Mobile Agent Communication in Highly Dynamic Networks: A Self-Adaptive Architecture inspired by the Honey Bee Colony

Phuong T. Nguyen[1,*], Volkmar Schau[2], Wilhelm R. Rossak[2]

[1]Research and Development Center, Duy Tan University, Da Nang, Vietnam
[2]Department of Computer Science, Friedrich-Schiller-Univerität Jena, Ernst-Abbe-Platz 2-4, D-07743 Jena, Germany

Abstract

Communication is considered as a building block for mobile agent systems. In highly dynamic networks, thanks to environmental stimuli such as changes in connection quality and network topology, performance of communication between mobile agents may be degraded considerably. With focus on attaining fault tolerance and reliability, we propose a context-aware architecture for agent communication model inspired by the honey bee colony. To validate the hypothesis, a software prototype has been designed and implemented according to the proposed mechanism. Encouraging experimental results on a test system show that our approach brings benefits to a colony of agent platforms.

Received on 29 October 2014; accepted on 28 November 2014; published on 16 December 2014

Keywords: Context-Awareness, Mobile Agents, Agent Communication

doi:10.4108/casa.1.2.e5

1. Introduction

Communication between mobile agents has been identified as an important issue in mobile agent technology, alongside other technical aspects of mobile agent technology, such as migrations and control mechanisms [9],[10],[11]. Under certain circumstances mobile agents can benefit from sending communication messages instead of migrating to remote platforms [9],[13].

For systems working in highly dynamic networks, agent communication is expected to adapt to environmental stimuli which appear only at execution time. Communication should deal not only with interoperability and location-transparency, but also with adaptability. Several studies have been published so far which address the design of an appropriate, highly flexible model for mobile agent communication. However, to the best of our knowledge none of the suggested solutions has been able to achieve the necessary performance and quality attributes to count as a practical solution. In most cases, these existing approaches seem to neglect the inherent dynamics of modern networks. Thus, in our view adaptive mobile agent communication remains a challenging topic. We are interested in developing an adaptive communication model for mobile agents which is suitable for tasks in highly dynamic networks.

In this paper we present an adaptive communication for mobile agents in highly dynamic networks. The main objective is to develop a model for agent communication that supports fault tolerance and reliability as the key qualities. Being inspired by the exotic behaviours of the honey bee colony, we suggested building a self-organizing network in the agent platform colony. Based on the proposed theory, we design, implement a software prototype for supporting mobile agent communication in highly dynamic networks. To validate our hypothesis, we also perform some evaluations for the software prototype in a laboratory scale test system.

The paper is organized as follows. In Section 2 we present the motivations as well as the aims of our work. Background on swarm intelligence is going

*Corresponding author. Email: phuong.nguyen@duytan.edu.vn to be introduced in Section 3. Section 4 provides a biological background concerning the honey bee colony. Our proposed approach is highlighted in Section 5. Afterwards, we present our method for improving an existing consensus algorithm in Section 6. Section 7 deals with the implementation of the software prototype. Experimental evaluation results are introduced in Section 8. Finally, Section 9 concludes the paper.

2. Motivations and Objectives

2.1. Motivations

For the support of mass casualty incidents (MCI) rescue, a project has been executed by Friedrich Schiller University Jena and its partners. The project, named SpeedUp[1], consists of two main areas: SpeedUp Practice and SpeedUp Technology. SpeedUp Technology is responsible for the IT support of rescue forces in MCI situations so that in disaster events, rescue actions can be performed in a more effective way. The IT solution supports the usual rescue measures by providing a framework for improved information handling, additional preprocessed sensor data, and a basis for a highly flexible communication infrastructure [12]. In SpeedUp the mobile agent concept has been selected as one of the main technologies on the communication infrastructure level.

Our work was motivated by a system working in a communication infrastructure for emergency rescue scenarios [6]. Figure 1 illustrates the so-called mass casualty incident (MCI). In this scenario work conditions change dramatically as new tasks appear, rescue staff are unlikely to process incoming information in an effective way. Moreover, since each team performs tasks with its own criteria and under control of its own authority, conflicts in their goals might arise at anytime. As a consequence, performing rescue operations based on experience without any additional technical supports seems to be an impossible task. In an MCI rescue scenario, there are multiple regions of working staff being scattered over a wide area. In a region, the movement and relocation of rescue personnel make the availability of their devices unstable. Changes in network topology trigger other stimuli, such as changes in connection quality, in routing strategy, and in communication pattern. These stimuli generally adversely affect communication performance. A question arises: How to organize the logical connection among platforms in an operationally feasible manner to cope with these stimuli? Given that communication between mobile agents is important, it is expected that agent platforms can self-manage in order to be adaptive to perturbations, thereby attaining fault tolerance and reliability in message transferring. We consider the issues as our research problem.

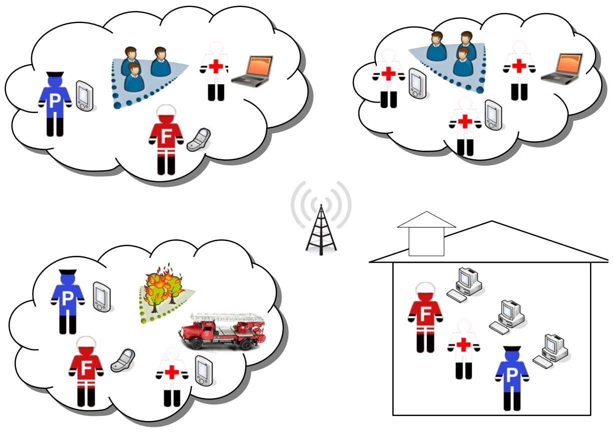


Figure 1. Communication in MCI rescue scenarios

In our work, we propose a model for agent communication that focuses on the cooperation aspect of agent interaction and supports reliability and fault tolerance as the key qualities, while keeping up an acceptable overall performance at the same time. We design, implement, and evaluate in this project environment a prototype for adaptive mobile agent communication in highly dynamic networks.

2.2. Aims

We have been participating actively in designing and implementing a communication infrastructure for the support of rescue personnel in MCIs. The IT solution supports the usual rescue measures by providing a framework for improved information handling, additional preprocessed sensor data, and a basis for a highly flexible communication infrastructure. Essentially, we are aiming at providing necessary information to the right personnel so that in disaster events, rescue actions can be performed in a more effective way. The main contributions of our work are summarized as follows:

- Introduction of a self-organizing mechanism for the management of working regions in highly dynamic networks.
- Development of a communication model that is able to adapt in a mostly autonomous fashion to topological changes of the underlying network.
- Implementation of a software prototype for the proposed communication model that is adaptive to environmental stimuli.

[1]The project was funded within the Federal Government's program "Research for Civil Security" by the German Federal Ministry of Education and Research (BMBF), http://www.speedup-projekt.de

- Improvement of overall qualities of mobile agent collaboration concerning fault tolerance and reliability.
- General speaking, the proposed model is suitable for mobile agents working in highly dynamic networks of the SpeedUp-Type.

3. Swarm Intelligence

Swarm intelligence is considered as a branch of biomimicry which covers a wide range of applications, including architecture, mechanical engineering, nano technology, and computer science. The main idea of swarm intelligence in computer systems is to emulate activities in nature so that they can have alike features. As a result, they can react to environmental changes and therefore be resilient to perturbations.

In MCI rescue scenarios, rescue forces may be distributed widely. Each geographical location forms most likely a technological region in which different forces work together to do rescue tasks. The key point is how to manage the dynamics of mobile agent platforms in these regions efficiently, as they join and quit spontaneously, or they swarm onto a new location. We expect a technique that can help the system fulfil the following requirements:

- The control of the whole system does not rely on any central component, it is distributed to all constituent members.
- The system is able to adapt automatically to the surrounding environment.
- The system is resilient to errors, it can get to a stable status after changes or damages through autonomous interaction of its members.

By bearing on these points, our analysis converges on the concept of self-organization [18]. Self-organization is the way a system adjusts its behaviour to adapt to the surrounding environment. Through interaction among the constituent members on a local scale, the system reaches a globally sensible pattern. The control of a self-organizing system does not rely on any specific member, it is distributed to all of them [15].

