Resilient Topology Discovery in Dynamic Systems Based on Self-Diagnosis

Joao Gustavo Gazolla Borges, Elias P. Duarte Jr.
Department of Informatics
Federal University of Parana
Curitiba, Brazil
Email: {jggb,elias}@inf.ufpr.br

***Abstract*—This paper introduces a distributed strategy for topology discovery in dynamic and decentralized networks, such as P2P and mobile ad hoc networks. This strategy is inspired on a distributed system-level diagnosis algorithm for general topology networks. Each node tests its adjacent links periodically. Each link may be in one of two states: fault-free or unresponsive. An event is defined as a change in the state of a link. When a tester detects an event, a message carrying event information is disseminated. The strategy was implemented in a simulator, and experiments were performed in networks with random power-law topologies, with sizes ranging from 10 to 2000 nodes. Simulation results evaluating the latency, cost and precision of the strategy are presented.**

I. Introduction

Several computer network applications require that nodes keep the complete network topology. Examples include multicast [1], group membership [2], network management, routing, and others. This work proposes a distributed strategy for topology discovery in dynamic and decentralized networks. The network is dynamic because nodes can join and leave at any time with or without previous notice. Furthermore, links are logical and can be established and terminated at any moment. Nodes can fail unexpectedly, as well as links. The networks are considered decentralized: there are no central servers nor special nodes of any kind. Examples of such networks include decentralized P2P networks, such as Gnutella [3], and wireless ad hoc networks [4].

The network is assumed to have arbitrary topology. Each node stores a graph which is a local representation of the network topology, with vertices standing for nodes and edges standing for links between nodes. Furthermore, each node executes periodic tests in order to be able to detect events in its adjacent links. An event is a change in the state of a link, such as the establishment, the termination, the failure or the recovery of a link.

Due to the dynamic behavior of the network, the local topologies kept by the nodes may not reflect the real topology. Thus, the algorithm needs to be run continually, in order to keep topology information as accurate as possible. The proposed strategy is based on an existing algorithm for self-diagnosis of general topology networks [5]. The contribution of this work is to allow nodes to join and leave the system, while the previously cited algorithm assumes a static topology.

The strategy was implemented in a simulator, in which experiments we executed to assess the latency, cost, and precision of the obtained topologies. The experiments were performed in power-law networks, created randomly using a topology generator based in [6]. The simulations were also executed with networks with sizes ranging from 10 to 2000 nodes.

The rest of the paper is organized as follows. Section II describes the proposed strategy. Section III presents experimental results. Section IV points to related work, and section V concludes the paper and presents an outline of future work.

II. The Proposed Strategy

This section describes the strategy that is introduced in this work. The goal of the strategy is to allow every node to have, at any time, a view of the complete network topology as accurate as possible.

A. Network Model

The network in which the strategy executes is assumed to be an overlay network [7]. The network topology is represented by an undirected graph $G = (V, E)$, where set V represents the network nodes and an edge $(u, v) \in E$ represents a symmetric logical connection between nodes u and v, such as a TCP connection.

Each node has an unique identifier, which is often the ordered pair *(IP address, port)*. A node a may connect to any other node b whose identifier is known by a, for that node a uses a connection establishment protocol. A connection established between two nodes may be terminated by either node at any time by executing a connection termination protocol.

A new node can join the network as long as it knows the identifier of at least one node already in the network. The arrival of a node in the network follows a subscription protocol, which is described later in this paper. A node may leave the network properly by following the network departure protocol correctly, which causes all connections to be terminated. However, a node may also leave the network unexpectedly, without any previous notice. This may happen for instance when a user accidentally pulls out the node's energy plug. In such a case, if the node wants to join the network again, it has to execute the subscription protocol.

A node may be in one of two states: fault-free or unresponsive. This model assumes crash faults and an asynchronous system [8]. An event is defined as either an unresponsive node becoming fault-free, or the opposite. A link may be fault-free or unresponsive. An unresponsive link loses all messages, whereas a fault-free link delivers all messages in the same order they are sent. No assumptions are made regarding the relative speed of processors, and link delays. Consequently all the connections to a slow node may be considered unresponsive, if a node does not answer within expected timeout intervals.

B. Specification

The distributed strategy is continuously executed on-line, in order to make each fault-free node aware of the entire network topology. Each fault-free node stores an undirected graph $G = (V, E)$, which is its local representation of the network topology. Each vertex $v \in V$ stands for a node, and each edge $(i, j) \in E$ stands for a link between the nodes i and j.

A natural number (timestamp) is associated with each edge in the graph. An even timestamp means that the corresponding link is fault-free, whereas an odd timestamp means the link is unresponsive. Any link that has properly executed a termination protocol, is erased from topology descriptions. When a link is established for the first time, its corresponding edge timestamp is set to 0.

