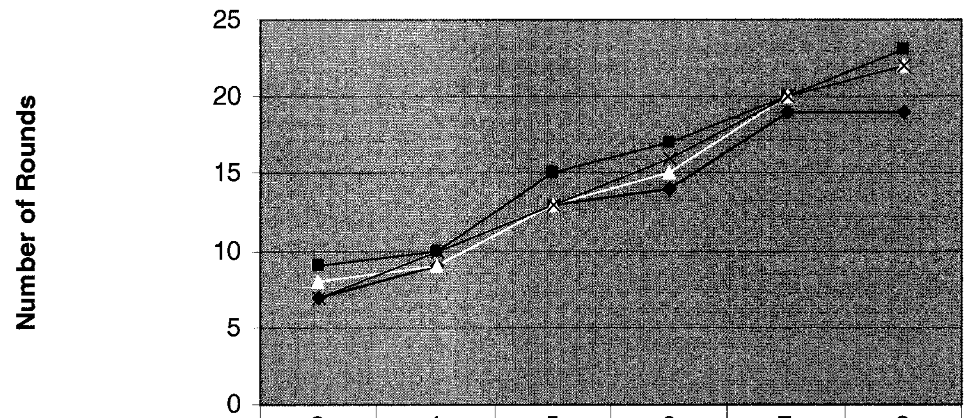
Figure 30: Testing Result of Shuffle-Exchange for Gossiping

In Table 8 and Figure 30, one can see the test result for the Shuffle-Exchange graph. Like the Butterfly graph, four results are different. The refinement keeps close to the best result of Round_Heuristic and better than the Round_Heuristic with parameters of Dis_Exp 8 and Num_Exp 1.

5.2.1.4 The Cube Connected Cycles Graph

K	RH	Original	Refinement	P	RH81	LB	UB
3	7	9	8	1.2	7	7	10
4	9	10	9	0	10	9	10
5	13	15	13	0	13	11	15
6	14	17	15	2	16	13	15
7	19	20	20	0.1	20	16	20
8	19	23	22	0	22	18	20

Table 9: Testing Result of CCC for Gossiping



	3	4	5	6	7	8
◆ RH	7	9	13	14	19	19
■ Original	9	10	15	17	20	23
Refinement	8	9	13	15	20	22
× RH81	7	10	13	16	20	22

K

Figure 31: Testing Result of CCC for Gossiping

For the Cube Connected Cycles Graph, one gets the testing result shown in Table 9 and Figure 31. When employing this kind of graph, the original of the new algorithm will always generate the worst result. However, the refinement has better results.

5.2.2 Broadcasting Testing Result

For this test, one encounters the same results as in [7]. In regards to the broadcasting process, the new algorithm performs in an almost identical fashion as the Round_Heuristic in several regular topologies. All of the results for $d \leq 14$ are the same as those in Round-Heuristics with the exception of four cases. In two cases the new algorithm gives better results and in two other cases the result from RH is better. Three of the four differences occurred in the butterfly graph where $d = 10$, $d = 12$ and $d = 14$. In addition, the new algorithm generates new upper bounds on broadcast time for the Butterfly graph and cub connected cycles graph when $15 \leq d \leq 16$, and for de Bruin (2, d) and Shuffle-Exchange with $15 \leq d \leq 20$.

5.3 Four Topology Models Test Result and Comparison

5.3.1 Introduction to the Models

If one wishes to accurately test the algorithm in the real world, an Internet like topology should be considered. The basic topological structure of the Internet can be modeled as the following graph that reflects the locality and hierarchy of the Internet [2].

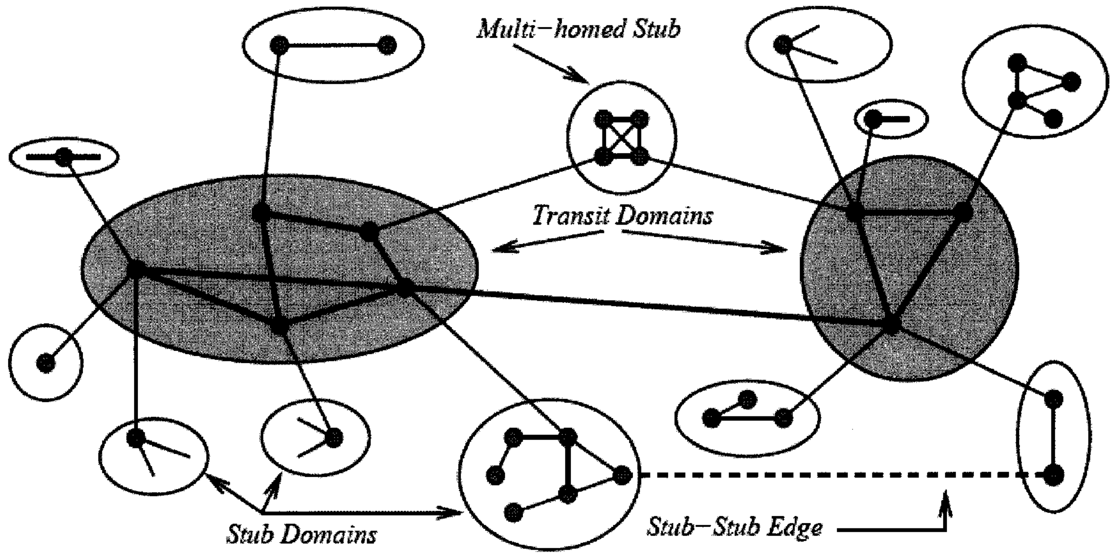


Figure 32: Internet Domain Structure

As seen from Figure 32, today's Internet can be viewed as a collection of interconnected routing domains. Each domain is a group of nodes (routers, switches and hosts): Each routing domain in the Internet can be classified as either a stub domain or a transit domain. A transit domain consists of a set of backbone nodes that also connect to a number of stub domains via gateway nodes in the stubs or connect to other transit domains. Stub domain can be further classified as single- or multi- homed. Some stub domains may have links to other stubs.

Three levels of hierarchy can be modeled, in correspondence with the transit domain, stub domains, and LANs attached to stub nodes. Also two sets of parameters can be used to control the coarse properties of the generated networks.

For the size of the three levels in the hierarchy:

- T: the total number of transit domains; N_T: the average number of nodes per transit domains.
- S: the average number of stub domains per transit domain; N_S: the average number of nodes per stub domain.
- L: the average number of LANs per stub node; N_L: the average number of hosts per LAN.

For the Intranet work connectivity and Internet work connectivity, we have:

- E_T: the average number of edges from a transit node to other transit nodes in the same domain.
- E_S: the average number of edges from stub node to other stub nodes in the same domain.
- E_{TT}: the average number of edges from a transit domain to another transit domain.
- E_{ST}: the average number of edges from a stub domain to a transit domain.
- E_{LS}: the average number of edges from a LAN to a stub node.

Currently, there are three main implementations of the basic generation method:

Transit-Stub (TS): part of Georgia Tech Internetwork Topology Models (GT-ITM). All nodes are of the same type, that is $N_L = E_{LN} = 0$. This model produces connected sub-graphs by repeatedly generating a graph according to the edge count, and checking the graph for connectivity. Extra edges from stub domains to transit nodes are added through the random selection of domains and nodes.

Inet: Inet [13] is a generator aimed at reproducing the connectivity properties of Internet topologies. This generator initially assigns node degrees from a power-law distribution and then proceeds to interconnect them by using different rules. Inet first determines whether the resulting topology will be connected, second forms a spanning tree using nodes of degree greater than two, attaches nodes with degree one to the spanning tree and then matches the remaining unfulfilled degrees of all nodes with each other.

Tiers: In the three level hierarchy structure, tiers are referred to as WAN, MAN and LAN levels. T= 1 because it does not support multiple WANs. It produces connected sub-graphs by joining all the nodes in a single domain through the use of a minimum spanning tree. When adding edges for Intranet work redundancy, edges are added to the closest nodes in the network in the increasing order of Euclidean distance. For internet work redundancy, extra edges are added to the closest nodes in the next higher domain.

Figure 33 shows a single level of network, Figure 34 shows a typical full internet work, and Figure 35 shows a larger internet work .