Self-organization has been proven to help a system to alleviate "The complexity problem" [18], it is a common phenomenon in nature. Throughout thousands of years of evolution, biological systems have adapted themselves to a changing environment. Plants' leaves always lean to the direction where they can get sunlight, roots grow towards water sources and nutrients. In the animal world, flocks of bird and fireflies, schools of fishes, swarms of honey bees are typical examples of self-organization [15]. The common property for these examples is that there is no central control or external intervention to the control of the community, the final decision is made based on the communication among individuals. Robustness, flexibility and adaptability are the outstanding characteristics of natural systems [17]. Through the interaction among its members a system can gain stability, and it is able to recover when error occurs. The whole system fits itself "bottom-up" to the environmental stimuli. It becomes more robust since the control of the whole system does not rely on any individual component. New members can be integrated seamlessly into the system. Furthermore, the disappearance of members does not cause the whole system to stop working, it can scale flexibly to the number of members.

4. Biological Background

4.1. Overview

The bee colony is a typical example of self-organization in nature. A bee population consists of a queen bee who is reproductive, some male drones, and thousands of female workers. Worker bees undertake the most important tasks of the whole colony, they build the hive, keep it clean, regulate the temperature, search for food sources, and find a new resident location when the bee colony needs to split up.

Teamwork is the distinctive character of the colony, honey bees are a well-organized team and they work in strict disciplines. The control of the whole colony is not dependent on any individual, even the queen bee. This is done through the interaction among thousands of worker bees, when and how many bees the queen bee must create is actually controlled by the workers. The queen bee has only an indirect role in organizing the colony, she gives birth to bees. The self-organizing model of honey bees has inspired much work in computer science and shown to improve considerably computational performance in particular circumstances [3],[4],[5].

Regarding the network model in MCI rescue scenarios, we found that the honey bee community and the community of agent platforms have a substantial coincidence of behaviour. If we call honey bees and agent platforms simply *members*, then the following clauses hold for both communities:

- Members work in a distributed environment.
- Members cooperate to perform common tasks.
- Members work to maximize the global work performance.
- Congestion occurs when the population grows.

Furthermore, honey bees have efficient mechanisms to deal with their daily routine as follows:

- Control of the colony takes place without any external intervention.
- The colony can respond dynamically to the surrounding environment.
- The colony is still able to work, if some of the workers are missing.
- The colony can adjust its population to avoid conflict and maintain reproduction.

The fact that the two phenomena have a lot in common and honey bees possess a good mechanism to deal with their daily routines encourages us to apply honey bee behaviour to manage the dynamics of a networked environment that reflects a SpeedUp-Type of system. In the next sections, we are going to provide the reader with an overview of the honey bee colony, and then introduce our approach.

4.2. Bee Communication

Bees can produce honey only after they collect enough nectar and pollen. Since food sources are normally distributed far away from their hive, foraging is a vital skill of honey bees. Foraging takes place in the summer, bees spread out to search for food sources. Each individual bee may find different food sources, but as time goes by, the colony tends to head for rich food source in which they can get more nectar and pollen. It is exciting to know that a honey bee has a mechanism to notify other bees. A worker bee can communicate with her colleagues to inform about the food sources she has found. A scout bee, after finding a new food source, returns to the hive and performs a *waggle dance* to notify other honey bees about the food source [25]. It has been shown that, with the waggle dance, a forager provides the following information to her colleagues [26]:

- Direction to the food source compared to the sun.
- Distance to the food source.
- The amount of food: the more exciting and the longer she dances, the richer the food source is [2],[16].

With this information, other honey bees are able to know where and how far they must fly in order to get to the food source. Rich food sources attract more honey bees to come and get pollen than other sources that contain lower amounts of food. Thanks to this information sharing, the honey bee colony can maximize the amount and quality of food. The discovery of bee communication brought the biologist Karl von Frisch the Nobel Prize for Physiology or Medicine in 1973 [27].

4.3. Swarming

Swarming in bee colony occurs when the bee population increases and causes congestion and difficulties in maintaining good hygiene in the beehive. Swarming is also one of the honey bees' instincts, they maintain reproduction by splitting.

Before swarming, worker bees build a cup in which the old queen lays eggs, in order to raise a new queen bee. When the new queen bee arrives, the old queen bee takes a number of workers with her and they fly to a new temporary location, e.g. a tree branch, that is near to their hive. The swarm remains on the temporary residence for a short time while waiting for a new residence. Scout bees, which are the most experienced foragers in the swarm, are then deployed to find new suitable locations. When the exploration is finished, every scout bee returns to the bee cluster to inform the whole colony about the place she has found. This is done by the same waggle dance the scout bee does when she informs the community about food sources.

The scouts fly back and forth until the number of honey bees that gather at a site constitutes a quorum, then the scout bees return to the swarm and the swarm decides to leave for the new location [1]. The relocation may prolong, if the colony finds the new location does not satisfy expected requirements regarding available space, light intensity and hygiene conditions. The swarm will continue to depart for the next location. Eventually, it is summoned to the new residence.

5. A Context-Aware Architecture for Mobile Agent Communication

5.1. Network Model

Figure 2 depicts the SpeedUp communication infrastructure. The components are described as follows:

- Agent Platform: A software platform which provides executing environment for mobile agents.
- Lightweight Agent Platform: A special type of agent platform which is energy limited, such as PDAs and smart phones.
- Region: A group of platforms that work in the same geographical location.
- RegionMaster: A platform that has enough computing power to perform self-organizing tasks, manages agent's location information for the whole region.
- Global System: The highest logical view and consists of all regions of the system.

Based on the self-organizing behaviours of the honey bee colony, we proposed a model for agent communication which is able to cope with environmental stimuli

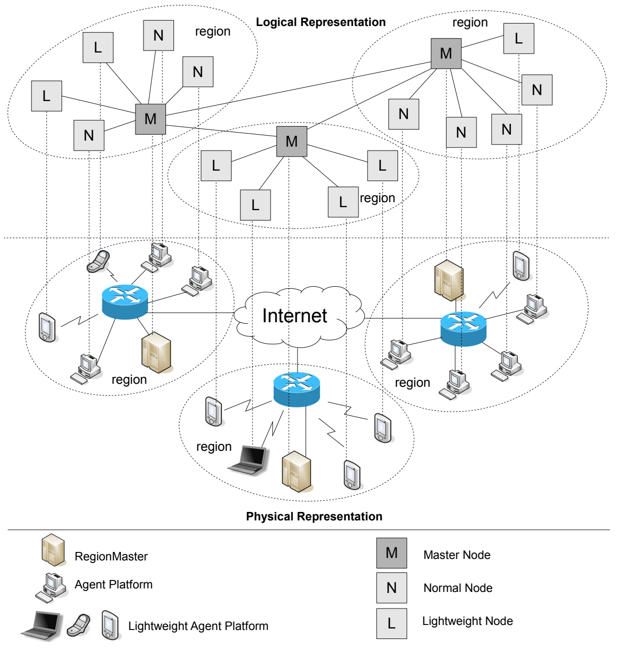


Figure 2. SpeedUp communication infrastructure

[8]. In the algorithm, two mappings from the honey bee colony to the rescue scenario network model will be employed. In the first mapping, RegionMaster is considered as the beehive and mobile agents are scout bees. All other platforms of the region are viewed as potential beehives. In the second mapping, we consider all agent platforms as honey bees, and RegionMaster as the queen bee. Our solution is going to bear on the following issues:

- The distributed nature of mobile agents.
- The parameters related to fault tolerance and reliability of communication between mobile agents.
- The scalability of the system with regards to the availability of heterogeneous end devices.
- The dynamics of the network.

In the SpeedUp architecture, regions of working devices are the prime elements which make up the global system. In our view, these facets should be handled in an effective way. Environmental stimuli affect the overall performance, therefore they need to be monitored so that the system can self-adjust accordingly. Since mobile agent communication is communication between individual agents, issues related to direct agent communication must be managed adequately from the start. Communication for mobile agents should reuse existing results from agent communication research. The main objective is, therefore, a model for agent communication that focuses on the cooperation aspect of agent interaction and supports fault tolerance and reliability as the key qualities.

5.2. Getting the Blueprint of a Region

The first mapping is inspired by bee communication. It is used for measuring the connection quality between an agent platform and all the remaining platforms of a region at a point in time. The connectivity map of the region can be constructed once a calculating process has been performed for all nodes of a region.

