A Secure Code Deployment Scheme for Active Networks

Leïla Kloul, Amdjed Mokhtari† *

Abstract

Active Networking is an innovative technology which can open the network and make it more flexible. But, introducing active code within the network increases the network vulnerability from security point of view. The security is always considered as a separated component or an upper layer of the active network architecture. In this paper, we develop a global security architecture for a safe code distribution. A three level mechanism is defined to provide a unique identification, authentication, and classification of the code, its developers and its users.

Index Terms : Active networks, Code distribution and identification, Network security.

1 Introduction

Beside the advantages of opening the network such as introducing quickly new protocols and applications within the network, injecting a smart program in the network nodes can affect their performance. Moreover, this new task for the routers can deteriorate functions of forwarding packets and increase dramatically the security failures. Usually the security enforcements are operated in active network architectures as an upper layer added at the end of the design process. Those enforcements are typically an admission control interface or a simple authentication mechanism based on Public Key Infrastructure (PKI). However, the security mechanisms are is not integrated into the lower layers like the code distribution and execution. Our goal is to build a global architecture where the security system is included in the conception of the code distribution and execution systems. In our architecture, the security mechanism interacts with all actors which are concerned by the injection of the active programs within the network and their execution. Those actors or entities can be identified as the code provider or developer, the code user and the code itself. Because they are external to the network, these three entities push us to define a three level security mechanism which defines a strong and a unique identification, authentication, and classification of the entities.

The development of a global security mechanism implies the definition at the same time of a global architecture for the code distribution. Both aspects are strongly linked and must be led seriously to implement a secure active network.

The code distribution is the main key to the introduction and implantation of the active network in the current Intranet and Internet. Without an appropriate code deployment scheme, in terms of performance (time of downloading, loading and executing the code) and security, the active network stays in the state of a project without an effective implementation. We divide the code distribution into two main phases, the code identification and its deployment. Code identification consists in an universal mechanism which gives to the code a unique and non-ambiguous identifier which is independent of its developer and its users and making it easy to share. The code identification has not found yet a normalisation in the active network. Each project defines its own identification method. The code deployment phase consists of the policy to follow which allows the active programs to reach the desired network nodes.

The code distribution security enforcement allows the right developer to publish its code, the network to check the safe execution of the new code in the nodes and finally the right user to deploy the code through the network nodes and to execute it.

The three poles relation between the code, the developer and the user leads us to define a three poles security architecture. The security function at the developer level must at least check if this developer is

*PRiSM, Université de Versailles, 45 av. des Etats-Unis, 78000 Versailles France. E-mail: {kle,amok}@prism.uvsq.fr, † corresponding author.

authorised to publish its code. The developers of active codes must be authenticated and classified into different groups. The distinction between the groups is based on their relation with the network, nodes resources and the user applications. Different classes can be defined in order to give several levels of grants. The differentiation between the developers creates a set of code categories. Each category of program corresponds to one class of users. The security function at the code level verifies if the active program adheres to the security rules defined by the network. The safety requirements must be related mainly to the resources consumption and the data access. Finally, the security function at the user level must allow the nodes to formally verify if the user, through its data packets, has the authorisation to execute the referenced code with few computation and communications.

In this article, we define a new approach of code deployment and identification in active networks. Our objective is to provide a secure mechanism which guarantees a unique identification of the code and allows its use by other users in addition to its owner. This approach combines different techniques and is mainly based on the notion of the code server. This server is able to achieve other tasks such as the attribution to each application of a unique identifier, the verification of the conformity of the code, the authentication of code developer and user and the storage and publication of the codes. We study also the security issues to establish the minimum requirements for a secure active network. We propose a new approach which places our code server in the heart of a global architecture interacting with the developer, the code, the user and the node to set up the previous security functions.

In the following section, we give an overview of the code distribution in the current active network. Based on the minimal requirements that are fundamental to build a secure active network environment, in Section 3, we define a new approach for code security and distribution. We study the performances of our appraoch in terms of throughput and latency in Section 4. The results obtained are compared with those of the *hop by hop* technique developed for ANTS platform [19]. The related works are presented in Section 5. Our conclusions and the possible extensions of this work are pointed out in Section 6.