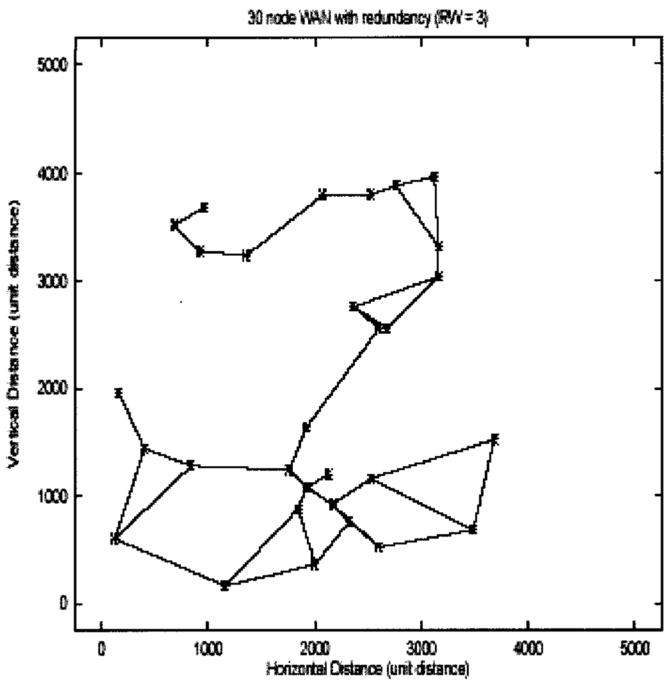


Figure 33: A Typical Tieres Network

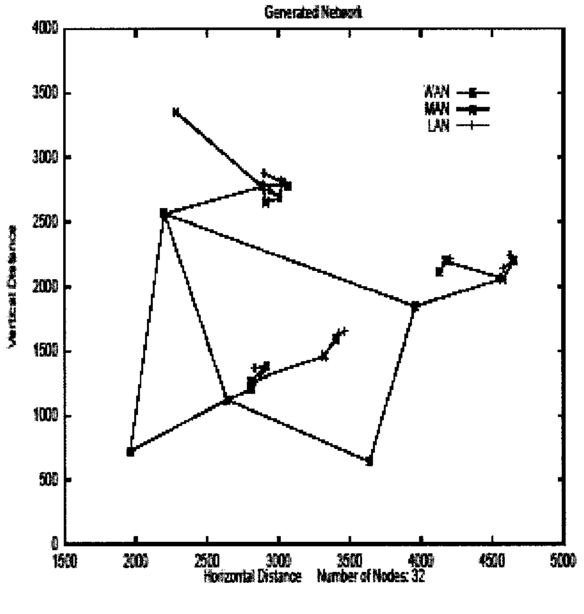


Figure 34: A Typical Tiers Internetwork

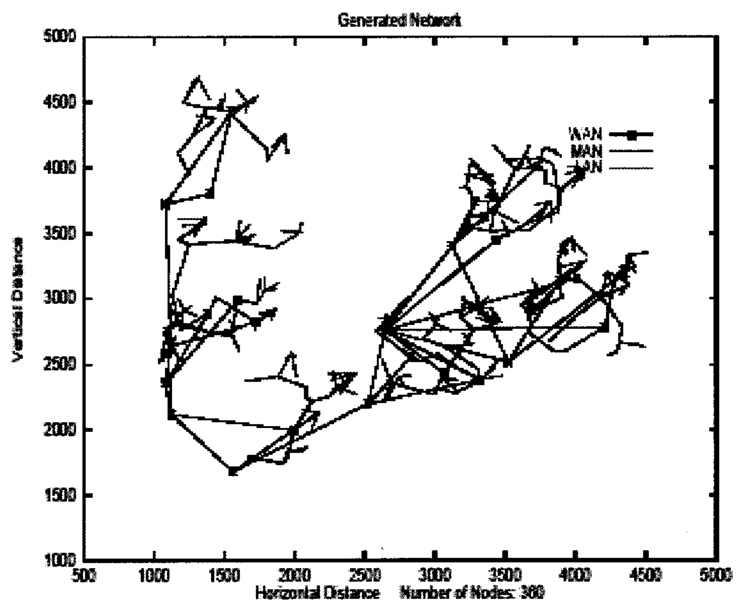


Figure 35: A Large Tiers Network

Ultimately one is not very concerned with the details of its implementation. What is important is one's ability to operate it and the results that it generates. In order to compare the algorithm of the original one, refinement and Round_Heuristic and TBA's result, four different network design models from ns-2 are considered: GT-ITM (including GT-ITM pure random and GT-ITM Transit-Stub), Tiers and Inet.

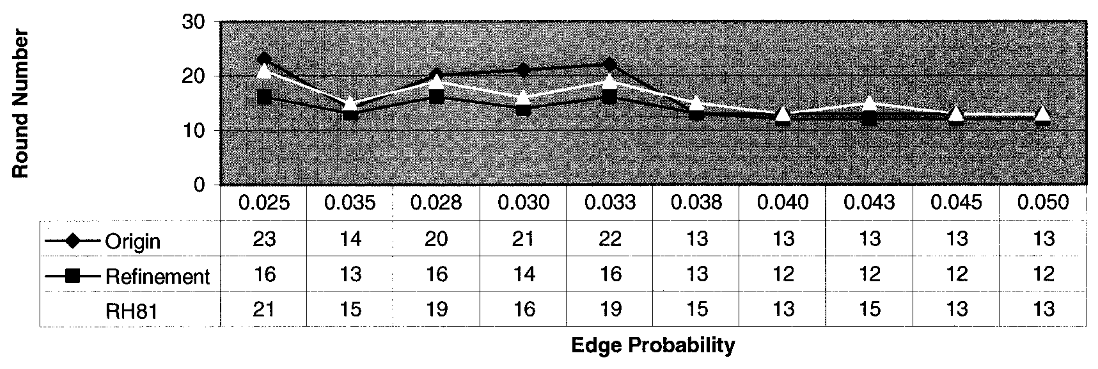
5.3.2 Gossiping Testing Results from Original, Refinement and Round_Heuristic

5.3.2.1 GT-ITM Random Model

Table 10, Table 11, and Table 12 present the test results in GT-ITM random model. Probability in the tables stands for the probability of having an edge between each pair of nodes. The number of nodes in the graphs corresponding to Table 10, Table 11, and Table 12 are 100, 300 and 400 respectively. Figure 36, Figure 37 and Figure 38 illustrate the differences and trends in a more staright forward way. In these figures, the horizontal scale represents the probability, and the vertical scale refers to the round used in the gossip process. There are three lines in these figures. The line of diamond points represents the testing result of the original algorithm, the line of triangle point represents the testing result of the Round-Heuritcs, and the line of square points shows the testing result in regards to refinement.

Probability	Edge	Origin	Refinement	P	RH81
0.025	148	23	16	4	21
0.035	187	14	13	3	15
0.028	145	20	16	2	19
0.030	158	21	14	3.5	16
0.033	177	22	16	4	19
0.038	204	13	13	1.5	15
0.040	211	13	12	2.5	13
0.043	213	13	12	7	15
0.045	225	13	12	0.5	13
0.050	235	13	12	1	13

Table 10: Testing Result of GT-ITM Random with 100 Nodes for Gossiping



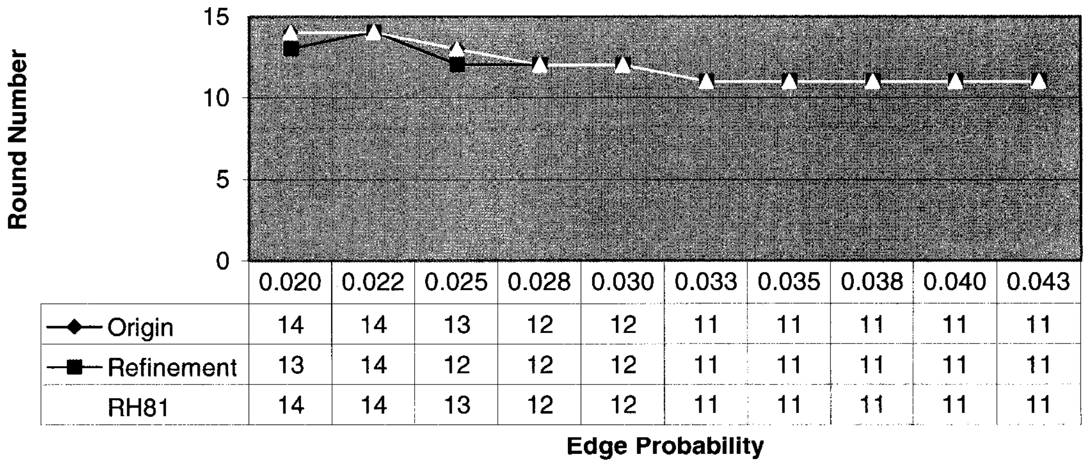
	0.025	0.035	0.028	0.030	0.033	0.038	0.040	0.043	0.045	0.050
—◆— Origin	23	14	20	21	22	13	13	13	13	13
—■— Refinement	16	13	16	14	16	13	12	12	12	12
RH81	21	15	19	16	19	15	13	15	13	13

Figure 36: Testing Result of GT-ITM Random with 100 Nodes for Gossiping

In Table 10 and Figure 36, one can see the testing result of graphs with 100 nodes. When the probability increases, the edge number increases accordingly, and the round number generated by three algorithms tends to generate a similar result. However, the refinement algorithm always generates a better result. At the probability of 0.035, all three algorithms have lower values. The reason for this is that the relatively large number of edges (187). The connections among the nodes results in an increase in speed of the gossiping and leads to a decrease in the round number employed for the entire procedure. Upon analyzing Table 10, one can see that the P value changes for different probabilities as well, even though there is no rule for this change.

Probability	Edge	Origin	Refinement	P	RH81
0.020	926	14	13	0	14
0.022	943	14	14	0	14
0.025	1125	13	12	0	13
0.028	1267	12	12	0	12
0.030	1350	12	12	0	12
0.033	1475	11	11	0	11
0.035	1569	11	11	0	11
0.038	1698	11	11	0	11
0.040	1788	11	11	0	11
0.043	1989	11	11	0	11

Table 11: Testing Result of GT-ITM Random with 300 Nodes for Gossiping



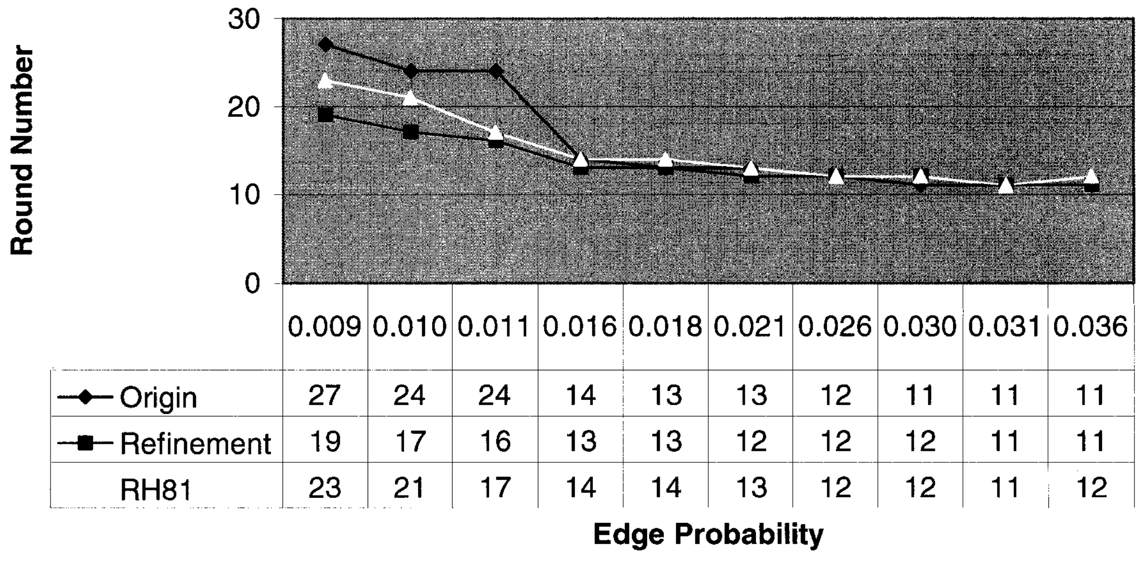
	0.020	0.022	0.025	0.028	0.030	0.033	0.035	0.038	0.040	0.043
—◆— Origin	14	14	13	12	12	11	11	11	11	11
—■— Refinement	13	14	12	12	12	11	11	11	11	11
RH81	14	14	13	12	12	11	11	11	11	11

Figure 37: Testing Result of GT-ITM Random with 300 Nodes for Gossiping

Table 11 and Figure 37 show the test result in GT-ITM model with 300 nodes. Onecan see that both the original and the Round_Heuristic have the same value and the refinement has a better value when the probability is less than 0.028.