When a node detects a change in the state of a link, it increments the corresponding timestamp and disseminates a message with the new event information. This timestamp is used to ensure that nodes can determine whether a received event has up-to-date information. For instance, if a node receives two messages with information about a certain link, the message with the smaller timestamp is ignored. Besides that, the timestamp is also used to reduce the number of redundant messages in each dissemination.

C. Joining the Network

A node a willing to join the network must connect to a node inside the network (called b). Node b checks in its graph whether edge (a, b) already exists, which could happen if a had failed previously. If edge (a, b) exists in its graph, node b increments the corresponding edge timestamp to the next even number, to indicate that the edge is now fault-free. On the other hand, if edge (a, b) does not exist, node b assumes that the link is being created for the first time, and thus creates a new edge (a, b) in its graph, setting the timestamp to 0.

Afterwards, node b sends to node a its graph representing the network topology. It is then necessary to disseminate the new event information to all fault-free nodes, i.e. node b disseminates a message containing the identification of the link (a, b), as well as the new link timestamp.

After the first connection, a node may, at any time, connect to other nodes in the network. The process is the same, except that it is not required to send the graph again.

D. Test Phase

Each node tests its adjacent links periodically in order to determine the occurrence of events. Tests are executed for that purpose. Each node sends, from time to time, a test message to each neighbor. If the node receives a reply within a timeout interval, the node assumes that link is fault-free.

However, if a node tests a link and receives no reply until the timeout interval expires, it is possible that either the link is faulty or its neighbor has failed. Because of that ambiguity, the node considers that the link is unresponsive. When detecting an event on a link, the node increments the corresponding edge timestamp. Afterwards, the node disseminates the event, sending a message with the identification of the link that suffered the event (i.e., the id's of the nodes connected by the link), and the new link timestamp.

E. Event Dissemination

In order to disseminate events, the node responsible for the dissemination sends a message to each neighbor, which in turn relay the message to their neighbors, and so on. When a node receives a message, it updates its graph with the received information. In this way, if the same node receives another message with the same information, the node can detect that it has already been informed about the event and does not disseminate the message again.

The dissemination message contains the id's of the nodes connected by the link, together with an integer (timestamp) that represents the link state. If the event is about a proper link termination, the timestamp is equal to -1, indicating that the link can be removed from the topology representation. The dissemination is executed throughout the entire network, and takes at most d steps where d is the network diameter. When a node receives a dissemination message, it compares the message timestamp to the link timestamp in its local topology representation graph. If the message timestamp is smaller or equal to the local timestamp, then the node had already received the dissemination previously, so no dissemination is required.

III. Experimental Results

A simulator was built to allow to the evaluation of the proposed strategy. The simulation goal is to evaluate the cost, the latency and the precision for randomly generated topologies.

The simulator, which was implemented in C++, employing an abstraction we call a *cycle*. The simulation program initially creates an object of type "Node" for each existing node. In each cycle, all nodes (i.e., the objects that represent the nodes) are executed sequentially. A node, when executed, can read messages that have been written in its queue by other nodes, and can also send messages to its neighbors. Queues are used to send messages, i.e., each node has a message queue accessible by its neighbors. When a message is sent in a cycle, it can only be read in the execution of the receiver in the next cycle.

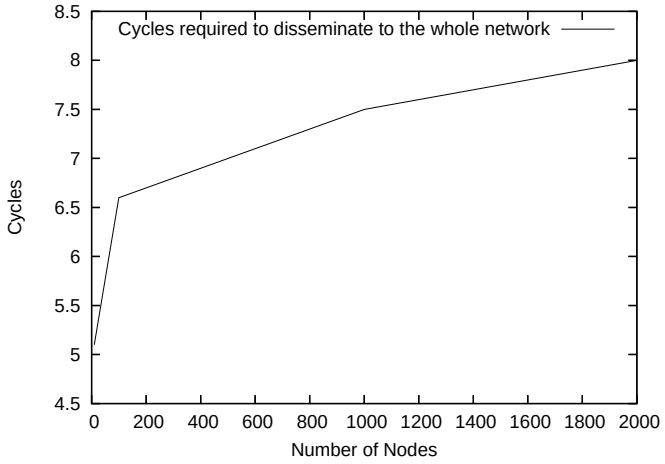

Fig. 1. Number of cycles required to disseminate a message.

