Intelligent Dynamic Network Reconfiguration

Juan Ramón Acosta and Dimiter R. Avresky

Northeastern University
Dept. of Electrical and Computer Engineering
Boston, MA 02115-5000 USA
{jracosta,avresky}@ece.neu.edu

Abstract

Dynamic network reconfiguration is a technique in which the routing tables of the nodes in the vicinity of a failure are updated in real-time. The technique has been proved effective only if no failures occur after the reconfiguration process has started.

This paper, presents enhancements to Agent NetReconf to allow it tolerate new failures if the reconfiguration was already started for a different failure. Agent NetReconf is an intelligent dynamic network reconfiguration algorithm. The improvements were made on the following three phases: Restoration Tree Construction (Phase 1), Multiple Failures synchronization (Phase 2) and Routing Information Update (Phase 3). The proposed strategy consists of: 1) Activate Agent NetReconf recursively, if a new node/link failure occurs and the reconfiguration of a different failure was started, 2) Use a pair of gateway nodes to help the restoration leaders, to reach consensus and to define the order in which each leader will execute the reconfiguration. The complexity, in terms of the number agents created, is analyzed for all phases. Termination is also proved for all phases.

1. Introduction

Building and deploying smart fault-tolerant networks have become the focus of several scientific groups and the industry in general. Applications and end users, expect the network to be able to handle failures transparently with minimum impact. For this, the scientific community has developed sophisticated techniques that use: intelligence [19, 17], knowledge [15, 21], active networking [20], combinatorics and game theory [5, 10].

1-4244-0910-1/07/$20.00 ©2007 IEEE.

Minar in [14], describes an algorithm to discover the topology of a network using mobile agents that travel around the network and cooperate with other agents. The found topology is then used to define the routing policies. Hood and Ji [9], proposed an intelligent software agent that performs fault detection accurately and in certain cases predicts them before they appear. Whit et al. [18], created communities of mobile agents that roam the network collecting and exchanging information based on the ”social insects” paradigm (ant behavior). Active networking [16] and social insects were combined to manage networks using a collection of smart agents. In [13], agents cooperate to provide high network connectivity and dynamic routing. Gianni [6] designed an adaptive routing algorithm based on mobile agents that learns the routing tables of a computer network. Garijo et al. [7] designed a centralized Multi-agent Cooperative Network-Fault Management system (CNFM), in which, the agents work as watchdogs and generate events into the CNFM engine when faults are recognized.

The novelty of *Agent NetReconf*, the proposed algorithm in this paper, resides on its ability to use knowledge and autonomous mobile agents to intelligently tolerate failures. The algorithm performs network reconfiguration based on the same principles used in *NetRec* [3, 4]. *Agent NetReconf* differentiates from *NetRec* [3] in several aspects: it is agent based and not message based, uses knowledge instead of synchronous messaging, has a lower runtime complexity, as proved in [2]. Finally, the algorithm can tolerate new failures even if the reconfiguration for a different failure started already, as shown in Section 2. The fault model supported by *Agent NetReconf* is known as “Fail-Silent”. Byzantine failures [11] are not considered in the algorithm fault model.

The rest of the paper is organized in three sections as follows: Section 2, Tolerating Failures During Reconfiguration, that describes the new enhancements to *Agent NetReconf*. Section 3, Presents the new algorithm properties where complexity and termination are proved. The last section are the conclusions.

2. Failures During Reconfiguration

2.1. Agent NetReconf Overview

Agent NetReconf [2], is a dynamic reconfiguration algorithm capable of tolerating multiple simultaneous network failures. The algorithm uses mobile agents to find restoration paths that re-establish connectivity on the network devices adjacent to the failed component. *Agent NetReconf* consists of four phases:

Phase 0, Restoration Leader Selection. It is the first phase and it is activated when a failure is detected by the Nodes Adjacent to the Failure ($NAFs$). The NAF with the highest ID is chosen as the restoration leader (RL.) RL coordinates the reconfiguration for failure F.

Phase 1, Restoration Tree Construction. In this phase, the restoration leader creates a set of exploration agents (E_{ij}), that are sent out to discover alternative routes to all the disconnected $NAFs$. As each E_{ij} explores the network, it cooperates with the visited nodes to share or learn new knowledge that then is used to make decisions on what is the best next hop towards the target NAF. If an E_{ij} reaches a NAF, then the NAF creates an agent explorer for restoration (ER_{ji}) and sends it back to RL as acknowledgment of the newly discovered restoration path. When the leader RL_i receives acknowledgment from all the $NAFs$, it declares the restoration tree established.

Phase 2, Multiple Failures Synchronization. After Phase 1 finishes, there could be more than one restoration tree RT_i intersecting at a common node. In this case, the algorithm establishes an ordered sequence of priorities in which the leader with the highest ID always executes *Phase 3* first, the other leaders will wait for their turn.

Phase 3, Routing Information Update. In this phase, the explorer and restoration agents, E_{ij} and ER_{ji}, visit each Node on the Restoration Tree (***NORTs***) and update the routing tables with new routing information. After this phase ends, the connectivity is re-established for all $NAFs$.

