Multicasting in Delay Tolerant Networks

Anshuman Tripathi
(07CS3024)

under the guidance of

Prof. Arobinda Gupta

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur-721302

CERTIFICATE

This is to certify that this thesis entitled "Multicasting in Delay Tolerant Networks" submitted by **Anshuman Tripathi** (07CS3024), in partial fulfillment for the award of the degree of Bachelor of Technology, is a record of bonafide research work carried out by him under my supervision during the period 2010-2011.

In my opinion this work fulfills the requirements for which it has been submitted. To best of my knowledge, this thesis has not been submitted to any other university or institution for any degree or diploma.

Dr. Arobinda Gupta
Department of Computer Science and Engineering
Indian Institute of technology
Kharagpur.

Acknowledgement

It gives me a great pleasure to thank all the people who have greatly helped me during the course of this project. Foremost, I express deep gratitude and thanks to my supervisor **Dr. Arobinda Gupta** for helping me shape this work to its current form. His invaluable suggestions have introduced me to a very interesting and dynamic field. He has been very much responsible for ensuring that my work progressed smoothly and surely in the right direction.

The co-operation extended by our beloved Faculty Advisor, **Dr. Jayanta Mukopahdyay**, is gratefully acknowledged.

Last but not the least, I am grateful to my parents and all my friends for their constant support, encouragement, and help in many different ways.

Anshuman Tripathi

Contents

1 Introduction **1**
- 1.1 Objectives . . . 1
- 1.2 System Model . . . 2
- 1.3 Scope . . . 2
- 1.4 Organization . . . 3

2 Related Work **4**

3 Proposed Algorithm **7**
- 3.1 Motivation . . . 7
- 3.2 Algorithm . . . 8
  - 3.2.1 Pseudo code . . . 8
  - 3.2.2 Graph Pruning . . . 8
- 3.3 Dynamic knowledge scenarios . . . 10
- 3.4 Version $MTBR_1$ . . . 12
  - 3.4.1 Avalanche effect . . . 12
  - 3.4.2 Dilution of 'Avalanche' effect . . . 13
- 3.5 Version $MTBR_2$ . . . 13

4 Simulation **14**
- 4.1 Algorithm Analysis . . . 14
  - 4.1.1 Graph generation . . . 14
  - 4.1.2 Effects of α and β . . . 18
  - 4.1.3 Centralities and 'Importance' . . . 20
  - 4.1.4 Avalanche Effect . . . 22
- 4.2 Network Simulation . . . 23
  - 4.2.1 Simulation Test Bed . . . 23
  - 4.2.2 Event Queue . . . 24
  - 4.2.3 SLAW traces . . . 24
  - 4.2.4 Statistical Soundness . . . 26
  - 4.2.5 Implementational Overview . . . 26
  - 4.2.6 Redundant message copies . . . 27
  - 4.2.7 Message Delivery Latency . . . 30
  - 4.2.8 Message Overhead Bandwidth . . . 32

5 Conclusions **35**
- 5.1 Deductions . . . 35
- 5.2 Limitations . . . 36
- 5.3 Future Work . . . 36

List of Figures

1.1 thin trees have less redundancy 2

3.1 Number of Edges Vs. Weight 9
3.2 DTBR tree from sampled graph 10
3.3 MTBR tree for DTBR tree in figure 3.2 11

4.1 DTBR tree for random graph 15
4.2 Trees for the $MTBR_1$ protocol for various values of α 16
4.3 Trees for the $MTBR_2$ protocol for various values of β 17
4.4 Selecting Threshold 18
4.5 Illustration of tree modification with 10 targets 19
4.6 Effect on Redundant message copies 19
4.7 Effect on Latency 20
4.8 Correlation of centralities with I_u for $MTBR_1$ 21
4.9 Correlation of centralities with I_u for $MTBR_2$ 21
4.10 Temporal Avalanche effect 22
4.11 Spatial Avalanche effect 23
4.12 SLAW trace average waiting times 25
4.13 Message copies for trace #1 (unstable)($\alpha = 1.0$,$\beta = 2.0$) 28
4.14 Message copies for trace #2 (unstable)($\alpha = 1.0$,$\beta = 2.0$) 28
4.15 Message copies for trace #2 (stable)($\alpha = 1.0$,$\beta = 2.0$) 29
4.16 Message copies($\alpha = 1.0$,$\beta = 2.0$) 29
4.17 Message copies with h 30
4.18 Latency with group size 31
4.19 Latency with α and β 32
4.20 Latency with h 32
4.21 Message header overhead with group size 33
4.22 Message header overhead with α and β 34
4.23 Message header overhead with h 34

List of Tables

3.1 Two special cases of the algorithm studied. 7
3.2 Metrics of a sampled Graph. 9

4.1 SLAW trace #1 24
4.2 SLAW trace #2 25
4.3 Demonstrating Statistical Soundness 26

Abstract

Data communication is challenging in network environments with no instantaneous end-to-end path. Such Networking environments find application in wide areas of interest eg military battle field and disaster recovery, deep-space communication, habitat monitoring, inter-vehicle communication etc. A multicast is the delivery of message or information to a well-defined group of nodes in a network.Many DTN applications need multicast service. For example, in military battlefields, it is vital to quickly and reliably transmit orders from a command center to a group of field commanders. It is also helpful to share information of surrounding environments among different squads of soldiers. Traditional methods of internet multicast, however, can.t be used as such in DTN multicasts. Firstly, it is difficult to maintain the connectivity of a source-rooted multicast tree (or mesh) during the lifetime of a multicast session. Secondly, data transmissions suffer from large end-to-end delays along the tree because of the repeated disruptions caused by periodically broken branches. This calls for separate multicast protocols for DTNs that function on the underlying principle of store-and-forward architecture.

Chapter 1

Introduction

Delay Tolerant Networks are a special class of networks that witness huge churn w.r.t. availability of links because of limited transition ranges and mobile nodes. In such networks there is no guarantee of a instantaneous source to destination path, which makes the problem of routing, broadcasting and multicasting much more complex. The routing protocols use store and forward mechanism to transmit messages from source to destination.

Multicasting is the problem of routing a message from a single source to a well-defined group of receivers (thus broadcast is a special case of muticasting where group of recipients is the entire network). There are many applications that that require an efficient multicast support, for example sending information of the victims of a natural calamity to the group of rescue workers, in battle field sending information about the terrain or enemy posts in a particular area to the platoons deploys there or even in the case of advertising where application developer may want that the ad is delivered to only the registered group of clients. The problem of multicasting has been studied in detail for internet and ad-hoc networks [2, 3, 5, 4, 7]. In the present work we study the protocols for multicasting in Delay Tolerant Networks, and propose a novel tree based multicast algorithm to reduce the network congestion by reducing the number of redundant copies in the network over other tree based multicast protocols.

1.1 Objectives

The main objective of this work is to investigate a novel multicast algorithm based on tree based routing protocols to minimize the number of redundant message copies. There might be many scenarios that demand the multicast algorithm to involve less message copies as opposed to quick delivery of the message. A subclass of the multicast protocols under Tree-Based-Routing (TBR) protocols. These protocols include STBR, DTBR, OS-multicast. All these protocols require a node to construct a predicted multicast-tree of routes to the destinations. However all the protocols use minimum edge weighed path from the node to the destinations for constructing the tree. The weight of an edge is model specific eg avg waiting time of contact, probability of existence of edge etc. However in many cases the minimum edge weighed path can be sacrificed to get a deep-thin tree instead of a shallow-wide tree. This can reduce the number of redundant message copies at the cost of multicast-tree not being minimum weighed multicast tree.

In the diagram 1.1 are shown two multicast trees for 16 destination nodes. The one on left is a shallow tree while the other one is a deeper tree. Note that number of redundant copies in shallow tree is 15 for a depth of 4 while for deeper tree is 10 for a depth of 7. Thus number of redundant message copies can be reduced but creating thinner trees.

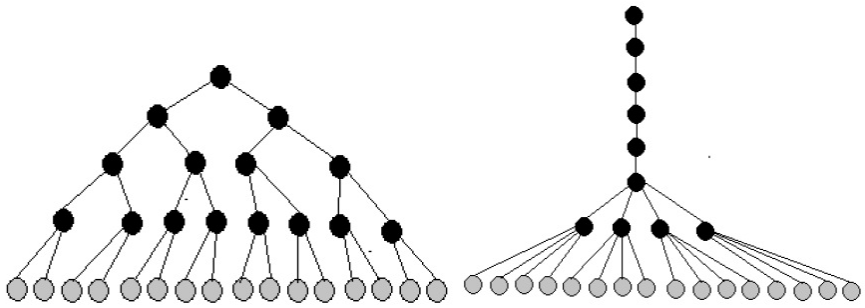


Figure 1.1: thin trees have less redundancy

1.2 System Model

The model of Delay tolerant network nodes considered here assumes that each node has a wireless network interface with some range limitations. A node is A said to come in contact with another node B if the node B comes within the radio range of A. The model assumed in the paper also assumes that the nodes can detect if they have come in contact with another node and also sense it when the contact breaks (i.e. when the node B is no longer in the radio range of A). The inter contact time of the contact $TC_{A->B}$ is defined as the time interval between the event of contact happening an the event of contact breaking. The amount of time A has to weight for a contact to happen with B is the waiting time $TA_{A->B}$. The average of $TA_{A->B}$ over all the contacts $A- > B$ is the average waiting time. The model assumes that the network interface of the nodes have a *constant bit rate* (cbr channel) and for the message to successfully being transfered the sender must be in contact with the receiver for the entire message delivery time. The channel is assumed to be fault free, in that the messages transferred will arrive at the destination in sane form. It is also assumed that the node can transfer messages only if there is no message already in transit (the bandwidth of the network interface can't be shared). The model also assumes that nodes have an instantaneous update of the maintained graph, irrespective of where the event occurred. This assumption is physically not feasible, however presents a simplified picture of the problem statement. Although the proposed algorithm doesn't explicitly require these conditions, these model specifications shall be useful in simulation results and reasoning.

1.3 Scope

The scope of present work is limited to the introduction and analysis of the new algorithm proposed w.r.t the DTBR protocol as previous papers have concluded DTBR to be better than STBR (in terms of network latency). The simulations for protocols is done on SLAW mobility model using a custom designed simulator in java based on the System model discussed earlier. Since throughout the simulations it has been assumed that the nodes have a global knowledge of the contact graph, the OS-multicast protocol reduces to a simple case of DTBR protocol (in OS-multicast a node changes the tree passed to it base on the local knowledge, which in this case is the complete knowledge). The simulation analysis has been done on the following performance metrics:-

Message copies : This is related to the main motivation and objective of the present work, it is the number of copies of the message spawned in the system before final termination of the protocol(i.e. every message copy is system is with its recipient).

Average delivery latency : This is the amount of time each recipient has to wait before it receives the message after the beginning of a multicast. The protocol presented here tries

to minimize the number of message copies by compromising the average latency of the system.

Message Overhead traffic :This is the total size of overhead in the message header that is passed throughout the interval of a single message multicast. Thus the traffic is computed per message multicast, in which the overheads of all message transitions are added.

1.4 Organization

The thesis has been organized in form of chapters, beginning at some preliminary introduction (chapter 1) and literature review (chapter 2) followed by chapter 3 that describes the motivation and theory behind the new algorithm and presents the pseudo code for the Modified protocol. This is followed by a detailed simulation analysis in chapter 4 that presents both algorithm analysis and results of network simulation (along with small note on the implementational aspect). Finally the thesis concludes with chapter 5 stating the deductions and some future work possible.

Chapter 2

Related Work

A significant amount of work has been done in literature for handling multicasting in DTNs. Many protocols and algorithms have been proposed for the purpose of multicasting, these protocols differ in the amount of global state knowledge required by them. For example there are some trivial algorithms like epidemic routing, relay cast routing [8], direct delivery ...etc that don't depend on any global knowledge. Whereas there is another class of algorithms like tree based algorithms :- Dynamic tree based routing (DTBR), Static Tree Based Routing (STBR), On-demand Situation-aware Protocol (OSP) [10] ...etc that compute speculative multicast states based on some weak knowledge of the global state.

There have been detailed studies of comparisons for many protocols in the literature [1, 11]. The work presented in [9] also goes on to enumerate several criteria based on which a multicasting algorithm may be chosen over another given the context of the situation. Here is a description of some of the important protocols found in literature:-

Static tree based routing (STBR) : In this routing protocol a tree rooted at the source is computed by the source from the knowledge base. The tree contains the shortest distance paths from the source to the recipients. The source then transmits the message along these paths. Each node in the path duplicates the message. If a previously calculated link is unavailable then the node holds the message until that link becomes available even if an alternative path exists to the receiver. In situations of low mobility where the structure of the tree computed doesn.t change much, this protocol gives a good message delivery rate with very less number of message forwarding. Receivers get a single copy of the message. Generally the delay for sending the message to the receiver(s) is quite high, and in this delay the network topology may change such that a previously calculated path in the tree may no longer exist. The node that has the copy of the message will then wait until the link becomes available even if an alternative or may be even better path is available. This may lead to prolonged delays for message transmission. Computing a good static tree at the source requires a good knowledge base with the source.

Dynamic Tree Based Routing (DTBR) :This technique is very similar to the STBR discussed above. A rooted tree is computed from the source to the destinations using the available knowledge base. The message is then forwarded to the next neighbors in the tree. However the recipient list for the neighbors now contains only the receivers that exist in the sub-tree rooted at that node. The node then re-computes a tree rooted at that node to the subset of receivers in the message header. This protocol partially overcomes the shortcoming of the STBR protocol that if a better path is available to the destination then the node that has the copy of the message can identify such a path. This may lead to reduced delays as compared to the static tree approach. The message header is although heavier than the Static tree counterpart but as the message copies descent down the multicast tree the header contains only the relevant receivers for that portion of the tree. If a

node acts as a central node during the multicast then the node will get only a single copy of the message. There is a complication as we go further down the tree the receivers list in the message header gets more pin pointed. Thus if at a particular node low down the multicast tree another receiver comes in contact with the node that belongs to a different portion of the tree OR if a better route becomes available to a receiver in another part of the tree then that route will not be considered. This may lead to longer delays than the optimal case of epidemic flooding. Furthermore all the nodes along the path must have a good knowledge (almost complete topological knowledge) of the entire DTN.

