Dynamic Network routing – an Agent Based Approach

Akash Gupta[a] and Aditya Zutshi[b]

[a]Synopsys India Pvt. Ltd.
131/82/2C, 3rd A Cross, 18th Main, Koramangala, Bangalore 560095, India
Tel: +91-80-5522201, E-mail: akgupta@synopsys.com

[b]R & D Engineer, Cisco Systems Pvt. Ltd.
Bangalore 560027, India

Abstract

Modern day networks are increasingly moving towards peer to peer architecture where routing tasks will not be limited to some dedicated routers, but instead all computers in a network will take part in some routing task. Since there are no specialized routers, each node performs some routing tasks and information passes from one neighbouring node to another, not in the form of dumb data, but as intelligent virtual agents or active code that performs some tasks by executing at intermediate nodes in its itinerary. The nodes only have to provide a platform on which the mobile agents can run, and they are free to do other tasks as the agents will take care of the routing tasks. The mobile agents because of their inherent 'intelligence' are better able to execute complex routing tasks and handle unexpected situations as compared to traditional routing techniques.

In a modern day dynamic network users get connected frequently, change neighbours and disconnect at a rapid pace. There can be unexpected link failure as well. The mobile agent based routing system should be able to react to these situations in a fast and efficient manner so that information regarding change in topology propagates quickly and at the same time the network should not get burdened with traffic. We intend to build such a system.

Keywords- Mobile Agents, distributed, decentralized, dynamic routing.

1. Introduction

Contemporary computer networks are heterogeneous; even a single network consists of many kinds of processors and communications channels. Networks are also inherently decentralized; capability is scattered across the system. Information networks continue to increase in size, complexity and importance. More and more devices are connected to computer networks, from desktop computers to cellular telephones, Web servers to pagers etc. As connections proliferate, network topologies necessarily become more and more dynamic. Devices may move from place to place, or maintain intermittent connections, or change their relationships to the network and their peers on the fly. Networks must be flexible enough to allow these devices to communicate with each other in a variety of ways and across a variety of substrates.

A fully peer-to-peer architecture offers potential advantages in scalability. We believe that decentralized architectures will be critical to the success of next generation wireless networks. Future networks will be so dense and heterogeneous that centralized system designs will prove unworkable. The present work is an attempt to simulate and test ability of virtual agents to update changes in network topology.

2. Overview

We have defined a simple model of a dynamic network and implemented a simulator to study that model. The simulation we describe here is quite a simplified compared to a real, hardware implementation of such a network. However, we believe our model is realistic enough to provide guidance for designing real systems, and that our experiments have general applicability to dynamically changing, decentralized networks.

Each node in our system owns a topology graph. Each node keeps a list of per-node topology information. The mobile agents embody the "intelligence" in the system, moving from

node to node and updating topology information as they go. Agents have one goal: to explore the network, updating every node they visit with what they have learned in their travels. We emulate the behavior of mobile agents by message passing between different nodes.

Whenever there is a change in network topology the relevant information is propagated throughout the network via agents. Since we have considered a dynamic model there may be three kind of changes in the network.

1. Entry of a new node in the system
2. Disconnection of a existing node.
3. Change in the neighbours of an existing node.

Once a change has occurred an agent performs three actions. First, the agent finds the neighbors of the node it is on and decides where to go next. Second, the agent moves itself to the new node, learning about the edge it travels. Third, it updates the topology graph of the node it now occupies, using its own recent knowledge of the network. We test two algorithms for how an agent select one of its neighbours. The two algorithms are *Random Walk* and *History Based Walk.*

RANDOM WALK

The mobile agent randomly chooses one of its neighbours each time it makes a move. This algorithm provides a base of comparison for more directed algorithms. Two major benefits of random walk are speed and simplicity. They would make our protocol to adapt quickly to frequent topology changes. Moreover, with the important memoryless property of random walks, the protocol consumes network bandwidth as minimum as possible.

HISTORY BASED WALK

In this algorithm the mobile agent preferentially visits the adjacent node it visited earliest or never visited. The agent uses its history to try to avoid backtracking. Intuitively, it performs its task more efficiently by not repeating its own work. Each agent keeps a history of where it has been. Each agent's memory for history information is quite small.

Because the agents must carry their histories with them as they move, the size of the history window is an important parameter: the longer the history, the higher the overhead of moving the agent. Our experiments investigate tradeoffs between history size and system performance.