2.2. Algorithm Assumptions

Agent NetReconf as described in [2], runs successfully only if no failures occur after the reconfiguration process started. This condition reduces the effectiveness of the algorithm to a small number of cases when in practice faults can appear at any time. Therefore, in order to enhance *Agent NetReconf* we need to establish the following assumptions:

Assumption 2.1 *Only one additional failure is tolerated during reconfiguration of the network.*

Assumption 2.2 *All the adjacent nodes are sending each other "I am alive" messages.*

Assumption 2.3 *Any type of failure detected by hardware or software will cause the node or link to become "Fail-Silent" (i.e the node stops sending or receiving "I am-alive" messages.)*

Assumption 2.4 *Failures can appear at any time.*

Assumption 2.5 *The fault model does not consider Byzantine Failures [11, 12].*

Assumption 2.6 *If a restoration leader RL_i fails, regardless of the phase, the algorithm will re-start from the beginning.*

Assumption 2.7 *A NAF saves the path traveled by the first arriving explorer agent E_{ij} on each active port(link). In contrast with Agent NetReconf [2] and Net Rec [3], the paths are saved in a priority queue, based on the time of arrival.*

2.3. Tolerating Failures in Phase 1

If a failure appears during *Phase 1*, then it is necessary to consider three different scenarios: 1) *There are no restoration paths yet established between the leader and the $NAFs$*, 2) *The acknowledgment for a restoration path is in transit towards the leader* and 3) *There are already restoration paths established between the leader and the $NAFs$.* A restoration tree is considered confirmed only when the next conditions are true: a) The leader RL_i received a restoration agent ER_{ji}, one from each NAF, acknowledging that a path has been established and b) Each NAF receives a *Restoration Tree Built (RTB)* message from the leader. A path on the tree is a "confirmed restoration path." Messages such as RTB, are sent over a restoration path using explicit path routing.

No Restoration Paths Established between Leader and NAFs. Consider that (RL_1) is executing phase 1, as shown in Fig. 1, and that it already sent exploration agents (E_{1j}) to look for paths that reconnect NAF_1. Then assume, that an instant later, a new failure (F_D) appears, and that (RL_D) was selected leader. During phase 1, RL_D sends exploration agents (E_D) to look for paths that reconnect (NAF_Ds).

When the search for restoration paths is in progress, an explorer E_D can arrive to leader RL_1. Suppose that no restoration path, so far, has been established between RL_1 and NAF_1. In this case, the visit of E_D does not cause a state change or another reaction in RL_1, because RL_1 has not yet received acknowledgments on how to reach a NAF_1. Therefore, E_D will migrate out from the leader to continue looking for a NAF_D. Now, in case that an explorer agent E_1 arrives to a node next to failure F_D, the explorer agent will exchange knowledge with the visited node and use this information to chose the best hop towards NAF_1. The failed node will be discarded because E_1 cannot move there, and the agent will chose a different route. In conclusion, the explorer agents will continue searching as long as the network is not partitioned. Therefore, the

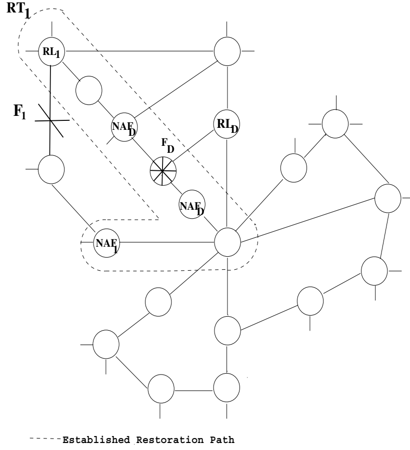

Figure 1. Failure During Reconfiguration

leaders will reach the $NAFs$ and reconnect them as specified in *Agent Net Reconf* [2].

The Acknowledgment for a Restoration Path is in Transit Towards the Leader. Assume that a path between the leader RL_1 and the NAF_1 was successfully found. Consider NAF_1 to have sent a restoration agent (ER_1) to acknowledge the newly found path. After the restoration agent leaves, the NAF_1 starts a timer T_{conf} to wait for the confirmation that the leader received the restoration agent. The restoration agent ER_1, will travel towards the leader over the restoration path using explicit path routing. Then while ER_1 is in transit towards RL_1, a new failure F_D may occur on RT_1, as shown in Fig. 1. The following cases should be considered:

1. The restoration agent ER_1 is intercepted by an explorer agent E_D at a $NORT$ on the path. In this case, both agents exchange knowledge and if ER_1 learns that the failure is on the restoration path towards RT_1. ER_1 deactivates itself immediately and E_D continues traveling towards the NAF_1.

2. The restoration agent ER_1 never meets an explorer agent E_D while it is traveling towards RL_1. In this case, there are two possibilities: 2.a) The explorer agent is before the failure or 2.b) the agent is after the failure. When ER_1 is before the failure, the agent will be immediately deactivated as soon as the agent reaches the $NORT$ adjacent to the failure. If ER_1 is after the failure, then the agent will travel until it reaches RL_1. The leader, RL_1 will send a RTB message towards NAF_1 using explicit path routing, as defined in [2]. The message will stop traveling further, when it reaches the node adjacent to the failure. The $NORT$ will drain the message and it will send a *Delivery Failed Message* (DFM) to the leader. As soon as the leader receives the DFM message, it discards the established restoration path and the explorer ER_1, the agent that confirmed the failed restoration path, immediately deactivate itself.