In this view, RegionMaster plays the role of the beehive and mobile agents are scout bees. Figure 3 illustrates how agents are sent to scout network. At regular intervals, RegionMaster deploys scout agents to all platforms of the region to get information about them. A RegionMaster is responsible for monitoring its region and regions are surveyed independently. Information related to connectivity and and platform's performance will be collected by scout agents.

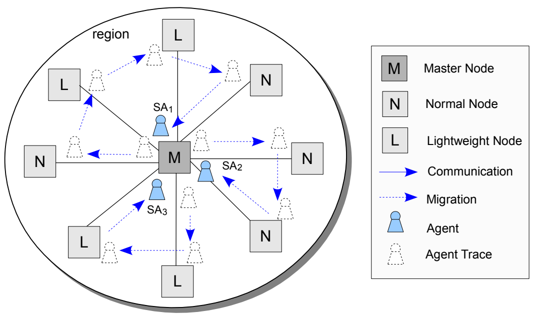


Figure 3. Scout agents collect network's information

When a scout agent arrives at a platform, it sends ping messages to all nodes of the region, including RegionMaster to measure the transfer time between the current platform and every other platform. After collecting the node's information, the agent migrates to the next platform. At the new platform, the scout agent performs the same routine. The process repeats until the last node of the itinerary has been visited.

A platform holds a boolean value *split* to indicate whether it expects its region to split or not. At each platform once a scout agent has obtained the transfer time to all neighbours by sending and receiving ping messages, it calculates the average transfer time τ_{avg}.

$$\tau_{avg} = \frac{1}{n}\sum_{i=1}^{n-1} t(i) \qquad (1)$$

In which n is the number of nodes in the region, $t(i)$ is the transfer time between the current platform and the i^{th} neighbour platform. Since every node in a region has the same number of neighbours, this value is the measure of the closeness between a node and other nodes in the region. Once the value is calculated, the platform compares the τ_{avg} with the transfer time to RegionMaster τ_{RM}. If $\tau_{RM} >> \tau_{avg}$ then it records this state. If the state is observed for a number of times then RegionMaster expects that the region will be splitted; it sets the value *split* to *true*. The two values τ_{avg} and *split* will then be given to the scout agent. Eventually, the agent migrates back to RegionMaster and submits all the information it has collected. RegionMaster then performs self-organizing tasks based on information fetched from all the scout agents it deployed.

5.3. Re-Organizing a Region

The second mapping is inspired by bee swarming. In this view, RegionMaster plays the role of the queen bee, and other platforms of the region are worker bees. This mapping is used to impose a self-orgarnizing mechanism on the platforms.

For the rescue scenario network model, in a region RegionMaster is responsible for performing self-organizing tasks and handling location information of all mobile agents working in the region. If the number of agent platforms in a region is large and too many requests are sent simultaneously, a bottleneck may occur at RegionMaster. The phenomenon resembles the necessity for swarming in the honey bee community. The platform population exceeds the region's capacity thus causing degradation in connection quality. As a result, it needs to be adjusted, the colony requires some kind of "swarming." Like in the bee colony, the region will be splitted into two regions. One of the nodes will be promoted to be a new RegionMaster. The new RegionMaster organizes to form a new region from the nodes it inherits. The two regions are then independent from each other, but logically connected.

Each agent platform plays a contributory role in organizing the region. A final decision of splitting the region is made based on a consensus of the platforms in the region. In mapping 1, at every platform, the scout agents collect the boolean value *split*. RegionMaster counts the number of platforms that want the region to be segregated. If the number constitutes a quorum, the region is about to be splitted. Like the queen bee chooses her successor, RegionMaster promotes a new RegionMaster from the candidates nominated the by scout agents. Afterwards, RegionMaster broadcasts the name of the new RegionMaster to all platforms. Every platform decides itself, which region it belongs to by sending a joining message to the new RegionMaster.

5.4. Managing Mobile Agent's Location

The mobility of agents may cause message losses when communication executes, especially in large scale and dynamic networks. The management of mobile agents' location should also be adaptive to the network topology changes. With respect to the dynamic region architecture established by the self-organizing algorithm, we introduce a technique for the management of mobile agent's location.

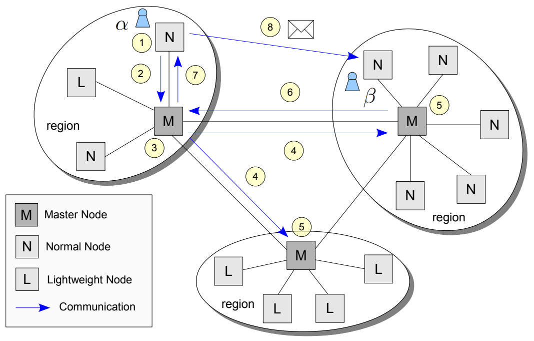


Figure 4. Location query

Figure 4 depicts an example of location searching where agent α wants to communicate with agent β whose location is unknown. The circled numbers depict the steps that are to be executed. Agent α asks the platform where it is currently in for the ID of agent β. This platform checks its cache to see if there is any information regarding the location of the requested agent (1). When the address is already in the cache, the platform returns the address to the agent. Otherwise, the platform queries its RegionMaster (2). RegionMaster then looks into its cache for the address (3). When the address is found, RegionMaster returns it to the requesting party (7); if not, RegionMaster broadcasts a query to all RegionMasters of the global system (4). Each RegionMaster of the global network checks its cache for the address (5). Whenever the address is found by one of the RegionMasters, it will be sent back to the requesting RegionMaster (6). This RegionMaster in turn forwards the address to the requesting platform (7). After the location of the agent β has been determined, the agent sends message to the targeted platform (8).

If a region is splitted, RegionMaster advertises the change to all other RegionMasters to notify other RegionMasters of the new region's appearance.

6. Fault Tolerance: A Honey Bee inspired Approach for Building Consensus

6.1. Finding consensus in distributed systems

The proposed mechanism seems to rely much on RegionMasters, which may generate a single point of failure. In a region, RegionMaster acts as the beehive, where scout agents unload information they collect. It also performs self-organizing tasks for the whole region. If RegionMaster goes haywire or disconnects suddenly, the node community has no information about the network and, therefore, cannot re-organize wherever necessary. As a result, a constructive approach to fulfil the fault tolerance requirement is needed. This leads to the problem of finding consensus in distributed systems, all nodes promote a new RegionMaster by exchanging messages.

The problem of building consensus is described in detail as follows: A set of $n \geq 2$ nodes $P = \{p_1, p_2, .., p_n\}$ has to negotiate by using asynchronous message passing and decide on the same value from a common set of inputs. In the scope of this paper, we discuss binary consensus, that means the input values $v \in \{0, 1\}$ [21],[22],[23],[24].

The properties of consensus [23]:

- At most f nodes may fail. The failed nodes are called faulty nodes; the working nodes are called non-faulty or correct nodes. Once fails a node remains faulty for ever.
- Each node p broadcasts its proposal as a message to all other nodes.
- A node makes a decision based on the set of messages it received.
- A message never fails once it has been sent. It arrives eventually at the targeted receivers.
- A node p can send its report or proposal value v to some nodes, and it may crash before sending the message to the remaining non-faulty nodes. As a result, some nodes have v and some not.

A consensus algorithm needs to satisfy three key requirements as follows [22]:

- Termination: Every node must ultimately decide.
- Agreement: All non-faulty nodes decide on the same value.
- Validity: The chosen value must be the input of at least one of the nodes.

No matter how simple the definition is, the solution for the problem remains a challenge in distributed computing. In [14] Fischer, Lynch and Paterson prove that with an asynchronous message passing system, there exists no deterministic algorithm for solving the problem of consensus in presence of node failures.