On Demand Situation Aware Multicast (OS-multicast) :This protocol is another extension to the above discussed DTBR. In this protocol a tree rooted at the source is computed from the knowledge base of the source. The node then forwards the message to the neighbors along with the entire recipient list and the static graph computed at that node. The child nodes (neighbors that received the message for forwarding) take the static tree attached in the message as a base and modify it according to the local knowledge of links (this knowledge may be limited to a particular number of hops) i.e. if a link is not present (or not expected to be present) the link is removed from the old static graph and the new paths are computed taking into account the presently available links at the node. This results in another dynamic tree for the node. This dynamic tree now becomes the static tree for the next hop of neighbors and the process continues. The protocol overcomes a lot of shortcomings of the earlier protocols. Each node may just have a rough knowledge of the network as compared to DTBR and STBR where a node is required to have the entire knowledge. If the Static graph computed initially is good enough then alternating routes can be found and used. Since the message header contains all the recipients, an alternating path to a receiver that was earlier expected to be in another part of the tree may also be considered. A mentioned the message header should contain not only the entire recipient list but also the static graph computed until now. Thus the message transfer overhead is very high. In this protocol a receiver may get multiple copies of the message.

Core Aided Routing : In this approach the fact that DTNs are heterogeneous is exploited. There are generally some nodes that are more powerful (in terms of buffer space and computations) than other nodes. These nodes are used as dedicated routers for the messages. There can be small changes in this protocol where the sender gives a copy to the core node whenever it comes in contact with one of them. The core node will then relay the message to the receiver when it comes in contact with it. Thus this becomes a 2-hop multicasting.Since dedicated nodes are used for the purpose of routing and multicasting, the message overhead of recipients can be overcome by letting only the core nodes maintain the group information. Any node that joins a group shall inform at least one of the core nodes. This works because the routing is 2-Hop.This method also involves very low network traffic. The buffer utilization of the non-Core nodes is very low thus they can have very low buffer capacities. Since the routing is handle by a small subset of the nodes, these nodes can collect the entire network information (e.g. contact information, neighbor topology etc.) for an efficient routing algorithm.

Spray-and-wait protocol :In this protocol a fixed number L. is chosen and initially L copies of message are forwarded in the network. When a node forwards a message copy, it deletes its copy of the message. His forwarding continues for L. hops. This stage is called Spray and Wait. In This stage the entire network contains L. copies of the message (unless the TTL has expired). If all the recipients receive the message within these forwards, then the algorithm terminates otherwise each of the nodes that are carrying a copy of the message performs a direct transmission to the receivers. The method is scalable in the Spraying stage since the network contains no more than L message copies. The method may give high chances of message delivery in low mobility situations. The method can fail terribly

in situations of mobility in a predefined pattern. Since the forwarded message copies are deleted from the source, the node may not contain the message when it comes in contact with a recipient. The protocol is not scalable if all the recipients don.t get the message in the spray stage of the protocol.

In the present work we propose a novel tree based multicast protocol that tries to optimize upon the number of redundant message by compromising the message delivery latency.

Chapter 3

Proposed Algorithm

3.1 Motivation

In this we propose the algorithm for computing multicast trees, to reduce the number of redundant message copies. The main motivation behind the algorithm comes from the fact that if a node is selected in the multicast tree then the message is anyways expected to come to that node than why not reuse that node itself for forwarding the message to the targets. For this we introduce the concept a quantity called 'Importance' which defines the relative favourability of a node from other nodes in being selected in the multicast tree. We denote the node 'Importance' (of node n) by I_n and that of an edge by I_e. Originally the multicast trees are constructed based on optimization of some parameter, generally minimizing the cost of path from source to destination. It can be mathematically defined as:-

$$minimize \quad W_P = \sum_{e(n_1,n_2)\in P} W_e \tag{3.1}$$

equation 3.1 captures the optimization of path weights, note that weight of an edge is a generic concept and can differ for various implementations of the contact graph. For the case when the required path is minimum waiting time graph, weights can correspond to the average waiting time of contact between various nodes, whereas for the case when the message delivery has to be maximize, $W_e = \frac{1}{p_e}$ is a possible weight where p_e is the probability of existence of the edge. Now to reduce the number of message copies for the underlying tree based algorithm we reformulate equation 3.1 but including the effect of node 'Importances' as follows:-

$$\begin{aligned} minimize \quad & W_P = \sum_{e(n_1,n_2)\in P} F_W(W_e, I_e) \\ where \quad & I_e = F_e(I_{n_1}, I_{n_2}) \end{aligned} \tag{3.2}$$

Note that the motivation of the algorithm requires that:-

$$F_W(W, I) \leq W \quad \forall W \geq 0, I \geq 1 \tag{3.3}$$

This modified version of equation 3.1 includes the Importances of the nodes in the optimization equation. In the present work we shall dealing in detail for two special cases for of the functions $F_e()$, $F_I()$ and $F_W()$ (These two flavors are named $MTBR_1$ and $MTBR_2$ respectively), the choices for respective versions is given in table 3.1.

Table 3.1: Two special cases of the algorithm studied.

Version	$F_I()$	$F_e()$	$F_W()$
$MTBR_1$	$I' = I + \alpha$	$I_e = max(I_{n_1}, I_{n_2})$	$W' = \frac{W}{I_e}$
$MTBR_2$	$I' = \beta$	$I_e = max(I_{n_1}, I_{n_2})$	$W' = \frac{W}{I_e}$

3.2 Algorithm

In this section we present a formal description of the generic algorithm with various tuning parameters and discuss the various aspects of its practical implementation.

3.2.1 Pseudo code

Following is the formal description of the modified tree computation algorithm, reffered as MTBR (**M**odified **T**ree **B**ased **R**outing).

Input :source `s`, destinations `d[]`, message `msg`, destination `seed`, Graph `G`

Parameters : Importance update $F_I()$, Edge importance function $F_E()$, Weight update function $F_W()$, number of iterations $nIter$, α

Output : Multicast tree T, Importances $I[]$

Procedure :
`compute_tree()`

```
1   T = φ
2   for all n ∈ V(G) :
3     I[n] = 1
4   for i = 1 to nIter:
5     sp[] = sortest_path(G, source=s, target=seed)
6     I[] = update_importances(I[], sp, F_I())
7     G = update_graph(G, I[], F_E(), F_W())
8     for node ∈ d[] − seed :
9       sp[] = sortest_path(source=s, target=node)
10      I[] = update_importances(I[], sp, F_I())
11      G = update_graph(G, I[], F_E(), F_W())
12  T = shortest_path_tree(G, source=s, target=d[])
13  return T, I[]
```

Procedure :
`update_importance(I[], sp, `$F_I()$`)`

```
1    for all nodes in sp[]:
2      I[node] = F_I()(I[node], α)
3    return I[]
```

Procedure :
`update_graph(G, I[], `$F_E()$`, , `$F_W()$`)`

```
1    for edge in E(G):
2      n1, n2 = edge
3      edge_importance = F_E(I[n1], I[n2])
4      G.edge[n1, n2]['weight'] =
5        F_W(edge_importance, G.edge[n1, n2]['weight'])
6    return G
```

3.2.2 Graph Pruning

The performance of the algorithm is dependent on the pre-processing done on the graph. If the original graph is given to the algorithm proposed herein then the multicast tree produce is a trivial single hop multicast tree (the multicast algorithm then shall correspond to the case of direct delivery). The graph provide to the algorithm is best passed pruned beforehand. Following motivations provide reasons why a pruned graph is considered with the algorithm.

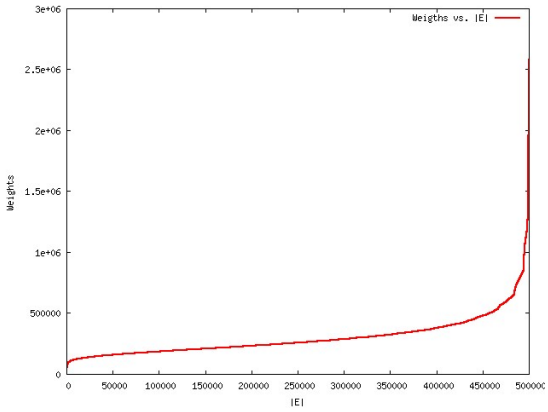


Figure 3.1: Number of Edges Vs. Weight

Metric	Value
Number of nodes $\|V\|$	1000
Number of Edges $\|E\|$	499277
Density $\rho(G)$	0.99
Diameter	2

Table 3.2: Metrics of a sampled Graph.

1. The **size of the graphs** are quite big for sufficient amount of simulation time. For example here is the size metrics of a graph that was sampled from the simulation setup after $\approx 3hrs$ of mobility simulation (SLAW mobility). From the table it is clear that the actual average contact time graph tends to a clique after sufficient amount of simulation. The computational complexity of finding the shortest path is $O(n^2)$ (for dense graphs) but the complexity can be improved for sparse graphs. Thus it makes sense for mobile nodes to maintain a pruned graph for computation of the multicast tree (as it can save the memory and power requirements of the node).

2. There are many **unnecessary edges** in the graph. Figure 3.1 shows the graph of number of edges with weight less than a particular value. It can be seen that there are a number of edges which have very big value of weight. Since the algorithm is primarily based on finding the shortest path from a source to targets, it is most likely that the edges with weights higher than a particular threshold value woll not be used in a shortest path in any iteration of the algorithm. Thus these edges can be safely removed from the graph to be considered in the graph for computation of the multicast tree.In the graph notice that some of the edges have very large weights $\approx 10^6$! which corresponds to an average waiting time of $\approx 10^5(1.15days)$! These edges with large weights are too improbable to appear in an optimum path from source to destination. Furthermore these edges just add to the time and space requirement of the underlying shortest path algorithm. Notice that the curve has an exponential trend that signifies that for pruning a significant number of edges the threshold shall be varied in the range where the graph is flatter.

3. The best possible cases for the complete graph can give rise to **trivial multicast trees** corresponding to the simple routing trees of direct delivery or first contact delivery. As already discussed in the statistics of the graph in table 3.2 the diameter of the graph is 2, and the density of the graph is 0.99 $\approx$ 1 which signifies that for most of the source destination pairs the lenght of path is 1 (or 2 for some rare cases). Thus the multicast tree for the normal shortest path optimization will incur no redundant messages (as the delivery is direclty to the targets). As shown in figure 3.2 the number of 'red' nodes (that

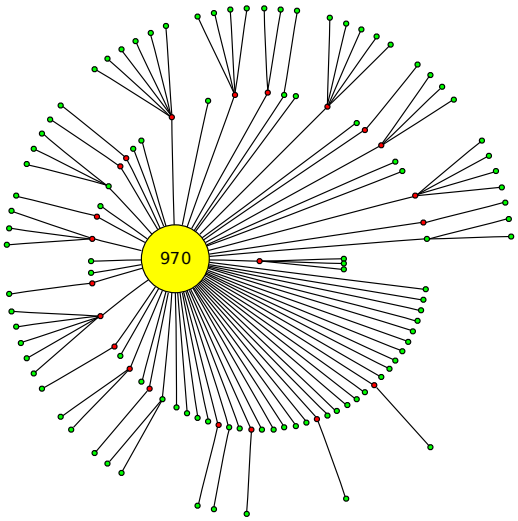


Figure 3.2: DTBR tree from sampled graph

denote redundant nodes) are quite less, thus the margin for imporvment is neglegiable, after applying the $MTBR_1(\alpha = 1.0)$ and $MTBR_2(\beta = 2.0)$ on the tree we get a very trivial direct delivery tree as shown in figure 3.3. .

The procedure is simple where the edges with a weight greater than a particular threshold are removed from the graph for consideration in the `compute_tree()` procedure as described in section 3.2.1.

```
1 prune_graph(G, threshold)
2   for edge in G.edges():
3     if edge.weight > Threshold :
4       G.remove_edge(edge)
5   return G
```

Selecting the `threshold` parameter of the pruning routine defines the structure of the residual graph and thus the structure of the computed multicast trees. A good value of the `threshold` should fullfil the following objectives :-

1. The *number of edges should be reduced to a appreciable amount*, so that the pruning of the graph has some effect on the computational aspect of the algorithm (Note that pruning the graph every time will not improve the complexity of the shortest path algorithm since the procedure `prune_graph` is itself $O(|E|)$. Thus the pruning procedure in practice must be on line and thresholding shall be done for an edge whenever the contact for that edge happens).

2. There should be a big component in the network and the number disconnected components should be very less.

3.3 Dynamic knowledge scenarios

In the cases where global knowledge is unavailable, or ever for the cases when global state of interconnection graph is known there can be some minor modifications in the algorithm that bring about a degree of coordination in the otherwise completely distributes execution of the algorithm by various nodes of the network. In dynamic tree based protocol every node computes

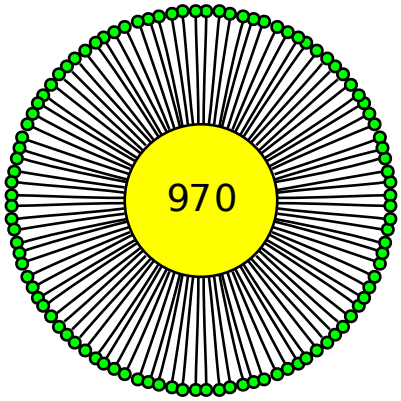


Figure 3.3: MTBR tree for DTBR tree in figure 3.2

a multicast tree of its own. Since the network knowledge may vary from node to node, the nodes that appear with high importance values in local multicast tree of a node may not retain their importance in the neighbors of the node. This may lead to a wide multicast tree despite of using importance values for nodes and deriving a thin local-multicast tree. To counter the above effect we need to add some sort of hysteresis for node importance in the model discussed above. For this we can introduce a hysteresis factor for the nodes in DTN (say h_i for a node i). Now while forwarding a message to a neighbor, the forwarding node includes a part of the local values of for nodes in the local multicast tree, in the message header. The receiving neighbor then initializes the nodes in the local DTN model as:

$$I_u = \begin{cases} (1-h) + h \times I_u^{'} & \text{if } I_u^{'} \text{ is provided} \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

Where I_u' is the importance of the node u as is indicated in the message header and h is the hysteresis factor for the receiving node. The value of h may be fixed for the entire DTN or may be configures with the individual nodes. It is the measure of dynamism in the multicast protocol. For the equations the value of h in in the range $h \in [0, 1]$.Note for that very high values of $h \approx 1$ the protocol will tend to be a static tree based routing protocol. We can say as $(h \to 1) \Rightarrow (I_u \to I_u^{'})$ (high dependence on history) and $(h \to 0) \Rightarrow (I_u \to 1)$ (no evidence of history). For forwarding the importance of a node in the local multicast tree there are message header overheads. To reduce these overheads we may consider following options

1. If the underlying multicasting protocol is similar to 'OS multicast protocol' where the entire local tree is embedded in the message body, then including the importance of the nodes in the tree introduce a very little extra overhead in the message header.

