A security protocol for mobile agents based upon the cooperation of sedentary agents

Abdelhamid Ouardani, Samuel Pierre*, Hanifa Boucheneb

Mobile Computing and Networking Research Laboratory (LARIM), Department of Computer Engineering, École Polytechnique de Montréal, P.O. Box 6079, Station Centre-ville, Montréal, Québec, Canada H3C 3A7

Accepted 25 April 2006

Abstract

Despite its many benefits, mobile agent technology results in significant security threats from agents and hosts. This paper presents a protocol which protects mobile agents from malicious hosts. This protocol combines four concepts: the cooperation between a mobile agent and a sedentary agent; the reference execution (reliable platforms which shelter our cooperating sedentary agents); the cryptography and the digital signature to ensure safe inter-agent communication and time-limited execution (timeout). A dynamic approach which makes use of a timer to make it possible to detect a mobile agent’s code re-execution was used. The attack on agent permanent modification was also dealt with. Moreover, the protocol is sufficiently robust so that it is durable and fault tolerant.

Keywords: Mobile agent; Security protocol; Malicious host; Cooperating agent; Sedentary agent

1. Introduction

With the expansive development of the Internet, the need to develop applications which can be exploited within such heterogeneous and very disparate environments becomes imperative. The client/server architecture demonstrates a real ability and sufficient performance to head up this development. However, it has been confronted with problems of maintainability, and thus displays some limitations when dealing with heavy network
loads. These limitations worsen, especially with mobile terminals and portable devices, which add other paradigms. A mobile agent appears to be an alternative solution as it is more adaptable to these new requirements (Poslad et al., 2001). To reach a certain degree of maturity, this paradigm should guarantee a minimum level of security.

An architecture of mobile agents mainly consists of two actors: the mobile agents and the platforms (Bäumer et al., 1999; Poslad et al., 2001). An agent is a software entity that can act in favor of another entity (a person, another agent). It can be mobile, sedentary or stationary. A mobile agent is an agent that is able to migrate between hosts according to a route, which is either static (i.e., established beforehand) or dynamic. A sedentary agent is a mobile agent which carries out only one single migration. The stationary agent does not migrate anywhere. An agent consists of three components: the code or the program which implements it, the execution state of the program and the data. There are two types of migration: weak migration occurs when only the code of the agent migrates to its destination, a strong migration occurs when the mobile agent carries out its migrations between different hosts while conserving its data, state and code. The platform is the environment of execution. The platform makes it possible to create mobile agents, it offers the necessary elements required by them to perform their tasks such as execution, migration towards other platforms and so on.

This paper proposes a protocol based on sedentary and cooperating agents to protect mobile agents against malicious hosts. This protocol combines four concepts: the cooperation between mobile and sedentary agents, the reference execution within a trusted platform (reliable platforms which shelter our cooperating sedentary agents), the cryptography and the digital signature to make inter-agent communication secure, and the time-limited execution (Timeout). A dynamic approach which makes use of a time-limit to make it possible to detect a mobile agent’s code re-execution was selected. The main contributions of this paper are: the design of an approach to estimate the time a mobile agent takes to carry out a task on a platform in order to detect a code re-execution attack; the design of some mechanisms which allow the protocol to recover after service or permanent code modification has been denied and to become fault tolerant in case the trusted platform breaks down.

The rest of the paper is organized as follows. Section 2 presents the background and related work by summarizing the security problems and protection approaches of mobile agents. Section 3 presents the foundations and specifies the proposed protocol, its behavior in case of failure or attacks such as the denial of service, code re-execution, and/or permanent code modification. Section 4 gives some implementation details and analyzes some experimental results. Section 5 concludes the paper.

2. Background and related work

This paper focuses on the protection of mobile agents against malicious hosts (platforms). This issue is problematic given the fact that the platform is the environment for execution required by these agents. The platforms hold all the elements necessary for the agents to operate correctly. Moreover, they can reach the various components of the mobile agents: the code, the data and the state. Some researchers treat mobile agent security by studying attacks (Jansen and Karygiannis, 1999; Jansen, 2000; Schnoebelen et al., 2001). They analyzed the attacks, presented the relationships between them (El Rhazi et al., 2003; Sander and Tschudin, 1998) and identified their level of importance.
(El Rhazi et al., 2003). It was concluded that eavesdropping attacks, for example, were the most dangerous as their occurrence can lead to other even more dangerous attacks, such as attacks on data modifications.