3. The leader RL_1 will not send a RTB confirmation to the NAF_1, unless it is notified of a new alternative restoration path towards NAF_1. In other words, a new restoration agent $ER_1^{'}$ should arrive to the leader. If no agent arrives in a given time, then the leader removes the NAF_1 from the restoration tree.

4. If no confirmation from RL_1 was received and the timer T_{conf} expires then the NAF_1 removes the current restoration path from the priority queue, creates a new restoration agent $ER_1^{'}$ that sends towards RL_1. The restoration agent travels to the leader via the alternative restoration path using explicit path routing.

Restoration Paths Already Established between Leader and NAFs. Assume that the restoration paths between the leader RL_1 and the $NAFs$ are already established before the leader RL_D started the recovery of failure F_D and sent the explorer agents E_D looking for restoration paths to NAF_D. From this scenario, the following cases should be considered:

1. An explorer agent E_D arrives to a NAF_1. The NAF_1 and the explorer agent exchange knowledge. If the NAF_1 determines from E_D, that failure F_D is on the established restoration path towards RL_1 then the NAF_1 removes the current path from the priority queue, described in Assumption 2.2. Then, using a new alternative restoration path, the NAF_{F1} creates a new restoration agent $ER_1^{'}$, sends the agent towards RL_1, and it starts a timer T_{conf} to wait for the confirmation that the leader received $ER_1^{'}$. The new restoration agent $ER_1^{'}$ will travel over the alternative restoration path using explicit path routing.

2. An explorer agent E_D arrives to a leader RL_1. The leader and the explorer agent exchange knowledge. If RL_1 determines from E_D, that failure F_D is on the established restoration path towards NAF_1 then the leader deletes the current established path and marks the NAF_1 as pending for acknowledgment. ER_1 deactivates itself immediately.

3. The leader RL_1 will not send a RTB confirmation to the NAF_1, unless it is notified of a new alternative restoration path towards NAF_1. In other words, a new restoration agent $ER_1^{'}$ should arrive to the leader. If no agent arrives in a given time, the leader removes the NAF_1 from the restoration tree.

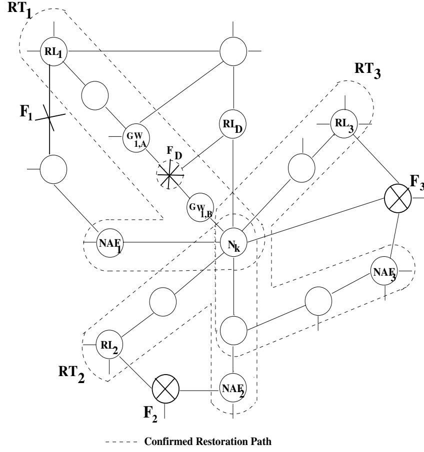

Figure 2. Failure During Multiple Tree Synchronization

4. If no confirmation from RL_1 was received and the timer T_{conf} expires then the NAF_1 removes the current restoration path from the priority queue, creates a new restoration agent $ER_1^{'}$ that sends towards RL_1.

2.4. Tolerating Failures in Phase 2

For situations in which multiple failures appear, *Agent NetReconf* creates a restoration tree per failure. The trees can intersect each other, as shown in Fig. 2. In phase 2, the standard algorithm executes a synchronization step that establishes an ordered sequence. The sequence begins with the leader with the highest ID, and ends with the leader with the lowest ID. If a new failure occurs, during the tree synchronization, the sequence is invalidated and the reconfiguration currently in progress gets suspended. Then, *Agent NetReconf* is invoked recursively to deal with the new failure. When the recovery for the new failure completes, there is the possibility, that the original tree intersection was modified (e.g. a new tree joins) or that the intersection is completely dissolved. Before proceeding with the explanation of the new enhancements, we need make the following definitions:

Definition 2.1 *Gateway Node (GW), It is a NORT adjacent to the failed node/link on the confirmed restoration tree.*
Definition 2.2 *Intersection Node (N_k), It is a NORT on which multiple restoration trees RT_i overlap.*

Assume that restoration trees (RT_1),(RT_2) and (RT_3) intersect at node (N_k), and that a new failure (F_D) appears on RT_1, as shown in Fig. 2. When the failure is detected two things happen in parallel: 1) The $NORTs$ adjacent to the failure become gateway nodes and 2) *Agent NetReconf* is recursively activated for F_D.

As shown in Fig. 2, the gateways $GW_{1,A}$ and $GW_{1,B}$ were created after F_D was detected on the restoration tree RT_1. Immediately after creation, each gateway sends out a point-to-point *Notify Tree Failure Event (NTFE)*. $GW_{1,A}$ sends the message towards the leader RL_1 and $GW_{1,B}$ sends the message towards the NAF_1. As the $NTFE$ event travels using explicit path routing, the following actions take place when a node receives the event:

1. A $NORT_i$, marks the confirmed restoration path to RL_i discardable and un-marks the links that were indicated to belong to the restoration tree.