2.1 MESSAGE FORMAT

As we have stated earlier that the propagation of information regarding network topology is done through message passing by mobile agents. There are two types of message. These are as follows :-

The above message format is passed between nodes by mobile agent in network whenever there is a change in the network topology. Various fields shown above are as follows

- START SEQUENCE – It indicates the origin of the message. 0 indicates message has been originated from main. 1 indicates that message has been originated from any node of the network.

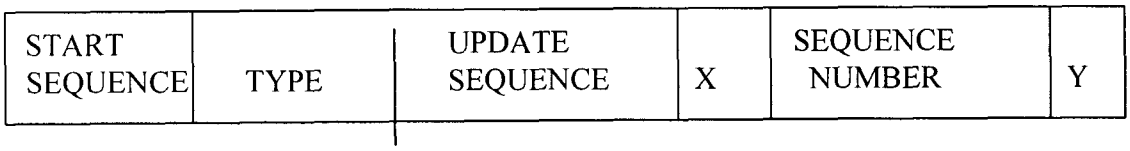

Fig. 1. Message format for Random Walk

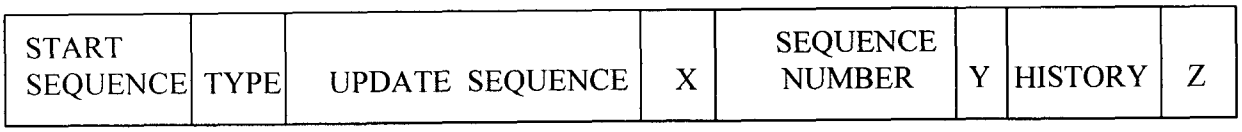

Fig. 2. Message Format For History Walk

- TYPE – It indicates one of the three types of messages that can be propagated in the network. It can be of three types.

1 – *Disconnect Message.* This message is generated when any node leaves the network.

2 - *Change connection Message*. This message is generated when any node changes its neighbours.

3 – *Create Message*. This message is generated whenever any new node wants to join the network.

- UPDATE SEQUENCE – This sequence contains the necessary information that is associated with the changes described above.

  In the case of change connection message it contains the new connections of the node which has changed its neighbours.For the create message it contains the connections of the node entering the system.

- SEQUENCE NUMBER – Messages are originated in sequence. This field indicates the sequence number of the particular message. This field is used to distinguish various messages arriving on a node so that an updating of topology occurs only once for a particular message. This stops the wastage of CPU cycles in updating the topology for multiple times for a single message.

- X & Y – These two characters are used to partition the message into subparts. Each subpart carries different information and should be appropriately interpreted.

All the fields except the History Size And Z are same as in the random walk message format.

- HISTORY – It contains the node number and the number of visits that the mobile agent has spawned in its journey to that node. The length of this field is variable and the user can set this field. It is a parameter of system performance measurement.

- Z – This character is used as End Of Message.

3. Mobile Agent Implementation

3.1 dd (Due_to_Disconnect):

dd is an intelligent agent that moves from one host to another carrying with it information regarding the node which has got disconnected. Whenever a node wants to leave the network, then before disconnection. It sends a Due_to_Disconnect (abbreviated as dd) to one of it's information providing information regarding the imminent change in the network topology. This dd agent then propagates around the network information the nodes where it finds itself to upgrade their topology graphs. Thus generation of dd is a step towards maintaining the network topology constant. For it's propagation, it uses the standard methodology to query the 'tellneigh' agent by sending a Hello message. On receipt of this message, teneigh' promptly replies by sending a list of it's neighbours. After that it is left to dd to select one of the existing neighbour randomly and to dispatch itself to it.

We also keep a count of the nodes which have already been visited by dd. If it is dd's first visit to a particular node then a disconnect message is send to StatAgent so that it can incorporate the change into the topology graph at that particular node. tellneigh (which is actively queried by all visitors to a particular node for getting information regarding the node's neighbours) also need to be informed regarding a particular node disconnection. When tellneigh receives such a message, it removes the disconnected node from it's list of neighbours (if at all, it is there). Processing of the information is left to the resident agents at each node. This keeps size of code which has to migrate from one host to another as small. dd is designed to dispose itself automatically once it has conveyed the information that it is carrying to each and every node in the network. This helps in curbing the movement of redundant (termed as garbage) agents through the network.