6.2. Ben-Or's algorithm for finding consensus

Ben-Or proposes a practical solution to solve consensus using randomization. In his approach, it is assumed that at most $f < n/2$ nodes may fail during execution. The algorithm is described in the pseudo code as follows [21],[31]:

Algorithm 1 Ben-Or's Consensus Algorithm

```
1: procedure BenOrConsensus($v_p$)
2:     $x \leftarrow v_p$
3:     $k \leftarrow 0$
4:     while $true$ do
5:         $k \leftarrow k + 1$
6:         Report($k$,$x$)
7:         WaitFor($k$,$*$) from n-f nodes    ▷ * is either 0 or 1
8:         if there are more than n/2 messages containing v then
9:             Propose($k$,$v$)
10:        else
11:            Propose($k$,$abstention$)
12:        end if
13:        WaitFor($k$,$*$) from n-f nodes
14:        if there are at least f+1 messages containing v then
15:            Decide($v$)
16:        else if there is at least 1 message containing v then
17:            $x \leftarrow v$
18:        else
19:            $x \leftarrow$ ChooseRandom(0,1)
20:        end if
21:    end while
22: end procedure
```

- Procedure $Report(k, x)$ broadcasts a message containing information about the current round k and the proposed value to all other non-faulty nodes.
- Procedure $WaitFor(k, *)$ waits for all incoming messages containing k and a report/proposal value.
- Procedure $Propose(k, v)$ broadcasts a proposal value v in round k to all non-faulty nodes.
- Procedure $Decide(v)$ makes a decision on the value v.
- Procedure $ChooseRandom(0, 1)$ returns either 0 or 1 with equal probability.

A comprehensive proof for the correctness of Ben-Or's algorithm can be found in [31].

6.3. A proposal for improving the convergence speed

The original Ben-Or's algorithm may reach a pretty slow convergence given that nodes do not choose the same value in the randomization phase. Regarding the problem of reaching an eventual agreement among nodes in distributed computing, we witness a substantial coincidence between consensus in distributed systems and consensus in the honey bee colony as shown in Table 1.

Bee Colony	**Distributed Systems**
Bees	Nodes
Sites	Values
Scout bees nominate a site by dancing	Nodes propose a value by broadcasting messages
Bees select a site if the number of bees around the site exceeds a threshold	Nodes decide on a value *v* if they receive more than *f* report messages containing *v*
All bees reach an eventual agreement for a site	All nodes eventually decide on the same value

Table 1. Analogies

The fact motivates us to propose some amendments to the original consensus algorithm. We call $M_p(k)$ is the set of messages received by node p at round k; $C_p(k,0)$ is the cardinality of the set of messages that contain 0 and $C_p(k,1)$ is the cardinality of the set of messages that contain 1. When $C_p(k,0) = C_p(k,1)$ we say it is a tie for the two sets.

As we have seen in Algorithm 1, it is expected that the majority of nodes will report the same value in Line 6 and then a quorum can be reached in Line 14. However, a node proposes a value only when it receives more than $n/2$ messages containing the same report value, that means only when the majority requirement is satisfied. Otherwise it proposes the value *abstention*, which does not help to build a consensus.

We may "waste" the chance that nodes report the same value in round $k+1$ given that almost $n/2$ messages containing the same value v are sent in Line 13. This should be considered as a room for improvement [7].

It should be noted that from rounds $k > 1$, the value x reported in Line 6 is the result from either Line 15 or 17 or 19 in round $k-1$. We propose a way to *guide* the nodes to opt for the most sensible value v instead of choosing a random value. We consider the case that the majority requirement is not satisfied: $C_p(k,v) < n/2$; but there is a *plurality* of nodes that report v, that means: $C_p(k,1-v) < C_p(k,v) < n/2$.

The honey bee inspired algorithm for building a consensus is illustrated in the pseudo code Algorithm 2. The procedures and functions of the pseudo code are explained as follows:

- Function $Plurality(0,1)$ returns 0 if $C_p(k,0) > C_p(k,1)$ and returns 1 if $C_p(k,0) < C_p(k,1)$.

- Procedure $Follow(v)$ indicates that a node prefers a value v.

- Function $Evaluate(0,1)$ calculates a correlative value between the cardinalities $C_p(k,0)$ and $C_p(k,1)$; a proposal is $Evaluate(0,1) = abs(C_p(k,0) - C_p(k,1))$.

- Procedure $Decrease(strength)$ subtracts a specified number from *strength*.

Algorithm 2 Honey Bee Consensus Algorithm

```
1:  procedure HoneyBeeConsensus($v_p$)
2:      $x \leftarrow v_p$
3:      $k \leftarrow 0$
4:      $strength \leftarrow 0$
5:      while $true$ do
6:          $k \leftarrow k+1$
7:          Report($k$,$x$)
8:          WaitFor($k$,$*$) from n-f nodes
9:          if there are more than n/2 messages containing v then
10:             Propose($k$,$v$)
11:         else
12:             if $strength \leq 0$ and $C_p(k,0) \neq C_p(k,1)$ then
13:                 $v \leftarrow$ Plurality(0,1)
14:                 Follow($v$)
15:                 $strength \leftarrow$ Evaluate(0,1)
16:             end if
17:             Propose($k$,$abstention$)
18:         end if
19:         WaitFor($k$,$*$) from n-f nodes
20:         if there are more than f messages containing v then
21:             Decide($v$)
22:         else if there is at least 1 message containing v then
23:             $x \leftarrow v$
24:         else
25:             if $strength$ >0 and Follow(v) then
26:                 $x \leftarrow v$
27:                 Decrease($strength$)
28:             else
29:                 $x \leftarrow$ ChooseRandom(0,1)
30:                 $strength \leftarrow 0$
31:             end if
32:         end if
33:     end while
34: end procedure
```

7. Implementation

This section deals with the implementation of the software prototype for an adaptive communication framework for mobile agents in highly dynamic networks of the SpeedUp-Type. The implementation of the software prototype will bear on the honey bee inspired alogrithm with the aims and objectives of the targeted qualites: fault tolerance and reliability in transferring agent messages.

Ellipsis is an open source mobile agent system developed by the Chair of Software Engineering of Friedrich Schiller University Jena. Ellipsis provides a framework for hosting and executing mobile agents and runs on any operating system that supports Java, thereby adhering to the slogan "write once, run anywhere." Ellipsis can be deployed on any device as long as the device is equipped with Java Virtual Machine (JVM). We base our design and implementation on Ellipsis and the software architecture in conjunction with the mobile agent system is depicted in Figure 5. The following sections provide an overview of the software components.

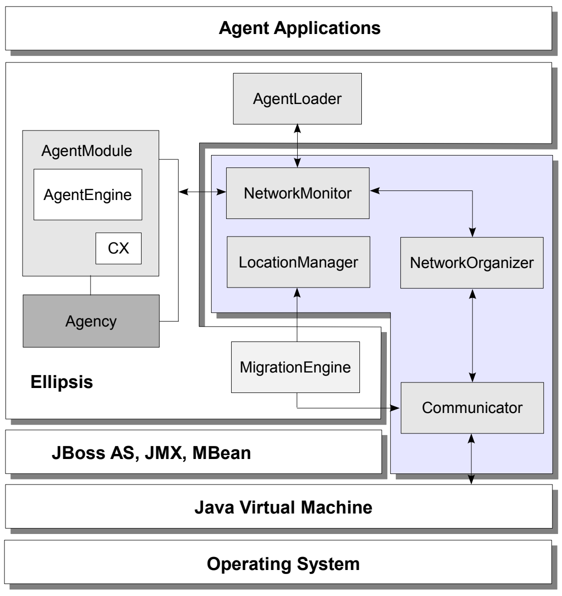


Figure 5. Software Architecture

7.1. NetworkMonitor

NetworkMonitor observes the agent platforms of a region. It collects connectivity information between a platform and its neighbours and information of the platforms. NetworkMonitor undertakes the following tasks:

- Managing a dynamic list of agent platforms.
- Deploying scout agents periodically to get update on the network situation.
- Receiving and replying ping messages.
- Calculating the values τ_{avg} and *split*.
- Making a decision of splitting region by counting the boolean values *split*.