2. We may include pairs $< ID, I_{ID} >$in the message header for a subset of nodes in the local multicast tree which pass through certain constrains, most obvious constrain being that the node is among top K'important' nodes in the local multicast-tree. Note that the value of K can be altered to affect the message header overheads. A constant value of K for the node u introduces an upper bound on the message overhead as $(constant \times max(k_u \forall u))$. This overhead is independent of the size of the multicast tree, thus it does not hinder with the scalability of the underlying multicast protocol. The value of K may be optimally

tuned for a network by considering the reduction in redundant message copies versus the message overheads involved in passing importance values.

3. Passing the importance of nodes with $I_u = 1$ in the final local multicast tree may be avoided.

In the above mentioned protocol it is not only possible to give greater importance to a node based on its 'heredity' but it is also possible to force the receiver node to give lesser importance to a node. This however has no obvious application unless of course the forwarding node has information such as a node being congested or out of order in which case it is practical to exclude the node from any future multicast trees.

3.4 Version $MTBR_1$

This special case of the generic algorithm described in section 3.2.1 corresponds to the choice of functions as given in the table 3.1. The 'Importance' update function of this version of algorithm is $I' = I + \alpha$ where α is a predefined constant that parameterizes the algorithm. Due to the nature of the update equation one can very well reason that the order in which the targets are processed is very important to the properties of the generated tree. If a node is selected for the first time in the multicast path, then with a higher probability it will again be selected in another shortest path and as the number of times it is selected in the shortest path increases its probability of selection in next path also increases. This 'Self amplifying' phenomena is like an 'Avalanche' in that the first destination selected decides to a large extend the nature of tree, thus it shall be called 'Avalanche' effect. The consequence of the avalanche effect can be extrapolated to theoretically predict a tree with large latency. Because of the avalanche effect a node may get excessively high values of 'Importance' and may ruin the performance of the algorithm by introducing a bias.

3.4.1 Avalanche effect

As discussed in the previous section the update function described for $MTBR_1$ can lead to a exponential blowup in the importance of the node. To demonstrate this we assume that the probability $P(u, i)$ that a node u is selected in the i^{th} iteration of the `compute_tree()` routine is directly proportional to the importance $I_u(i-1)$ of the node after the $(i-1)^{th}$ iteration . That is for a node u we can write:-

$$\begin{aligned} P(u,i) &\propto I_u(i-1) \\ \therefore \quad P(u,i) &= C \times I_u(i-1) \end{aligned} \tag{3.5}$$

Thus the Expected value of the importance of the node after the i^{th} iteration can be computed as:-

$$\begin{aligned} E(I_u(i)) &= (1 - P(u,i)) \times I_u(i-1) + P(u,i) \times (I(u,i-1) + \alpha) \\ \therefore \quad &= (1 + C\alpha) \times I(u,i-1) \\ \therefore \quad E(I_u(i)) &= (1 + C\alpha)^{i-i_0} \times I(u,i_0) \\ \therefore \quad E(I_u(i)) &= (1 - C\alpha)^{i_d} \end{aligned} \tag{3.6}$$

where i_0 is the largest integer such that $I(u, i_0) = 1$ and $i_d = i - i_0$.
Thus it is clear form equation 3.6 the expected value of the importance of a node varies exponentially after it has been selected for the first time. Thus the first node is expected to have a very high value of 'Importance' and thus the effect on the multicast tree. Thus selection of seed in the algorithm is very important to the property of the tree.

3.4.2 Dilution of 'Avalanche' effect

There can be several modifications to the algorithm suggested in section 3.2.1 to dilute or counter the avalanche effect, these are:-

1. One may consider forming a partial multicast-tree for say some most versatile destinations (probably based on contact frequencies and hysteresis importance) with the objective function for optimization given by 'Z' as defined in equation 3.1. Note that the tree is constructed without using the values of I_u but these are anyways updated as previously discussed. After constructing this partial tree with chosen destinations we may start using equation 3.2 as the objective function for optimization. This process essentially dilutes the effect of favoring a single node because it was previously favored.

2. Another alternative is to construct parallel multicast trees corresponding to the M most versatile destination nodes as the seeds. Then finally select the tree based on some property say the tree with least delay OR with max delivery probability OR with minimum nodes involved.

3.5 Version $MTBR_2$

Yet another very intuitive way of overcoming the avalanche effect is to consider the α as a function of the iteration index i i.e. $\alpha = \alpha(i)$. for example let us assume a simple case where we want that all the nodes are given equal importance irrespective of the iteration in which they were first chosen (this is a very logical requirement since we should intuitively give equal credit to all the nodes through which a message passes, since selecting the node has already accounted for a duplicate message copy for that node). Thus our requirement can be mathematically stated as:-

$$\begin{aligned} & E(I_u(i)) = \beta \times P(u,i) + I_u(i-1) \times (1 - P(u,i)) \\ \therefore \quad & \beta \times P(u,i) + I_u(i-1) \times (1 - P(u,i)) = I_u(i-1) + \alpha(i) P(u,i) \\ \therefore \quad & \alpha(i) = \beta - I_u(i-1) \end{aligned} \tag{3.7}$$

substituting this value of $\alpha(i)$ in $F_I()$ we get:-

$$I_u(i) = \begin{cases} \beta & \text{if node is selected once} \\ 1 & \text{has n't been selected} \end{cases} \tag{3.8}$$

This is the main motivation behind the second version of the algorithm. I can seen easily from equation 3.8 that the importances now have binary values i.e either 1 or β thus nodes with value β are all important nodes.In case of $MTBR_1$ the nodes at a shallower depth have greater chances of having a high value of 'Importance' than nodes at deeper depths in the multicast tree. However our initial motivation was not to distinguish among important nodes but to distinguish important nodes from not-important nodes. Therefore when a node appears in a shortest path, then the node has become important, thus the value of importance if node is set to a statically higher value . Hence in the case of $MTBR_2$ a node.s importance value can be either 1 (unimportant node) or (important node).

In few words the $MTBR_2$ protocol only aims at reducing the number of redundant nodes on the multicast tree whereas $MTBR_1$ protocol aim to reduce the redundant nodes while considering the depth of the multicast tree (which is not necessary for the problem statement).

Chapter 4

Simulation

In this section we shall present the simulation results obtained for the performance of MTBR algorithm proposed herein. The simulation results has been divided in two parts :-

Algorithm Analysis : In this section we present a detailed analysis of the structure of multicast trees computed using the algorithm versions $MTBR_1$ and $MTBR_2$. The simulation done herein are from the sampled graphs from the same mobility trace used in the next part of the simulation of message transfers.

Network Simulation : In this section we present the results of simulating message multicasts using the proposed algorithm for versions $MTBR_1$ and $MTBR_2$

4.1 Algorithm Analysis

In the following section we investigate the properties of the multicast trees generated from the proposed algorithm. The comparative analysis done herein is for $MTBR_1$ trees, $MTBR_2$ trees and $DTBR_1$ trees. Note that all the other tree based protocols as described in the literature review (chapter 2) use the same underlying algorithm as DTBR protocol for computing the multicast tree (i.e. shortest path tree). As we are only interested in the properties of the multicast tree, considering only the DTBR trees for comparisons suffices.

4.1.1 Graph generation

The algorithm was run on several graphs which were obtained using:

1. Random graph generation
2. Graph Sampling from actual mobility trace
3. Pruning the Sampled Graph

Random Graph Generation

The random graph used in the simulation analysis herein were created using networkx [6] module of python (can be downloaded from `http://networkx.lanl.gov/`). Steps followed to generate a random weighted graph are:-

1. Generate an ErdsRnyi Graph G(n, p) using the networkx module.
2. Iterate through all the edges in the graph G.
3. For each edge assign a weight $W \in [min_w, max_w]$.

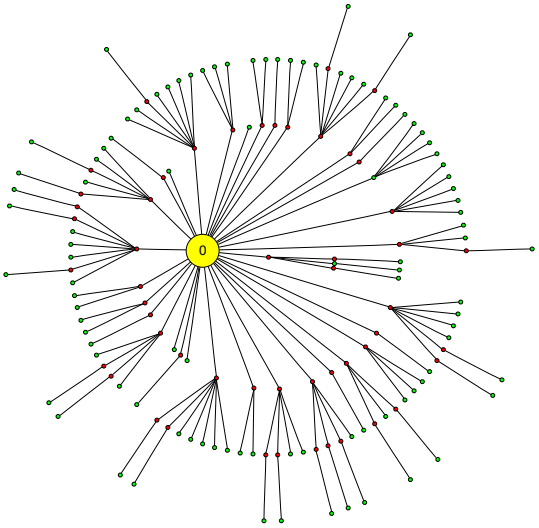
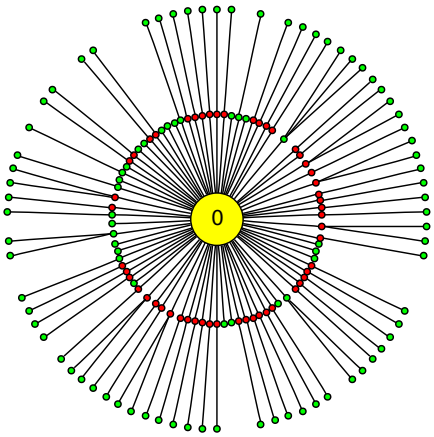


Figure 4.1: DTBR tree for random graph

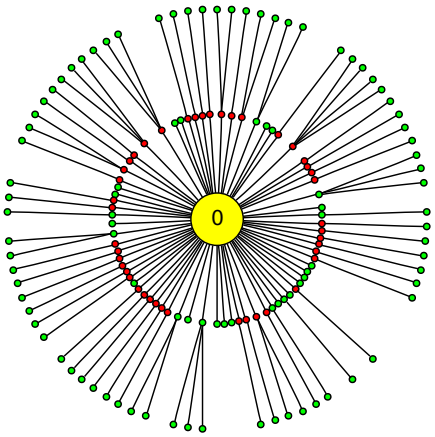
The graphs used in the simulation analysis have 1000 nodes. Here is the working of MTBR1 and MTBR2 protocols with variation of parameters on a sample multicast tree from a random source to 100 random recipients in a random weighted graph of 1000 nodes and probability of edge = 0.3, the edge weights for the graph were randomly chosen from $[100, 1000]$. Figure 4.1 shows as simple shortest path tree[1] simulated on a random graph with 100 targets. The red nodes denote the nodes that are not destination (thus the redundant nodes) and the green nodes represent the destinations. The tree has a diameter of 6 with $|V_{total}| = 152(|V_{red}| = 52)$.

Figure 4.2 shows the structure of $MTBR_1$ trees for various values of α. Now Comparing the $MTBR_1(\alpha = 1.0)$ tree from DTBR tree in figure 4.1 we see that $MTBR_1$ protocol has indeed generated a tree with significantly less number of *red* nodes (only 2 in this case). Note that $MTBR_1(\alpha = 0)$ is same as DTBR tree but introducing a small value of α for $MTBR_1(\alpha = 0.01)$ changes the tree significantly. This can be attributed to the fact that in DTBR the criteria for tree formation is just shortest path, a small value of alpha acts as a *smoothing* to reduce the diameter of the tree. It can be seen that the structure of the generated tree interestingly changes as the value of α is increased. Also note the visual evidence of decrement in number of redundant message copies in the multicast tree as the value of α is increased. (Number of redundant message copies is mathematically equal to the number of non-recipient OR red nodes in the graph). It can be seen in $MTBR_1(\alpha = 0.5)$ and $MTBR_1(\alpha = 1.0)$ trees that some nodes are acting as hubs i.e. they have a huge out degree to the recipients. Figure 4.3 shows the structure of $MTBR_2$ trees for different values of β. Note the reduction in number of non-recipient nodes (the red nodes). The smoothing effect as seen in $MTBR_1(\alpha = 0.01)$ tree is also visible in $MTBR_2(\beta = 1.1)$ tree i.e a small value of β does n't reduce the number of red nodes drastically but immediately reduces the tree depth by 1. The number of redundant message copies do reduce as the value of β is increased (as is visually evident). When compared with $MTBR_1$ protocol the $MTBR_2$ protocol performs equally good w.r.t. the number of redundant message copies, however as expected the diameter of the tree increases for $MTBR_2 protocol$. These illustrations provide theoretical grounds for possibility of improvement in the actual mobility pattern based graphs.