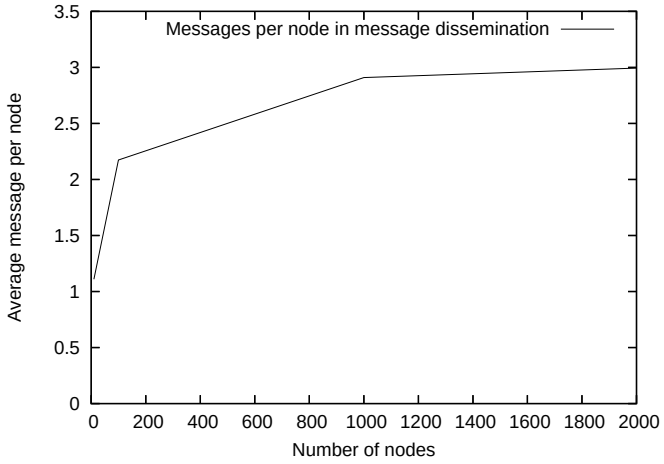

Fig. 2. Average number of messages sent or received by each node.

The greatest advantage of such a kind of simulation is its scalability, since details that would prevent the simulation of a large number of nodes are ignored. As the simulator is based on cycles, the concept of real time in the simulation does not exist. For example, it is possible to measure how many cycles it takes for a message to reach the entire network, but not the corresponding amount of time.

Initially, when the simulator is executed, n nodes are created and connected in a random topology. To execute the experiments, a power-law topology generator based on the GLP (Generalized Linear Preference) algorithm [6] was employed.

A. Experiments

The first experiment evaluates the cost and the latency to disseminate a message throughout the entire network. The experiment was performed by creating initially n nodes of a network with power-law topology, and later by making a random node disseminate a message. We measured the number of messages sent and received by each node in the dissemination, as well as the required number of cycles for the dissemination to reach the entire network. Assuming that all links have the same speed and latency, the greater the number of cycles, the more time is necessary for the dissemination to finish, and hence the less accurate the local topologies get.

Each test was performed 10 times. Tests with 10, 100, 1000 and 2000 nodes were executed. The graph in figure 1 shows the average number of cycles required in order to have the dissemination message received by every node in the network. The number of cycles depends basically of the network diameter, i.e. the distance between the node that disseminates the message and node with the greatest shortest distance to the source. The graph shows that the number of cycles increases quickly in networks with up to 100 nodes, but after that the number of cycles increases more slowly. The graph also shows that disseminations are propagated throughout the network in relatively few cycles.

In the graph shown in the figure 2 it is possible to see the average number of messages sent or received by each node, according to the size of the network. Since no node failed in the simulation, the total number of received messages is equal to the number of messages sent, as a sent message is always received by another node. Therefore, the graph representing the average number of messages sent is equal to the graph representing received messages. It can be noticed that from 1000 nodes, the number of messages employed increases slowly.

Nonetheless, the average number of received or sent messages does not present an accurate idea about the distribution of the number of messages sent or received by nodes. To investigate that, in the graph of figure 3 it is shown, along the x axis, 5 intervals, which are related to the amount of messages sent to disseminate a message in a network with 100 nodes. According to the graph, 67 nodes sent 0 or 1 message during the dissemination. On the other hand, 3 nodes sent between 21 and 50 messages, each one. That shows that most of the nodes sent few messages, and had a small burden in the dissemination, whereas few nodes received a great burden, and needed to send several messages. This behavior was expected due to inherent properties of power-law networks, which have few nodes with many neighbors and many nodes with low connectivity.

Figure 4 presents the same graph, but this time for a network with 1000 nodes. The same behavior of the previous graph is noticed. 760 nodes sent, each one, 0 or 1 messages during the dissemination, whereas 3 nodes sent more than 100 messages. The node that sent the greatest number of messages sent 148 messages. 12 nodes sent between 51 and 100 messages each, 15 nodes between 21 and 50 messages, 31 between 11 and 20, 46 between 6 and 10, and 134 nodes sent between 2 and 5 messages. The distribution of received messages is similar to the distribution of sent messages.

These results show that, despite the heterogeneous distribution of the dissemination cost of a message, the disseminations occurred in relatively few cycles, which guarantee that local representations of the topology converge relatively quickly to reflect the real topology.

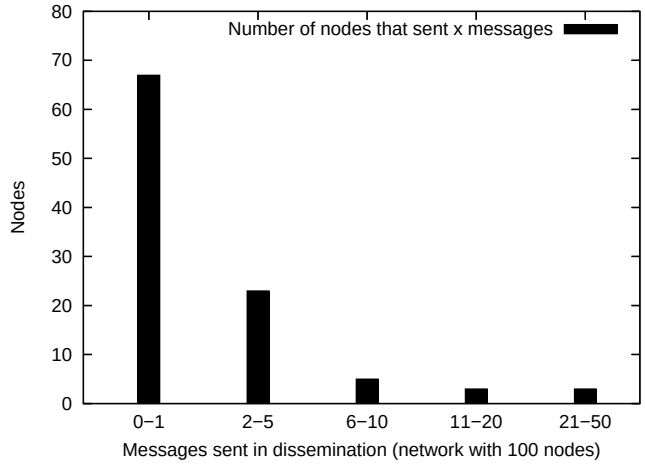

Fig. 3. Number of messages sent per node, in a network with 100 nodes.