3.2 StatAgent:

It is a stationary agent that resides at a particular host and updates the topology graph on receipt of a disconnect message from another agent. Topology graph is implemented in the form of a two- dimensional integer array. An object of this class is stored in a file named topology3 from which it can be easily read by any agent. On receipt of a 'disconnect' message from another agent, StatAgent parses it to obtain the IP address of the disconnected node. Subsequently, all the bidirectional links between the disconnected node and any other node are removed from the matrix and the resulting object of the topology class is stored in the file 'topology3'.

3.3 Initial_topology:

This is a resident agent that is involved in maintaining the correct network topology during the initial stages of network startup. Whenever a neighbourinfo agent arrives at a node for the first time it sends a 'my_neighbour' message to initial topology. The message contains a list of neighbours of the node that created the neighbourinfo. Upon receiving the message from neighbourinfo, initial topology reads the current topology from persistent storage, makes appropriate changes in the topology in accordance with the message and stores the topology back in persistent storage. The persistent storage is a disk file named topology3. Whenever a new node is created it sends its neighburinfo across the network. At each node the initial topology receives the message from it and updates itself about the existence of the new node and its neighbours. Initial topology is not designed to send any messages to any other agent.

3.4 Neighbourinfo:

This has the functionality for updating the topology of each node. It sends a hello message to agent tellneigh in response to which tellneigh sends the information of its neighbours.Now getting the response neighbourinfo form it's message . It selects one of the neighbours returned by tellneigh and dispatches itself to that node.

This agent is generated at each node which are in the network. Thus when this agent is communicated between two nodes ,each node can have the information of others neighbours .This agent can also be used as check of link between two nodes as if one neighbourinfo found a neighbour where it dispatches then at that node it can check whether the node from which it has been created is the neighbour of the node at which it is currently at, if it is not so, then there should be any problem with that link.

Now when it reaches at remote host it transfer the information of the neighbours of the node which has created it ,by passing a message to the stationary agent initial_topology which can read this message and use it to construct initial topology of the network .Thus with the help of this we can construct initial topology of the dynamic network at various time as whenever a node wants to enter the system it dispatches this agent to the network. Now to further continue its travel this again sends message to the stationary agent tellneigh to learn about its neighbours and select one among them randomly and dispatches to that node.

Whenever this agent transferred to a new node it add this node in its history. And when it finds it has traversed whole network it disposes itself so that there are no more than necessary visits of this agent on a node in network and using system resources intelligently.

3.5 TellNeigh:

It is a stationary agent that resides at each host in the network supplying information regarding neighbours of that node in response to a query request. It has got two main functions. On creation TellNeigh first identifies the neighbours for a particular node by randomly selecting a variable number of IP addresses from the network address space. Here an implicit assumption is that every node when it enters the network is aware of the nodes which comprise the network topology and then uses that information to select it's own neighbours, based on factors like with whom it wants to communicate, nearness in terms of vector distances, link reliability, access levels etc. After the neighbours have been selected they are displayed on the viewer window.

Whenever an agent wants to dispatch itself to a neighbouring node, it has to first send a query in form of a 'Hello' message to TellNeigh agent to obtain the list of neighbours for that node. On receipt of such a message, TellNeigh combines the neighbour list in form of a single string and sends the reply back to the querying agent. Since a node chooses its neighbours randomly, it might happen that a particular node A select node B as it's neighbour but node B does not do so. Since we are assuming existence of only full – duplex links in the network, node B must include node A as a neighbour in it's neighbour list (which is currently been maintained by TellNeigh). To account for this case, we have another agent known as 'Neighbourinfo' which traverses the network carrying with it the list of neighbours for the node where it was created. Whenever it comes at a particular node, whose IP address matches the IP address of one of the neighbours of the node where it was created, it sends a 'new_neighbour' message to TellNeigh. Along with the message it sends as an argument the IP address of the node where it was created. TellNeigh at that node, compares that IP address to it's own neighbour list. If it finds (i.e. node B has already selected node A as it's neighbour) a match, it ignores the message, else it appends that IP address to its neighbour list. This helps in keeping the network Topology information consistent at all the nodes, which is the basic motivation behind our approach.

Similarly, when the agent carrying the disconnection information, visits a particular node, it sends a 'disconnect' message to TellNeigh which subsequently removes the node in question from the list of its neighbours.