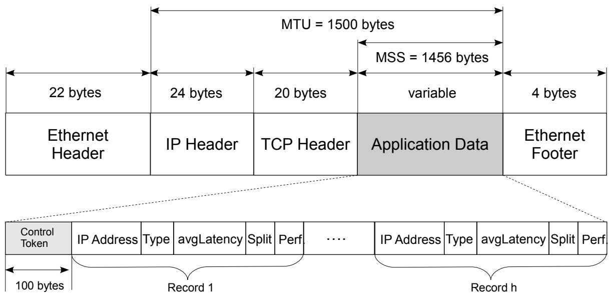


Figure 6. Packet Format

7.2. NetworkOrganizer

NetworkOrganizer re-organizes the logical network based on the conditions percepted by NetworkMonitor. This component is present at every platform of a region. However its role is dependent on the type of the platform. In a RegionMaster, NetworkOrganizer has the following functions:

- NetworkOrganizer receives the list of platforms from NetworkMonitor.
- NetworkOrganizer selects a new RegionMaster from the candidates nominated by NetworkMonitor. It broadcasts the ID of the new RegionMaster to all platforms of the region to conduct a poll.
- NetworkOrganizer activates a platform when the platform has been promoted to the post of a RegionMaster.
- NetworkOrganizer notifies other NetworkOrganizers of the changes by sending broadcast messages.

In a normal platform, NetworkOrganizer performs the succeeding tasks:

- It receives a proposal for a node to be the new RegionMaster from RegionMaster.
- It chooses which region the corresponding platform should belong to by measuring the latencies to the proposed platform and to RegionMaster.
- It sends the ID representing the chosen platform back to RegionMaster.

7.3. LocationManager

LocationManager manages location information of mobile agents, it updates location of mobile agents when they migrate, answers location query and queries agent's location information. LocationManager holds a cache for storing location information. Essentially, the component performs its function to answer the question: *Which agent locates where?*

The following tasks are undertaken by LocationManager:

- Interfacing to Ellipsis to manage a dynamic list of location information for mobile agents.
- Receiving and answering location query from other nodes as well as from agents.
- Sending location information query to other LocationManagers.

7.4. Communicator

The software entity Communicator is responsible for initializing connection between platforms and transferring messages between agents. It encodes, decodes, and transfers agent messages over the network. This component maintains two caches of agent messages at every platform, one for incoming messages and one for outgoing messages.

Communicator has the following functions:

- Establishing connection between two agent platforms.
- Encoding the envelope and the payload of an ACL message at the sender side.
- Sending byte streams containing ACL messages over the network.
- Decoding the envelope and the payload of ACL messages at the receiver side.

8. Experimental Evaluation

8.1. Setup

We deploy a laboratory scale test network with the presence of eight computers connected through a local Gigabit Ethernet LAN 1000 Mbps network. Each computer corresponds to an agent platform and is equipped with the software framework as well as other essential softwares. The system configuration is specified in Table 2.

The agent platforms representing by their alias are going to be used throughout the succeeding test scenarios.

Alias	OS	RAM	Processor
D1	Fedora 12	2.0 GB	AMD 2.2 GHz
D2	Debian 6.0.4	4.0 GB	Intel 2*2.0 GHz
D3	Debian 6.0.4	4.0 GB	Intel 2*2.0 GHz
D4	Debian 6.0.4	4.0 GB	Intel 2*2.0 GHz
FJ	Fedora 12	3.0 GB	Intel 2*2.8 GHz
PT	Windows 7	4.0 GB	Intel 2*2.4 GHz
T1	Fedora 12	2.4 GB	Intel 2*2.4 GHz
T2	Windows XP	1.0 GB	Intel 1.7 GHz

Table 2. Hardware configuration of the experiments

8.2. Feasibility

In a dynamic network environment the cost for monitoring network might be considerably high. Monitoring activities may place a burden on the network traffic, thereby reducing overall system's performance. The proposed mechanism is beneficial only if network monitoring gains a good performance whilst keeping a reasonable running cost. The issues need to be addressed throughtfully.

The feasibility of the proposed mechanism means that it maintains a reasonable cost for monitoring while providing necessary information for the self-organizing tasks. To validate the feasibility, scout agents are deployed to monitor network and the parameters regarding processing time and exploration time will be measured. One round of exploration happens when a scout is created at a RegionMaster, travels through all nodes of the region, performs its routine and migrates back to RegionMaster. The following information is going to be acquired:

- The time needed for a scout agent to accomplish its tasks at a platform: $t_{processing}$.
- The time needed for an agent to perform one round of exploration: $t_{exploration}$.

The processing time at an agent platform is the duration from when an agent arrives until it completes its tasks and leaves for the next node. It is calculated as follows:

$$t_{processing} = t_l - t_a \tag{2}$$

where t_a is the time when a scout arrives at a platform and t_l is the time when it leaves for the next node of its itinerary.

The processing time is made of:

$$t_{processing} = t_{RTT} + t_s + t_d \tag{3}$$

in which

- t_{RTT} is the period of time from the first ping message sent until the last response received.

- t_d is the time to deserialize a scout agent from an incoming stream of byte.
- t_s is the time to serialize a scout agent into a byte stream to be sent over the network.

By measuring the processing time, we examine how fast a scout agent performs its tasks at a specific platform.

The average exploration time is computed:

$$t_{exploration} = t_{end} - t_{begin} \tag{4}$$

where t_{begin} and t_{end} are the time when a scout starts and completes one round of exploration, respectively. This paramater indicates how often a scout agent supplies a RegionMaster with up-to-dated information for the self-organizing activities.

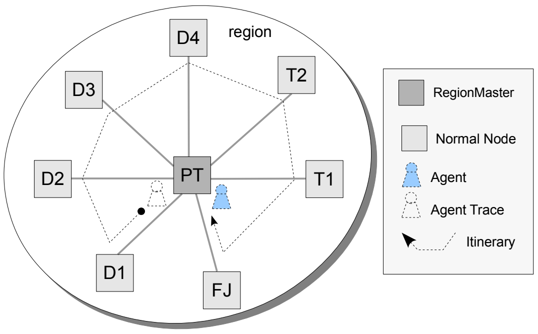


Figure 7. Network Setup

Figure 7 shows the logical connection for the first evaluation test. In this scenario, one scout agent is created at RegionMaster. The RegionMaster assigns the list of agent platforms $\{D1, D2, D3, D4, FJ, PT, T1, T2\}$ to the scout agent. One round of surveillance takes place when the agent is sent by RegionMaster, travels around the nodes of the region, performs its routine at every node and then migrates back to RegionMaster.

To get a more reliable result, by every platform a measurement is done for 80 times. The box plot diagram in Figure 8 depicts the processing time $t_{processing}$ of the platforms. This parameter represents the time that the agent needs to perform its tasks at that platform. It is dependent on the processing power of agent platforms as well as the latencies to the other platforms, which are in turn dependent on the network connection speed. It can be seen that, except platform $T2$ that has a higher processing time because of its limited processing power (as specified in Table 2), the processing time for the other platforms is considerably low. It guarantees that the processing activities place comparatively little burden on the system performance.

To measure $t_{exploration}$, the agent is sent around the network for different numbers of rounds r. In

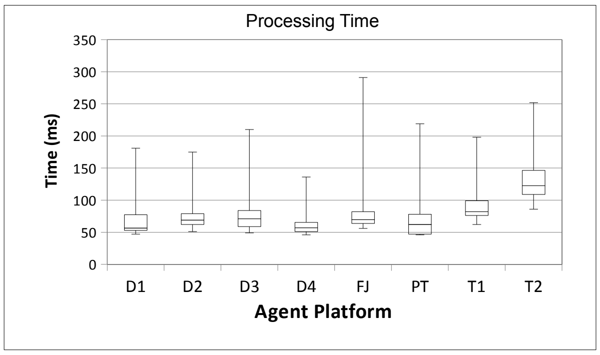


Figure 8. Processing time

the second experiment r is set to different values, i.e. $r = \{100; 200; 300; 500; 1000; 2000; 3000; 5000\}$. The average exploration time is given by:

$$t_{exploration} = \frac{t_{end} - t_{begin}}{r} \tag{5}$$

This parameter demonstrates how fast the scout agent can supply RegionMaster with up-to-date information about the network. If the agent needs a long time to perform its tasks at the platforms or to hop from platform to platform, the information submitted to RegionMaster might be out-of-date. As a consequence, the reactions produced by RegionMaster would not be adequate.

However, the measurement results show that this is not the case. In the diagram in Figure 9, the curve represents the accumulated exploration time. The straight line which depicts the average time for finishing one round of surveillance (second/round) provides evidence that the parameter is stable, no matter how many rounds the agent has migrated. This indicates that the scout agent produces no overhead when it works in the long run.