Security threats can be classified into four categories (Schnoebelen et al., 2001): integrity attacks, availability attacks, confidentiality attacks, and authentication attacks. Integrity attacks include any attack which modifies or interferes with the code, the state or the data of the agent. Availability attacks occur when an authorized agent cannot access objects or resources to which it has the right. Confidentiality attacks happen when the components of the agent are accessed illegally by a malicious platform. Authentication attacks occur when a mobile agent cannot identify and validate the current platform to which it is assigned.

Although researchers have tried to propose approaches to protect mobile agents, unfortunately, they have yet to find a complete and effective measure of protection. Each method focuses upon protecting one or more elements of the agent. The existing methods differ in their degree of capacity to protect an agent and to immunize it against, or at least detect, potential attacks. At this stage, it is interesting to specify that the moment of detection occurs either immediately after the attack was made or even later. Moreover, even if the protection is incomplete, it is still necessary as it limits the damage and thus minimizes the consequences. Some researchers have proposed approaches based on the calculation of cryptographic functions (Sander and Tschudin, 1998), which aim to prohibit decrypting the agent’s code. One problem with this approach is that it is impossible to find cryptographic functions for all of the problems.

A mobile agent system, called Ajanta (Karnik and Tripathi, 2000), offers mobile agents three mechanisms of protection. For the first one, the agent’s author can declare a part of the agent’s state as read only (ReadOnlyContainer). The second makes it possible for the mobile agent to create append only logs (AppendOnlyContainer); however, this approach is not appropriate when two platforms are working in collaboration. The third is selective revelation which enables the platform of origin to appropriate certain information from the agent to well-defined platforms (TargetedState). In other words, only part of the agent’s data will be visible to the platform for which it is intended. Others have proposed approaches which protect the agent’s state or behavior by comparing it to one which has been used as a reference (Farmer et al., 1996; Hohl, 2000; Minsky et al., 1996; Vigna, 1998). These approaches detect attacks even if some platforms are working in collaboration (Minsky et al., 1996).

A secure protocol has been proposed in El Rhazi et al. (2003), and makes use of a sedentary agent which carries the mobile agent’s critical code (the part of the code that consists of its objective function, for example). This cooperating agent is carried out within a trusted platform to insure its itinerary. All of its results are compared to those previously received from the mobile agents from each platform visited to detect any wrong execution by the agent. Then, it sends the originating platform a report identifying all of the malicious hosts which attacked the mobile agent. This approach also uses two time counters (Hohl, 1998) to detect mobile agent re-execution and denial of service. To insure confidential communication between the mobile agent and its cooperating agent, this approach uses signed data sent through encrypted messages based on a public/private key system. It can detect attacks like code re-execution, denial of service, wrong execution, itinerary modification, and eavesdropping. The drawback to this approach is the estimate made by the time counter. The technique determines the estimate by using another agent to simulate the mobile agent. Thus, the owner of the agent can deduce the execution time.
required on a platform. However, this mechanism might make mistakes depending on platform quality (a fast platform can re-execute the agent without being detected, while a slow platform can be considered as suspect yet it is honest). Also, a permanent modification of the mobile agent can effect the agent as it continues its migration uselessly in so far as that all future hosts visited (after a code modification attack) will be considered as being malicious even if they are not. In this new approach, these drawbacks will be overcome in order to improve the protocol described in El Rhazi et al. (2003), in terms of code re-execution detection, denial of service recovery, permanent code modification protection and fault tolerance in case the trusted platform crashes.

3. Secure protocol based on sedentary agents

This new approach combines four concepts: a reference execution within a trusted platform, cooperation between agents, digital signature and encrypted communication. In this section, the foundations and the components of the protocol are presented. Then, its behavior towards some hostile environments or system failure is described. Finally, the robustness of the protocol is discussed.

This protocol uses three different entities: a mobile agent (MA) and two sedentary agents (SA and SAR). The MA migrates between a set of hosts (Pi, which could either be malicious or honest) forming its itinerary Id(Pi), Prev and Nexti the Pi’s identity, Pl-1’s identity and Pi+1’s identity, respectively. Time represents the MA’s time of execution and Pi, Tai and Tid, respectively, designate the inter-arrival on platform T of messages Arrival()-Input() and Input()-Departure(). Note Timeout and IAS as the estimated time counters after which a code re-execution and denial of service will be detected. Also note TimeoutSA as the maximum amount of time required by SA to cooperate with an MA and update the SAR. Note Enc and DDec as the encryption of the data X using Pi’s public key, DPi(X) for the decryption of X using Pi’s private key and SIGPi(X) for the digital signature of X using Pi.