Probability	Edge	Origin	Refinement	P	RH81
0.009	779	27	19	2	23
0.010	866	24	17	3	21
0.011	902	24	16	0	17
0.016	1319	14	13	3	14
0.018	1405	13	13	0	14
0.021	1645	13	12	3	13
0.026	2049	12	12	0	12
0.030	2389	11	12	0	12
0.031	2488	11	11	0	11
0.036	2881	11	11	0	12

Table 12: Testing Result of GT-ITM Random with 400 Nodes for Gossiping



	0.009	0.010	0.011	0.016	0.018	0.021	0.026	0.030	0.031	0.036
—◆— Origin	27	24	24	14	13	13	12	11	11	11
—■— Refinement	19	17	16	13	13	12	12	12	11	11
RH81	23	21	17	14	14	13	12	12	11	12

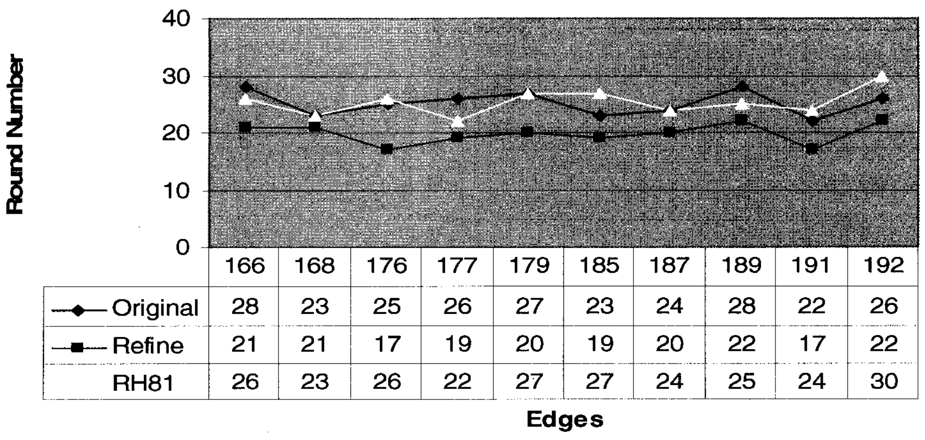
Figure 38: Testing Result of GT-ITM Random with 400 Nodes for Gossiping

The graphs corresponding to Table 12 and Figure 38 have 400 nodes, and they are also generated through the use of the GT-ITM pure random model. When the probability is 0.030, the original algorithm has a better value than both of the refinement and Round_Heuristics. However as seen in the previous two cases, when the probability is little, refinement always posses a greater value. When the probability increases, they tend to generate the same value.

5.3.2.2 GT-ITM Transit-Stub Model

Edges	Original	Refine	P	RH81
166	28	21	5.5	26
168	23	21	3.5	23
176	25	17	7.5	26
177	26	19	2.5	22
179	27	20	5.5	27
185	23	19	4	27
187	24	20	5	24
189	28	22	3	25
191	22	17	3.5	24
192	26	22	4	30

Table 13: Testing Result of GT-ITM Transit Stub with 100 Nodes for Gossiping



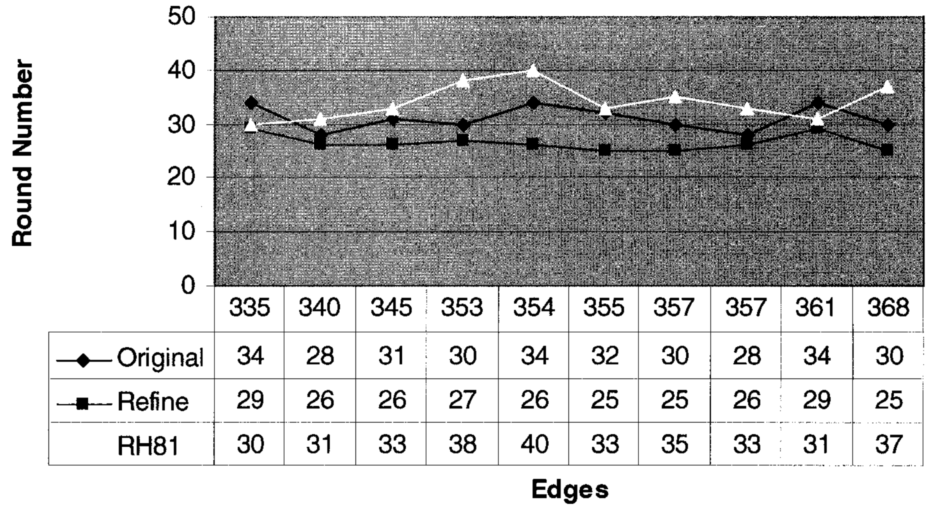
Edges	166	168	176	177	179	185	187	189	191	192
—◆— Original	28	23	25	26	27	23	24	28	22	26
—■— Refine	21	21	17	19	20	19	20	22	17	22
RH81	26	23	26	22	27	27	24	25	24	30

Figure 39: Testing Result of GT-ITM Transit Stub with 100 Nodes for Gossiping

Table 13 and Figure 39 show the test result from graphs generated according to the following parameters. The initial seed is 47. Each graph has 3 stub domains per transit node, with no extra transit-stub or - stub-edges. There is only one transit domain with 4 nodes, and the probability of an edge between each pair of nodes is 0.6. Each stub domain will have (on average) eight nodes, and an edge probability of 0.42. The total number of nodes is 100. It is evident that the new refinement algorithm always generates better results than the other two.

Edges	Original	Refine	P	RH81
335	34	29	3	30
340	28	26	1.5	31
345	31	26	2.5	33
353	30	27	5.5	38
354	34	26	6	40
355	32	25	4	33
357	30	25	2	35
357	28	26	4.5	33
361	34	29	5	31
368	30	25	2	37

Table 14: Testing Result of GT-ITM Transit Stub with 200 Nodes for Gossiping



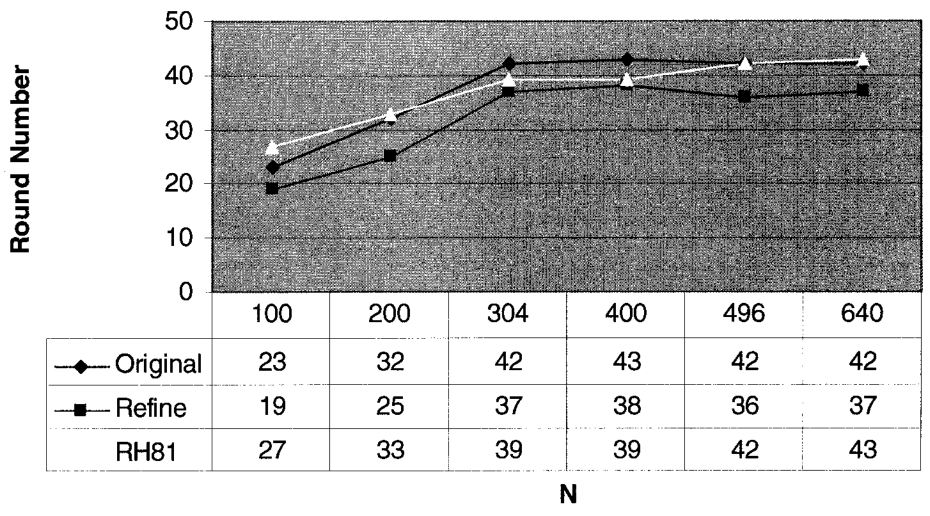
	335	340	345	353	354	355	357	357	361	368
—◆— Original	34	28	31	30	34	32	30	28	34	30
—■— Refine	29	26	26	27	26	25	25	26	29	25
RH81	30	31	33	38	40	33	35	33	31	37

Figure 40: Testing Result of GT-ITM Transit Stub with 200 Nodes for Gossiping

Table 14 and Figure 40 show the testing result from graphs with 200 nodes that have various edges. They were generated according to the following parameters. Initial seed is 47. Each graph has 3 stub domains per transit node, with no extra transit-stub or stub-stub edges. There is only one transit domain that has eight nodes on average, and the probability of an edge between each pair of nodes is 0.6. Each stub domain will have (on average) eight nodes, and an edge probability of 0.42. The total number of nodes is 200. One can see that the new refinement algorithm always generates better results then the other two. The number of rounds remains at a similar level.