4. Results of Simulation:

The simulation was done to study the behaviour of mobile agents under various network topologies. The two mobile agents propagation approaches namely Random walk and History based were also put under test and comparative analysis carried out. The results were averaged over a large no. of iterations. Topology changes were randomly simulated in quick succession keeping a base line of 100 changes for obtaining data regarding the Reachability information.

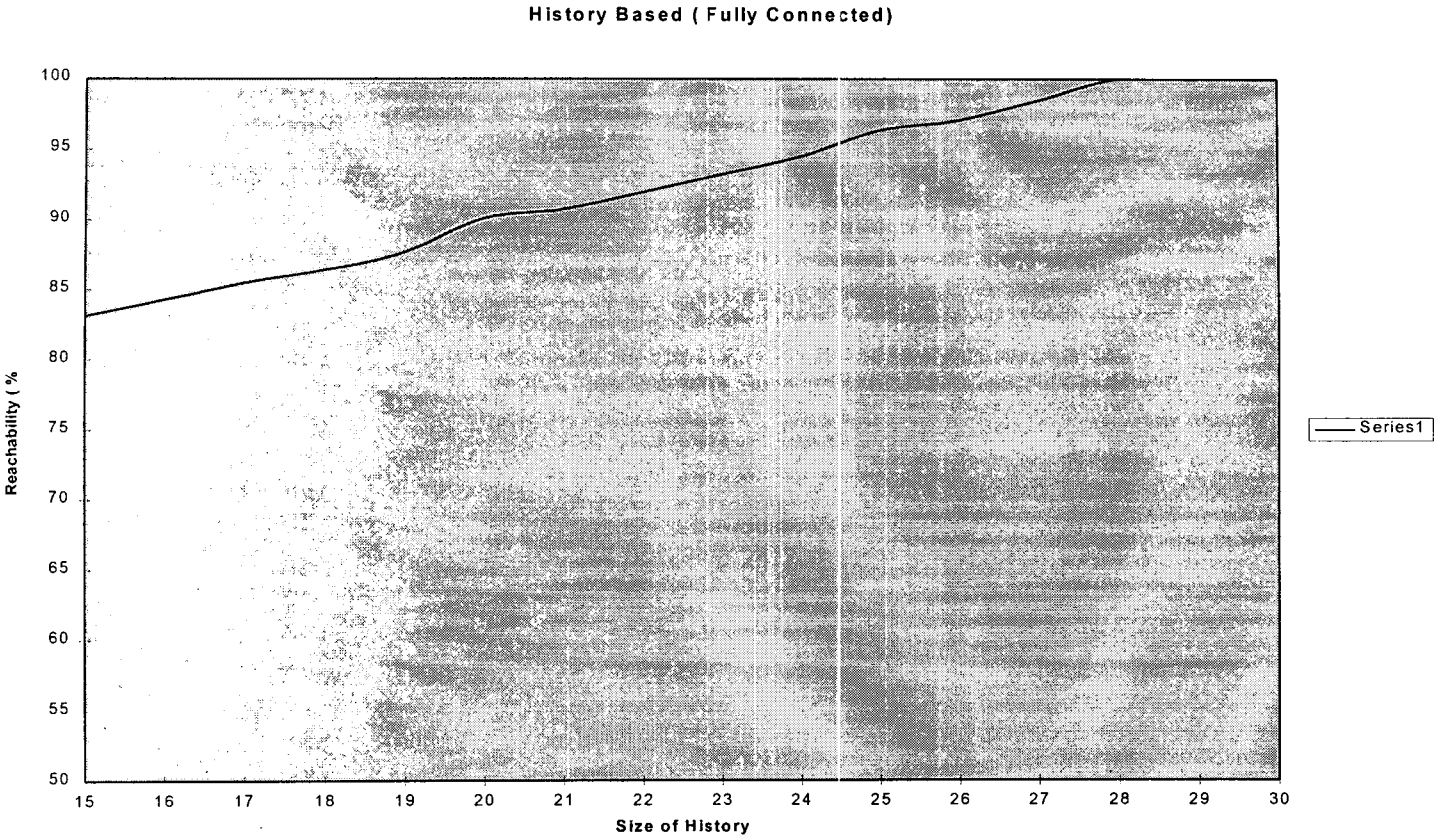

Fig. 3. Variation of Reachability information with History Size for a fully connected topology

From the graph it can be seen that even for the history size just equaling the number of nodes in the network(15), we have a high value of reachability, 83%. As the history size is increased, the reachabilty increases linearly. For a history size of 28(approx. twice the no. of nodes), we achieve 100% reachability.