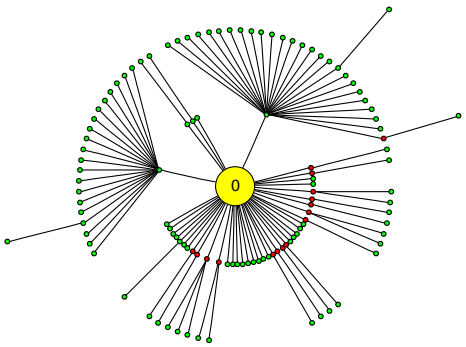
[1]From now referred to as DTBR tree



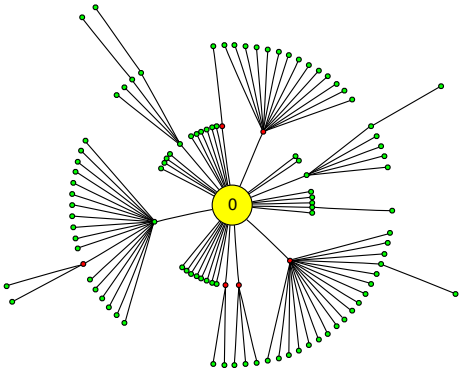
(a) $\alpha = 0.01$, $|V|_{red} = 52$, $diameter = 4$



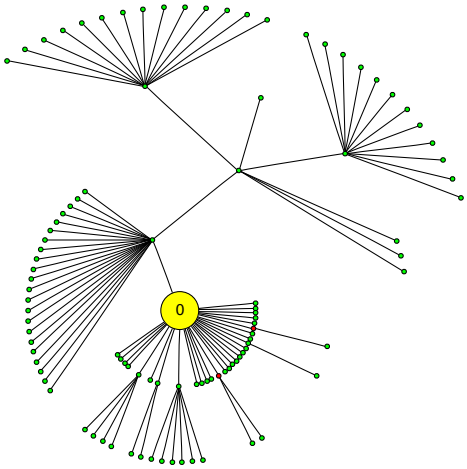
(b) $\alpha = 0.05$, $|V|_{red} = 44$, $diameter = 4$



(c) $\alpha = 0.1$, $|V|_{red} = 16$, $diameter = 6$

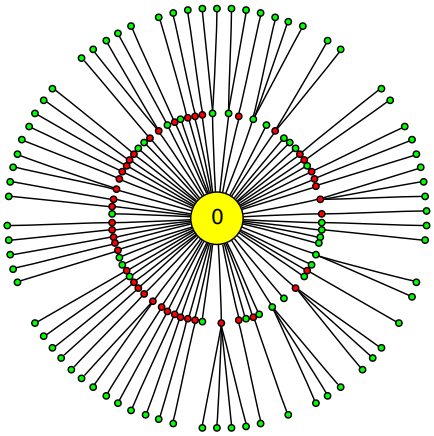


(d) $\alpha = 0.5$, $|V|_{red} = 6$, $diameter = 6$

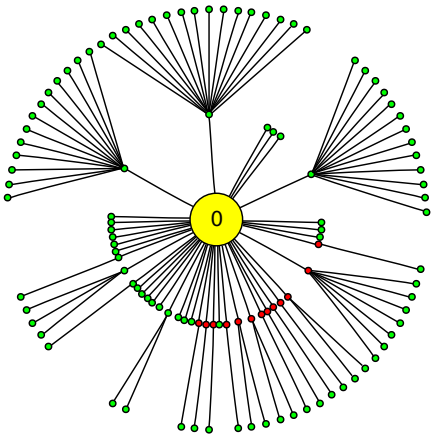


(e) $\alpha = 1.0$, $|V|_{red} = 2$, $diameter = 6$

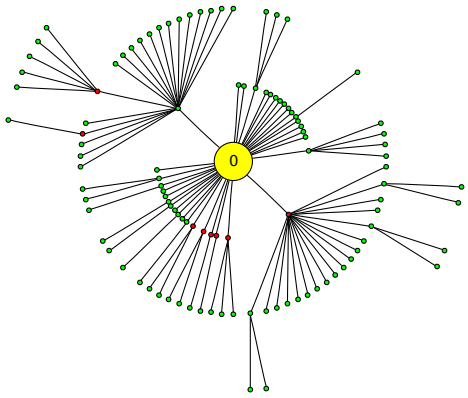
Figure 4.2: Trees for the $MTBR_1$ protocol for various values of α



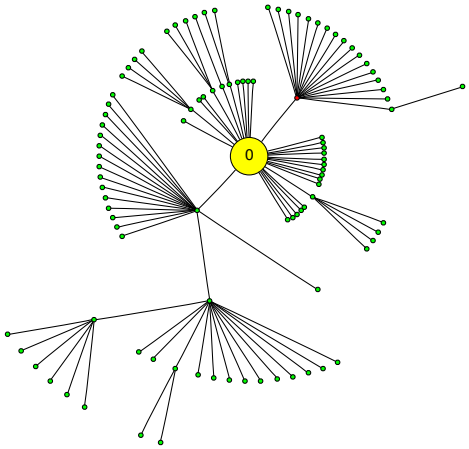
(a) $\beta = 1.1$, $|V|_{red} = 44$, $diameter = 4$



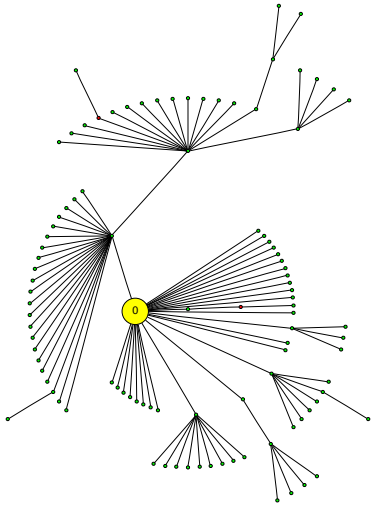
(b) $\beta = 1.3$, $|V|_{red} = 13$, $diameter = 6$



(c) $\beta = 1.6$, $|V|_{red} = 8$, $diameter = 6$

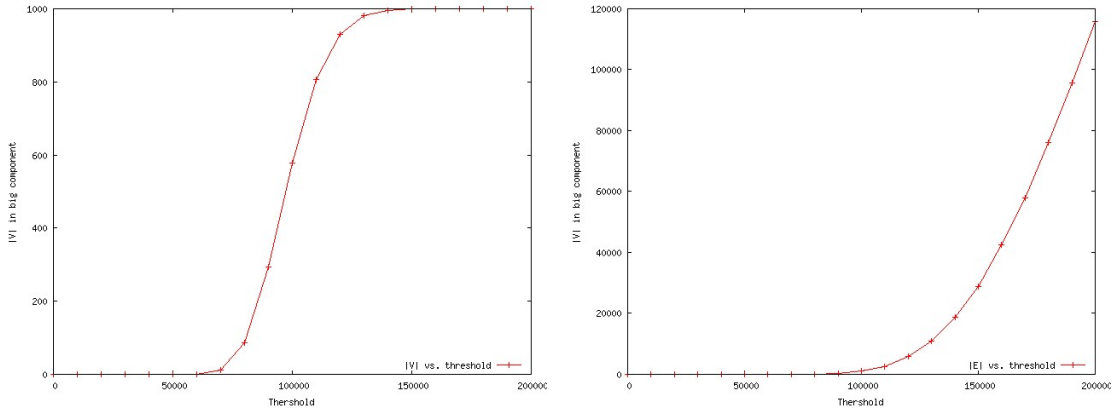


(d) $\beta = 2.0$, $|V|_{red} = 1$, $diameter = 7$



(e) $\beta = 3.0$, $|V|_{red} = 2$, $diameter = 8$

Figure 4.3: Trees for the $MTBR_2$ protocol for various values of β



(a) Number of vertices in big component (b) Number of edges in the big component

Figure 4.4: Selecting Threshold

Graph Sampled from SLAW mobility trace

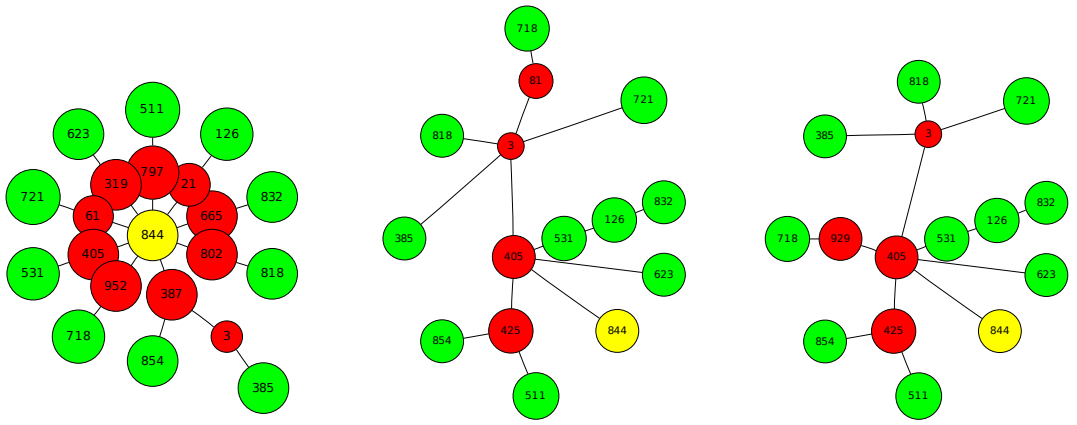
The average waiting time graph is sampled at some temporal points [500000sec to 600000 $mboxsec$@5000] these graphs are then used for analyzing the algorithm. The sampled graph is a weighted graph with edge weights equal to the average waiting time for contact between two nodes. The results presented herein are averaged over results of simulations done over 20 such graph samples (taken in the range [500000sec to 600000 $mboxsec$@5000]).

Pruning of graph As disscused in earlier section 3.2.2 the threshold should be selected so as to remove a considerable number of edges from the graph and also ensure existence of a big component in the graph. Figure 4.4(a) show the number of vertices in the big component as the threshold is fixed. we can see that after a threshold of ≈ 150000 the number of edges in the big component $= 1000 = |V|_{total}$ i.e. there is a single component in the graph. Figure 4.4(b) shows the number of edges in the pruned graph as a function of the fixed threshold value. We can see that the number of edges at around the threshold of ≈ 150000 reduces to merely ≈ 35000. The threshold for the sampled graph is thus selected as 155000.

Trees in pruned sampled graph To illustrated the working of algorithm on the pruned graph a sample DTBR tree with 10 recipients is shown in figure 4.5(a) and the corresponding $MTBR_1(\alpha = 1.0)$ and $MTBR_2(\beta = 2.0)$ trees.The corresponding $MTBR_2(\beta = 2.0)$ tree is given in figure 4.5(c).Notice the new nodes that are introduced in the multicast tree but were not present in the original DTBR tree. Also notice the reduction in the number of red nodes in the modified version of tree. Formation of hubs is also evident from the above tree. The $MTBR_1(\alpha = 1.0)$ tree for the DTBR tree given above is given in figure 4.5(b).Notice that both MTBR trees shown in figures 4.5(b) and 4.5(c) are thinner than the original DTBR tree.

4.1.2 Effects of α and β

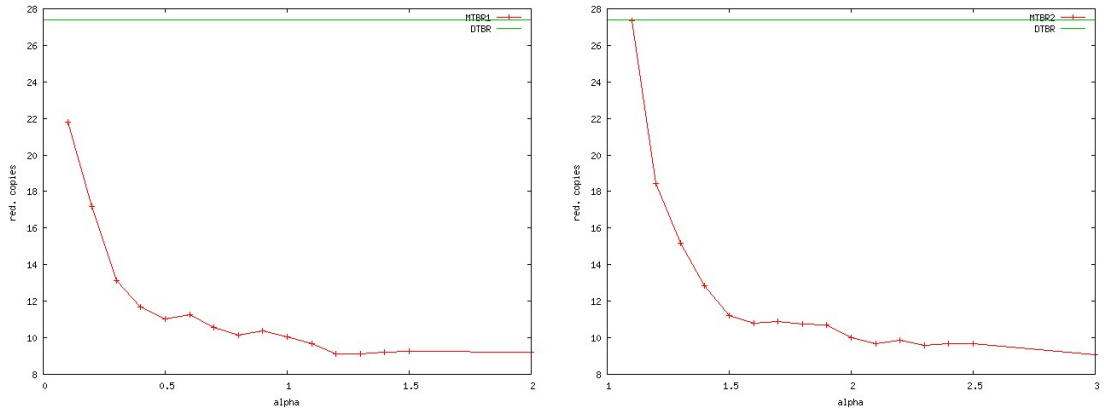
In in section we present the effect of α and β on the nature and properties of the multicast tree. A demonstration of the effect of these parameters on $MTBR_1$ and $MTBR_2$ has already been presented in figures 4.2(for $MTBR_1$) and figure 4.3 (for $MTBR_2$). Here is an analysis of how the statistical properties of the multicast tree change with variation in α for the $MTBR_1$ protocol and β for the $MTBR_2$ protocol. All the results are taken by averaging the values over 20 different sampled graphs all pruned with a threshold of 155000. In all the cases the numbers of recipients in the multicast group is fixed to 100 (10% of the network size).



(a) DTBR tree (b) $MTBR_1(\alpha = 1.0)$ tree (c) $MTBR_2(\beta = 1.0)$ tree

Figure 4.5: Illustration of tree modification with 10 targets

Effect on redundant message copies



(a) Red. message as function of α (b) Red. msg as function of β

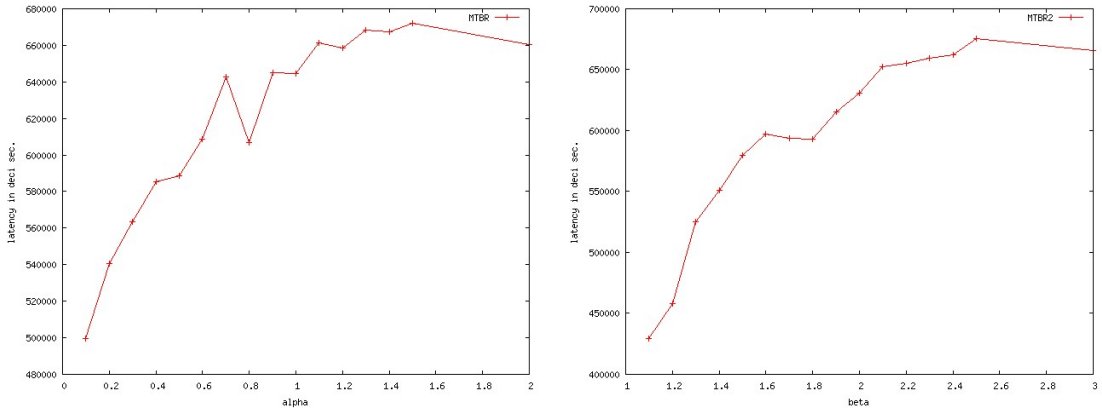
Figure 4.6: Effect on Redundant message copies

Figure 4.6(a) shows the graph for number of redundant message copies vs the α value for $MTBR_1$ protocol compared to that of the DTBR protocol (Green line parallel to the X-axis). It can be seen that the number of redundant message copies reach a minimum possible value on increasing α value (≈ 9 in this case). Also notice that the nature of the curve is exponential and achieves a theoretical reduction of $\approx 65\%$ on the number of redundant message copies (reduction compared to DTBR tree).