N	Edges	Original	Refine	P	RH81
100	185	23	19	4	27
200	355	32	25	4	33
304	453	42	37	1.5	39
400	368	43	38	2.5	39
496	1029	42	36	1.5	42
640	1696	42	37	3	43

Table 15: Testing Result of GT-ITM Transit Stub with Various Nodes for Gossiping



N	100	200	304	400	496	640
◆ Original	23	32	42	43	42	42
■ Refine	19	25	37	38	36	37
RH81	27	33	39	39	42	43

Figure 41: Testing Result of GT-ITM Transit Stub with Various Nodes for Gossiping

In order to study the performance of these algorithms as the number of nodes increase with a large number of nodes, four graphs have been created as shown in Table 15 and Figure 41. These four extra graphs have two transit domains. The graph with 304 nodes has 8 nodes for each transit domain on average and 6 nodes for each stub domain. Similarly the graphs with 400, 496 and 640 nodes have 8 nodes for each transit domain on average and 8, 10 and 13 nodes for each of the stub domains. It is evident that the new algorithm always generates better results. In the case of 2 transit domains the number of nodes has only a small effect on the number of rounds used in the gossiping procedure.

5.3.2.3 Tiers

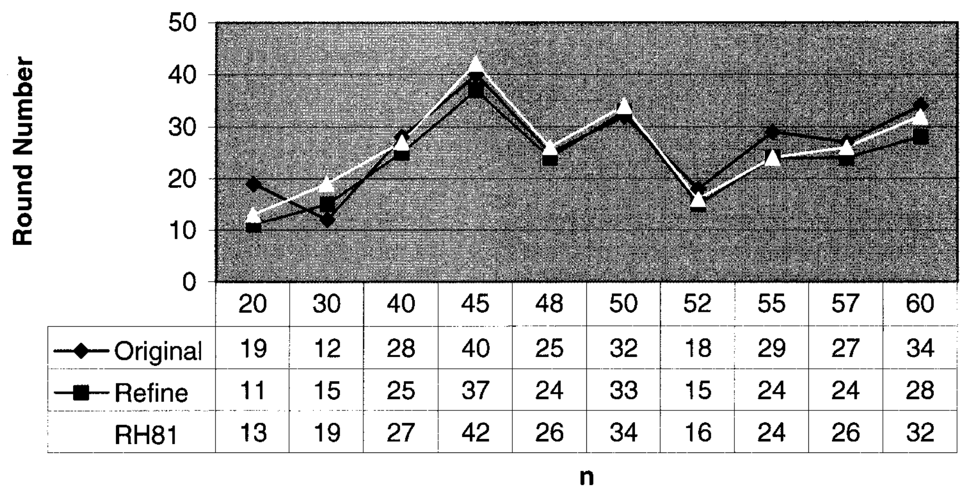
For Tiers graphs, we will compare three kinds of graphs based on the number of levels each has.

The parameters and their meanings are as follows:

- NW: maximum number of WANs (currently only 1 supported, it is not shown in the tables)
- NM: maximum number of MANs per WAN
- NL: maximum number of LANs per MAN
- SW: maximum number of nodes per WAN
- SM: maximum number of nodes per MAN
- SL: maximum number of nodes per LAN
- RW: intranetwork redundancy for WAN (default 1)
- RM: intranetwork redundancy for MANs (default 1)
- RL: intranetwork redundancy for LANs (currently only 1 supported, it is not shown in the tables)
- RMW: internetwork redundancy for MAN to WAN (default 1)
- RLM: internetwork redundancy for LAN to MAN (default 1)

NM	NL	SW	SM	SL	RW	RM	RMW	RLM	n	Edges	Original	Refine	P	HR81
0	0	20	0	0	3	1	1	1	20	27	19	11	8	13
0	0	30	0	0	3	1	1	1	30	38	12	15	3	19
0	0	40	0	0	3	1	1	1	40	51	28	25	2	27
0	0	45	0	0	1	1	1	1	45	44	40	37	2	42
0	0	48	0	0	5	1	1	1	48	87	25	24	0	26
0	0	50	0	0	2	1	1	1	50	53	32	33	1	34
0	0	52	0	0	7	1	1	1	52	109	18	15	3	16
0	0	55	0	0	4	1	1	1	55	73	29	24	2	24
0	0	57	0	0	5	1	1	1	57	80	27	24	4	26
0	0	60	0	0	3	1	1	1	60	75	34	28	2	32

Table 16: Testing Result of Tiers with WAN for Gossiping



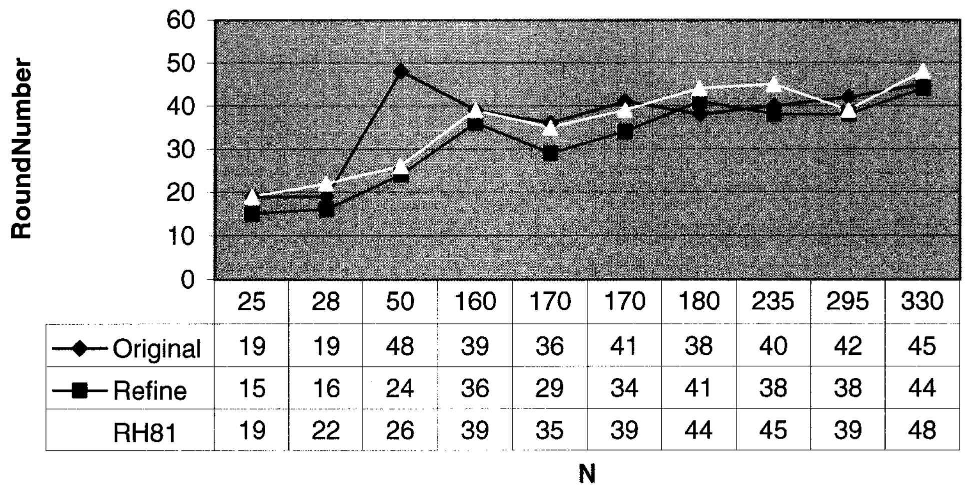
n	20	30	40	45	48	50	52	55	57	60
◆ Original	19	12	28	40	25	32	18	29	27	34
■ Refine	11	15	25	37	24	33	15	24	24	28
RH81	13	19	27	42	26	34	16	24	26	32

Figure 42: Testing Result of Tiers with WAN for Gossiping

Table 16 and Figure 42 show the test results from graphs that have a WAN with some redundancy. The number of nodes in the graphs changes according to the difference of the maximum number of nodes per WAN and intranet work redundancy for WAN. The new algorithm always has a strong result. However, the number of rounds used for the gossiping process also changes due to the number of their edges

NM	NL	SW	SM	SL	RW	RM	RMW	RLM	n	Edges	Original	Refine	P	RH81
1	0	20	5	0	3	2	1	1	25	32	19	15	6	19
2	0	20	4	0	3	1	1	1	28	33	19	16	1	22
3	0	20	10	0	3	2	1	1	50	121	48	24	2	26
30	0	10	5	0	3	2	1	1	160	164	39	36	3	39
30	0	20	5	0	3	2	1	1	170	174	36	29	3	35
30	0	20	5	0	3	2	1	1	170	174	41	34	0	39
30	0	30	5	0	3	2	1	1	180	190	38	41	1	44
30	0	25	7	0	3	2	1	1	235	239	40	38	1	45
30	0	25	9	0	3	2	1	1	295	306	42	38	0	39
30	0	30	10	0	3	2	1	1	330	343	45	44	2	48

Table 17: Testing Result of Tiers with WAN and MAN for Gossiping



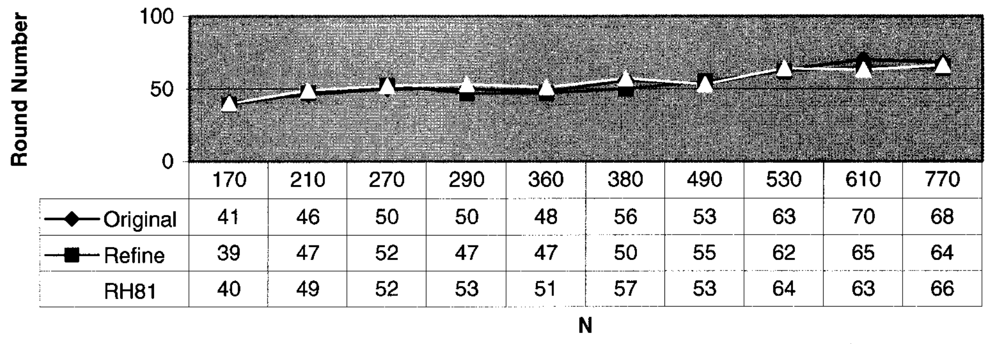
N	25	28	50	160	170	170	180	235	295	330
—◆— Original	19	19	48	39	36	41	38	40	42	45
—■— Refine	15	16	24	36	29	34	41	38	38	44
RH81	19	22	26	39	35	39	44	45	39	48

Figure 43: Testing Result of Tiers with WANs and MANs for Gossiping

From Table 17 and Figure 43, one can see the test result from graphs that have a WAN with some MANs. The number of nodes of graphs changes according to maximum number of WANs, the difference of the maximum number of nodes per WAN and maximum number of nodes per MAN. The new refined algorithm always has a good result. However the number of rounds used for the gossiping process increases based on the number of nodes in the graphs.