Therefore, it can be concluded that in a history based fully connected network, one should use a history size of approx. twice the no. of nodes in the network in order to get maximum reachability.

The above behavior is easy to understand. In a fully connected network, the agent has all the options at each node to decide the next hop. Because it is using history, it always makes the best possible decision at each node.This leads to the sharp increase in reachability with history size.The above chart should be contrasted against that of a random agent for a similar topology in which the reachability levels off (after an initial surge) and becomes insensitive to increase in no. of iterations making it difficult to achieve 100% reachability .

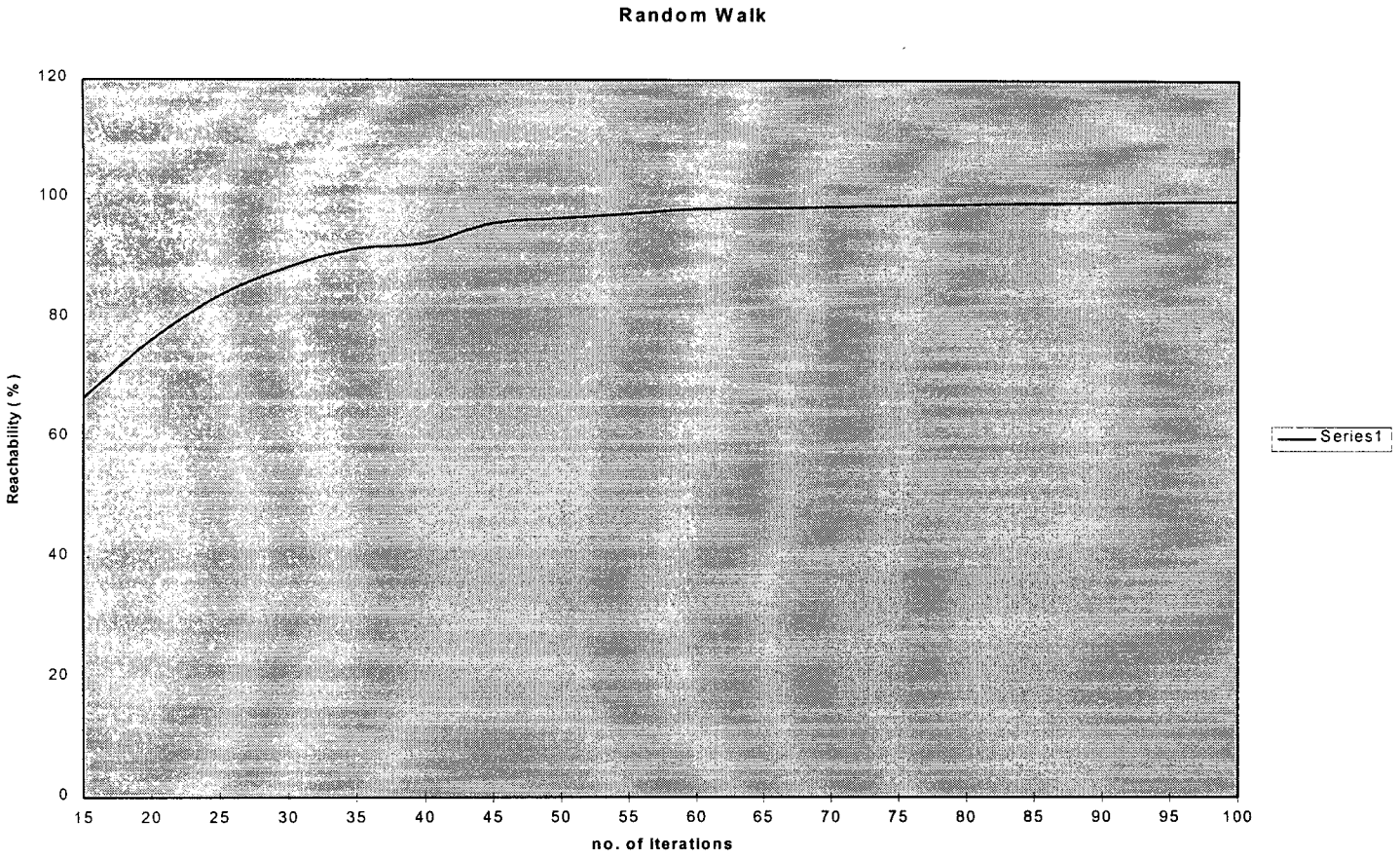

Fig. 4. Variation of Reachability information with No. of iterations for a Random Walk fully connected topology.