Assuming the existence of two trusted platforms within the protocol: a primary one called T where the SA is executed, and a second platform for recovery called TR on which the SAR is carried out. The mobile agent code is divided into two parts: the sensitive one is called the critical code and the non-critical code. The critical code is also retained by the SA. Assuming that the messages exchanged between agents are carried through authentic channels, to insure communication, an asymmetrical cryptographic system (El Rhazi et al., 2003), must be made available so it will be possible for these agents to get their private and public keys required for encryption as well as a one-way hashing function to make it possible to send signed data.

At the initial stage, the protocol is launched starting from the platform of origin which creates the SAR, the SA and the MA. It initializes the route recording parameters (Prev0 with Id(O) and Next0 with Id(P1)) for the agents. It also initializes the various parameters of Timeout time, IAS and TimeoutSA with the preliminary values chosen. Also, it provides the MA with a list of the known malicious platforms to be avoided, and charges a list of alternate platforms (the utility will be explained further) on the SA. It then sends the SAR and SA, respectively, towards the TR and T platforms. For the last step of this stage, it sends the MA towards the P1 platform. The MA knows the identity of the T and TR platforms and those of the SA and SAR platforms.
3.1. Intermediary stage i

During step $i$, the two cooperating agents ($MA$ and $SA$) exchange three types of messages: $Arrival()$, $Input()$, $Departure()$. $Arrival()$ contains the identity ($Idi$) of the current $Pi$ platform; and the identity of the $MA$ $SIG_{Pi}(Id_{AM})$ signed by $Pi$; $Previ$ the identity of platform $P_{i-1}$ (the preceding platform along the $MA$’s path), and the current state of the $MA$ ($StateMA$). $Input()$ contains $Idi$ the identity of platform $Pi$, $X$ the data input provided by $Pi$, and $SIG_{Pi}(X)$ the signature of data $X$ using the public key of $Pi$. Finally, the $Departure()$ message contains the identity of platform $Pi$ ($Idi$), the results $R$ obtained by the mobile agent, the $SIG_{Pi}(R)$ signature on $R$ using the public key of platform $Pi$, the identity of the next platform ($Pi+1$: $Nexti$) and the state of the $MA$ ($StateMA$). Fig. 1 illustrates the structure of the three messages which are exchanged during each step $i$. The whole protocol will be summarized later on Fig. 4.

Once it arrives at $Pi$, $MA$ sends an $Arrival()$ message to the $SA$ which is awaiting the $MA$’s messages. The $SA$ verifies its authenticity by examining $Id_{AM}$. It also checks if the $MA$ really migrated towards platform $Pi$ as expected ($Next_{i-1} = Id_{i}$). If this is not the case, it recognizes an itinerary modification attack (El Rhazi et al., 2003; Allée et al., 2005; Roth, 1998), and it then examines the $MA$ to see if $Previ$ is equal to $Id_{i-1}$ which ensures that $P_{i-1}$ actually sent the $MA$. If the preceding equations are satisfactory, then the sedentary agent arms the timer ($Timeout$) and waits to receive other messages from the mobile agent. This timer is used to measure the execution time of the $MA$ on platform $Pi$. If it expires, the $SA$ will suspect a re-execution of the mobile agent. The $SA$ also initializes a timer which will measure $T_{ai}$ (inter-arrival of $Arrival()$ and $Input()$ messages coming from the $MA$). Before executing the critical code, the $MA$ sends an $Input()$ message to its cooperating agent. This message is encrypted with the public key for the $T$ platform. In this message, the mobile agent provides the $SA$ with the data required to execute the critical part of the code. The $SA$ agent extracts the data signed by platform $Pi$ with its private key, then, it stops the timer and evaluates the $T_{ai}$. Using this value, the $SA$ calculates the right time $T_{id}$ (inter-arrival of $Input()$ and $Departure()$ messages coming from the $MA$) before it receives the $Departure()$ message. In fact, the cooperating agent ($SA$) uses a function $f$ which implements an empirical law in order to deduce $T_{id}$ from $T_{ai}$ ($T_{id} = f(T_{ai})$). Then, the $SA$ deduces the $MA$’s total execution time on current platform $Pi$ using the following formula:

$$Time = g(T_{ai} + T_{id}) = g(T_{ai} + f(T_{ai})), \quad (1)$$

where $g$ is a function which implements the relationship between the execution time of the $MA$ and the sum of the inter-arrival time of the $Arrival()$–$Input()$ and $Input()$–$Departure()$ messages. This $Time$ value is then corrected by a multiplying a coefficient for safety estimated during the experiments carried out by the protocol. Hence, one obtains the value of the new $Timeout$ which takes into account the characteristics of platform $Pi$ in terms of

<table>
<thead>
<tr>
<th>Arrival():</th>
<th>$Idi$</th>
<th>$SIG_{Pi}(Id_{MA})$</th>
<th>$Previ$</th>
<th>$StateMA$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input():</td>
<td>$Idi$</td>
<td>$X$: Data</td>
<td>$SIG_{Pi}(X)$</td>
<td></td>
</tr>
<tr>
<td>Departure():</td>
<td>$Idi$</td>
<td>$R$: Results</td>
<td>$SIG_{Pi}(R)$</td>
<td>$Nexti$</td>
</tr>
</tbody>
</table>

Fig. 1. Structure of the messages exchanged during each step.
performance, the number of agents carried out and also the load on the communication channels between $P_i$ and $T$. In other words, the cooperating agent readjusts the preliminary value of the $Timeout$ already initialized to detect a possible attack on code re-execution (Fig. 2).

Whereas the mobile agent continues its execution, the $SA$ also begins to carry out all of the critical code, since it has received all of the necessary input. Once finished, the $MA$ sends a $Departure()$ message $(Id(P_i), R, SIG_{P_i}(R), Next_i, State_{MA})$. It encrypts it with the public key for platform $T$. When the $SA$ receives this message, it decrypts it using its private key of platform $T$. It then extracts the results and checks the signature of platform $P_i$. The results are compared with those obtained by the $MA$. If they are not identical, a modification of code attack is detected. Finally, the $SA$ records this information (intermediary results, route, and credibility of $P_i$) and waits until the mobile agent migrates towards the next platform.

As previously mentioned, the $SA$ readjusts the $Timeout$ of the protocol, which is armed as soon as it receives the $Input()$ message. Therefore, if the $Timeout$ expires before receiving the $Departure()$ message, then the $SA$ recognizes this platform as a re-execution attacker. Otherwise, it continues to wait for the $Departure()$ message within another interval of time ($IAS$) El Rhazi et al. If the $Departure()$ message still has not arrived, then the $SA$ will detect a denial of service attack (Fig. 2).

Before starting another step, the cooperating agent makes a summary of the $Arrival()$, $Input()$ and $Departure()$ messages, produces and sends a new $UpdateSAR()$ message to the $SAR$ to provide it with information about the latest execution of the $MA$ in platform $P_i$. The $UpdateSAR()$ message, whose the structure is shown in Fig. 3, contains the identity of the cooperating agent encrypted with the private key of $T$, the identity of the $MA$, the identities of the previous (Previ), the current ($Id_{P_i}$) and the next ($Next_i$) platforms and both the input ($X$) and the results ($R$). It also contains a vector ($Attacks$) of the attacks detected within the current execution. Once it receives the $UpdateSAR()$ message, the $SAR$ records the data it contains, starts its $TimeoutSA$ timer to detect a possible breakdown of

---

**Fig. 2.** $Timeout$ estimate based upon inter-arrival times.
the sedentary agent on platform $T$. Timeout$_{SA}$ is estimated by the SAR at the beginning of the protocol using the same approach that the basic protocol (El Rhazi et al., 2003) uses to estimate its Timeout. Indeed, before its migration towards the $T$ platform, the MA’s cooperating agent initially sends its first UpdateSAR() message to the SAR which is already running on platform $TR$. Then, the SAR simulates an execution of the MA (including the messages of cooperation) and sends another UpdateSAR() message to the SAR. Thus, one obtains an estimate of the inter-arrival time of the two UpdateSAR() messages between two executions. To this estimate, the SAR adds the value of the IAS, so, the protocol will ignore if the SAR suspects a breakdown of the SA (due to the long delay) while it is still running. The SA does not declare a denial of service to the MA (the IAS has not yet expired) as it is still waiting for its cooperating messages.