NM	NL	SW	SM	SL	RW	RM	RMW	RLM	n	Edges	Original	Refine	P	R81
10	2	30	10	2	3	2	1	1	170	178	41	39	2	40
10	2	30	10	4	3	2	1	1	210	220	46	47	1	49
10	2	30	10	7	3	2	1	1	270	280	50	52	0	52
10	4	30	10	4	3	2	1	1	290	298	50	47	0	53
10	5	10	10	5	2	1	2	1	360	373	48	47	2	51
10	5	30	10	5	3	2	1	1	380	394	56	50	2	57
10	9	30	10	4	3	2	1	1	490	495	53	55	1	53
10	8	30	10	8	3	2	1	1	530	544	63	62	1	64
10	9	30	10	4	3	2	1	1	610	618	70	65	0	63
10	8	30	10	8	3	2	1	2	770	863	68	64	2	66

Table 18: Testing Result of Tiers with Three Lays for Gossiping



N	170	210	270	290	360	380	490	530	610	770
◆ Original	41	46	50	50	48	56	53	63	70	68
■ Refine	39	47	52	47	47	50	55	62	65	64
RH81	40	49	52	53	51	57	53	64	63	66

Figure 44: Testing Result of Tiers with Three Layers for Gossiping

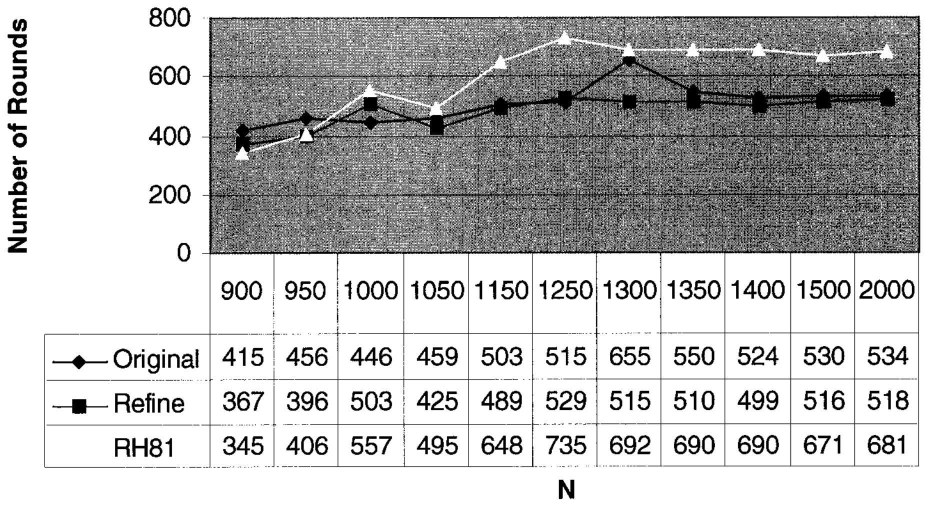
All of the graphs shown in Table 18 and Figure 44 have three-tier-networks with redundancy. These three results are similar.

5.3.2.3 Inet

This type of graph has special properties that contribute to a dramatic increase in the round number used for gossiping and broadcasting. Let us take the graphs with fewer nodes to test this theory.

N	Edges	Original	Refine	P	RH81
900	1094	415	367	8	345
950	1174	456	396	8	406
1000	1252	446	503	7.5	557
1050	1332	459	425	7.5	495
1150	1492	503	489	8	648
1250	1654	515	529	8	735
1300	1735	655	515	3	692
1350	1816	550	510	3	690
1400	1899	524	499	2.5	690
1500	2064	530	516	7.5	671
2000	2914	534	518	8	681

Table 19: Testing Result of Inet for Gossiping



	900	950	1000	1050	1150	1250	1300	1350	1400	1500	2000
◆ Original	415	456	446	459	503	515	655	550	524	530	534
■ Refine	367	396	503	425	489	529	515	510	499	516	518
RH81	345	406	557	495	648	735	692	690	690	671	681

Figure 45: Testing Result of Inet for Gossiping

By observing Table 19 and Figure 45, one can see that in most cases, the new algorithm has stronger results except when the number of nodes is 900. It is also evident that the number of rounds used increases in relation to the increasing of the node number in the graphs.

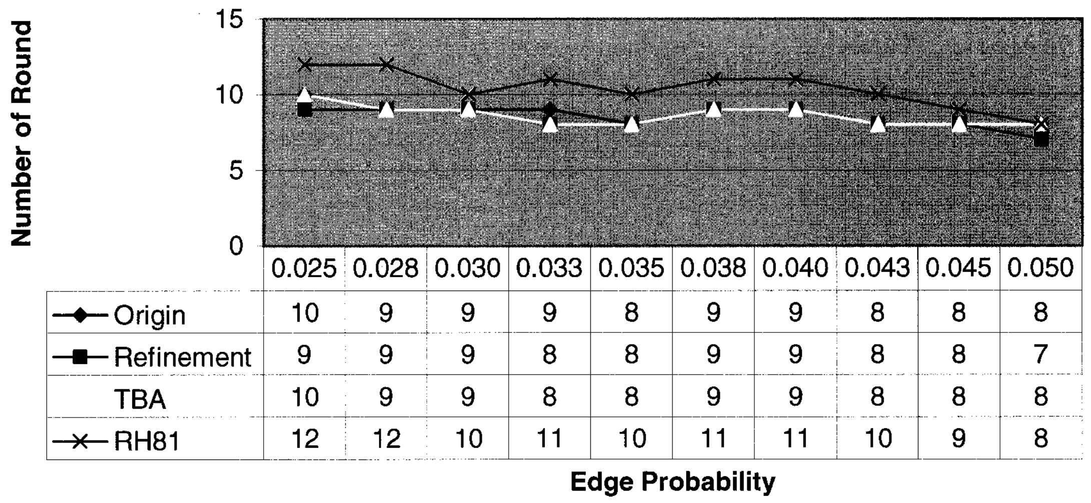
5.3.3 Broadcasting Testing Results

Here are the test results for the broadcasting algorithm [7], the new algorithm of refinement, the new original algorithm, and Round_Heuristic.

5.3.3.1 GT-ITM Random Model

Probability	Edge	Origin	Refinement	P	TBA	RH81
0.025	148	10	9	1	10	12
0.028	145	9	9	1	9	12
0.030	158	9	9	1	9	10
0.033	177	9	8	2	8	11
0.035	187	8	8	1	8	10
0.038	204	9	9	1	9	11
0.040	211	9	9	1	9	11
0.043	213	8	8	3.5	8	10
0.045	225	8	8	1	8	9
0.050	235	8	7	7.5	8	8

Table 20: Testing Result of GT-ITM Random with 100 Nodes for Broadcasting



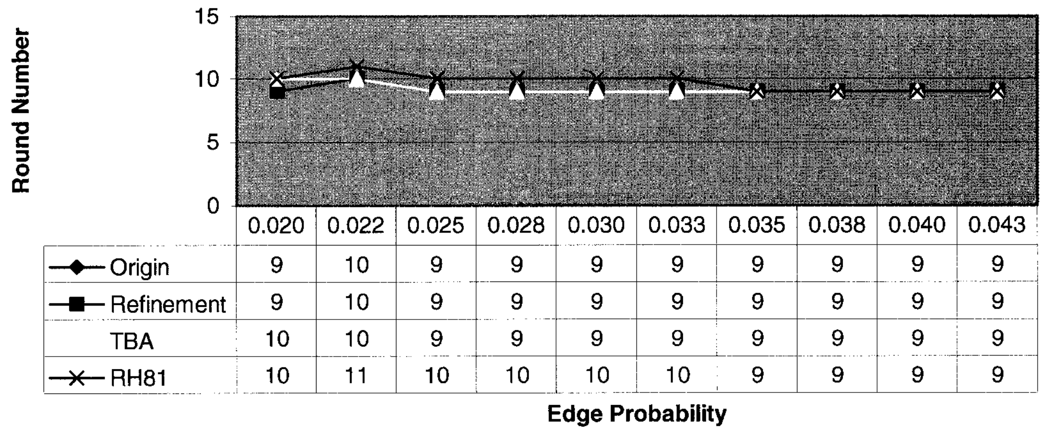
	0.025	0.028	0.030	0.033	0.035	0.038	0.040	0.043	0.045	0.050
◆ Origin	10	9	9	9	8	9	9	8	8	8
■ Refinement	9	9	9	8	8	9	9	8	8	7
TBA	10	9	9	8	8	9	9	8	8	8
× RH81	12	12	10	11	10	11	11	10	9	8

Figure 46: Testing Result of GT-ITM Random with 100 Nodes for Broadcasting

By observing Table 20 and Figure 46, one can see the test result for the graph with 100 nodes for GT-ITM pure random model. The broadcasting time obtained by using the new algorithm is always less than that of the Round_Heuristic, and the difference between the original and the refinement is not substantial. However, for this group of graph models, there is one case in which the result is greater than TBA when they have the same P value. From Figure 46 one can see that when the probability increases from 0.025 to 0.050, there is no significant change in the broadcast time.

Probability	Edge	Origin	Refinement	P	TBA	RH81
0.020	926	9	9	0	10	10
0.022	943	10	10	1.5	10	11
0.025	1125	9	9	1.5	9	10
0.028	1267	9	9	0	9	10
0.030	1350	9	9	0	9	10
0.033	1475	9	9	0	9	10
0.035	1569	9	9	0	9	9
0.038	1698	9	9	0	9	9
0.040	1788	9	9	0	9	9
0.043	1989	9	9	0	9	9

Table 21: Testing Result of GT-ITM Random with 300 Nodes for Broadcasting



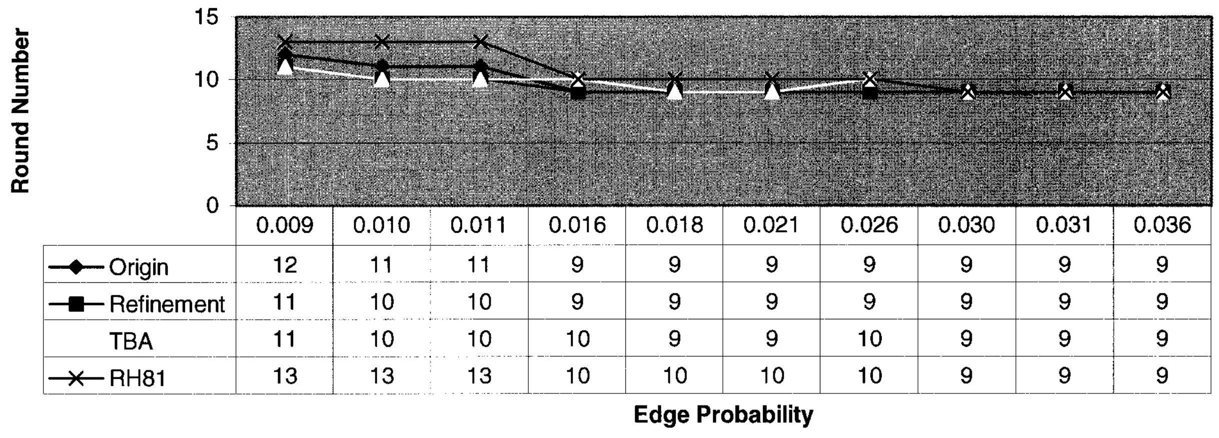
	0.020	0.022	0.025	0.028	0.030	0.033	0.035	0.038	0.040	0.043
◆ Origin	9	10	9	9	9	9	9	9	9	9
■ Refinement	9	10	9	9	9	9	9	9	9	9
TBA	10	10	9	9	9	9	9	9	9	9
× RH81	10	11	10	10	10	10	9	9	9	9

Figure 47: Testing Result of GT-ITM Random with 300 Nodes for Broadcasting

Table 21and Figure 47 show the test results for graphs with 300 nodes. One can see that both the original and the refinement algorithm have the same broadcast time. When the probability is less than 0.035 they have better results than Round_Heuristic, and all of them tend to have the same value. Again with the probability of 0.020, the new algorithm's result is better than TBA.