From the plot it can be seen that for a fully connected graph a very high value of reachability(96%) is achieved for 45 iterations . Thereafter the reachability increases very slowly with increase in the number of iterations. For a 4% increase in reachability from 96% to 100% we have to increase the number of iterations from 45 to 100. This indicates that reachability becomes almost insensitive to increase in the number of iterations after a value of around 45. This is because in a fully connected network, the agent has the option of going to any node from a particular node. As the number of iterations increase above a certain value the number of redundant paths increase.

As the agent chooses path randomly, it is is more and more likely to choose one of the redundant paths as iterations increase, leading to a stagnation in reachability. Therefore, if cent percent reachability is not critically important, that is, the network can tolerate a bit of error in topology updates, then it is advisable to limit the number of iterations to around 45(which is 3 times the no. of nodes in the n/w). In the initial stages the reachability increases sharply from 67% to 92% as the number of iterations increase from 15 to 35.

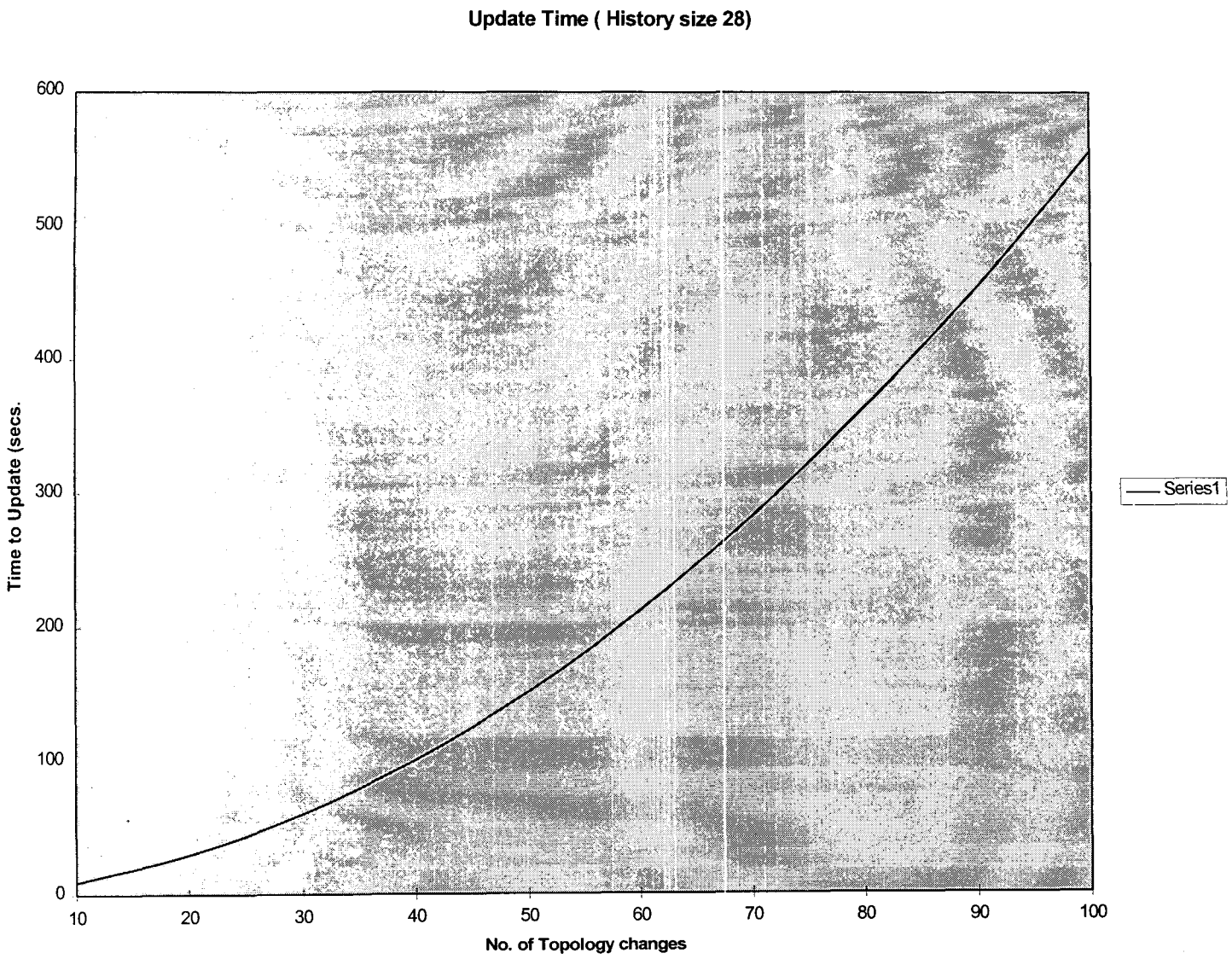

Fig. 5. Variation of time taken to Update with No. of changes in the topology for a History based fully connected topology with History Size constant as 28.

The first question we investigate is how does the network behaviour change as it gets more and more populated with messages regarding changes in the state of the network. For our observations, we consider a fully connected network consisting of 15 nodes. The mechanism used for mobile agent propagation is History based with a fixed history size of 28. This figure of 28 may be considered as the optimum history size for such kind of network so that the mobile agents are able to update the network topology in less than 100 node traversals.

As shown in the graph above, as the number of topology change messages in the network increase, time taken to update the topology at each node increases quite rapidly. This increase is much more pronounced beyond say 50 topology changes as the curve tends to become exponential. The basic feature to note in this respect is that, since we are considering a network consisting of only 15 nodes, then say 50 changes in the network topology within a short span of time calls for a highly dynamic network. Even under such conditions, mobile agents are able to update the topology graph at each node within a relatively short span of time.