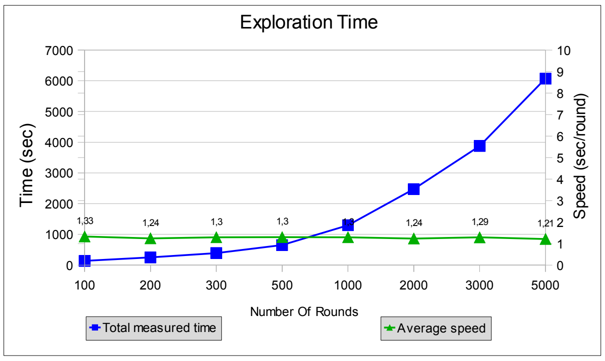


Figure 9. Average time for a scout agent to explore the region

8.3. Self-Organizing

In a dynamic network it is a common occurrence that a group of platforms leaves other platforms of the same region and swarms onto a new location, causing the connectivity to change correspondingly. Under the circumstances, maintaining a fixed logical connection between the abandoning platforms and the stationary platforms is not an optimal solution. It is more sensible to re-organize the colony to adapt to the new structure. The abandoning platforms should form a new region while the stationary platforms group into a second region. A reasonable re-organizing manoeuvre helps facilitate a harmonious correlation between the platforms, thereby balancing network load and saving processing time.

In network applications, a service is identified by the combination of an IP address and a network port. A traffic shaper controls network bandwidth by delaying traffic to meet pre-defined requirements. A traffic can be precisely allotted to a service by referring to the service's identification. By traffic shaping, two important factors for a service are determined. A minimum usable bandwidth ($MiUB$) which is the guaranteed bandwidth that the service can consume and a maximum usable bandwidth ($MxUB$) which is the bandwidth that the service can reach as long as there is free bandwidth available.

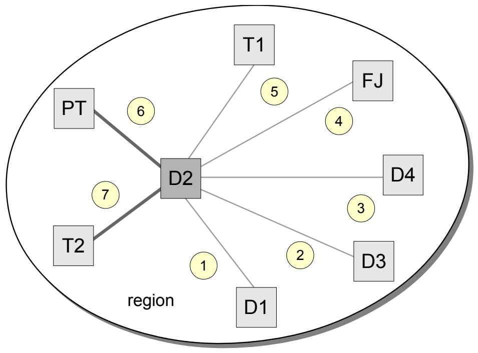


Figure 10. The network layout and bandwidth shaping imposed on ping services

The logical network layout for this test scenario is depicted in Figure 10. In the beginning, platform $D2$ is configured as RegionMaster. A scout agent is deployed to survey the region. We use traffic shaping to simulate the occurrence when the platforms, together, move far away from RegionMaster and swarm onto a new location, resulting in a decrease in the corresponding bandwidths.

In this test scenario, the traffic shaper which is comprised of software packages *tc* (traffic control), *iptables* and *iproute* in Linux distributions is used to regulate the traffic of the ping messages exchanging service between RegionMaster and the platforms $\{D1, D3, D4, FJ, T1\}$ on TCP port 3242. Table 3 lists the corresponding values $MiUB$ and $MxUB$ for every connection representing by the circled numbers depicted in Figure 10.

Con.	(1)	(2)	(3)	(4)	(5)	(6)	(7)
$MiUB$ (Mbps)	0,512	1	2	20	54	1000	1000
$MxUB$ (Mbps)	1	2	3	30	60	1000	1000

Table 3. Bandwidth allotted to each link

The bandwidth degradation will have a certain effect on the communication between agent platforms. While bandwidth shaping is being activated, there are changes in network connectivity within the region. The platform colony re-organizes to react to the changes in network bandwidth. Every platform sets the value $split$ based on the ratio $\tau = \tau_{avg}/\tau_{RM}$. In this experiment the splitting threshold is set to $\tau < 50\%$. Since more than a half of the agent plaforms set $split = true$, the region is bisected. Two regions emerge from the original region. Figure 11 depicts the splitting process, those platforms with a ratio $\tau \geq 50\%$ namely $\{D2, PT, T2\}$ group into Region 1; Region 2 is made of the platforms with $\tau < 50\%$ i.e. $\{D1, D3, D4, FJ, T1\}$.

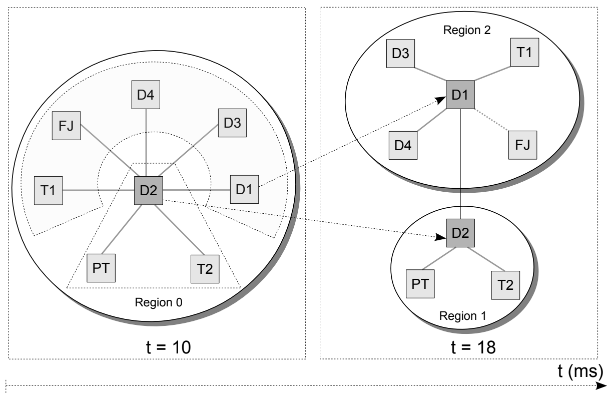


Figure 11. The self-organizing process of the platform colony

Table 4 displays the metrics measured for Region 1 and Region 2 before and after splitting. The measurement unit for the ratio τ is %. According to the metrics shown in the tables, both Region 1 and Region 2 benefit from the splitting, the ratio τ grows markedly and exceeds the threshold 50%. An increase in the ratio τ implies that the connection from each platform to RegionMaster has been improved. All the platforms reset $split$ to $false$.

The metrics show that after network connectivity degrades, the splitting helps the platforms improve connection quality to RegionMaster significantly. Since

	Before splitting		After splitting	
Platform	τ(%)	**Split**	τ(%)	**Split**
Region 1				
D2	—	*false*	—	*false*
PT	75,0	*false*	75,0	*false*
T2	120	*false*	83,3	*false*
Region 2				
D1	14,8	*true*	—	*false*
D3	16,8	*true*	75,0	*false*
D4	15,7	*true*	100	*false*
FJ	38,9	*true*	87,5	*false*
T1	29,4	*true*	83,3	*false*

Table 4. Metrics measured for Region 1

RegionMaster is responsible for performing the self-organizing tasks of the whole region, an improvement in connectivity will lead to an increase in performance of the system. This test supports the hypothesis that the software framework can assist the system to react appropriately to changes happening in the surrounding environment. In our judgement, the features demonstrate that the software framework complies with one of the requirements: the ability to self-organize in a changing environment.

8.4. Context-Awareness

In the next experiment, platform FJ is configured as RegionMaster, a scout agent is deployed to survey the region. In this scenario, the connectivity between RegionMaster and the remaining platforms is going to be degraded using software. This aims to investigate the countermeasures of the system given that the quality of the connection to RegionMaster has declined. Since RegionMaster is responsible for the management of the whole region, the deterioration adversely affects system performance, processing speed may considerably slow down. Given the circumstances, it is expected that the software prototype helps the platform colony to re-organize and recover from the degradation.

To conduct this experiment, a simulation of connectivity degradation is required. A certain network traffic between RegionMaster and the other platforms will be created, resulting in a smaller bandwidth left to the remaining platforms. For the purpose of creating network traffic, we utilize the open source software Iperf [28]. This tool is used for measuring throughtput and performance of a network. It can also be used to produce both TCP and UDP data streams over networks; data sent by the client will be received and eventually discarded by the server.

Since Iperf consumes a certain bandwidth on the connection between RegionMaster and the rest of the region, there is smaller bandwidth left for other applications. As a result, each agent platform will experience effects from the traffic generator. The latencies between the platforms and RegionMaster grow sharply. These changes are sensed by the scout agent every time it visits the platforms. Every platform sets the value *split* based on the ratio $\tau = \tau_{avg}/\tau_{RM}$; where τ_{avg} is the average latency and τ_{RM} is the latency to RegionMaster, respectively. In this experiment the splitting threshold is set to $\tau < 50\%$.

Table 5 shows the ratio and the corresponding value *split* for every platform. At the end of a round of exploration, the scout agent goes back to RegionMaster and submits information it collected. Based on this information, RegionMaster makes a decision.