2. A leader RL_i, suspends phase 2 immediately until it receives a signal to continue.

3. A NAF_i, marks the confirmed restoration path to RL_i discardable.

4. The intersection node N_k, sends a copy of the $NTFE$ event to the other trees in the intersection. N_k starts a timer $T_{re-built}$ to wait for RL_1 to confirm that the restoration tree RT_1 was re-built. The leader RL_1 is expected to send an event if the tree is recovered.

As mentioned earlier, at the same time the gateways do their work, *Agent NetReconf* is ran recursively to recover the new failure. Consider (RL_D) to be the restoration leader selected for failure F_D. Assume that RL_D already send out exploration agents (E_D) to look for restoration paths towards the $NAFs_D$. Therefore, the following cases take place as the explorer agents visit nodes during their search on the network:

1. An explorer agent E_D arrives to the restoration leader RL_1. The leader exchanges knowledge with the explorer agent. If RL_1 finds that the failure F_D occurred on the confirmed restoration path towards NAF_1 then the leader immediately removes the restoration path and re-starts the timer T_{ack}. RL_1 will use the timer to wait for a new restoration agent $ER_1^{'}$ to confirm a new alternative restoration path for NAF_1. If T_{ack} expires the NAF_{F1} is excluded from the restoration altogether, as described by *Agent NetReconf* [2] then the leader deactivates the restoration agent ER_1 that initially confirmed the failed restoration path.

2. An explorer agent E_D arrives to the NAF_1. The NAF_1 exchanges knowledge with the explorer agent. If the NAF_1 finds that the failure F_D occurred on the confirmed restoration path towards RL_1 then the NAF_1 removes the current restoration path from the priority list and uses the new alternative restoration path to send a new restoration agent $ER_1^{'}$ towards

RL_1. The restoration agent travels the network using explicit path routing.

3. An explorer agent E_D arrives to a NAF_1. The NAF_1 and the explorer agent exchange knowledge. If the NAF_1 determines from E_D, that failure F_D is on the confirmed restoration path towards RL_1 then the NAF_1 removes the current path from the priority queue. Then, using a new alternative restoration path, the NAF_{F1} creates a new restoration agent $ER_1^{'}$, sends the agent towards RL_1, and it starts a timer T_{conf} to wait for the confirmation that the leader received $ER_1^{'}$. The new restoration agent $ER_1^{'}$ will travel over the alternative restoration path using explicit path routing.

4. The leader RL_1 will not send a RTB confirmation to the NAF_1, unless it is notified of a new alternative restoration path towards NAF_1. In other words, a new restoration agent $ER_1^{'}$ should arrive to the leader. If no agent arrives in a given time then the leader removes the NAF_1 from the restoration tree.

5. If no confirmation from RL_1 was received and the timer T_{conf} expires then the NAF_1 removes the current restoration path from the priority queue, creates a new restoration agent $ER_1^{'}$ that sends towards RL_1.

The recovery of failure F_D will proceed to completion following the principles defined for *Agent NetReconf.* When the failure is recovered, the connectivity between $GW_{1,A}$ and $GW_{1,B}$ is re-established automatically. This is possible since the gateways are also $NAFs$ on the restoration tree RT_D. After the restoration tree RL_1 has also been rebuilt, the leader RL_1 needs to re-synchronize with the intersection node (N_k). Remember, that all the leaders involved in the intersection were suspended and cannot continue with phase 2. In addition, there is the possibility that the intersection's leadership may have changed and a new ordered sequence among the leaders may be necessary. For example, RT_1 could had left the intersection and RT_D could have joined.

To re-synchronize RL_1 and N_k, the leader will send a *Restoration Tree Recovered Message (RTRM)*. The message carries information about the new restoration tree RT_1. The intersection node N_k, will respond to RL_1 in one of the following ways: 1) If N_k determines that RT_1 is not longer part of the original intersection then it sends a *No Intersection Found Message (NIFM)* , 2) If the tree RT_1 participates on the intersection then N_k sends the *Tree Intersection Found Message (TIFM)*. In case RL_1 never sends the $RTRM$ message, N_k will remove RT_1 from the intersection and it will notify the other leaders to re-synchronize and re-start phase 2.

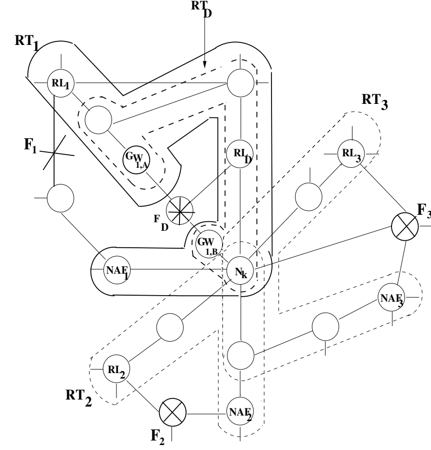

Figure 3. Re-confirmed Restoration Tree

The leader RL_1, will process the response from N_k as follows: If the leader receives a $NIFM$ message then it immediately resumes phase 2 and continues independently on its own. However, if it receives a $TIFM$ message then the leader makes itself ready to re-initiate phase 2 to participate on the definition of a new intersection leadership. As an example, assume that RT_D joined the intersection and has the highest ID, as shown in Fig. 3. In this case, RL_1 will execute phase 2 after RL_D finishes and it will be subject to the new established ordered sequence.