The Graph shown in the figure 4.6(b) represents the functional change in number of redundant message copies vs β for the performance of $MTBR_2$ protocol over DTBR. In this case also the number of message copies decrease in an exponential trend and reach a static value of ≈ 9 after a certain value of $\beta \approx 2$. For the case of $MTBR_1$ the redundant copies stabilize around $\alpha \approx 1.0$ and for $MTBR_2$ the copies stabilize around $\beta \approx 2.0$.

Effect on Expected Message latency $E_{tree}(lat)$

Here we present the behavior of 'Expected' Message latency as a function of α for $MTBR_1$ and β for $MTBR_2$ trees. All the trees calculated in the algorithm are weighted trees, where the weights of the edges denote the average waiting time for contacts to happen. Therefore we can



(a) $E_{tree}(lat)$ as function of α for $MTBR_1$ (b) $E_{tree}(lat)$ as function of β for $MTBR_2$

Figure 4.7: Effect on Latency

compute the expected latency of a path P as :-

$$E_P(lat) = \sum_{e \in P} W_e \tag{4.1}$$

Now a tree can be seen as superimpositions of single source paths to the targets, the $E_P(lat)$ for all these paths can be calculated using the equation 4.1. Since the paths are covered parallely in time the expected value of the latency of entire multicast tree is the largest expected latency of the sets of all paths in the tree from source to targets. Thus mathematically :-

$$E_{tree}(lat) = max(\{E_P lat : P \subset tree\}) \tag{4.2}$$

The graphs shown herein are based on the expected value of latency for the entire tree as given by equation 4.2. Figure 4.7(b) is a graph of how the expected latency of message delivery varies with β for the $MTBR_2$ protocol. Although the trend is not clear over such small and coarse interval, the curve is seemingly of the form $K - e^x$. It can be easily noted that as β increases, the latency also increases. This is an expected result since the generated tree is thinner and longer. Figure 4.7(a) is the same graph for $MTBR_1$ protocol that shows the variation of expected latency of the multicast tree with the protocol parameter α in the range $[0.1, 2.0]$. This graph also the same trend line as exhibited by $MTBR_2$. Note that the results shown here for the expected latency has been computed by averaging over 20 sampled graphs from the mobility pattern simulation (for same source and destination).

4.1.3 Centralities and 'Importance'

In graph theory centrality measures are used for comparing the central-ness of a node compared to the other nodes. Since in the proposed algorithm also we are trying to define a concept of hubs for DTNs by introducing a new parameter called 'Importance' (which is nothing but a measure of how central is a node fro a particular multicast) a natural question arises that is this new parameter doing the same thing as well known centrality measures of complex graph theory. For investigation this aspect we present in this section the analysis of correlation between node 'Importances', betweenness centrality and closeness centrality.

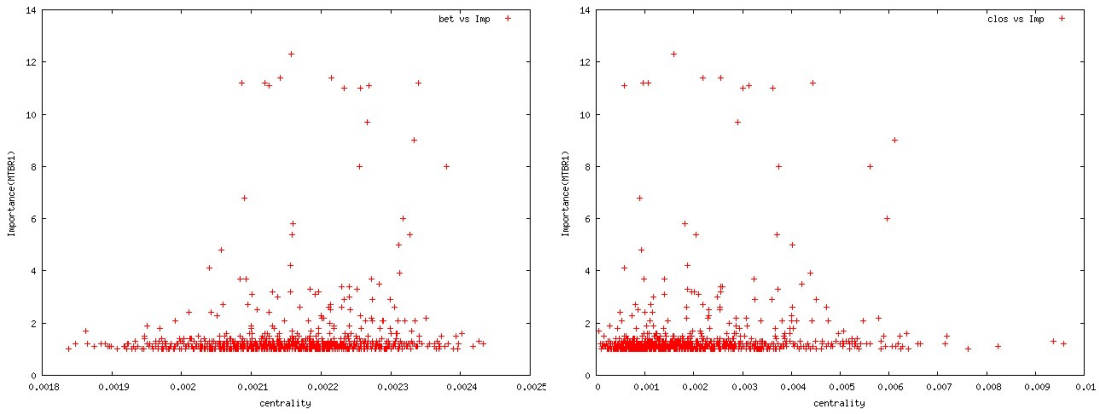
Betweenness Centrality is the ratio of total number of all pair shortest paths to the number of shortest paths passing through a vertex (or edge). Similarly Closeness centrality is the reciprocal of sum of lengths of shortest paths from the vertex to all the other vertices.

The centralities computed herein are on a single instance of sampled graph (at time stamp $595000sec$). The importance values are computed by taking $10iterations$ of each of the algo-

rithms (with $\alpha = 1.0$ and $\beta = 2.0$)

$MTBR_1(\alpha = 1.0)$

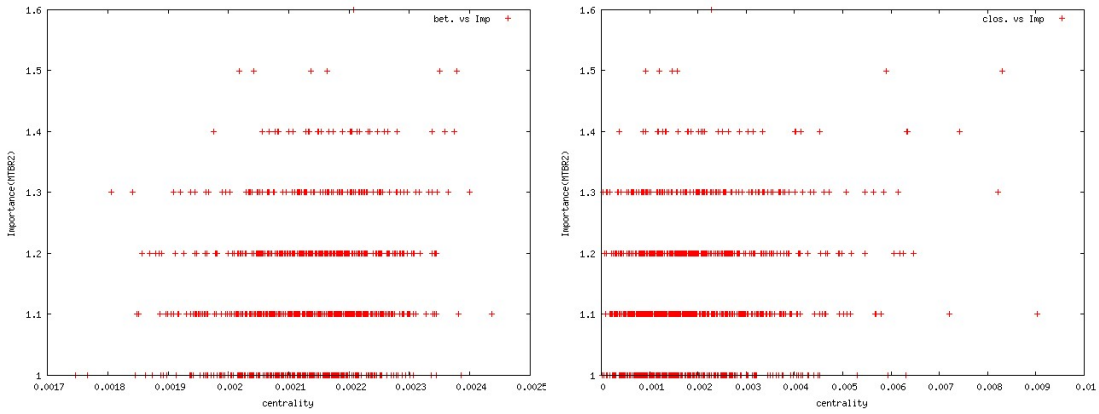
Plots shown in figure 4.8 graphically shown the dependence or the correlation between centrality metrics and 'Importance' values for the $MTBR_1(\alpha = 1.0)$. The Pearson's correlation coefficient for left graph (bet. centrality and importance value) is $\sigma_{xy} = 0.111$ and for the right graph (closeness centrality and importance value) is $\sigma_{xy} = 0.119$, thus on may conclude that centrality measures are weakly/not-correlated to the importance values of the nodes as obtained from the $MTBR_1$ algorithm as described in the present work. (The plots are obtained by plotting the points for ordered pair $(centrality, importance)$ for a particular node)



(a) plot of bet. with I_u (b) plot of closeness with I_u

Figure 4.8: Correlation of centralities with I_u for $MTBR_1$

$MTBR_2(\beta = 2.0)$



(a) plot of bet. with I_u (b) plot of closeness with I_u

Figure 4.9: Correlation of centralities with I_u for $MTBR_2$

Figure 4.9 shows the plots for graphical representation of correlation between centrality measures and the importance values computed for $MTBR_2(\beta = 2.0)$. The Pearson's correlation coefficient for left graph (bet. centrality and importance value) is $\sigma_{xy} = 0.14$ and for the right graph (closeness centrality and importance value) is $\sigma = 0.168$, thus one may conclude that centrality measures are weakly/not-correlated to the importance values of the nodes as obtained

from the $MTBR_2$ algorithm as described in the present work. (The plots are obtained by plotting the points for ordered pair (centrality, importance) for a particular node)

From the above two analysis we can conclude that the concept of 'Importance' is disjoint from the centrality measures considered for the analysis viz. betweenness centrality and closeness centrality.

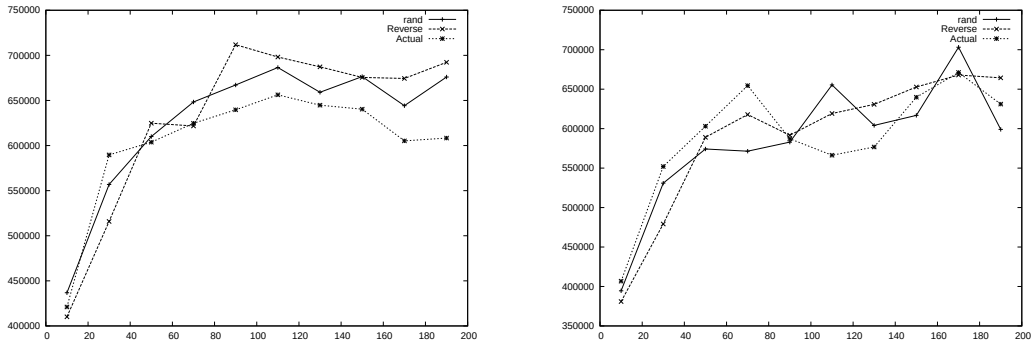
4.1.4 Avalanche Effect

As discussed earlier in section 3.4.1 equation 3.6 shows that the importance of a node depends heavily on the iteration in which it was included in the multicast tree. Thus it is expected theoriticaly that changing the order of targets in the iteration of `compute_tree()` routine will result in multicast trees with markedly different properties (probably different values for latency and redundant message copies). In this section we investigate this difference in properties of the multicast tree and find out whether the $MTBR_2$ protocol surpasses the problem of 'Avalanche' Effect as postulated in section 3.5 from equation 3.7.

Temporal Avalanche Effect

In this section we study the possible avalanche effect in the $E_{tree}(lat)$ of the multicast tree. Since it is based on the expected waiting time, we simulate the trees for the mobility traces for the following three ordering of the targets :-

1. Random order, where the targets are ordered randomly.
2. Reverse shortest path, where the targets are ordered in decreasing order of their *weighted* shortest path from the source in the DTBR tree.
3. Actual shortest path, where the targets are ordered in ascending order of their weighted shortest path from source.

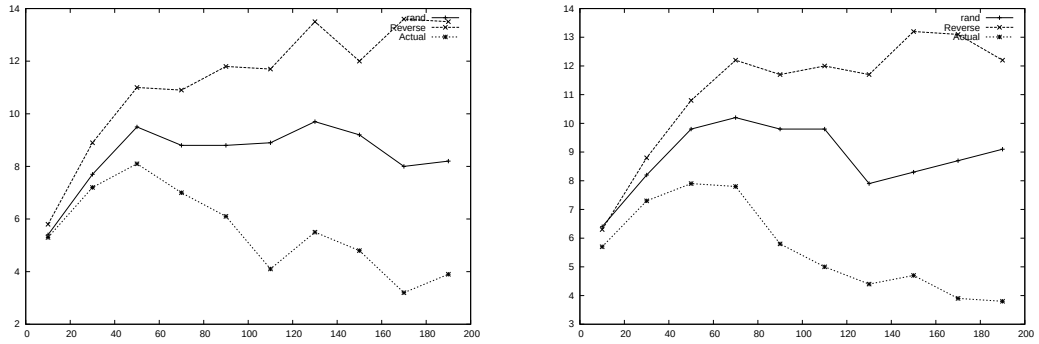


(a) $\Delta E_{tree}(lat)$ for $MTBR_1$ (b) $\Delta E_{tree}(lat)$ for $MTBR_2$

Figure 4.10: Temporal Avalanche effect

Figure 4.10 shows the variation of Expected latency of the $MTBR_1$ multicast tree for the different orderings of the targets. It can be seen that the avalanche effect is visible only after a significant number of targets ($\approx 10\%$) in the multicast group, after which it can be seen that the trendlines for Reverse ordering and actual ordering become increasingly apart, while the random ordering follows a path between these two trend lines. Figure 4.10 also shows the variation for $MTBR_2$ we can see that there is no trend in the differences seen for the orderings, which can be assumed to be oscillatory deviations. Thus the $MTBR_2$ stands the theoritical postulation of surpassing avalanche effect.

Spatial Avalanche Effect



(a) $\Delta(msg\_copies)$ for $MTBR_1$ (b) $\Delta(msg\_copies)$ for $MTBR_2$

Figure 4.11: Spatial Avalanche effect

Figure 4.11 shows the difference in the number of redundant copies of the proposed multicast protocol as the size of recipients increase. It can be seen that the avalanche effect in the number of redundant message copies is seen in case of both $MTBR_1$ and $MTBR_2$ protocols, however the this spatial effect is not predicted by the equation 3.6, throughout the calculations in section refsec:avalanche we have considered the expected value of 'Importance' of a node. The Importance of the node directly affects only the latency of the tree, since higher importance means costlier edges become available for shortest path. Thus the avalanche effect explained in equation 3.6 is temporal in nature. The difference shown here in the plots show that the number of redundant message copies is lower in the case of targets order in ascending order w.r.t the shortest path lengths.

Note that the effect shown here is not at all undesirable. Since the number of message copies are lesser for the case when targets are ordered by shortest path lengths, this gives way to fine tune the algorithm.

4.2 Network Simulation

In this section we present the results of simulation of message multicasts on an underlying mobility model. For the entire simulation we have selected SLAW (*S*elf-similar *L*east *A*ction *W*alk) model as the underlying mobility model because it can capture to some extend the actual motion of humans. Models like Random Way point does n't give any insight into the performance of the algorithm since in RW model all the nodes are similar, thus the average waiting time for contacts for the nodes tend to become same for all the pairs of nodes. The problem of computing multicast tree thus reduces to an unweighted graph (since all the edge weights are equal, graph can be assumed unweighted). The performance of the DTBR and MTBR algorithms on the trace of a RW model is thus trivial.