Probability	Edge	Origin	Refinement	P	TBA	RH81
0.009	779	12	11	1.5	11	13
0.010	866	11	10	2	10	13
0.011	902	11	10	1.5	10	13
0.016	1319	9	9	0.5	10	10
0.018	1405	9	9	2	9	10
0.021	1645	9	9	2	9	10
0.026	2049	9	9	0	10	10
0.030	2389	9	9	0	9	9
0.031	2488	9	9	0	9	9
0.036	2881	9	9	0	9	9

Table 22: Testing Result of GT-ITM Random with 400 Nodes for Broadcasting



	0.009	0.010	0.011	0.016	0.018	0.021	0.026	0.030	0.031	0.036
—◆— Origin	12	11	11	9	9	9	9	9	9	9
—■— Refinement	11	10	10	9	9	9	9	9	9	9
TBA	11	10	10	10	9	9	10	9	9	9
—×— RH81	13	13	13	10	10	10	10	9	9	9

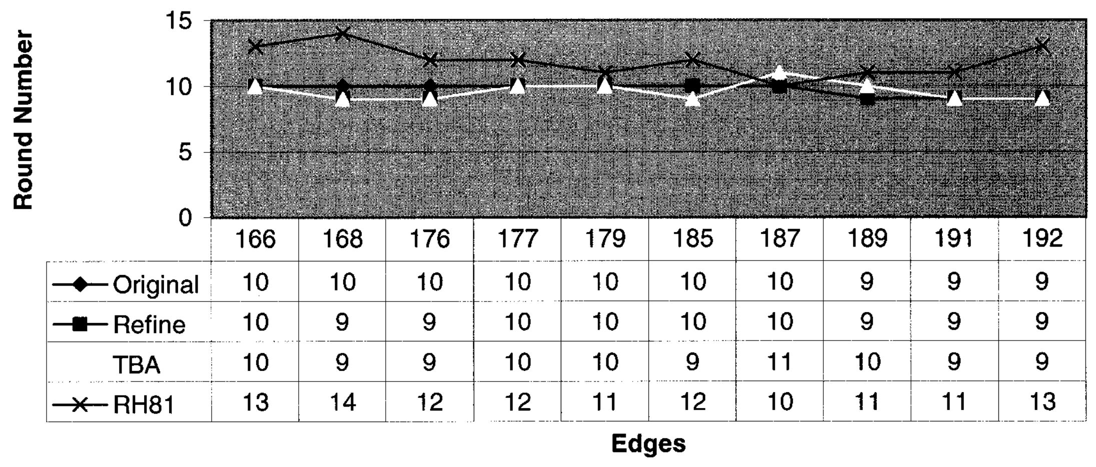
Figure 48: Testing Result of GT-ITM Random with 400 Nodes for Broadcasting

In Table 22 and Figure 48, one can see the broadcast test results for graphs with 400 nodes generated by the GI-ITM pure random model. When the probability increases, all of these three algorithms tend to finish broadcasting in fewer rounds. At the beginning, the refined algorithm has a better result, but eventually both have identical results. In two cases, the new algorithm has a better result than TBA: when Probability = 0.016 and Probability = 0.026, and when they have the same P value. In the end, all of them tend to generate same value.

5.3.3.2 GT-ITM Transit-Stub Model

Edges	Original	Refine	P	TBA	RH81
166	10	10	1	10	13
168	10	9	1.5	9	14
176	10	9	1	9	12
177	10	10	1	10	12
179	10	10	1	10	11
185	10	10	1	9	12
187	10	10	0.5	11	10
189	9	9	1	10	11
191	9	9	1	9	11
192	9	9	1	9	13

Table 23: Testing Result of GT-ITM Transit Stub with 100 Nodes for Broadcasting



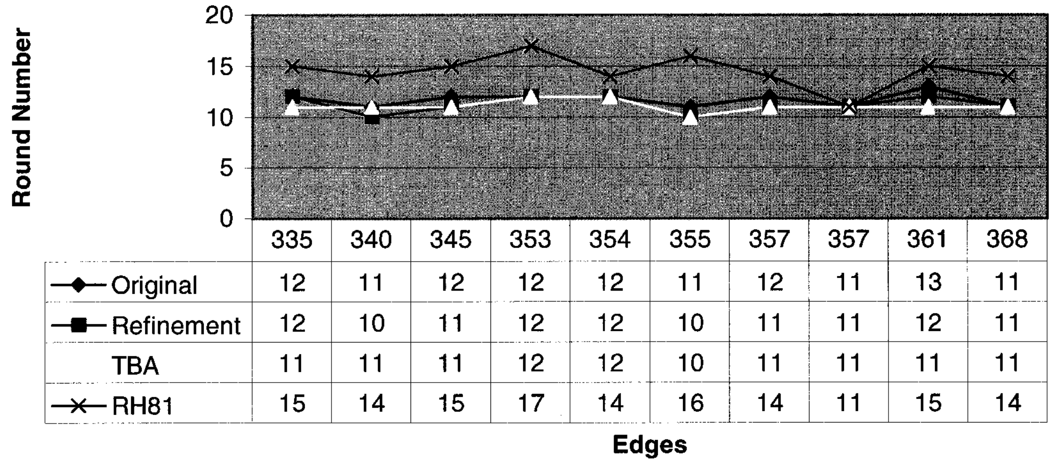
	166	168	176	177	179	185	187	189	191	192
◆ Original	10	10	10	10	10	10	10	9	9	9
■ Refine	10	9	9	10	10	10	10	9	9	9
TBA	10	9	9	10	10	9	11	10	9	9
× RH81	13	14	12	12	11	12	10	11	11	13

Figure 49: Testing Result of GT-ITM Transit Stub with 100 Nodes for Broadcasting

Table 23 and Figure 49 use the same graph as seen in Table 13 and Figure 39. In these graphs, the new algorithm always generates good results (both the original and refinement). In three cases, the new algorithm has different values than TBA and in two cases, the new algorithm has better results. When there are 187 edges, all of these three algorithms generate identical results.

Edges	Original	Refinement	P	TBA	RH81
335	12	12	1	11	15
340	11	10	1	11	14
345	12	11	1.5	11	15
353	12	12	1	12	17
354	12	12	0.5	12	14
355	11	10	1	10	16
357	12	11	2	11	14
357	11	11	1	11	11
361	13	12	1	11	15
368	11	11	1.5	11	14

Table 24: Testing Result of GT-ITM Transit Stub with 200 Nodes for Broadcasting



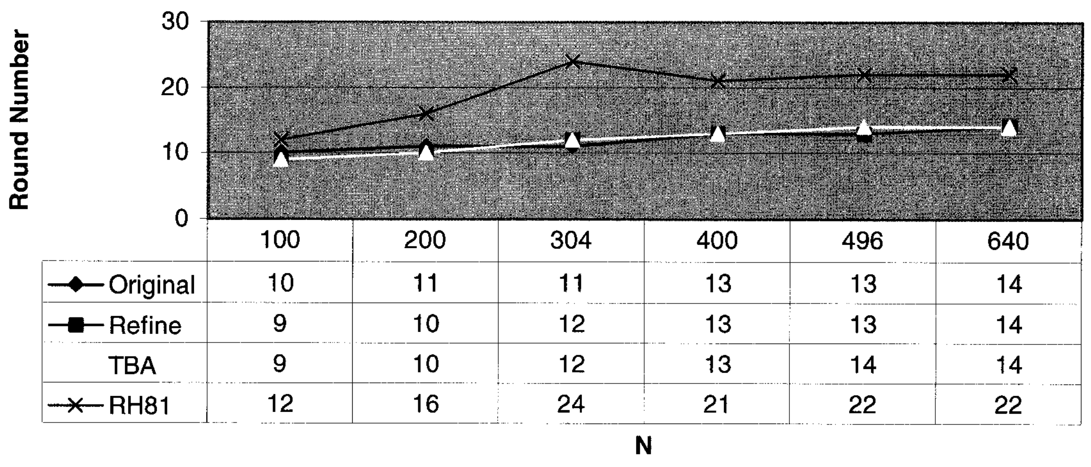
	335	340	345	353	354	355	357	357	361	368
Original	12	11	12	12	12	11	12	11	13	11
Refinement	12	10	11	12	12	10	11	11	12	11
TBA	11	11	11	12	12	10	11	11	11	11
RH81	15	14	15	17	14	16	14	11	15	14

Figure 50: Testing Result of GT-ITM Transit Stub with 200 Nodes for Broadcasting

The results of Table 24 and Figure 50 are based on the same graphs as the results of Table 14 and Figure 40. For these graphs, the new algorithm always generates good results regardless of whether it is the original or a refinement. When there are 357 edges, all of these three algorithms generate the same result. Compared with TBA, the new algorithm has two cases that generate the worse result and one case that generates a better result.