2.5. Tolerating Failures in Phase 3

The update to the routing tables, as defined in [2], occur as follows: After a restoration tree RT_i has been confirmed and synchronized, the leader RL_i signals the restoration agents ER_{ji} to start traveling back towards the $NAFS_i$, following the restoration path. The restoration agent uses explicit path routing to reach its destination. The restoration agents ER_{ji}, as they arrive to a node, make the current node to update its routing tables using the connectivity information specified in the confirmed restoration path. The updates made by the restoration agents, correspond to routes that allow traffic from the leader RL_i to the NAF_i. The restoration agent continues traveling until it reaches the NAF_i. After ER_{ji} arrival, the NAF_i signals the explorer agents E_{ij} to start traveling back towards the leader RL_i following the restoration path. The explorer agents work in the same way as the restoration agents did. In this case, the explorer agents will make updates that correspond to routes that allow traffic from the NAF_i to the leader RL_i.

Therefore, if a failure F_D occurs during phase 3, there will be several unusable routers with partially updated tables. In this case, the reconfiguration must be stopped and

any changes made to the routing tables must be undone. Then, in order to return the routing tables to how they were before Phase 3 was initiated, the following enhancements are proposed:

1. A $NORT$ will use a LIFO list to save the current value of an entry, in the routing table, for which a change is requested by a visiting explorer E_{ij} agent or restoration ER_{xi} agent.
2. When a Failure F_D is detected on a confirmed restoration path, a pair of gateways $GW_{1,A}$ and $GW_{1,B}$ will be created, as shown in Fig. 3.
3. Each gateway $GW_{1,A}$ and $GW_{1,B}$, will send a point-to-point *Notify Tree Failure Event (NTFE)* on the opposite direction of the failure. For example, $GW_{1,A}$ will send the message towards the leader and $GW_{1,B}$ will send the message towards the NAF_i.
4. A $NORT$ that receives a *NTFE* event, will rollback the affected entries, using the last saved value in the LIFO list, and will make itself ready to start phase 2.
5. Either a leader RL_i or NAF_i that receives a *NTFE* event, replies a gateway with a point-to-point *Change Rolled Back Event (CRBE)*. This event is understood by a gateway node as a confirmation that the routing tables were rolled back to the last saved value and that the execution of the algorithm for the original failure has safely transitioned to the beginning of Phase 2.

At the same time the gateways initiate the process of rolling back the routing tables, *Agent NetReconf* is ran recursively to recover the new failure. Consider (RL_D) to be the restoration leader selected for failure F_D. Assume that RL_D already sent out exploration agents (E_D) to look for restoration paths towards the $NAFs_D$. Therefore, the following cases take place as the explorer agents visit nodes during their search on the network:

1. An explorer agent E_D arrives to the restoration leader RL_1. The leader exchanges knowledge with the explorer agent. If RL_1 finds that the failure F_D occurred on the confirmed restoration path towards NAF_1 then the leader immediately removes the restoration path and re-starts the timer T_{ack}. RL_1 will use the timer to wait for a new restoration agent $ER_1^{'}$ to confirm a new alternative restoration path for NAF_1. If T_{ack} expires the NAF_{F1} is excluded from the restoration altogether, as described by *Agent NetReconf* [2] then the leader deactivates the restoration agent ER_1 that initially confirmed the failed restoration path.
2. An explorer agent E_D arrives to the NAF_1. The NAF_1 exchanges knowledge with the explorer agent. If the NAF_1 finds that the failure F_D occurred on the confirmed restoration path towards RL_1 then the NAF_1 removes the current restoration path from the priority list and uses the new alternative restoration path to send a new restoration agent $ER_1^{'}$ towards RL_1. The restoration agent travels the network using explicit path routing.
3. An explorer agent E_D arrives to a NAF_1. The NAF_1 and the explorer agent exchange knowledge. If the NAF_1 determines from E_D, that failure F_D is on the confirmed restoration path towards RL_1 then the NAF_1 removes the current path from the priority queue. Then, using a new alternative restoration path, the NAF_{F1} creates a new restoration agent $ER_1^{'}$, sends the agent towards RL_1, and it starts a timer T_{conf} to wait for the confirmation that the leader received $ER_1^{'}$. The new restoration agent $ER_1^{'}$ will travel over the alternative restoration path using explicit path routing.
4. The leader RL_1 will not send a RTB confirmation to the NAF_1, unless it is notified of a new alternative restoration path towards NAF_1. In other words, a new restoration agent $ER_1^{'}$ should arrive to the leader. If no agent arrives in a given time then the leader removes the NAF_1 from the restoration tree.
5. If no confirmation from RL_1 was received and the timer T_{conf} expires then the NAF_1 removes the current restoration path from the priority queue, creates a new restoration agent $ER_1^{'}$ that sends towards RL_1.