2 Code distribution overview

The code distribution consists in injecting active programs in a semi-closed system constituted of the network routers. This operation must provide an homogeneous mechanism of identification of the active programs inside and outside this system.

We distinguish between two independent steps in the distribution of the code. The first step is the effective deployment of the new application or the active service in the concerned active nodes. The second is the identification of the code.

2.1 Effective deployment of active code

The main key of active networking is to load dynamically a new code into the network nodes. The way the code reaches the nodes determines the chosen policy of code deployment in the network. The recent active networks projects fall onto two approach to deploy their code, the *capsule based code distribution* and the *switch based code distribution*.

Capsule based code distribution: in this approach, the code is transmitted, in band, at the same time as the data; the capsules carry the active code and the data. When an active packet arrives at the active node, this one loads immediately the code part of the packet and executes it using the corresponding execution environment. As the code is loaded in the nodes along the packet path, only the concerned nodes are programmed. Thus this scheme adapts itself dynamically to the packets path changes.

The inconvenient of this approach is that the presence of the active code in the data packets increases the packet length [4]. Consequently the response time (treatment and propagation time) is sharply increased. And we know that the network protocols are limited in term of packet size.

To improve this aspect, a *hop by hop* based approach has been proposed in [19]. In this technique, a node receiving a capsule which does not contain the active code, extracts the *reference* of this code and checks its presence in the cache. If the code is in the cache, the node downloads it and executes it on the

data. Otherwise, the node requests it from the active node previously visited by the packet (a previous hop address field is updated at each node). This process is repeated hop by hop.

Switch based code distribution: This approach is characterised by two distinct phases. The first phase consists of sending, out of band, the active code to all nodes, through the possible data packet paths, generally one potential path for each user session. The second phase is the sending of the data packets. Here, the nodes are pre-programmed, because a code is sent before the data. With this approach, we can expect that unknown and several paths can be taken by user data packets. Thus we cannot know exactly the nodes to pre-program.

Both approaches have the following characteristics in common:
- An active code cannot be stored indefinitely in the active node.
- It is difficult to share the active code. The user must ask for the code from its developer or to develop it himself again and deploy it in the network.
- An active node cannot check itself the integrity and the authentication of the received code and its owner. The program verification requires a lot of processor time and memory space and its execution may penalise the node performance.
- With the switch based approach we must consider two delays. The time to pre-program the nodes and the session time, i.e. the time to receive the user active data and to execute the corresponding active code. The session time may be small but we may expect the total time to be considerable. Similarly, with the capsule approach (hop by hop), the code downloading time and the session time are mixed. The session time includes the downloading time which may be considerable.

2.2 The code identification issue

Because of the multiple code sources and the multilevel node architecture, a unique and multilevel identification is essential. The code identification must allow the code to be shared and reused by other users.

The code identification has not found a normalisation in the active networks yet. Each project defines its own identification method. Currently several projects such as ANTS [20] and Switchware [2] use a three layers based architecture developed in [5]. In this architecture, the identification of the application data is specific to each session user. A data packet must contain the identifier of the active application (AA) and also the identifier of the corresponding execution environment (EE) which is able to interpret the AA.

The reference of the EE is given only once by the Active Networks Assigned Number Authority (ANANA). The encapsulation packet ANEP (Active Node Encapsulation Packet), a standard in terms of format of packets, includes a dedicated field of 16 bits length to the EE identifier. At the arrival of a data packet to an active node, this node loads the EE which its identifier is specified in the packet. Once loaded, the EE must then trigger the loading of the corresponding active application.

Each EE is basically able to manage several active applications at the same time, to load them in memory and to eliminate them according to user's needs. This implies that, like the EE, each application must have a unique identifier. However as the identifier of an application is currently defined by its developer and each application can be developed by an independent user, the problem of the multiple identifiers can be inevitable. Indeed, because of the multitude of the sources of code, their independence and the absence of central regulation authority, situations where two active applications have the same identifier are not impossible. Moreover, as the application identifier is known only by its developer, the code of this application can hardly be shared with the other users.