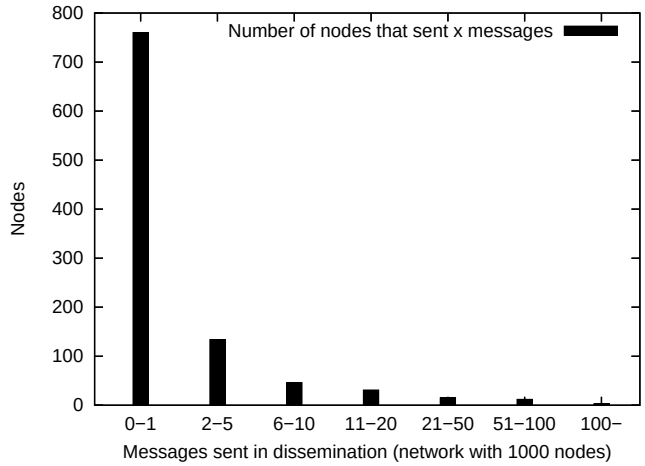

Fig. 4. Number of messages sent per node, in a network with 1000 nodes.

IV. Related Work

Several related work present solutions to the problem of topology discovery. The approaches presented in [9], [10] are concerned with topology discovery in the physical layer, instead of the application layer, and are also centralized. The solutions that come closer to our proposal are the ones based on mobile agents, such as the one presented in [11]. In those types of solutions, several mobile agents wander in a wireless network spreading information about the topology. However, the main problem of those solutions is the high latency of the topology convergence.

In [5] a distributed diagnosis algorithm is presented that allows nodes to keep diagnostic information about the topology, but the approach considers that the set of the nodes is known a priori, which means it is not possible to dynamically add and remove nodes from the network, as it happens in a P2P network. Moreover, the algorithm uses breadth-first trees in order to disseminate the events, and uses tree concatenations when two or more trees are received by the same node. This approach is unfeasible in dynamic environments, since the concatenation of trees could possibly never complete.

V. Conclusion

This work presented a strategy for topology discovery in dynamic and decentralized networks. The strategy is based on previous system-level diagnosis results, and uses tests to detect link events. After the occurrence of events, nodes disseminate the information throughout the network. Using a simulator, experiments were carried out, which allowed the evaluation of latency, precision and number of messages required. The experiments show that event disseminations occur in relatively few cycles, and that as the number of nodes increases, the number of cycles does not grow at the same rate. The experiments were executed on randomly generated networks with power-law topology, and show that node connectivity does have an impact on the number of messages sent and received by nodes in the network.

Future work includes implementing the system for real P2P network monitoring, as well as employing other hybrid techniques, such as intelligent agents, in order to disseminate messages.

References

[1] M. Castro, M. B. Jones, A. M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *Proc. IEEE INFOCOM*, vol. 2, 2003, pp. 1510–1520.

[2] K. Birman, A. Schiper, and P. Stephenson, "Lightweight causal and atomic group multicast," *ACM Transactions on Computer Systems*, vol. 9, no. 3, pp. 272–314, 1991.

[3] M. Ripeanu, A. Iamnitchi, and I. Foster, "Mapping the gnutella network," *IEEE Internet Computing*, vol. 6, no. 1, 2002.

[4] R. Bruno, M. Conti, and E. Gregori, "Mesh networks: commodity multihop ad hoc networks," *IEEE Communications Magazine*, vol. 43, no. 3, pp. 123–131, 2005.

[5] E. P. Duarte Jr. and A. Weber, "A distributed network connectivity algorithm," in *Proc. International Symposium on Autonomous Decentralized Systems*, 2003.

[6] T. Bu and D. Towsley, "On distinguishing between internet power law topology generators," in *Proc. IEEE INFOCOM*, vol. 2, 2002, pp. 638–647.

[7] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys & Tutorials*, pp. 72–93, 2005.

[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[9] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology discovery in heterogeneous IP networks," in *Proc. IEEE INFOCOM*, vol. 1, 2000, pp. 265–274.

[10] Y. Bejerano, Y. Breitbart, and M. Garofalakis, "Physical topology discovery for large multisubnet networks," in *Proc. IEEE INFOCOM*, vol. 1, 2003, pp. 342–352.

[11] R. RoyChoudhury, S. Bandyopadhyay, and K. Paul, "A distributed mechanism for topology discovery in ad hoc wireless networks using mobile agents," in *Proc. ACM International Symposium on Mobile Ad Hoc Networking & Computing*, 2000, pp. 145–146.