During the stage $i$, three different situations can occur: a fault on the $T$ platform, a denial of service or a permanent code modification. How will the protocol react to each of these situations?

**Fault on the $T$ Platform:** The SAR does not cooperate with the MA during its various migrations except during breakdown situations. It only interacts with the SA. In fact, the mechanism works as follows: the cooperating agent sends the SAR a summary of the three messages exchanged through an UpdateSAR() message. Once it receives this summary, the SAR starts a timer with the value of Timeout$_{SA}$. Thus, if the SA does not send any more messages and the timer expires, the SAR suspects a breakdown of the primary cooperating agent (SA). Consequently, it sends a changeCo-operanting() (a hello) message to replace the current cooperative agent of the MA to inform it that it will become the new cooperating agent. The MA starts by sending the three previous messages (Arrival(), Input() and Departure() as described above) which precede the detection of a breakdown. The changeCo-operating() message contains the identity of SAR ($Id_{ASR}$) and the signature of platform $TR$ on $Id_{ASR}$ ($SIG_{TR}(Id_{ASR})$).

**Denial of Service:** This occurs when, for a given reason, the MA cannot continue to function. The protocol detects this situation using a timer IAS (additional waiting interval). To avoid restarting the protocol from the beginning, the cooperating agent creates a copy of the updated MA with all of the information that the previous mobile agent had. It is for this purpose that the cooperating messages contain the StateMA parameter, thus making it possible for the new copy to start from where the previous copy was stopped functioning. Then, the cooperating agent sends the copy to an alternative platform chosen from a previously determined list (at initial stage). Once on the new platform, the new MA can continue its mission.

**Permanent Code Modification:** This is suspected when the cooperating agent detects two consecutive attacks on a simple modification (on two platforms $P_i$ and $P_{i+1}$). In this case, it will eliminate the mobile agent on platform $P_{i+2}$ since it has already migrated from there. Then, it will send a valid clone towards the previous platform ($P_{i+1}$) and wait for the cooperation messages from the new MA. Once it receives the Departure() message, it will
compare the results received with those obtained from the original \( MA \). If they differ, it concludes that \( P_{i+1} \) is actually a malicious host and it is identified as such.

### 3.2. The final stage

At the end of the protocol, the mobile agent returns to its platform of origin \((O)\). The latter identifies which sedentary agents \((SA\) or \(SAR\)) cooperated with the \( MA \). It requires a final report and a list of the malicious platforms. The platform \( O \) compares the two reports to eliminate the malicious host’s results. Fig. 4 illustrates the whole protocol.

The detection of a code re-execution is based on a dynamic method of estimation of the \( Timeout \). This approach does not leave a malicious platform an opportunity to perpetrate an attack on re-execution without being detected. Not only does it detect the denial of service, but it also makes it possible for the protocol to survive such an attack. One could even indirectly consider it a measure of detection and protection. As for the attack on permanent code modification, the protocol allows for the recovery in case of the breakdown of one of its entities, which makes it possible to avoid several useless migrations. Also, the new protocol invalidates the need for an entity called the estimator agent since the protocol performs their required estimates. However, the proposal certainly produces some constraints, such as the addition of a second platform of confidence, which

---

**Fig. 4.** Secure protocol based on sedentary agents.
generally consists of a paying service, possibly increasing costs. However, the recovery procedures after the faults are detected are expensive in terms of time, memory resources and create traffic problems within the network. Indeed, they are necessary in order to generate a valid clone of the mobile agent if a denial of service occurs, to eliminate the MA and to generate its clone if a permanent code modification occurs, and finally to retransmit certain messages if an SA breakdown occurs.

4. Implementation and experimental results

To implement the protocol, the Grasshopper (Baümer et al., 1999; IKV++, 1998) platform was used. The agents communicate together by remote invocation. Considering this fact, interfaces were designed within which the agents could communicate. In other words, sending a message to an agent consists of remotely calling a method on the interface of the target agent which then implements the message. Indeed, each agent has an interface that contains the methods to which it can react. However, before initiating the communication, the agent that starts the interaction initially needs to locate its target entity. To do so, it must build a proxy of this object by looking for it within the region to which it is assigned (a Grasshopper logical domain within which all the participating entities are registered).