4.2.1 Simulation Test Bed

Simulations for the designed protocol are done in a custom simulator coded in java. The Simulator takes a contact trace as input and generates message Multicast events. The Simulation is done on DTBR and MTBR. Contact traces for SLAW mobility model is generated using ONE simulator. The custom simulator generates graph of node interaction based on the average waiting time for contact between any two nodes. For now all the nodes use this same global

interaction graph for computing shortest path trees. (It is assumed that the nodes have the global knowledge of the graph).

4.2.2 Event Queue

The Simulator is an event based simulator, thus for each contact event (contact UP or contact DOWN) a network events is registered with the event queue. The simulator keeps track of number of contact events in the queue. When there are no contact events in the queue, the simulator reloads the contact sequence from the trace file specified (because without contact events the simulator is idle). Thus there is a periodicity in the contact sequence at least with the period of total time of the contact trace simulation (i.e. the time till which the mobility pattern was simulated in ONE simulator).

Phantom Events

Since the number of UP and DOWN contact events might not be same in the contact trace or because of minor perturbations induced intentionally in reloading the mobility trace, when the trace file is reloaded there might be some contacts already present in the network topology that have no corresponding contact break event. These **phantom contacts** are handled by registering a break contact event for all the existing contacts at the time of reloading the event queue. Thus at the boundaries of mobility traces the event queue may contain physically impossible events, but this is a transient state and the results shown herein are well within the boundaries of the mobility traces.

4.2.3 SLAW traces

The SLAW traces used in the simulations are programatically generated from MATLAB code available at the following ncsu site:- `http://research.csc.ncsu.edu/netsrv/?q=content/human-mobility-models-download-tlw-slaw`. Table 4.1 and 4.2 show the details of the sets of slaw traces on which the simulations are performed. Matlab code used for generation of the respective slaw trace has been given (issuing the command will not generate the exact same trace but the properties of the trace should be equivalent to trace 1&2).

Trace 1 : `trace = SLAW_MATLAB(3, 1000, 4000, 50, 0.75, 72, 50, 1, 30, 60*60);`

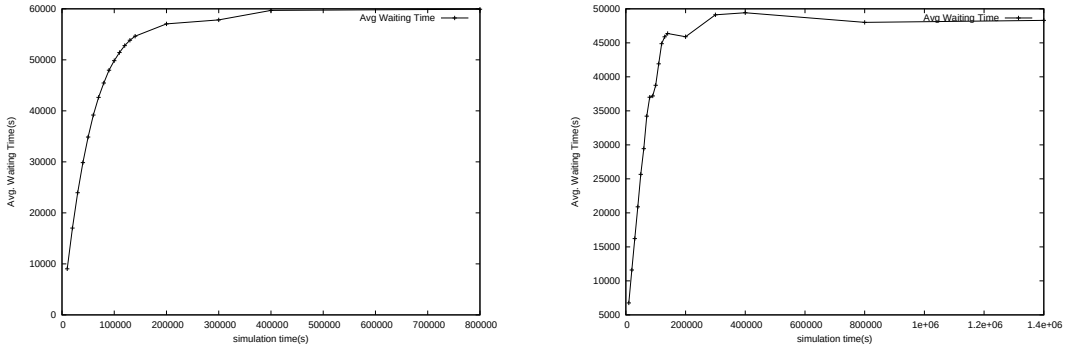
Trace 2 : `trace = SLAW_MATLAB(3, 126, 2000, 2000, 0.75, 24, 50, 1, 30, 60*60);`

Number of nodes	1000
Area of simulation	$4000m \times 4000m$
Warmup time	$1000s$
Simulation Time	$259200s$ ($72hr$ OR $3day$)
Number of waypoints	50
Alpha	3
Hurst parameter	0.75
Clustering Range	$50m$
Beta (for pause time)	1
Min wait time	$30s$
Max wait time	$1hr$ ($60 \times 60s$)
Number of contact Events	9713129 (4857720 up, 4855409 down)

Table 4.1: SLAW trace #1

Number of nodes	126
Area of simulation	$2000m \times 2000m$
Warmup time	$1000s$
Simulation Time	$86400s$ ($24hr$ OR $1day$)
Number of waypoints	2000
Alpha	3
Hurst parameter	0.75
Clustering Range	$50m$
Beta (for pause time)	1
Min wait time	$30s$
Max wait time	$1hr$ ($60 \times 60s$)
Number of contact Events	85811 (43019 up, 42792 down)

Table 4.2: SLAW trace #2



(a) trace #1 (b) trace #2

Figure 4.12: SLAW trace average waiting times

Simulation

The traces used in the simulation are generated in MATLAB simulated environment of finite duration as mentioned in tables 4.1 and 4.2. The amount of time the mobility patterns are simulated gives two temporal regions of interest as seen w.r.t. figure 4.12. For computation of the average waiting times of the figure 4.12, the SLAW traces are repeated over and over again to the required simulation time. For example it can be seen that in figure 4.12(a) the actual simulation time for the mobility pattern is $\approx 3days$ but the amount of simulation time in the plot is $\approx 10days$. Thus the trace is repeated ≈ 3 times for the computation these plots. As is visually evident from the plots the mobility pattern stabilizes in terms of average waiting time after some amount of simulation. This may be the effect of repetition of the trace, hence no conclusive remarks can be made for the nature of actual average waiting time trend in a general SLAW model. For the simulation purpose however the setup mentioned here provide a stable interconnection graph for algorithm analysis. From the graphs two temporal sections are vividly prevelant:-

Unstable Zone : As shown in figure 4.12, during the time interval $[0, 150000]$ for plot 4.12(b) and [0, 200000] for figure 4.12(a), the average waiting time of the graph is increasing monotonically. It can be reasoned that during this time interval the graph properties are not well 'learnt' by the simulation setup and hence the internal state representation of the system exhibits unstable behavior. Simulation in these regions of the mobility traces give an idea of how good an algorithm performs in initial stage of installation of a DTN system.

Run	Redundant Message Copies(R_i)	$(\overline{R} - R_i)^2$
1	71.2	1.69
2	68.5	1.96
3	65.5	19.36
4	67.2	7.29
5	72.3	5.76
6	71.1	1.44
7	73.2	10.8
8	72.1	4.84
9	68.7	1.44
10	69.2	0.49
	$\overline{R} = 699$	$\sum = 55.07$

Table 4.3: Demonstrating Statistical Soundness

This state is however transient since over time the system shall learn a specific pattern.

Stable Zone : As evident from figure 4.12, the plots have stabilized after a certain point (200000 for 4.12(a) and 150000 for 4.12(b)). Beyond these points it can be reasoned that the simulation setup has identified a pattern in the mobility trace. Even though the average waiting time stabilizes, there is still a lot of churn and heterogeneity in the interconnection graph (evident from high variance). Trace #1 has a very large number of contact events $\approx 10M$, which makes it computationally difficult (within the resources available) to simulate it for 20 iterations to the stability point and beyond, thus the analysis in the stability region shall be presented only for trace #2.

In the following sections we shall analyses different performance metrics from the network simulations done on mobility pattern depicted in table 4.1 and 4.2. All the simulation results presented herein are averaged over 10 runs where each run consists of 20 message multicasts (unless state otherwise), thus effectively all the results are averaged over 200 message multicasts.

4.2.4 Statistical Soundness

The results presented in the current work are significantly accurate and practically free from random phenomena. To demonstrate the idea we take the example of 10 runs of code (20 message multicasts each) as shown in the table 4.3. From table 4.3 we can infer that the variance of the sample is given as:-

$$S^2(n = 10) = \frac{\sum (R_i - \overline{R})^2}{n - 1} = \frac{55.07}{9} = 6.1$$

for a 90% confidence interval, $t_{9,90} \approx 1.833$ thus interval is,

$$\overline{R} \pm t_{9,90} \times \sqrt{\frac{S^2(10)}{10}} = 69.9 \pm 1.4$$

Thus with 90% confidence (probability) the value of redundant message copies varies no more than $\frac{1.4 \times 100}{69.9} = 2.1\%$ from the average value, i.e 69.9. This shows that the results presented in the present thesis are statistically significant.

4.2.5 Implementational Overview

This section presents a general overview of the implementational details that shall provide grounds to reason the outcomes of the following simulations presented. The algorithm implementation is simple and evident from the pseudo code presented in section 3.2.1, however

there are some evident problems while simulation that need to be addressed.The following is the description of the general logic flow in simulating node u:-

Contact Event with v : node u searches its local buffer for any message that has been marked for transmit to v, if found then checks if the interface of u and v are idle. if yes then makes them busy and starts transmission else retries after a timeout.

Receive Event of msg : node u appends its id to the list of ids in the message header. This is done to prevent the message from going in loops (as it happens very frequently in the case of $MTBR_2$ and to a lesser extent in $MTBR_1$). Next the node computes tree for further forwarding of the message and stores it in local buffer (if a contact is found then the message is sent).

Passing History : Each message header has a unique ID and a history of the nodes it has passed through, the message also contains a vector of $k < ID, I_{ID} >$ pairs from the multicast tree of the previous node (as explained in section 3.3).

Delivery and Forwarding : A node u may receive many requests for forwarding the message with same ID but it can only have a singe message copy that may address it as a recipient. Thus a recipient gets only one copy of the multicast message in all the tree algorithms presented.

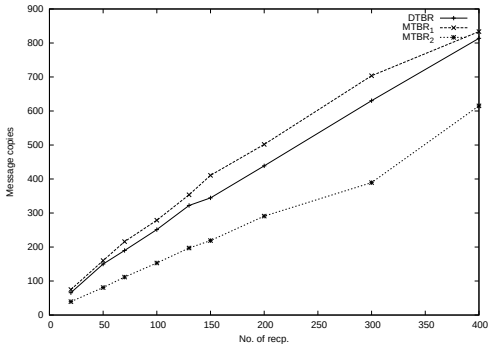
In the following sections we present a comparative analysis of the proposed algorithms and DTBR in separate scenarios. The analysis for STBR yields similar results as obtained in the algorithm analysis in section 4.1, since in the case of STBR implementations of the algorithms, the multicast tree is passed along with the message and the tree is followed irrespective of the current state of network graph. This means if initially the tree properties are better for the $MTBR_1$ or $MTBR_2$ tree, the properties for the entire multicast shall also be better for these algorithms. DTBR however presents an entirely different scenario where the current state of network is considered at each node of the actual transit path. Thus due to high churn in the edges of the graph we get unexpected results for the case of Dynamic implementation of the algorithm. We omit results for STBR protocol since they are same as obtained in the algorithm analysis section.

4.2.6 Redundant message copies

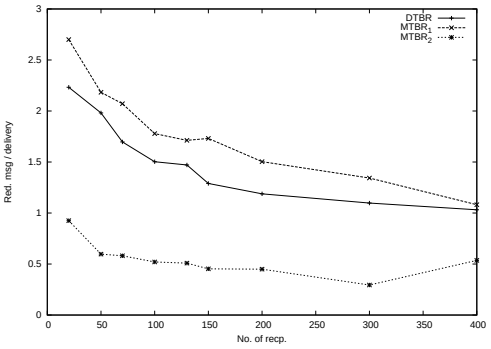
The main aim of the algorithm was to reduce the number of redundant message copies to an appreciable level. In this section we analyze this reduction and try to reason the outcomes. As explained earlier we present the analysis for both stable and unstable portions of the mobility trace plot.

Effect of group size

Figure 4.13 shows the trend in number of message copies and the number of additional message copies per each message delivered on variation of the multicast group size for the trace #1. An unexpected result that crops up in the plots is that the $MTBR_1$ performs poorer than DTBR protocols that seemingly contradicts the results obtained in Algorithm analysis under section 4.1. However the outcomes can be reasoned to be a result of the unstable nature of graph in the time interval. The plots in figure 4.13 were taken from multicast of 20 message copies to the specified number of recipients, where the messages where injected in the simulation environment staring from the simulation time $t_{start} = 60000sec$ at a constant difference of $100sec$. Thus the simulation results are mainly contained in the unstable portion of the trace. Since the $MTBR_1$ protocol is more prone to 'Avalanche effect' as deduced in section 3.4.1 and supplemented in section 4.1.4, then change in the topology metrics of the system affects the $MTBR_1$ version more



(a) Message copies Vs. group size



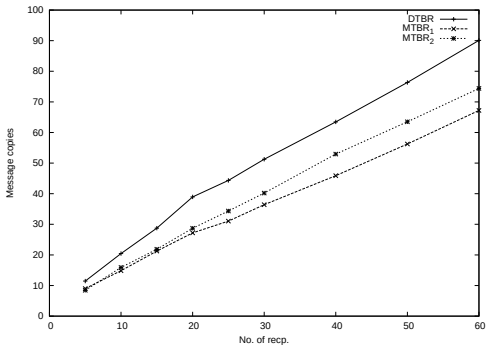
(b) redundancy ratio Vs. group size

Figure 4.13: Message copies for trace #1 (unstable)($\alpha = 1.0$,$\beta = 2.0$)

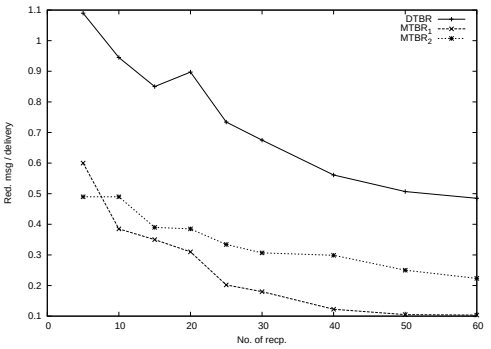
than $MTBR_2$ which performs quite good as compared to DTBR protocol. Thus the reasoning fits in the concept of high dependence on importance of nodes for $MTBR_1$ while in the case of $MTBR_2$ the importances are diffused, thus giving tolerance to network churn. The plot in figure 4.13(b) is the redundancy ratio plotted against number of recipients.

$$\text{Redundancy Ratio}(RR) = \frac{\text{Message copies}(MC) - 1 - \text{No. Recp.}(nR)}{nR} \tag{4.3}$$

Figure 4.14 presents the same results for trace #2 which has a lower contact activity (evident



(a) Message copies Vs. group size

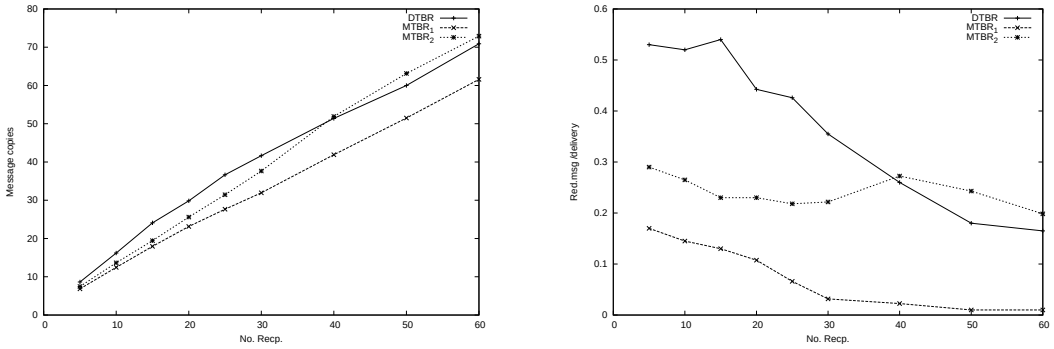


(b) redundancy ratio Vs. group size

Figure 4.14: Message copies for trace #2 (unstable)($\alpha = 1.0$,$\beta = 2.0$)

from table 4.2). In this case $MTBR_1$ performs better than both $MTBR_2$ and DTBR owing to less contact activity (thus less churn and more stability) and small graph (thus lesser nodes to participate).