N	Edges	Original	Refine	P	TBA	RH81
100	176	10	9	1	9	12
200	355	11	10	1	10	16
304	453	11	12	1	12	24
400	368	13	13	1.5	13	21
496	1029	13	13	1	14	22
640	1696	14	14	2	14	22

Table 25: Testing Result of GT-ITM Transit Stub with Various Nodes for Broadcasting



	100	200	304	400	496	640
Original	10	11	11	13	13	14
Refine	9	10	12	13	13	14
TBA	9	10	12	13	14	14
RH81	12	16	24	21	22	22

Figure 51: Testing Result of GT-ITM Transit Stub with Various Nodes for Broadcasting

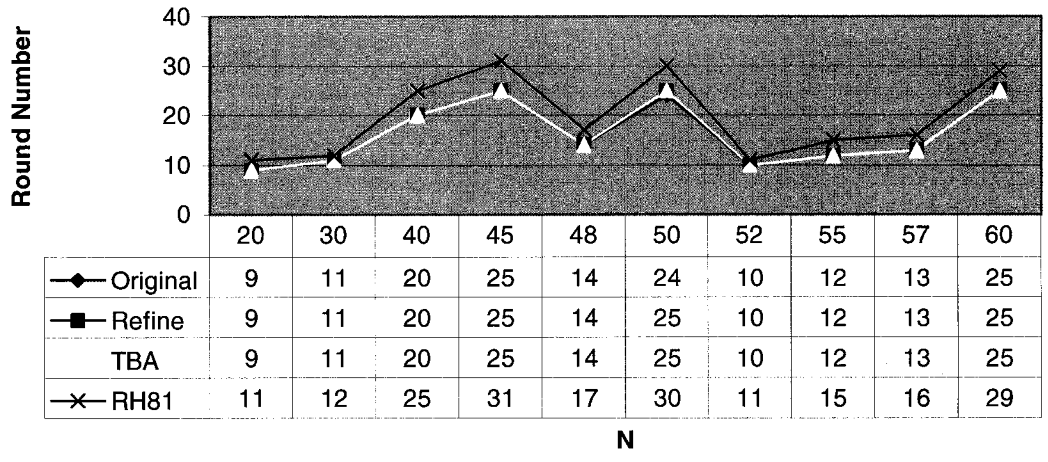
We can see the test result for broadcasting with different nodes from Table 25 and Figure 51. They use the same graphs with Table 15 and Figure 41. For broadcast, the new algorithm always generates far better results. When the number of nodes is 304, the original has better results than the refinement. For this group of graph models, there is one case in which TBA's result is worse. This occurred when N = 496.

5.3.3.3 Tiers

In this section, the same graphs have been employed as those seen in section 5.3.2.3.

NM	NL	SW	SM	SL	RW	RM	RMW	RLM	n	Edges	Original	Refine	P	TBA	RH81
0	0	20	0	0	3	1	1	1	20	27	9	9	1	9	11
0	0	30	0	0	3	1	1	1	30	38	11	11	1	11	12
0	0	40	0	0	3	1	1	1	40	51	20	20	1	20	25
0	0	45	0	0	1	1	1	1	45	44	25	25	1	25	31
0	0	48	0	0	5	1	1	1	48	87	14	14	1	14	17
0	0	50	0	0	2	1	1	1	50	53	24	25	1	25	30
0	0	52	0	0	7	1	1	1	52	109	10	10	1	10	11
0	0	55	0	0	4	1	1	1	55	73	12	12	1	12	15
0	0	57	0	0	5	1	1	1	57	80	13	13	1	13	16
0	0	60	0	0	3	1	1	1	60	75	25	25	1	25	29

Table 26: Testing Result of Tiers with WAN for Broadcasting



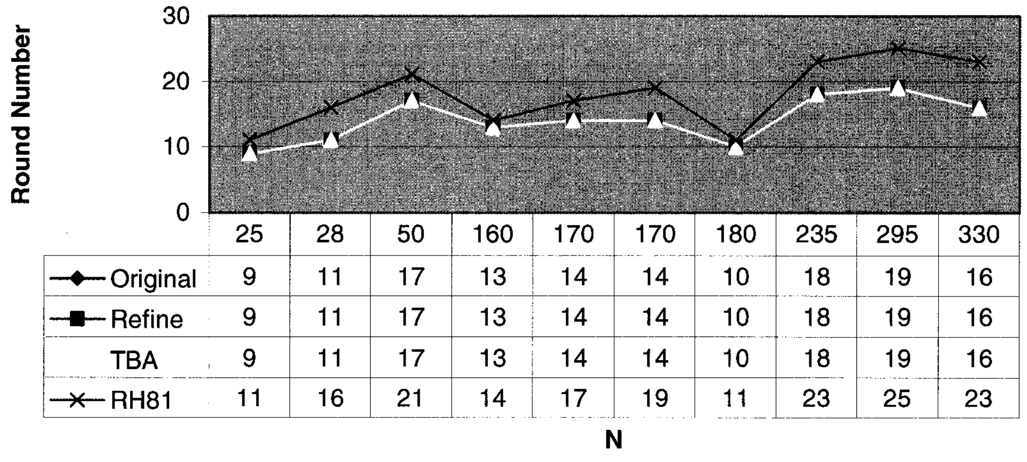
	20	30	40	45	48	50	52	55	57	60
—◆— Original	9	11	20	25	14	24	10	12	13	25
—■— Refine	9	11	20	25	14	25	10	12	13	25
TBA	9	11	20	25	14	25	10	12	13	25
—×— RH81	11	12	25	31	17	30	11	15	16	29

Figure 52: Testing Result of Tiers with WAN for Broadcasting

Table 26 and Figure 52 above have the same graph as Table 16 and Figure 42. It is evident that for the majority of the cases, the new refined algorithm shares the same value with the original one. They also give better results than Round_Heuristics with parameter values of 8 and 1. The number of rounds needed to complete broadcasting changes dramatically. In this group of graphs, the new algorithm always generates the same result as TBA.

NM	NL	SW	SM	SL	RW	RM	RMW	RLM	n	Edges	Original	Refine	P	TBA	RH81
1	0	20	5	0	3	2	1	1	25	32	9	9	1	9	11
2	0	20	4	0	3	1	1	1	28	33	11	11	1	11	16
3	0	20	10	0	3	2	1	1	50	121	17	17	1	17	21
30	0	10	5	0	3	2	1	1	160	164	13	13	2	13	14
30	0	20	5	0	3	2	1	1	170	174	14	14	1	14	17
30	0	20	5	0	3	2	1	1	170	174	14	14	2	14	19
30	0	30	5	0	3	2	1	1	180	190	10	10	1	10	11
30	0	25	7	0	3	2	1	1	235	239	18	18	1	18	23
30	0	25	9	0	3	2	1	1	295	239	19	19	1	19	25
30	0	30	10	0	3	2	1	1	330	343	16	16	1	16	23

Table 27: Testing Result of Tiers with WANs and MANs for Broadcasting



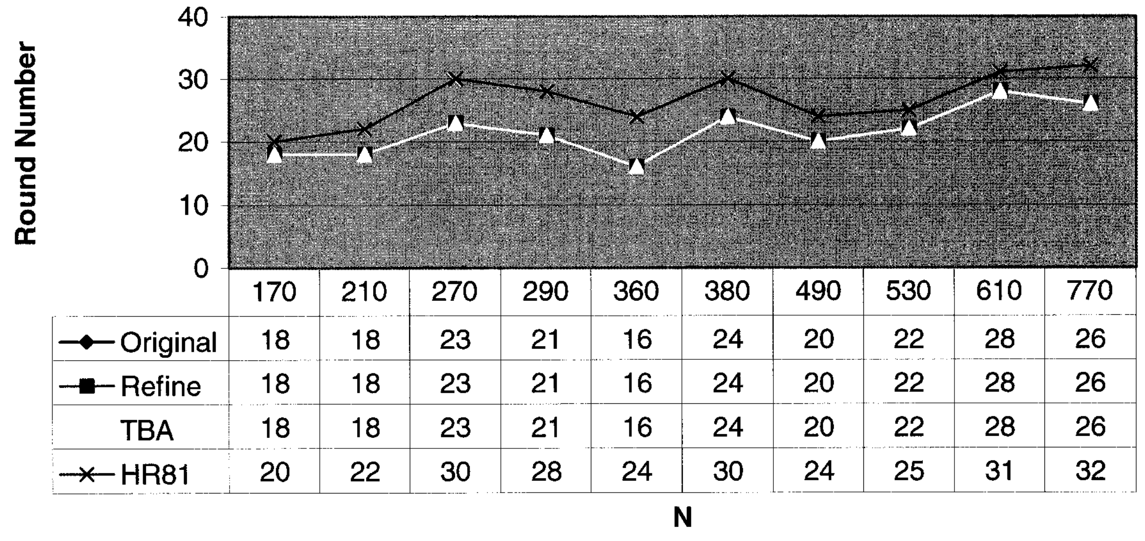
	25	28	50	160	170	170	180	235	295	330
—◆— Original	9	11	17	13	14	14	10	18	19	16
—■— Refine	9	11	17	13	14	14	10	18	19	16
TBA	9	11	17	13	14	14	10	18	19	16
—✕— RH81	11	16	21	14	17	19	11	23	25	23

Figure 53: Testing Result of Tiers with WANs and MANs for Broadcasting

With the same corresponding graphs in Table 17 and Figure 43, Table 27 and Figure 53 show the testing result of broadcasting. The new refined algorithm has exactly the same values as the original. All of the values are better than RH81. Once again, the new algorithm and TBA generates the same results.