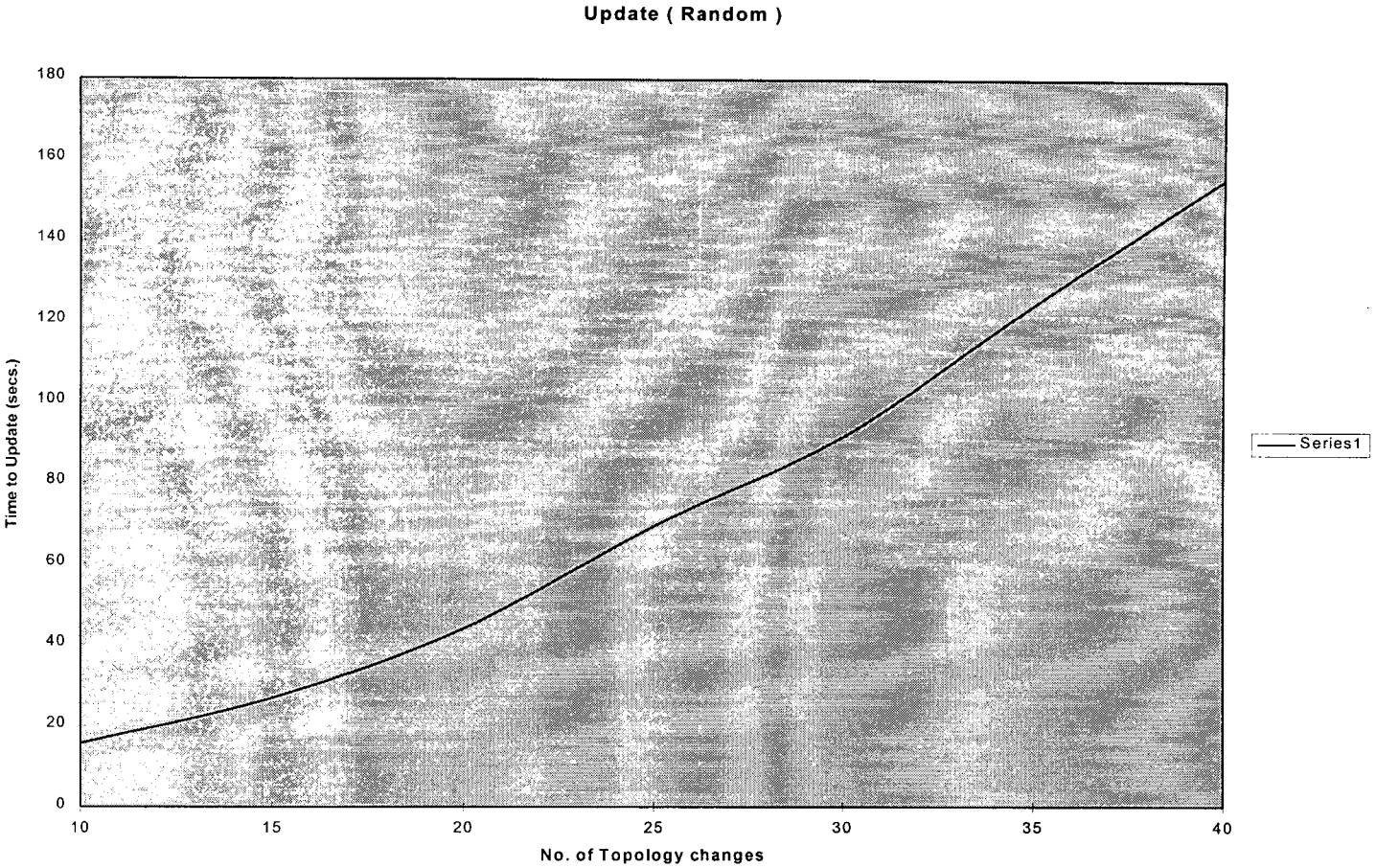

Fig. 6. Variation of time taken to Update with No. of changes in the topology for a Random Walk based fully connected topology taking the no. of iterations as 80.

The above graph must be seen in comparison with the data observed for the case of History based approach. Corresponding to 40 topology changes in the network, Random Walk approach takes around 155 secs. while History based takes around 100 secs to update the network . Another thing to note in this case is that here we are considering an optimum History size of 28 whereas to obtain the same performance from Random Walk strategy, we require to accommodate 80 node traversals.

Thus in a situation, where the network topology changes are very frequent, network links would be frequented by the mobile agents communicating information from one node to another. In such a scenario, it would be desirable to stick to a approach which can update the topology graphs across the network in a shorter span of time. So even with respect to the time based performance, History bases propagation strategy outsmarts the Random Walk approach.

5. Conclusion & Scope for future Work:

In our project we have presented an approach to managing routing in a dynamic network. The particular network we have chosen to simulate, consists of a large number of nodes which can change their topology at any instant of time. This is one example of a system that is poorly served by traditional centralized architectures. We have attempted to show that mobile agents are able to build and maintain topology graph of whole network in such a system, and that these graphs remain reasonably accurate even as the topology of the network changes over time.

We have obtained results depicting the performance of mobile agent based systems for three common network topologies namely the mesh, ring and star topology. We have given our suggestions regarding selection of mobile agent parameters in these topologies.