Figure 4.15 shows the results of $MTBR$ in stable region of SLAW trace #2 (As mentioned earlier, simulation on trace #1 was impractical due to resource limitations). As expected the performance of $MTBR_1$ has improved over DTBR as compared to the unstable case thus augmenting the credibility of reasoning for figure 4.13. It can be seen that after a certain point the $MTBR_2$ protocol can not perform better than DTBR which can be attributed to lack of room for improvement, however for reasonable multicast group size (i.e up to $\approx 30\%$) the performance of both $MTBR_1$ and $MTBR_2$ are comendable for reducing the number of redundant copies. From these results it becomes evident that $MTBR$ protocol stands its ground on reducing the



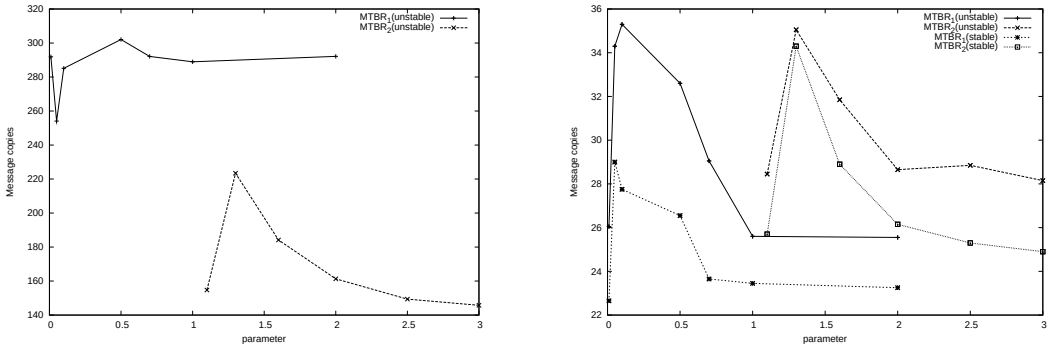
(a) Message copies Vs. group size (b) redundancy ratio Vs. group size

Figure 4.15: Message copies for trace #2 (stable)($\alpha = 1.0$,$\beta = 2.0$)

number of message copies when compared to DTBR in a dynamic routing implementation.

effect of algorithm parameters

Now we look at how the parameters to the algorithm change the number of message copies for the multicast events. In the simulation results presented here the group size is fixed to 100 for trace #1 and 20 for trace #2. Two parameters are varied in these simulations viz. hysteresis factor h (section 3.4) and α(or β) as the case may be.

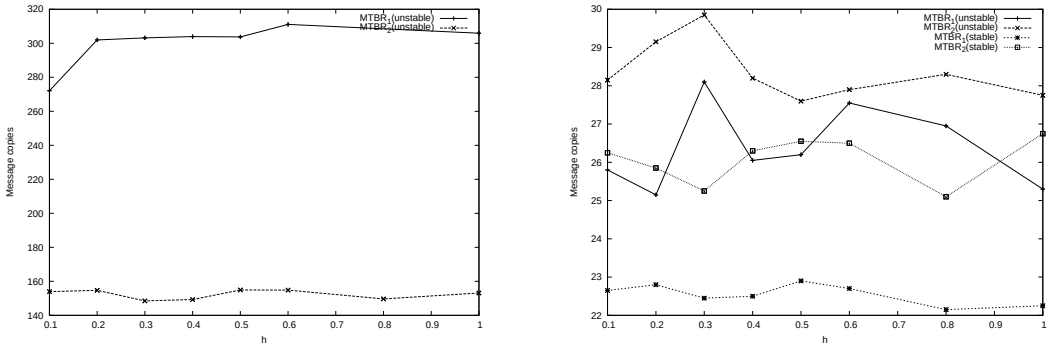


(a) trace #1 (b) trace #2

Figure 4.16: Message copies($\alpha = 1.0$,$\beta = 2.0$)

Variation with α(or β) Figure 4.16 shows the plots of variation of Message copies as the parameter for algorithm varies (i.e α for $MTBR_1$ and β for $MTBR_2$) where $\alpha_{min} = 0.01$ while $\beta_{min} = 1.1$. In plot of figure 4.16(a) we notice that for the case of unstable region in SLAW trace #1 the number of redundant copies does't vary as much as $MTBR_2$ with the variation in α for $MTBR_1$. This means that the $MTBR_1$ protocol is not much sensitive to the parameter change (some sort of 'inertia' in performance). Correlating this observation with the plot in figure 4.16(b) we notice that the general shape of curves for $MTBR_1$ and $MTBR_2$ (take seperatly) are same in all the trend lines (trace #1 and trace #2 both), however the trend line variation seems scaled down for each of the protocol in figure 4.16(a). If we include the degree of stability while noticing the plot 4.16(b) we can see that for both the protocols the

dependence on parameter follows the same pattern with just a small deviation.This can signify that for trace #1 the amount of instability is not a factor that changes the performance of $MTBR$ algorithm. Keeping the notion of stability aside, we can notice that as the parameter attains a very small increment in its value as compared to its value for DTBR trees ($\alpha_{DTBR} = 0$ and $\beta_{DTBR} = 1$), the number of redundant copies decreases instantly then rapidly increases and finally follows a predictable decaying trend. This effect of small value of parameters has been previously pointed out in section 4.1 in context of 'smoothing' effect. The trend for both the parameters finally decay down to show a infinitismly limiting trend to a minimum possible value for redundant copies (≈ 140 for trace #1 and ≈ 22 for trace #2).



(a) trace #1

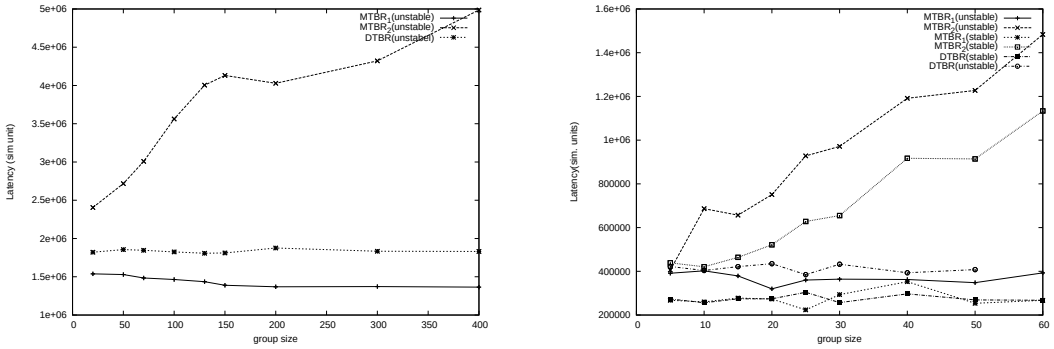
(b) trace #2

Figure 4.17: Message copies with h

Effect of h While analyzing the message copies for variation with 'Hysteresis' factor we fix the values of other variables as: $\alpha = 1.0$, $\beta = 2.0$, $nR = 100$(trace #1),20(trace #2), $K = n$(number of nodes). The value of $K = n$ is very trivial in which a node passes the importances of all the nodes according to its multicast tree. Figure 4.17 shows the plots for bot stable and unstable regions, one can readily deduce that there is no visible change in the number of copies with 'Hysteresis' factor. In all the cases the #Message copies remain almost static with minor pertebations. It can be attributed to the reasoning that with the specified choice of α and β we have already reached the static region as evident from figure 4.16. Thus inefficacy of h might be a result of saturation in performance limits. However it becomes clear that the history information doesn't decide the minimum number of message copies reachable.

4.2.7 Message Delivery Latency

As evident from the previous section the $MTBR$ algorithm is able to stand the grounds of its original motivation to reduce the number of message copies for multicasting using tree based protocols. It is however clear that the reduction in message copies is achieved by 'thinning' of the multicast-tree which may result in its elongation thus sacrificing the delivery latency of the multicast. In this section we present an analysis of how much does one have to let go in terms of Latency to accommodate the reduction claimed. In this section also we analyze the performance following the same steps followed earlier form evaluation of message copies in earlier section. The analysis is presented seperatly for variation of some system parameter while the others are kept constant.



(a) trace #1

(b) trace #2

Figure 4.18: Latency with group size

Effect of group size

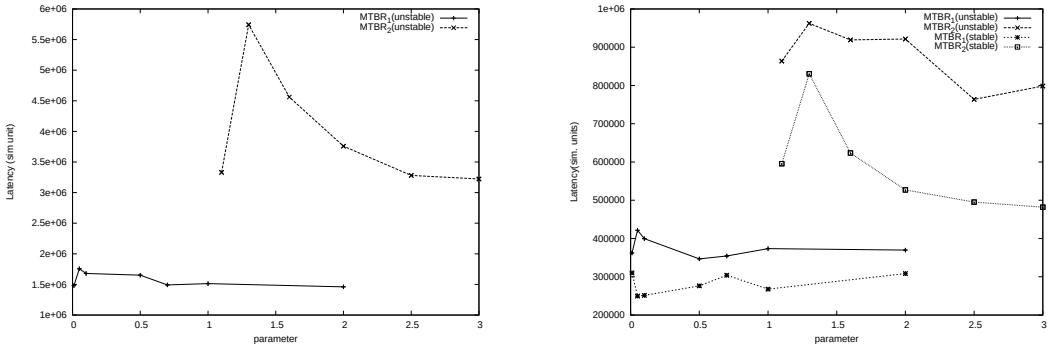
Figure 4.18 show the variation of average message delivery latency as the size of mulitcast group is varied. For the sake of completeness we define the average message delivery latency in terms of following parameters. Let m be a message, T_m be the time of its injection in simulation system (i.e the time instant when multicast event was handled). Let there be nR targets indexed from 1 to nR and τ_i be the time when message was delivered to the i^{th} target. The we define the average delivery time as:-

$$\text{Avg. Latency per delivery for } m = \frac{\sum_{i=1}^{nR} (\tau_i - T_m)}{nR} \tag{4.4}$$

A deducible trend that is evident from the figure 4.18 is that the average delivery latency for $MTBR_1$ and DTBR remains constant however it increases almost monotonically for the case of $MTBR_2$ and the absolute value for $MTBR_2$ is quite large as compared to $MTBR_1$ or DTBR. The fact that a protocol from the $MTBR$ family will have a higher latency is understandable and expected as the algorithm tries to substitute longer paths for shorter ones to accommodate multiple message copies to the same node. This explains the increase in latency of $MTBR_2$, however a much unexpected result arises from the fact that in figure 4.18(a) the latency for $MTBR_1$ protocol is all the way less than the latency of DTBR in the simulated range. To investigate the result a possible reason for this observation can be attributed to the instability of network. One can reason from plots in figure 4.12 that the average waiting time for SLAW increases monotonically in the unstable phase. From the average waiting time curves shown in figure 4.12 one can reason that for any pair of vertices u and v the average waiting time T_i until i^{th} contact is higher than T_{i-1} by some exponential factor (to explain the trend of curve). Thus the average waiting time is not such a good predictor as exponential averaging. In $MTBR_1$ the average waiting times are divided by an exponential importance (eq 3.6) which should provide a better prediction for the next average time. However the factor of division in case of $MTBR_2$ and $DTBR$ is constant thus providing a bad prediction. The explanation for the observation thus shall pertain to the specifics of the mobility pattern.