The identification depends also on deployment techniques. In the capsule based technique, all data packets are active and identifies the couple AA-EE treating it. On the other hand, in the switch based technique, a filter is installed at the NodeOS level which is responsible to recognise the potential passive flows on which to apply the active code. The filter uses packet header information (such as packet type, source address and/or destination address) to orient packets.

In following section, we present a new approach which integrates code deployment and security. This approach constitutes a response to the problems of identification, verification and the sharing of the code. It allows the authentication and authorisation of the entities manipulating this code.

3 A new security approach for code distribution

The requirements to deploy the code must consider the security at the level of three entities: the code developer, the code user and the code itself. We can summarise the minimal requirements as follows:

- a unique developer identification, its authentication and hierarchical developer rights to publish a code according to the code effect on the network. Moreover, it requires a certificate.
- a unique user identification, its authentication and hierarchical user permissions to execute a code. Moreover, it requires a certificate.
- a unique code identification and a safety control mechanism and a key as a hash code.

To satisfy these security requirements, we define a *Code Identification and Storage Server* (CISS). To be able to manage our three poles structure, a central authority acting as a code server plays a crucial role in the verification of the code safety and the authentication of both the developer and the user.

The *CISS* associates a unique identifier to an application and places at the disposal of all network users this identifier and its code. With this technique sharing the code becomes easier, because the user who wants to use an existing code has only to reference it.

In our approach we assume that the certificates are delivered by CAAN (Certificate Authority for Active Network). It is an autonomous authority which delivers a certificate based on the PKI (such as X.509). The certificates must contain the public key, information about the holder and its class.

Moreover, we consider a *publication web site* where a user can have knowledge about the existing active applications. This web site allows the visitors to compose their applications by choosing different code modules. The access to this web site takes into account the user rights. Only applications of the same or lower class are showed to the users.

We establish another security requirement to authenticate all routers, CISS, CAAN and the publication web site server by giving them a certificate or at least a couple of key.

3.1 User and developer classification and authentication

We can expect that most end users will not program the network nodes. The code users and the code developers can be different entities. For this reason we differentiate between who publishes the codes and who use them.

We classify the developers and the users into hierarchical classes according to their permissions and the executed code effects on the network. To simplify this model, we consider only two classes. These classes are mainly the user application class and the network application class. A typical example of the first class developer is the Internet Service Provider manager (ISP) and its customers are considered as users in this class. The code of this class can be executed on the user data only. The second class gathers the developers and users who are allowed to operate on the network level and can change the node behaviour by introducing new protocols and routing algorithm codes. This is the case, for example, of the network administrator. Note that each class can be divided into several subclasses.

Thus, the node resources access depends on this classification. This distinction is a first fire-wall against malicious codes as it allows a first control of the code execution. It is managed by CISS during the code publication phase.

3.1.1 The identification

The identification of the developer or the user must allow us to determine exactly to which class he belongs. The CISS requests the *certificate* from the CAAN. This certificate contains information about the holder (name, address, ...), its class and also its *public key*. The attribution of the developer identifier can be performed by this trust authority.

3.1.2 The authentication

The authentication process is based on the PKI. At the beginning of this process the CISS sends a private information (a simple random text to strengthen the process) to the developer or to the users. When the developer receives this information, it crypts it with its own private key and sends the encrypted information to the CISS. Using the developer public key, the CISS decrypts the received information. If the decrypted

information and the original one match then the developer is authenticated. Otherwise, it is another entity acting on behalf the real one. Therefore, the developer or the user is strongly identified and authenticated and cannot deny or repudiate the code sent or executed. Moreover, the untrusted entity which is not identified (does not hold a certificate) cannot send, publish or deploy any code.

3.2 Code distribution phases

Our approach consists of three phases as follows:

1. Code publication phase: the application developer sends to the CISS its active code, in order to publish it. This phase is noted by 1 in Fig 1. After verifying the code safety using the Proof-Carrying Code (PCC) technique and authenticating the user certificate with CAAN (exchanging Certificate Check-CC message), the CISS sends an Active Code Identifier (ACI) (2 in Fig 1) to the developer.