NM	NL	SW	SM	SL	RW	RM	RMW	RLM	n	Edges	Original	Refine	P	TBA	RH81
10	2	30	10	2	3	2	1	1	170	178	18	18	1	18	20
10	2	30	10	4	3	2	1	1	210	220	18	18	1	18	22
10	2	30	10	7	3	2	1	1	270	280	23	23	1	23	30
10	4	30	10	4	3	2	1	1	290	298	21	21	1	21	28
10	5	10	10	5	2	1	2	1	360	373	16	16	1	16	24
10	5	30	10	5	3	2	1	1	380	394	24	24	2	24	30
10	9	30	10	4	3	2	1	1	490	495	20	20	1	20	24
10	8	30	10	8	3	2	1	1	530	544	22	22	1	22	25
10	9	30	10	4	3	2	1	1	610	618	28	28	1	28	31
10	8	30	10	8	3	2	1	2	770	863	26	26	2	26	32

Table 28: Testing Result of Tiers with Three Layers for Broadcasting



	170	210	270	290	360	380	490	530	610	770
—◆— Original	18	18	23	21	16	24	20	22	28	26
—■— Refine	18	18	23	21	16	24	20	22	28	26
TBA	18	18	23	21	16	24	20	22	28	26
—✕— HR81	20	22	30	28	24	30	24	25	31	32

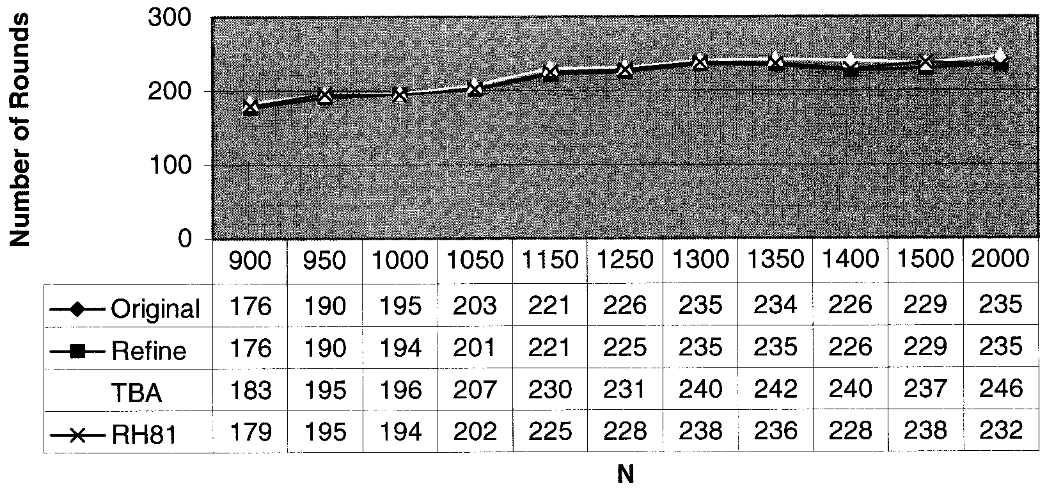
Figure 54: Testing Result of Tiers with Three Layers for Broadcasting

Similar to the two layers Tiers model the test results from three layers shown above have indicated that the new algorithm generates stronger results than RH81, but the same results as TBA.

5.3.3.3 Inet

N	Edges	Original	Refine	P	TBA	RH81
900	1094	176	176	0	183	179
950	1174	190	190	0	195	195
1000	1252	195	194	2.5	196	194
1050	1332	203	201	0	207	202
1150	1492	221	221	0	230	225
1250	1654	226	225	1.5	231	228
1300	1735	235	235	3	240	238
1350	1816	234	235	1	242	236
1400	1899	226	226	1	240	228
1500	2064	229	229	1	237	238
2000	2914	235	235	1	246	232

Table 29: Testing Result of Inet for Broadcasting



	900	950	1000	1050	1150	1250	1300	1350	1400	1500	2000
◆ Original	176	190	195	203	221	226	235	234	226	229	235
■ Refine	176	190	194	201	221	225	235	235	226	229	235
TBA	183	195	196	207	230	231	240	242	240	237	246
× RH81	179	195	194	202	225	228	238	236	228	238	232

Figure 55: Testing Result of Inet for Broadcasting

All the graphs in Table 29 and Figure 55 are the same as those in Table 18 and Figure 44. By observing them one can see that these three results have different values, and one can conclude that the refinement algorithm is better. In all of the cases, the new algorithm generates better results than TBA.

5.4 Summary

Through an analysis of the three sections presented above, one can conclude that the results of the new algorithm. While in terms of gossiping, the new algorithm performs in a similar way to the Round_Heuristic (in several commonly used topologies), it results in a better performance in three NS-2 models.

When it comes to broadcasting, the results are even better. The difference between original and refinement is more obvious for gossiping. For broadcasting in regular graphs, the new algorithm generates the same results as TBA. But in the four new NS-2 models, they are different. In some cases the new algorithm has better results than TBA, But in some cases, TBA's results are better. With the exception of the Inet model, the new algorithm always generates better results when the number of rounds increases dramatically.

As the authors mentioned, the quality of the Round_Heuristic depends heavily on the choice of parameters. There are two parameters in this heuristic. A range from 0.25 to 60 of one parameter is used to test the heuristic. In practice, it is hard to determine the best parameter, and this could degrade the performance of the heuristic. When using the new original heuristic, there is no parameter at all. Even when using refinement, there is only one parameter. Moreover, as tested, the values of the parameter are the integer numbers between 1 and 8. Combined with the low time complexity of the new heuristic, it is easy to try these values in practice.

6. Conclusions and Future Work

The new algorithm offers an alternative way to find a schedule for broadcasting and gossiping in an arbitrary network. The test results show that the algorithm works very well in practice.

My contribution include:

1. Researched on regular models and internet graph models and their generation in order to find the suitable topologies for testing.
2. Designed an algorithm for both gossiping and broadcasting based on the idea from [1] and [7].
3. Implemented the algorithm for both gossiping and broadcasting.
4. Implemented the algorithm of Round_Heuristic for both Gossiping and Broadcasting for test result comparison.
5. Implemented TBA algorithm for comparison with the new algorithm.
6. Evaluated and compared the difference between the new algorithm, TBA and Round_Heuristic.

In the implementation part, the future work includes reducing the redundant data in the implementation to save more space for big graphs and integrating the matching program in the implementation to reduce data I/O time from the hard disk.

References

[1] R. Beier, J. F. Sibeyn, "A Powerful Heuristic for Telephone Gossiping". The 7th International Colloquium on Structural Information & Communication Complexity (SIROCCO 2000), L'Aquila, Italy, 2000, pp.17-36.

[2] K. Calvert, M. Doar and E. W. Zegura. Modeling Internet Topology. IEEE Communications Magazine, June 1997, pp. 160-163.

[3] C. Jin, Q. Chen, and S. Jamin. Inet: Internet Topology Generator. Technical Report Research Report CSE-TR-433-00, University of Michigan at Ann Arbor, 2000.

[4] T. Leighton, "Introduction to Parallel Algorithm and Architecture: Array-Trees-Hypercubes". Morgan-Kaufmann Publishers, San Meteo, California, 1992.

[5] J. Hromkovic, R. Klasing, B. Monien, and R. Peine. Dissemination of Information in Interconnection Networks (Broadcasting and Gossiping). Manuscript, University of Paderborn, Germany, 1993.

[6] S. Hedetniemi, S. Hedetniemi, A. Liestman. A survey of gossiping and broadcasting in communication networks. Networks, 1988, pp. 319-349.

[7] B. Shao, "A Heuristic for Broadcasting in Arbitrary Networks". Master Thesis, Computer Science Department, Concordia University, Spring 2003.

[8] D. W. Krumme, George Cybenko, K. N. Venkataraman, "Gossiping in Minimal Time". SIAM J. Comput. 21(1): 111-139 (1992).

[9] V. E. Mendia and D. Sarkar, Optimal broadcasting on the star graph, IEEE Trans. Parallel Distrib. System. 3, 1992, pp. 389-396.

[10] W. Aiello, F. Chung and L. Lu, Random evolution in massive graphs, Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 510-519.

[11] A. Farley and S. Hedetniemi, Broadcasting in grid graphs. Proc. Eighteenth SE Conf. on Combinatorics, Graph Theory and Computing. Utilitas Mathematica, Winnipeg, 1978, pp. 275-288.

[12] H. A. Harutyunyan and B. Shao, A Heuristic for Broadcasting. IASTED International Conference on Communications and Computer Networks (CCN 2002), Cambridge, USA, 2002, pp. 360-365.

[13] G. Kortsarz and D. Peleg, Approximation Algorithms for Minimum Time Broadcast. SIAM J. Discrete Math. 8, 1995, pp. 401-427.

[14] R. Klasing, B. Monien, R. Peine and E. A. Stohr, Broadcasting in butterfly and deBruijn networks. Discrete Appl. Math. 53, 1994, pp. 183-197

[15] M. B. Doar, A Better Model for Generating Test Networks, IEEE GLOBECOM'96, London, UK, 1996.

[16] A. L. Liestman, J. G. Peters, Broadcast networks of bounded degree, SIAM Journal on Discrete Maths, 1(4) (1988), pp. 531-540.

[17] A. Farley, S. Hedetniemi, S. Mitchell and A. Proskurowski, Minimum broadcast graphs. Discr. Math. 25, 1979, pp. 189-193.

[18] L. H. Khachatrian and H. A. Harutyunyan, Construction of new classes of minimal broadcast networks, Proceedings 3rd International Colloquium on Coding Theory, Armenia, 1990, pp. 69-77.

[19] P. Scheuermann and G. Wu, Heuristic Algorithms for Broadcasting in Point-to-Point Computer Network. IEEE Transactions on Computers. C-33(9), 1984, pp. 804-811.

[20] S. C. Chau and A. L. Liestman, Constructing minimal broadcast networks. J. Comb. Inf. Sys. Sci. 10, 1985, pp. 110-122.

[21] P. Fraigniaud and E. Lazard, Methods and problems of communication in usual networks. Discrete Appl. Math. 53, 1994, pp. 79-133.

[22] Wikipedia, the free encyclopedia. NP (complexity). Available at

http://en.wikipedia.org/wiki/NP-complete. Last visited on Nov 14, 2005.

[23] M. R. Garey and D. S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Complete-ness W.H. Freeman, San Francisco, CA, 1979.

[24] Wikipedia, the free encyclopedia. Matching. Available at http://en.wikipedia.org/wiki/Matching. Last visited on Nov 14, 2005.

[25] H. N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. Journal of the *ACM*, 23(2), April 1976, pp. 221-234.