The platform is organized as follows: all the participating entities are registered in a logical domain called a region. This entity contains a set of agencies (a logical subdomain), each of them can receive and create agents; it also contains special places where the visiting agents can be executed. These special places provide services such as encryption and digital signature to make it possible for agents to exchange authentic messages.

When the MA runs on a Pi platform and before sending its first Arrival() message, it must initially activate the security context, i.e., the supplier IAIK_JCE3.03 (Institute for Applied Information, 2002), which provides the elements that function within a security maneuver. Consequently, for all future platforms visited, it would thus be necessary for the MA to activate this context before starting its execution: it is what is called the “cache” aspect. In addition, there is the so-called “proxy” image of the cooperating agent that the MA must build in order to locate its interlocutor. Thus, the MA could call upon the arrival() of I_SA method of implementing the Arrival() message. This is true for the first migration as, once on the platform Pi+1, the MA already knows the address of the proxy; therefore, it does not need to look it up. Also, at the time of recovery after a denial of service, the MA is obliged to build the proxy before carrying out the cooperation. Consequently, during the experiments these two aspects were taken into account: the proxy of the current cooperating agent and the activation of the security functions of the cryptographic places (cache). Thus, the following two scenarios were selected for testing: no cache/no proxy (worst case), and no cache with proxy (best case).

Table 1 shows the characteristics of machines used in this experiment. It also shows the location of the agencies and places. All of the machines were connected to an Ethernet 100 Mbps.

4.1. Evaluation of timeout

A series of experiments were conducted using both scenarios to find empirical functions to represent these relations: \( T_{id} = F(T_{ai}) \) and \( \text{time} = G(T_{ai} + T_{id}) \).
### 4.1.1. No cache with proxy (best case)

Initially, the results obtained compared to $T_{id} = F(T_{ai})$ measurements show a correlation between $T_{ai}$ and $T_{id}$, as presented in Fig. 5. The dotted line was obtained according to a linear approximation whose equation is as follows:

$$T_{id} = -0.2079T_{ai} + 307.53. \quad (2)$$

Note that the values are not too dispersed. There is a variation of approximately 20 and 30 ms, respectively, in the direction of $T_{ai}$ and $T_{id}$.

During the second trial, the results (shown in Fig. 6) were obtained and are representative of the time $g(T_{ai} + T_{id})$. The $Time$ parameter is proportional to the sum of $T_{ai}$ and $T_{id}$ due to the fact that, if the exchanged messages (represented by $(T_{ai} + T_{id})$) take awhile to arrive at their destination, it is due to the execution of $MA$ (Time) took up some of their allotted time. The variation is mainly due to the fact that the time required by the $MA$ to build the proxy as well as the time to start the $PlaceCrypto$ (place of platform $P_i$) service is not considered in the $T_{ai} + T_{id}$ value. In fact, the evaluation of the latter $(T_{ai} + T_{id})$ starts only after the $Arrival()$ message has been received. As a result, the searching proxy and starting the cryptographic services of the place procedures are not taken into account when it comes to measurement. On the other hand, to measure the value of the $Time$ parameter all of the steps are considered. They are obtained through a linear approximation, the relationship that binds the two parameters. It is determined by
the following function:

\[
Time = 1.3049(T_{\text{at}} + T_{\text{id}}) - 62.218. \tag{3}
\]

Thus, by combining relations (1) and (2), the following relation is obtained to connect \( T_{\text{at}} \) to \( Time \):

\[
Time = 1.033T_{\text{at}} + 401.29. \tag{4}
\]

In fact, the execution time of \( MA \) on a platform \( P_i \) is estimated if the \( MA \) already has the proxy of the cooperating agent. The estimate is made according to the inter-arrival time of the first two (\text{Arrival()} and \text{Input()}) messages. In order to estimate a value for our \( Timeout \), a series of measurements are compiled to compare the actual execution time of the \( MA \) with the estimate according to relation (3). Indeed the latter (3) was introduced into the code of the sedentary agent to calculate \( Timeout \) and some improvement were conducted using coefficients. The results obtained by comparing, the average deviation calculated to the actual time of execution are shown in Fig. 7.