PCC technique [15] is used for a secure execution of untrusted code. In our context, the CISS establishes a set of safety rules that guarantee a safe behaviour of programs. According to the developer class, the *CISS* requires different rules. The rules for user data level code developers are more restrictive than the rules for network level code developers. In the other hand, the code developer creates a formal safety proof that proves the code adherence to the required safety rules. Then, the *CISS* is able to use a simple and fast proof validation system to check, with certainty, that the proof is valid and hence the code is safe to be executed [15] by the nodes. It is possible to use a heavy code analysis mechanism to check memory use and CPU performed by CISS.

If one of those tasks fails, the CISS sends a negative message to the developer. Once the ACI is sent to the developer, the CISS generates a unique key for the code (Kc) to be used when checking if a certain user has the right to execute this code. Once all information about the application service are available, a notification is sent to the publication web site (3 in Fig 1) which generates a web page containing the application identifier and a description of all its modules.

2. Code referencing phase: when a user wants to customise the treatment of the network nodes on its data packets, it connects to the publication web site to choose the application service and notes its ACI (4 in Fig 1). With the ACI and its own certificate, the user requests from the CISS (5 in Fig 1) the generation of a symmetric key (K) to be able to deploy the code. Once the CISS is sure about the user rights to deploy and execute the code by checking the user certificate (CC message exchanged with CAAN), it generates a key K and sent it to the user.

We adapt a distributed symmetric key generation based credential technique, used by [11], in which the key is generated as a combination of the code key, user key, address source, the authorisation time and a validity period. The key is hashed by the CISS and sent to the users. With this technique the active node can verify the user authentication and authorisation without connecting to any third authority, without additional computation and without holding any further users list.

In [11], the code key is generated and shared frequently by an authorisation authority and different code servers. To secure this generation, the Kc is now generated by the CISS itself as a hash-code signed with CISS private key. This hash-code allow the node to verify at the same time the code integrity and CISS authenticity. The symmetric-key offers good performances because the node does not perform heavy computations or further communication or authorisation servers.

3. Code deployment phase: the user sends its packets (6 in Fig 1) with the reference of the application service and the generated symmetric-key K through the network. The node receiving such packets and holding the code referenced and its corresponding key (*code key*), reconstitutes a symmetric-key K' using the code key sent by the CISS, the user key, the source address, the session time and the validity period sent by the user. In the case where the node does not hold the referenced code and its code key, it must send a demand of code (7 in Fig 1) to the CISS. Consequently, the CISS provides the node with the desired code and key (8 in Fig 1). If both code keys match, the node can execute the code on the user data.

This approach can be considered as an intermediate between the integrated and discrete approaches. Indeed the active code is downloaded out of band (discrete approach) from a certain server. The data

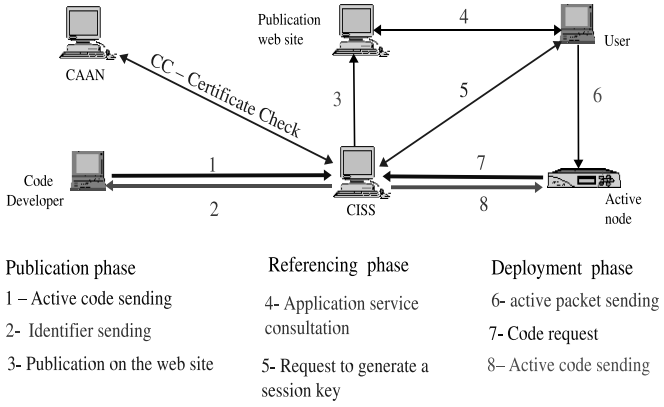


Figure 1: Active code distribution phases

packets (at least the first ones) arrive at the nodes before the code and can reference an inexistent code in the node cache (integrated approach).

This method solves the problem of multiple paths that may be taken by the packets of the same application, which is a characteristic of IP networks. It allows also the node to not keep a code indefinitely because the codes are always retrieved from the CISS. It is important to note that using an event based system, it is easy to consider the arrival of standard packets as an event and to treat them as active.