Platform	τ_{avg}/τ_{RM} **(%)**	Split
D1	47,8	*true*
D2	47,6	*true*
D3	37,5	*true*
D4	27,5	*true*
FJ	—	*false*
PT	37,5	*true*
T1	44,4	*true*
T2	29,7	*true*

Table 5. Metrics measured at the time of self-organizing

Since most of the platforms set the value *split* to *true* the region is splitted. The new region structure is depicted in the central part of Figure 12. There is only one platform staying at the old region, this is RegionMaster. The remaining nodes join the new region with $D2$ promoted to be the new RegionMaster. In this case, an adaptation has been made, the old RegionMaster relinquishes its leadership in the only-one platform region and becomes an inferior node of the new region. Eventually, a new region emerges from the original region.

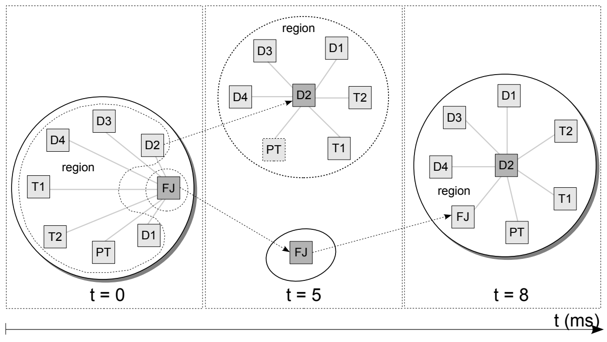


Figure 12. Exchange in role of the command node

Figure 13 displays the latency to RegionMaster τ_{RM} of every platform in three phases. For a platform, the left column is the latency before `Iperf` is activated;

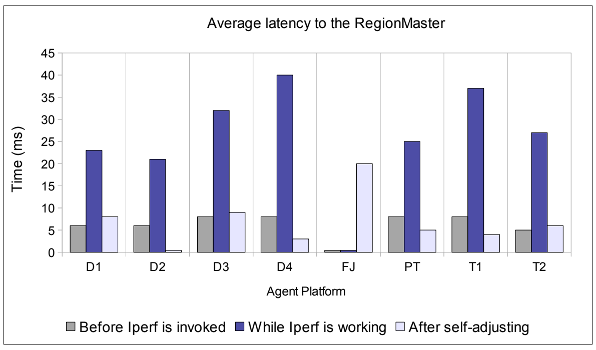


Figure 13. Average latencies to RegionMaster

the middle column represents the time while Iperf is operating and the right column is the latency of the platform after the self-adapting process has occurred. In the beginning and while Iperf was working FJ was RegionMaster so $\tau_{RM}(FJ) = 0$. Similarly, after $D2$ has taken its job as RegionMaster $\tau_{RM}(D2) = 0$. As usual expected, while Iperf is being executed, the latencies to RegionMaster measured for every platform increase significantly. However, once $D2$ takes over as RegionMaster, the latencies decrease proportionally.

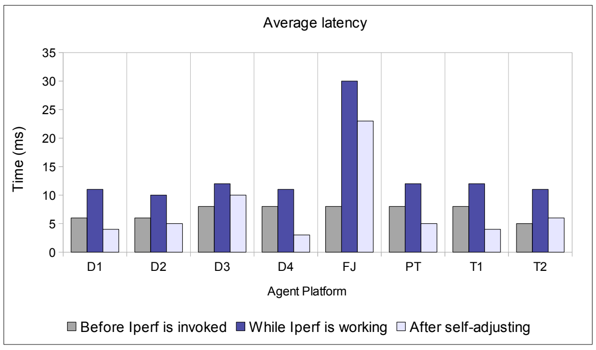


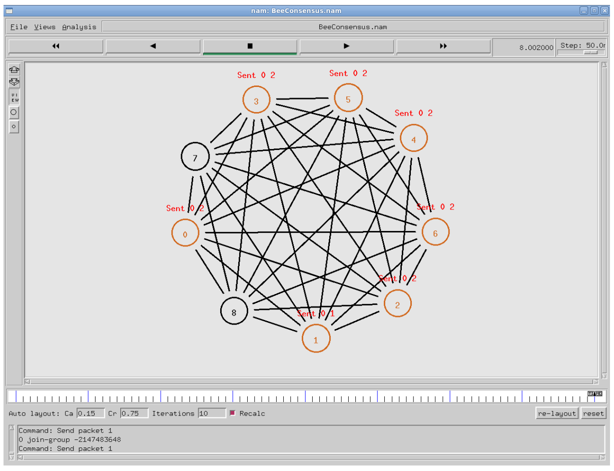
Figure 14. Avarage latencies of all platforms

Figure 14 shows the average latencies τ_{avg} of every platform in three phases: Before Iperf is executed, while Iperf is being executed and after the adaptation has been made. These latencies also shift in the same pattern as by τ_{RM}. Before additional bandwidth was produced, the average latencies had been at a normal level. While Iperf was operating the latencies rose markedly. After the region has been restructured, these values resume to a normal level. It can be seen that the network monitoring activities supply the framework with up-to-date information about network situation, thereby facilitating the decision making process. The measurement results suggest that the swap in role of RegionMaster from FJ to $D2$ brings a more stable connectivity to every platform compared to that of the old arrangement, right after Iperf started producing bandwidth.

8.5. Fault Tolerance

We decided to export the test program to run on simulation environment, since a more precise evaluation result can be obtained if performance tests are performed with presence of several network nodes. The software Network Simulator NS2 is a discrete event simulation framework. It was written in the programming languages C++ and Python and has been used widely for simulating applications running in wired and wireless networks. NS2 supports various routing protocols, including TCP, UDP, multicast. NS2 is normally utilized for testing distributed network applications on a laboratory scale. Tcl (Tool Command Language) is used as the language for scripting simulation scenarios in NS2 [30].

AgentJ is a software tool for embedding real applications written in Java in the NS2 simulation environment. AgentJ allows Java source code to execute on NS-2 with minor modification. Essentially, AgentJ works as a bridge between Java source code and NS2 [32]. The use of AgentJ is practical given that there is a need for simulating program written in Java with solicitation of a large number of nodes. The combination NS2 and AgentJ provides us with a convenient way to simulate evaluation tests without needing to setup a real network with multiple computers.



```
Command: Send packet 1
0 join-group -2147483648
Command: Send packet 1
```

Figure 15. NS2 Simulation with a handful of network nodes

Network Animator (NAM) is a tool embedded in NS2 to visualize simulation. Figure 15 depicts an example of multicast network nodes in NAM, a circle represents a network node and a line between one pair of nodes corresponds to a link between them. In a multicast scenario, every node is connected to all other nodes. For a clear representation, in the figure we depict only a network with a handful of nodes, however in simulation, we can increase the number of network nodes to meet our requirement.

We setup NS2 and AgentJ in the Linux Distribution Fedora 12. Network crash is simulated by shutting down some nodes randomly during execution given that at most $f < n/2$ nodes may fail. A node fails with a probability of a randomized value ranging from 0 to 1.

In the evaluation, the number of nodes is set to the following values $n = 5, 10, 15, 20, 25, 30$ and the number of failed nodes $f \in \{0, \lfloor n/2 \rfloor\}$. We ran the tests with different sets of input values and with repetition. This aims to cover a wide range of possibilities of execution.

For the implementation, Function $Evaluate(0, 1)$ in Line 15 is determined as $Evaluate(0, 1) = abs(C_p(k, 0) - C_p(k, 1))$. Once a node starts to follow a value v, it assigns the difference to the strength value $strength = Evaluate(0, 1)$. Every time it is called, Procedure $Decrease(strength)$ subtracts 1 from $strength$ until it is smaller than 0.

The outcomes of the execution are shown in Table 6 and Table 7. For each table, the first row represents the number of nodes taking part in building consensus n. The second row is the number of nodes that fail during execution f. The third and fourth rows r_1 and r_2 are the corresponding numbers of rounds that Ben-Or's algorithm and the honey bee inspired version reach a consensus, respectively.

n	5			10			15		
f	0	1	2	0	3	4	4	6	7
r_1	1	1	12	1	10	20	4	23	60
r_2	1	1	2	1	2	2	2	2	2

Table 6. Performance Comparison for $n = 5, 10, 15$

n	20			25			30		
f	6	8	9	8	10	12	10	12	14
r_1	17	32	189	32	291	1319	112	1531	7554
r_2	2	2	2	2	3	2	4	2	2

Table 7. Performance Comparison for $n = 20, 25, 30$

At first sight, it can be seen that, the larger the number of nodes is, the more difficult a consensus can be reached. If no node fails, that means $f = 0$ then both approaches can gain a swift consensus. When failure is present, there are differences in performance. By the original Ben-Or's algorithm, if the number of failed nodes increases, the number of rounds that it gains a consensus increases correspondingly. Especially, when f is nearly approaching $n/2$ a large number of rounds can be seen.