Notice that the standard deviation of the estimate, corrected by a 0.975 coefficient, is the best in absolute value. This solution was retained as the protocol of the best case, as it is the most likely to respect the predetermined time estimate.
4.1.2. No cache/no proxy case (worst case)

As in the preceding case, the \( T_{id} = f(T_{ai}) \) and \( Time = g(T_{ai} + T_{id}) \) correlations were selected. For the first, using linear approximation, the correlation was obtained by the following equation:

\[
T_{id} = 0.0429 T_{ai} - 185.65. \tag{5}
\]

For the second relation which binds \( Time \) and \( T_{ai} + T_{id} \), a series of experiments were carried out in order to obtain the following relation (linear approximation):

\[
Time = 1.0241(T_{ai} + T_{id}) + 216.49. \tag{6}
\]

By combining the two relations (4) and (5), the relation that connects \( Time \) to \( T_a \) must be

\[
Time = 1.068 T_{ai} + 26.36. \tag{7}
\]

Thereafter, this latter relation was introduced into the code of the cooperating agent in order to be able to dynamically estimate the \textit{Timeout}.

Finally, to be able to compare these results to the reference protocol, it was necessary to see the variations compared to the effective execution time and, using some weighting, the adequate approximation of \textit{Timeout} was obtained and adopted for the chosen protocol. The coefficients for the reference protocol will be those previously indicated, as this protocol does not distinguish between these two scenarios. The results obtained appear in Fig. 8 along with the two approaches and the values of the estimates illustrating the variation compared to the execution time. The quality of the approximation is very clear: the basic protocol penalizes a slower platform. For this protocol, the corrected estimate does not penalize a less powerful platform nor does it give a fast platform an advantage as the variation is minimal. Thus, for performance evaluation purposes, coefficients 1.001 and 2 were respectively adopted for the protocol and the basic protocol.

Note that the protocol dynamically chooses the coefficient to be adopted according to the scenario. Thus, for the first execution and for an execution after a breakdown or a denial of service, the protocol chooses the estimate of the worst case and, for any other execution, chooses to adopt the estimate of the best case.

![Fig. 8. Comparing the mean deviation of the estimates of Timeout from the real execution time of both protocols (worst case).](image)
4.2. Evaluation of timeoutSA

Note that the value of TimeoutSA depends on IAS (El Rhazi et al., 2003), because if the cooperating agent detects a refusal of service, it will send an UpdateSAR() message. However, if the SAR has already signaled a $T$ breakdown due to a refusal of service, it was mislead, therefore, the estimation is based on the evaluation of the time of inter-arrival of the UpdateSAR() messages. At the beginning of the activation of the protocol, the SAR migrates towards the TR platform. When the primary cooperating agent is created on the platform of origin $O$, it sends an UpdateSAR() message. Then, it migrates towards the $T$ platform and simulates the execution of the $MA$. Finally, it sends a second UpdateSAR() message.

A series of experiments were carried out on the two scenarios investigated (best and worst case) in order to correct this estimated time (time of inter-arrival of UpdateSAR()) messages. So, in the best case (Fig. 9a), the variations are positive for the two coefficients (1.7 and 1.8.). These results show that the protocol will allow more time than estimated for the inter-arrival of the two UpdateSAR() messages. Also, the probability that the protocol will be mislead is 12.5% for the 1.7 coefficient and 0% for the 1.8 coefficient. Thus, the

Fig. 9. (a) Comparing the mean deviation of the estimate of the TimeoutSA compared to the time of inter-arrival of the UpdateSAR() (best case). (b) Comparing the mean deviation of the estimate of the TimeoutSA compared to the time of inter-arrival of the UpdateSAR() (worst case).
second correction of 1.8 will guarantee that the timer will not expire due to a delay or a re-execution of the MA.

For the worst case experiment (Fig. 9b), two proposals were presented as they were closest to the effective time. The variations are certainly positive, but the probability of the protocol being mislead is 0% for a coefficient of 6.15 whereas it is 14.28% for a coefficient of 6.10. Thus, the first proposal was adopted as the protocol.

As in the estimate of Timeout, this protocol chooses one of the coefficients elected according to the scenario.

4.3. Evaluating the implementation

The performance of this protocol was evaluated in relation to the basic protocol. Therefore, the following aspects were taken into consideration:

- The detection of agent re-execution in order to measure the accuracy of the prediction of the Timeout.
- The amount of traffic generated between the platforms which take part in the protocol.