There are some areas in which more work needs to be done. In a real life dynamic network, the topology could change to shapes not identical to any of the three topologies that we have plotted the graphs for. Work needs to be done to obtain

the optimum values for various mobile agent parameters, in all possible network topologies.

While mobile agents provide for intelligent and adaptable routing, they increase the network traffic as the complete code of the mobile agents need to be transferred across the network. Simple dumb message passing requires less bandwidth but they are not adaptable to complex situations. Work needs to be done to devise ways of selecting either message passing or mobile agent passing based on the network topology.

Much work remains to be done in areas of provisions for security and safety across networks; tools for design, modeling and inspection; methods for analysis of resource tradeoffs, system overhead and protocol strengths and weaknesses.

6. REFERENCES

[1] Lange, D.B., and Oshima, M.: Programming and Deploying Java Mobile Agents Using Aglets,1st edition, Addison Wesley,1998.

[2] http://www.trl.ibm.co.jp/

[3] Minar, N., Kramer, H.K., and Maes, P.:Cooperating Mobile Agents For Dynamic Network Routing., Mobile Computing and Communications Review, March 1999.

[4] http://www.media.mit.edu/~nelson/research/routes/

[5] Caro, G., and Dorigo, M.:Mobile Agents for Adaptive Routing.
In *Proceedings of the 31st Hawaii International Conference on Systems*, January 1998.
ftp://iridia.ulb.ac.be/pub/dorigo/conferences/IC.22-HICSS31.ps.gz

[6] Mario B., et al. :Exploiting code mobility in decentralized and flexible network management., *Proceedings, First International Workshop on Mobile Agents*, Berlin, April1997.
http://www.polito.it/~picco/papers/ma97.ps.gz