Both tables show that the honey bee inspired algorithm has a considerably better computational performance, especially when network nodes fail en masse. Compared to Ben-Or's algorithm, it can gain a consensus after a small number of rounds of exchanging messages. In addition, its performance is stable towards the number of failed nodes. We witness an improvement in performance when applying the honey inspired mechanism.

9. Conclusion

In this paper, we have introduced our proposed architecture for mobile agent communication in highly dynamic networks. Our research was motivated by a mobile agent system working in networks for rescue forces in a mass casualty incident. In this type of networks, connection instability militates against successful transmission of agent messages. To facilitate message transmission, it is necessary to organize the logical connection among agent platforms in an operationally feasible manner. Our work aims to develop a communication model for mobile agent systems that is able to deal with the inherent dynamics of modern networks.

We found inspiration from the honey bee colony where honey bees have a special mechanism to reach self-organization. The organization of the whole colony is done through the interaction among thousands of bees. The behaviours of honey bees inspire us to employ the self-organizing model on the colony of agent platforms. We proposed a solution to a network management mechanism where network organizing tasks are performed in an autonomous manner.

Mobile agents are sent over the network to gather network and platform information. Information collected by mobile agents gives a base for managing the network. Each agent platform plays a contributory role in organizing the region. A final decision to reorganize the network is made based on a consensus of the platforms. In a platform colony, though the master node plays an important role, it is replaceable. Thanks to a failure detector, whenever it fails a second platform can substitute for it and system's operation is not interrupted. Based on the existing mobile agent system Ellipsis, the software prototype for an adaptive model for mobile agent communication has been designed and implemented.

The software prototype was deployed to run in a system of a laboratory scale where conditions of a real dynamic network had been simulated using various software tools. The experiments demonstrated that the software prototype provides the system with the ability of being self-adaptive. It helps the system to react adequately to environmental stimuli as well as to take suitable measures to counteract unexpected network degradation. In addition, given that network failure happened, the system can achieve fault tolerance and maintain a reasonable processing cost for message coding and transferring. From our perspective, the software framework fulfils the basic requirements.

It is our firm belief that the honey bee inspired model cannot only be utilized for mobile agent systems. It can also be applied in other types of P2P network, where it is necessary to network member nodes in a flexible manner. The model can be used to solve other types of problems in network management, such as for discovering resources in P2P networks [29].

References

[1] Seeley, T.D. and Kirk Visscher, P. (2003) *Choosing a home: How the scouts in a honey bee swarm perceive the completion of their group decision making* Journal of Behavioral Ecology and Sociobiology, vol. 54, pp. 511–520

[2] Seeley, T.D. and Kirk Visscher, P. (2004) *Group decision making in nest-site selection by honey bees* Journal of Apidologie, vol. 35, pp. 101–116

[3] Nakrani, S. and Craig, T. (2004) *On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers* Journal Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems , pp. 223–240

[4] Karaboga, D. and Bahriye, A. (2009) *A survey: algorithms simulating bee swarm intelligence* Journal Artif. Intell. Rev., pp. 61–85

[5] Pham, T. and Afify, A. and Koc, E. (2007) *Manufacturing Cell Information using the Bees Algorithm* In Proceedings Innovative Production Machines and Systems Virtual Conference

[6] Nguyen P.T. and Schau V. and Rossak W.R. (2011) *Towards an Adaptive Communication Model for Mobile Agents in Highly Dynamic Networks based on Swarming Behaviour*. The 9th European Workshop on Multi-agent Systems (EUMAS)

[7] Nguyen P.T. (2013) *Building Consensus in Context-Aware Systems Using Ben-Or's Algorithm: Some Proposals for Improving the Convergence Speed.* In Proceedings of the Second International Conference on Context-Aware Systems and Applications, ICCASA 2013, pp. 87–96

[8] Nguyen P.T. and Schau V. and Rossak W.R. (2013) *A Context-Aware Model for the Management of Agent Platforms in Dynamic Networks.* In Proceedings of the Second International Conference on Context-Aware Systems and Applications, ICCASA 2013, pp. 77–86

[9] Finin, T. and Labrou, Y. and Peng, Y. (1998) *Mobile Agents can Benefit from Standards Efforts in Inter-agent Communication* IEEE Communications Magazine, vol. 36, pp. 50-56

[10] Braun, P. (2003) *The Migration Process of Mobile Agents* PhD Dissertation, Friedrich Schiller University Jena.

[11] Baumann, J. (2000) *Control algorithms for mobile agents* PhD Dissertation, the University of Stuttgart.

[12] FSU Jena (2011) *The SpeedUp Project (2011)* http://www.speedup-projekt.de

[13] Baumann, J. and Hohl, F. and Radouniklis, N. and Rothermel, K. and Strasser, M. (1997) *Communication Concepts for Mobile Agent Systems* In Proceedings of the First International Workshop on Mobile Agents

[14] Fischer, M.J. and Lynch, N.A. and Paterson, M.S. (1983) *Impossibility of Distributed Consensus with One Faulty Process* Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems, pp. 1–7

[15] Heylighen, F. (1999) *The Science Of Self-Organization And Adaptivity* The Encyclopedia of Life Support Systems, vol. 62, pp. 253–280

[16] Seeley, T.D. and Buhrman, S.C (2001) *Nest-site selection in honey bees: how well do swarms implement the best-of-N decision rule?* Journal of Behavioral Ecology and Sociobiology, vol. 49, pp. 416–427

[17] Yeom, K. (2010) *Bio-inspired self-organization for suppoting dynamic reconfiguration of modular agents* Journal of Mathematical and Computer Modelling, vol. 52, pp. 2097–2117

[18] Heylighen, F. and Gershenson, C. (2003) *The meaning of self-organization in computing* IEEE Intelligent Systems.

[19] FIPA (2002) *ACL Message Representation in String Specification*

[20] FIPA (2002) *ACL Message Representation in Bit-Efficient Specification*

[21] Ben-Or, M. (1983) *Another Advantage of Free Choice (Extended Abstract): Completely asynchronous agreement protocols* Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing

[22] Vavala, B. and Neves, N and Moniz, H. and Veríssimo, P. (2010) *Randomized Consensus in Wireless Environments: A Case Where More is Better* Proceedings of the 2010 Third International Conference on Dependability, pp. 7–12

[23] Aspnes, J. (2003) *Randomized Protocols for Asynchronous Consensus* Journal of Distributed Computing, vol. 16, pp. 165–175

[24] Aspnes J. (2002) *Fast deterministic consensus in a noisy environment* Journal of Algorithms, vol. 45, no. 1, pp. 16–39

[25] Thom, C., Gilley, D.C., Hooper, D., Esch H.E.: The Scent of the Waggle Dance. PLoS Biology (2007)

[26] Hayes, J.: Pleasure chemical controls bee dance. Cosmos Online (2007)

[27] Gadagkar,R.: The Honeybee Dance-Language Controversy. Resonance: Journal of Science Education, vol. 1, pp. 63–70 (1996)

[28] NLANR/DAST: Network Performance Measurement. `http://sourceforge.net/projects/iperf/`, (2012)

[29] Mittelbach F. and Goossens M (2004) *The LaTeX Companion* (Addison-Wesley), 2nd ed.

[30] The Network Simulator NS2, http://www.cs.virginia.edu/ cs757/slidespdf/cs757-ns2-tutorial1.pdf

[31] Aguilera, M.K., Toueg,S.: The correctness proof of Ben-OrŠs randomised consensus algorithm (2011)

[32] Taylor, I., Downard, I., Adamson, B., and Macker, J. AgentJ: Enabling java NS-2 simulations for large scale distributed multimedia applications. In Second International Conference on Distributed Frameworks for Multimedia DFMA 2006, Penang, Malaysia, pp.1-7, (2006)