4.3.1. Quality of the approach

After comparing this protocol and the basic protocol (El Rhazi et al., 2003), another comparison was performed in a hostile environment (a platform which intentionally re-executes the code). In the worst case the correction coefficients remain the same: 1.001 for this protocol and 2 for the basic protocol. Then, the detection rate was measured. The two protocols show a 100% detection rate. The rates are certainly satisfactory. However, the variation of the Timeout is very large in the case of the basic protocol because the re-execution of the code for the mobile agent requires simply the re-execution of its sensitive code (critical()) which is in fact a small function compared to the total execution time of the MA. In other circumstances, if the MA is carried out on a slow platform, the protocol will probably be erroneous considering that the Timeout estimate is too short and will thus penalize less powerful platforms. In the second case, the rectification coefficients of 0.975 for this protocol and 0.65 for the basic protocol were adopted. A percentage of 35.3% was obtained as a rate of detection for the basic protocol and 100% for this protocol, which confirms both the refinement and the quality of this approach. Moreover, this protocol has a reasonable standard deviation compared to the execution time of the malicious platform. Thus, it could not be erroneous in terms of the integrity of a slow platform. At the same time, it will avoid a very powerful and malicious platform in order to pass unnoticed.

4.3.2. Analysis of traffic generated

The amount of traffic generated between the various participating platforms is usually (43%) due to the exchanges between the O and T platforms of origin. This is due to the fact that on platform T, two entities are carried out: the agency which carries the SA agent and the Region to which all the agencies, places and agents of the protocol belong. Thus, all the migrations will have to be deferred there (to the register of the region). Considering that migrations are weak (by instantiation and not an entire migration of the code), the agents are thus instantiated on all the platforms that have been visited, whereas the code remains on the platform of origin. This platform will have to send packages to T to update the
register of the Region. In addition, the traffic generated due to the exchanges between $T$ and $P_1$ is equivalent to the traffic between $T$ and $P_2$. This is due to the fact that both $P_1$ and $P_2$ perform the same role. However, the protocol adds 13% to the amount of traffic due to the SAR, but this traffic could be partially compensated by eliminating the agent estimator AE of the basic protocol. Note that this protocol presents an 11.7% increase of the size of the $SA$ compared to the basic protocol, while the $SAR$’s size (14 KB) is double the size of the estimator agent. In part, that constitutes the price to pay to have a secure and robust protocol.

5. Conclusion

This paper presented a secure protocol for mobile agents. This protocol combines the reference behavior, cooperation between agents, cryptography and timed execution techniques. It is a reactive protocol in the sense that it adapts itself to the environment and the circumstances in which it operates. The protocol uses the behavior of reference which makes it possible to partially (critical code) or entirely carry out the mobile agent in a non-hostile environment (trusted platform). The principle of cooperation associated with the first aspect makes the verification of the execution more dynamic than ever. Thus, the detection of several attacks becomes instantaneous. The encryption and the digital signature of the communication between participating agents make for authentic and confidential cooperation. Moreover, the execution time limit can only help the mobile agent to avoid any illegitimate execution, as it enables the system to detect an attack such as the re-execution of agents.

The reactivity of the protocol to its environment (being able to dynamically adapt itself to the characteristics of the current platform) enables the system to estimate the value of the Timeout—the code re-execution detection. This estimate deals with the platform parameters (performance, load, etc.) without questioning a malicious platform and likely receiving the wrong information. Thus, the integrity and the confidentiality of the mobile agent are not compromised. In addition, the protocol was equipped with a counter-measure against the attack of permanent modification to the code. This reactive method makes it possible to mitigate and rectify the damage caused by permanent code modification. Since a possible breakdown of the trusted platform will certainly stop the cooperation, a second agent was introduced to watch for the breakdown of the first cooperating agent. This will guarantee a continuity of the protocol since the agent will continue its mission under full security.

Future research shall address the limitations presented by this protocol. An in-depth treatment of the timer for denial of service detection will make it possible to obtain a complete measure of detection and protection against this type of attack, which partly facilitates the estimation of the TimeoutSA. In addition, a replication was implemented in only one direction between the $SA$ and the $SAR$. However, it would be best to be able to enforce the protocol with a total replication, i.e., the two agents $SA$ and $SAR$ should have identical roles. Thus, in the case of the restoration of the $SAR$ after a breakdown of $T$, the protocol will have to update the $SA$ after its recovery with all the events which occurred in its absence in order to perform a handoff.

Note that this protocol has been formally validated in an upcoming paper using the model checking technique (Schnoebelen et al., 2001).
References