The recovery of failure F_D will proceed to completion following the principles defined for *Agent NetReconf*. When the failure is recovered, the connectivity between $GW_{1,A}$ and $GW_{1,B}$ is re-established automatically. This is possible since the gateways are also $NAFs$ on the restoration tree RT_D. After the restoration tree RL_1 has also been rebuilt, the leader RL_1 needs to re-synchronize with the intersection node (N_k). Remember, that all the leaders involved in the intersection were suspended and cannot continue with phase 2. In addition, there is the possibility that the intersection's leadership may have changed and a new ordered sequence among the leaders may be necessary. For example, RT_1 could had left the intersection and RT_D could have joined.

To re-synchronize RL_1 and N_k, the leader will send a *Restoration Tree Recovered Message (RTRM)*. The message carries information about the new restoration tree RT_1. The intersection node N_k, will respond to RL_1 in one of the following ways: 1) If N_k determines that RT_1 is not longer part of the original intersection then it sends a *No Intersection Found Message (NIFM)* , 2) If the tree RT_1 participates on the intersection then N_k sends the *Tree Inter-*

section Found Message (TIFM). In case RL_1 never sends the $RTRM$ message, N_k will remove RT_1 from the intersection and it will notify the other leaders to start phase 3 according to the sequence of priorities defined in phase 2.

The leader RL_1, will process the response from N_k as follows: If the leader receives a $NIFM$ message then it immediately resumes phase 2 and continues independently on its own. However, if it receives a $TIFM$ message then the leader makes itself ready to synchronize with the other leaders to define a new intersection leadership. The synchronized leader RL_1, will be assigned a new order with in the sequence. After the re-synchronization is completed, execution of phase 3 will be started as defined in [2].

3. Algorithm Properties

3.1. Complexity

The complexity for *Agent NetReconf* is given in terms of the number of explorer agents created during restoration tree construction and routing table reconfiguration. The complexity of the algorithm is as follows:

Let L_{Active} be the number of active links on each router, n_{naf} the number of *NAFs* for failure *F*, and P a path between RL_F and a NAF.

Theorem 3.1 *The complexity for Agent NetReconf for multiple failures when there are no failures during the reconfiguration is given by*

$$O\Bigg(F * \big(L_{Active} * \ ((n_{max} * P_{max}) + 1)\big)\Bigg) \qquad (1)$$

where F is the total number of failures in the network, P_{max} is the longest path connecting RL_F and any NAF and n_{max} is the maximum number of NAFs. The proof is described in [2].

Theorem 3.2 *The complexity for Agent NetReconf when there are failures during the reconfiguration is given by*

The complexity when a failure occurs during Phase 1

$$O\Bigg(n_{max}\Bigg) \qquad (2)$$

The complexity when a failure occurs during Phase 2

$$O\Bigg(L_{Active} * \ ((n_{max} * P_{max}) + 1)\Bigg) \qquad (3)$$

The complexity when a failure occurs during Phase 3

$$O\Bigg(L_{Active} * \ ((n_{max} * P_{max}) + 1)\Bigg) \qquad (4)$$

Proof: As shown in [2], *Agent NetReconf* determines RL_F without creating explorer agents such that leader selection is achieved with $O(0)$ complexity, no agent are created.

For Phase 1, the complexity of tolerating a failure (F_D) appears before the restoration tree as been established is defined as follows: For the case in which there are no paths established between the leader and the $NAFs$, no agents need to be created such that the additional complexity is $O(0)$. However, if path has been established, a new restoration agent ER_{ik} will have to be created to re-establish the path. The cost for this is $O(1)$. Now taking the worse case in which all the restoration paths of three need to be re-established the total complexity for Phase 1, is given by $O(n_{max})$.

For phase 2. If a new failure appears on a confirmed restoration tree RT_i, that is member of an intersection then the reconfiguration process for the intersection is suspended until the recovery of the new failure completes. Now, since the gateway nodes, GW_{1A} and GW_{1B}, participate actively on the reconfiguration of the new failure and the original reconfiguration cannot continue until they are reconnected, the additional complexity due to the new failure is $O(L_{Active} * \ ((n_{max} * P_{max}) + 1))$. In addition, to re-establish original the restoration tree RT_i is necessary to create in the worse case $O(n_{max})$ restoration agents, if all the restoration paths in RT_i were assumed to fail. Therefore, the total complexity for recovering a failure while Phase 2 is running is given by $O(L_{Active} * \ ((n_{max} * P_{max}) + 1)) + O(n_{max})$.

For phase 3. The case in which the update of the routing tables is in progress and a new failure occurs. It is necessary to interrupt immediately the algorithm and rollback any changes made up to time of failure. The worse case is that in which the restoration tree RL_i, the tree on which the failure appeared, participates in a multiple tree intersection. As mentioned in section 2.5, the two nodes adjacent to the failure become $NAFs$ on the restoration process of the new failure which has a cost of $O(L_{Active} * ((n_{max} * P_{max}) + 1))$. Now, since the tree RT_i needs to be restored, then according to the algorithm enhancements proposed for Phase 3. It will be required, O(1) agents to re-establish the tree using an alternative restoration path. In case that all of the restoration paths failed it will be required a total of $O(n_{max})$. Then the total complexity is given by $O(L_{Active} * \ ((n_{max} * P_{max}) + 1)) + O(n_{max})$.
Q.E.D □

3.2. Termination

The following message delivery properties are used for proving *Agent NetReconf's* Termination.