Effect of α or β

Figure 4.19 shows the variation of delivery latency with graph parameters α and β. Note that the graph shown here shall not be confused as contradiction to figure 4.1.2 which is the expected time in which the entire multicast will finish while figure 4.19 plots the expected delivery latency, which are different things. Similar to other plots which vary the α or β there is a characteristic

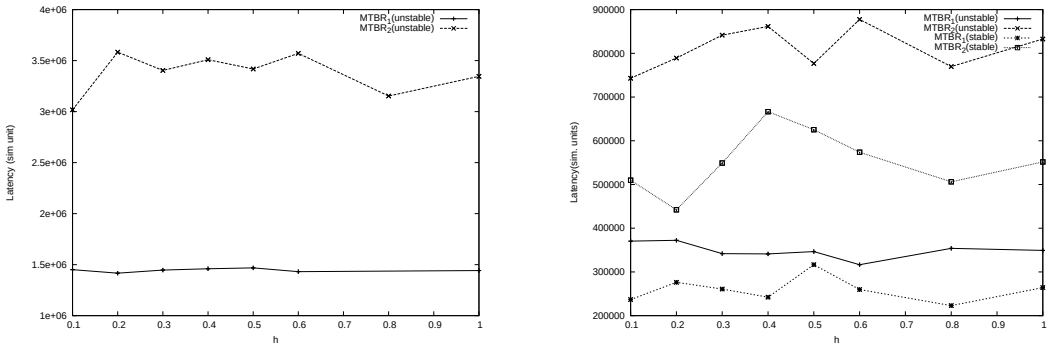


(a) trace #1 (b) trace #2

Figure 4.19: Latency with α and β

sudden change in the Latency metric. The curves then reach a constant value, which signifies that every message delivery takes almost the same time irrespective of the thickness or thinness of tree. Also notice that the trend of latency is scaled down for unstable regions (or regions with higher contact activity). This effect was also observed in simulation for redundant message copies as shown in figure 4.16.

Effect of h



(a) trace #1 (b) trace #2

Figure 4.20: Latency with h

From the figures 4.20 it can be deduced that the latency of delivery has no evident relationship with the 'Hysteresis' of the DTN nodes. In both the stable and unstable region simulation the delivery latency just ripples around a constant trend. It can be reasoned that Hysteresis in a way increases the parameter α or β of the algorithm but the values of these parameter used for taking the plot are already in the region of stability i.e. ($\alpha = 1.0$ and $\beta = 2.0$). This might be the reason for non dependence of delivery latency on h.

4.2.8 Message Overhead Bandwidth

Message Overhead is the amount of extra information kept in the message header for successfully behavior of the protocol. In the simulator designed the message header contains the following fields that add to overhead :-

message ID : This is a 4 byte long integer that uniquely identifies a multicast in progress.

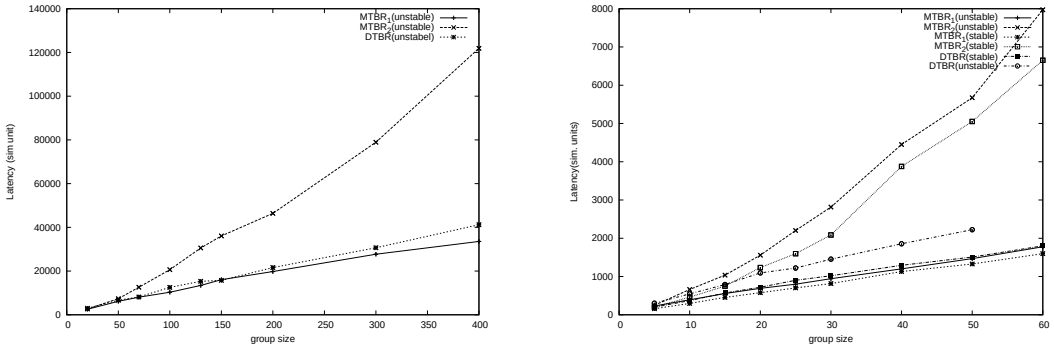
Path : It is the list of all nodes that forwarded the message.It is used to prevent messages from going in loops. Each entry in the path is 4 byte integer.

'Importance' history : This is an array of K 12 byte tuples, that pass the importance values to next node.

recipient list : This is a list of targets that belong to the multicast group.

From this list it can be reasoned that the message overhead that varies with the algorithm comes mainly from the path vector. Message overhead traffic determines the amount of network bandwidth that has been consumed without transfer of actual useful data. Thus it is central to the study of a multicast algorithm as to what and how much overheads are imposed on the bandwidth. In the section presented herein we study the variation of message overhead with different system and model parameters.

Effect of group size



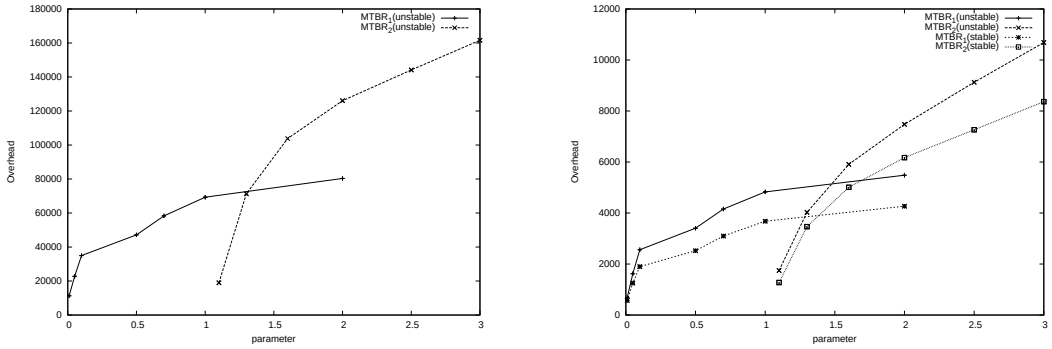
(a) trace #1 (b) trace #2

Figure 4.21: Message header overhead with group size

Figure 4.21 shows the effect of increasing the size of multicast group in the overhead traffic. The trend line for $MTBR_2$ has a higher slope in both the plots as compared to the DTBR and $MTBR_1$ plots. Had the overhead been solely of increase in target group size the variation would have been linear with same slope and values. However the increased overhead for $MTBR_2$ beyond the accountability by recipient list size compels the inclusion of message path in the affecting parameters. The total number of nodes in the multicast are lower for $MTBR_2$ as compared to the cases of $MTBR_1$ and DTBR, then the only way that message overhead could have exploded is that the path mainly consists of a line graph fanning out only near the end (thus all the nodes in the path have recipient list equal to the original recipient list, thus incurring a huge overhead all along the path). This observation thus implicitly provides a proof of thinner trees in actual multicasting.

Effect of α or β

Figure 4.22 shows the plot of message overheads with the algorithm parameters (α for $MTBR_1$ and β for $MTBR_2$). The plots depict that there is general increase in the overhead traffic with an increase in the corresponding parameter. Now having a look at the data elements in the message header only the path field can be a reason for the increase in the message overheads

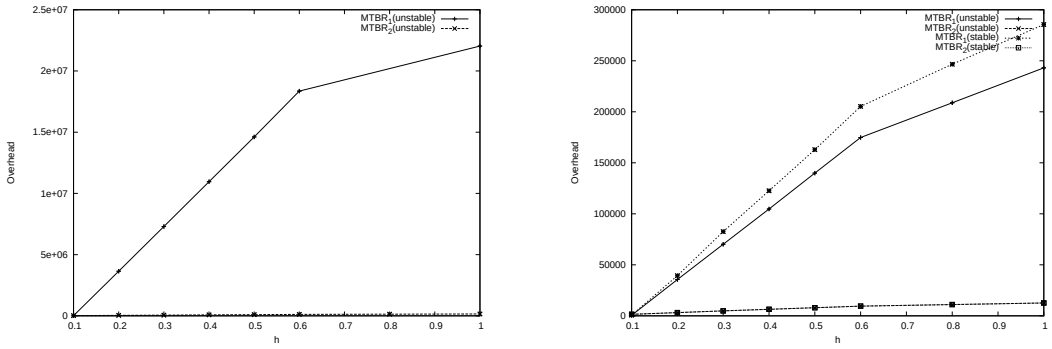


(a) trace #1 (b) trace #2

Figure 4.22: Message header overhead with α and β

for both the algorithms. This combined with the fact that number of nodes visited are reduced for $MTBR_1$ and $MTBR_2$ protocol contributes to the introduction of concept of long and 'thin' actual multicast paths in the system. On can also notice that the Message overhead increase has a steeper slope for the case of unstable region simulation as compared to the stable mobility trace region.

Effect of h



(a) trace #1 (b) trace #2

Figure 4.23: Message header overhead with h

Introduction of h in the protocols demands inclusion of node importances from the sender, in the message header for at most top K nodes. For the sake of simplicity of reasoning we set $K = |V|$ i.e. all the importances are forwarded to the next hop. We notice that there os an increase in the message overhead as the value of h is increased, which again demands the restoration to, increase in the path length, as the possible explanation. However notice that the $MTBR_2$ protocol is unchanged by the h factor, however the $MTBR_1$ protocol shows a significant increase in the overhead traffic. Since the number of recipients is same the only possible explanation for this observation can be, a longer path to the destinations.

Chapter 5

Conclusions

In the preceding part of work we studied the properties of algorithm proposed in both theory and practice (simulations). There are a few vivid conclusions that can be derived viz:-

5.1 Deductions

As explained and analyzed under algorithm analysis and network simulations, the algorithm developed does achieve a reduction in the number of message copies in one flavor or another. This reduction is achieved either at the cost of increased delivery latency (case of $MTBR_2$) or increased overhead traffic. The Simulation results show that best results are obtained when the underlying picture of the network for the node is stable. The algorithm although achieves a betterment in the number of message copies for the multicast in unstable environment, the results taken in complete spirit are below expectations for such situations of high churn in the system state. From the Simulations results presented in the thesis a verdict can be passes regarding each idea explored in the algorithm so far:-

1. Importances does provide a novel an disjoint concept for visualization of hubs in the case of DTNs. With proper selection of update functions, the algorithm can be tailored for a particular mobility pattern.

2. In almost all the simulation results, the inclusion of 'Hysteresis' factor has seemingly defeated the reason and expectations of its introduction. However no conclusive remark can be made for the efficacy of history of importances just based on the simulation on a single type of mobility trace.

3. Like most Tree based routing protocols the algorithm propose herein is more of a tree manipulation algorithm than a multicast management algorithm. Thus the algorithms performance is limited by the underlying physical layer of the node interactions.

4. A dry run of the algorithm (simulating algorithm on a single sample of a graph) for analyzing the properties of the algorithm yields promising and expected results as cited in section 4.1. This sprouts the possibility of good implementations of the algorithm by selecting proper system state metrics for edge weights.

5. Pruning the node graph is a novel approach to reduce the computational cost of the system without compromising the reachability (all the results mentioned above were on pruned data but still the messages could achieve 100% delivery rate). The computation of the `threshold` however shall require some research for the specific case.

5.2 Limitations

Other than the conclusions made above, the present study revealed allot of limitations and shortcomings of various kinds in the simulation. The first and foremost limitation is that of the average waiting time to represent the expected system state. Comparing the results shown in section 4.1 and section 4.2 we can arrive at a strong argument regarding the average waiting time being a bad indicator for edge weights in the inter-connection graphs. One can notice that section 4.1 claims pretty good results for the trends of message copies and expected latency, however the results for actual network simulation differ from this analysis to great extend. The only different in the Network Simulation and Algorithm analysis presented herein is that the algorithm analysis considers the results of the algorithm application to a set os static graphs. However in the case of Network analysis the graph metrics is allowed to change. Since this is the only difference, it can be the only explanation for difference in results of the two analysis presented. The only possible p recursion of this reasoning is that the average waiting time as the edge metric compromises the performance of the algorithm for simplicity of implementation.Better candidates for the representation of DTN graphs do exist, but the analysis of these representations within the time and scope of work was limited by resources available. Another limitation of the analysis in this work is a short spectrum of comparative analysis. Which could have conclusively tracked biases in the algorithm performances for different types of mobility patterns.

5.3 Future Work

The present work has focused mainly in only two flavors of the algorithm for two specific sets of choices for the update functions. The work can be extended by performing a comparative study of a larger spectrum of algorithm versions. Furthermore the present work has dealt with only SLAW mobility traces which, although very common, is not exhaustive. A Conclusive study for finding parameter values for the algorithm that provide optimum multicast for a particular mobility pattern may be considered. Since the System model assumes that the nodes have a global knowledge of the common inter connection graph, there was no specific need for simulating the results on different finds of tree based routing algorithms. For the sake of completion of study, it is important that other TBR algorithms be included in the comparative study whereby implicitly requiring to implement local knowledge version of the algorithm. The algorithm modifications suggested in section 3.3 may be studied in detail for optimization on a specific scenario. Another important problem that comes to picture is the specification of the multicast group to the source and the intermediate forwarding nodes. Since the inclusion of all the recipients in the message header may incur larger overheads, one may include the geographical information in headers (since in most practical cases the multicast group is dictated by the geographical position of the node). A much needed analysis of the algorithm on different graph edge metrics can conclusively postulate the its behavioral characteristics thus making the implementation on a specific case much more easier and elaborate.

Bibliography

[1] Muhammad Abdulla and Robert Simon. A simulation analysis of multicasting in delay tolerant networks. In *Proceedings of the 38th conference on Winter simulation*, WSC '06, pages 2234–2241. Winter Simulation Conference, 2006.

[2] Tony Ballardie and Jon Crowcroft. Core based tree (cbt) multicast - an analysis of multicast routing architectures.

[3] Ching chuan Chiang, Mario Gerla, and Lixia Zhang. Forwarding group multicast protocol (fgmp) for multihop, mobile wireless networks, 1998.

[4] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching gung Liu, and Liming Wei. The pim architecture for wide-area multicast routing. *IEEE/ACM Transactions on Networking*, 4:153–162, 1996.

[5] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8:85–110, 1990.

[6] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.

[7] Sung Ju Lee, William Su, and Mario Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *Mob. Netw. Appl.*, 7:441–453, December 2002.

[8] U. Lee, Soon Young Oh, Kang-Won Lee, and M. Gerla. Relaycast: Scalable multicast routing in delay tolerant networks. In *Network Protocols, 2008. ICNP 2008. IEEE International Conference on*, pages 218 –227, oct. 2008.

[9] Peng Yang. Context-aware multicast routing scheme for disruption tolerant networks.

[10] Qing Ye, Liang Cheng, Mooi Choo Chuah, and B.D. Davison. Os-multicast: On-demand situation-aware multicasting in disruption tolerant networks. In *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd*, volume 1, pages 96 –100, may 2006.

[11] Wenrui Zhao, Mostafa Ammar, and Ellen Zegura. Multicasting in delay tolerant networks: semantic models and routing algorithms. In *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 268–275, New York, NY, USA, 2005. ACM.