Definition 3.1 *If a point-to-point message is sent from a*

source agent S to a destination agent D, then it will be received once and only once by D.

Definition 3.2 *Every point-to-point message sent between an exploration agent E_{ij} or ER_{xi} and a node manager agent NM_x will be routed following a path on the restoration tree and will be reliably delivered to its destination.*

Lemma 3.1 *Agent NetReconf* Phase 1 will successfully complete even when a new failure F_D occurs during the reconfiguration of a preexisting failure F_1.

Proof: For a given failure F_1, *Agent NetReconf*'s Phase 1 starts with restoration leader RL_1 creating L_{Active} explorer agents E_1, that begin searching for restoration paths towards each NAF_1. Considering that a second failure F_D appears before a restoration path between RL_1 and NAF_1 has been established, and that exploration agents E_D, for the second failure, are also searching for NAF_D. It can be observed, as shown in section 2.4, that the presence of an agent E_D at the leader RL_1 or the NAF_1, will not cause any changes on their behavior, with respect to the original reconfiguration. This assures that the restoration tree RT_1 will be established and the reconfiguration will transition to Phase 2. Now, in case the failure F_D appears on an already established restoration path between RL_1 and NAF_1, the arrival of an agent E_D at the leader RL_1 or the NAF_1, will cause the current path to be discarded and replaced by a new alternative restoration path. Therefore, it is assured that a unique restoration tree will be always established and that the algorithm will transition to Phase 2.
Q.E.D □

Lemma 3.2 *Agent NetReconf* Phase 2 will successfully complete even when a new failure F_D occurs during the reconfiguration of a preexisting failure F_1.

Proof: Assume that multiple restoration trees, (RT_1), (RT_2) and (RT_3), intersect at node (N_k), and that a new failure (F_D) appears on RT_1. As described in Section 2.4, after the failure is detected, the $NORTs$ adjacent to the failure become gateway nodes and *Agent NetReconf* is recursively activated for F_D. The gateways will send a *Notify Tree Failure Event (NTFE)* on apposite direction to the failure. Based on Def. 3.2, the $NTFE$ message is reliably delivered only once to each node and according to Def. 3.1 the message travels using explicit path forwarding. Now, if N_k receives $NTFE$ it will start a timer $T_{re-built}$ to wait for RT_1 to be reconfirmed. In case, $T_{re-built}$ expires and the failed tree RT_1 it is not re-established, the tree is removed from the intersection. Now, since *Agent NetReconf* was ran recursively for failure F_D, there will be explorer agents E_D traveling the network searching for $NAFs_D$ to reconnect. The restoration leader RL_1, when visited by an E_D will discard the restoration path, towards NAF_1, on which F_D appeared and it will start a timer T_{ack} waiting for an alternative restoration path confirmation. If the timer T_{ack} expires, then the leader removes the NAF_1 from the reconfiguration as described in [2]. Likewise, if a NAF_1 is visited by an E_D and the failure F_D is on the restoration path towards RL_1. the node will discard the path. The NAF_1 will send a new restoration agent with an alternative restoration path towards the leader. If the NAF_1 does not receives an *Restoration Tree Built* message from the leader, the NAF_1 as described in [2] will remove itself from the reconfiguration. Based on these facts, it is proved that Phase 2 will successfully terminate, since the algorithm will always reach an state in which none of the peers communicating on the reconfiguration will wait for each other and the participants of the reconfiguration will reach consensus.
Q.E.D □

Lemma 3.3 *Agent NetReconf* Phase 3 will successfully complete even when a new failure F_D occurs during the reconfiguration of a preexisting failure F_1.

Proof: Assume that during the execution of Phase 3, for failure F_1, a new failure F_D appears on the restoration tree RT_1. As described in Section 2.4, after the failure is detected, the $NORTs$ adjacent to the failure become gateway nodes and *Agent NetReconf* is recursively activated for F_D. The gateways will send a *Notify Tree Failure Event (NTFE)* on the opposite direction to the failure to initiate rolling back the routing tables to their last saved value. Based on Def. 3.2, the $NTFE$ message is reliably delivered only once to each node and according to Def. 3.1 the message travels using explicit path routing. Now since, to tolerate a failure in Phase 3 it is required that the algorithm to transition back to Phase 2 and the re-run Phase 3, it can be conclude that Lemma 3.2 also proves that Phase 3 terminates.
Q.E.D. □

Theorem 3.3 *On all nodes, Agent NetReconf will successfully complete when, a new failure F_D occurs during the reconfiguration of the network due to multiple failures.*

Proof: Based on Lemmas 3.1 - 3.3, it can be concluded that the RL_i and the *NAFs* will proceed with all phases of *Agent NetReconf* and will generate the required explorer agents to carry out the establishment of the restoration tree and the reconfiguration of each node (RL_F, *NAFs* and *NORTs*) on the tree even when, an additional failure F_D occurs during the reconfiguration in the network due to the multiple failures.
Q.E.D. □

4. Conclusions

The paper presented enhancements to *Agent NetReconf*, that allow the algorithm to tolerate failures during the reconfiguration process initiated for a different failure. The complexity analysis demonstrated that the additional overhead per failure is equal to the complexity incurred by *Agent NetReconf* for a single failure as shown in [2]. The factors that contribute on the achievements of this complexity are as follows: a) The interactions between agents occur at each router and the number of point-to-point non in-router communications are minimal b)The agents share knowledge dynamically on each interaction when they meet.

To conclude, *Agent NetReconf* is a low complexity, intelligent distributed dynamic network reconfiguration algorithm that is capable of tolerating failures during the recovery of an initial failure. The algorithm is applicable to computers with arbitrary topologies, is application-transparent and is capable of isolating and tolerating multiple faulty links or nodes.

5 Future Work

The authors structured the design and implementation of the algorithm in three stages. The first stage, is represented by the work published in [2]. The second stage, is the work presented in this paper. In the third stage, the authors will elaborate a performance characterization of *Agent NetReconf* and will compare it with exiting message based dynamic reconfiguration algorithms. The throughput, latency and saturation will be analyzed. The "Safety" and "Liveliness" properties of the algorithm will proved as well.

References

[1] M. A. and T. C. Raynal M. Time-free and timer-based assumptions can be combined to obtain eventual leadership. *IEEE Transactions on Parallel and Distributed Systems*, 17(7), July 2006.

[2] J. R. Acosta and D. R. Avresky. Dynamic Network Reconfiguration in Presence of Multiple Node and Link Failures Using Autonomous Agents. In *2005 IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, San Jose, CA,USA*, pages 10 – 20. IEEE Computer Society, December 2005.

[3] D. Avresky and N. Natchev. Dynamic Reconfiguration in Computer Clusters with Irregular Topologies in the Presence of Multiple Node and Link Failures. *IEEE Transactions on Computers*, 55(2), May 2005.

[4] D. R. Avresky, N. H. Natchev, and V. Shurbanov. Dynamic reconfiguration in high-speed computer clusters. In *2001 IEEE International Conference on Cluster Computing (CLUSTER 2001),Newport Beach, CA, USA*, page 380, Oct. 2001.

[5] R. Bhandari. *Survivable Networks: Algorithms for Diverse Routing*. Kluwer Academic Publishers, 1999.

[6] G. D. Caro and M. Dorigo. Mobile Agents for Adaptive Routing. In *Proceedings of 31st International Conference on System Sciences (HICSS-31)*, 1998.

[7] M. Garijo, A. Cancer, and J. Sanchez. A Multi-Agent System for Cooperative Network-Fault Management. In *Proceedings of the First International Conference and Exhibition on the Practical Applications of Intelligent Agents and Multi-agent Technology*, pages 279 – 294, 1996.

[8] M. Heusse, S. Gu'erin, D. Snyers, and P. Kuntz. Adaptive Agent-Driven Routing and Load Balancing in Communication Networks. *Complex Systems*, 1998.

[9] C. S. Hood and C. Ji. Intelligent Agents for Proactive Fault Detection. *IEEE The Internet Computing*, 2(2):65–72, March – April 1998.

[10] R. La and V. Anantharam. Optimal routing control: Game theoretic approach. In *Proceedings of the CDC Conference.*, 1997.

[11] L. Lamport, Shostak, and Pease. "The Byzantine Generals Problem. In *Advances in Ultra-Dependable Distributed Systems*. IEEE Computer Society Press, 1995.

[12] J.-P. Martin and L. Alvisi. Fast Byzantine Consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202 – 215, July - September 2006.

[13] N. Minar, K. H. Kramer, and P. Maes. *Cooperating Mobile Agents for Dynamic Network Routing*, chapter 12. Springer-Verlag, 1999. ISBN: 3-540-65578-6.

[14] N. Minar, K. H. Kramer, and P. Maes. Cooperating Mobile Agents for Mapping Networks. In *Proceedings of the First Hungarian National Conference on Agent Based Computation*, 1999.

[15] H. S. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11(3):205–244, Oct./Nov. 1995.

[16] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[17] G. Weiss. *Multi Agent Systems, A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 2001. ISBN: 0-262-23203-0.

[18] T. White, A. Bieszczad, and B. Pagurek. Distributed Fault Location in Networks Using Mobile Agents. In *IATA 1998,Proceedings of the Second International Workshop on Intelligent Agents for Telecommunication*, volume 1437, 1998.

[19] M. J. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, June 1995.

[20] Y. Yemini and S. daSilva. Towards programmable networks. In *Proceedings of IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, 1996.

[21] P. Zhang and Y. Sun. A New Approach Based on Mobile Agents to Network Fault Detection. In *ICCNMC'01, Proceedings of the International Conference on Computer Networks and Mobile Computing*, 2001.