3.3 The multi-CISS approach

The solution with one code server can present a major drawback, the *CISS* can become a bottleneck of the network. Indeed, it can be congested if a lot of nodes send requests at the same time. One approach to avoid the congestion of the CISS is to add other CISS servers in appropriate places in the network. The first step performed when a passive node becomes *active* is to detect all CISS present in its neighbourhood. Based on the RTT (Round Trip Time) time counted from the set of CISS responses, the new active node orders its CISS neighbours according to their distance.

When the search of a code is needed (the active code is not available in the active node), the node requests it from the first CISS in the list of CISS. This one responds with a negative message or the active node does not receive a message after a notified RTT from this server, the active node sends a new request to the second CISS. This process is repeated for the other CISS until the active node receives the appropriate active modules.

To avoid the saturation of the network by multiple code packets sending by the CISS, the active node which wants an active module can begin a negotiation protocol by sending, at first, a *request for code message* or *notification of code existence* to the set of CISS. Each CISS able to satisfy the code request sends to the active node an equivalent Ack packet, otherwise, it sends an equivalent Nack packet. When the current node receives an Ack packet, it can send to the chosen CISS, and only one, a *downloading request*. At this time a CISS receiving the downloading request message sends to the node the code. We assume that negotiation protocol messages are small packets and their transfer in the underlying network is as fast as 'syn' messages.

The next paragraph presents another approach which we consider as a mixed solution, between CISS and the *Hop by Hop* method.

3.4 The mixed approach

We expect that the CISS will be placed in the edge of the global network, in the subdomains. For the nodes close to CISS servers (belonging to the same domain), the number of the routers separating them is very small and the RTT is also small. However, for the nodes in the other domain the path to CISS is longer and downloading the code from a nearest node is more suitable.

The CISS approach evolve easily with other techniques and can be combined with the *hop by hop* scheme for its connectless and flexibility. In this combination, the CISS plays its classical role of attributing and saving the active codes and also has the role of the first and permanent source of codes. In this case we can consider either one or several CISS. The hop by hop code migration intervenes to reduce the load on the *CISS* and also to improve the code transfer time by choosing the nearest source.

Moreover, the combination may be an alternative in the case where an active node cannot receive a code or receives a negative response from the CISS. This can be due to, for example, one of the following reasons: the CISS is congested, the link is temporary broken, bottlenecked, the server is off or does not have the right code in the case of several CISS managing a distributed code base.

Using this approach requires two steps which must be followed and the distinction between the first active node and the other nodes.

1. *Code injection*: this is the first time the user sends its data packets with references to the code to the first attached active node. At this moment, the "previous hop" field of the data packet is empty. This first node is the first visited node and must download the code and the corresponding key (Kc) from the CISS if the active code is missing. Before forwarding the data packet, this active node puts its address in the previous hop field.

2. *Code migration*: the next active nodes visited by the data packet look for the previous hop field which determines the new source of the code, and the code request must be sent to the previous node. However, the other nodes must use this same code migration technique to find the code and will have to ask the CISS for the code only in the error case. The previous hop field should be updated at each node. When receiving the code and its key from the previous node, the current node can also perform the symmetric key verification to authorise the code execution.

With this technique, the CISS is requested only once for all the data session.

4 Performance Analysis

In this section, we are interested in the performance evaluation of the code deployment techniques presented previously. We compare the two approaches *single CISS* and *multi-CISS* that we developed for our platform ARFANet (Active Rule Framework for Active Networks) [3] and the approach *hop by hop*. We are interested particularly in the throughput of the network for the active packets and end-to-end latency for this type of packets.

The considered topology of the network is composed of one CISS and tree active nodes and two passive nodes. In the case of the multi-CISS approach, a second CISS is added to this topology. We varied the arrival rate of the packets to this network from 250 $packets/s$ to 4500 $packets/s$ for a size of the packets of 200 $bytes$.

According to our analysis of the impact of the active packets introduction into a network on the standard packets, several proportions of active packets were considered [9]. This work showed that the proportion $K = 20\%$ of active applications does not deteriorate the active nodes performances with respect to the standard packets. Also, in this study, we retained the same proportion of active packets. This proportion includes the packets of data, programs as well as the code requests packets sent to the CISS. The data packets sent in the network can reference four different types of applications. The applications are published on the server for the CISS approaches and are provided by the source node for the *hop by hop*.

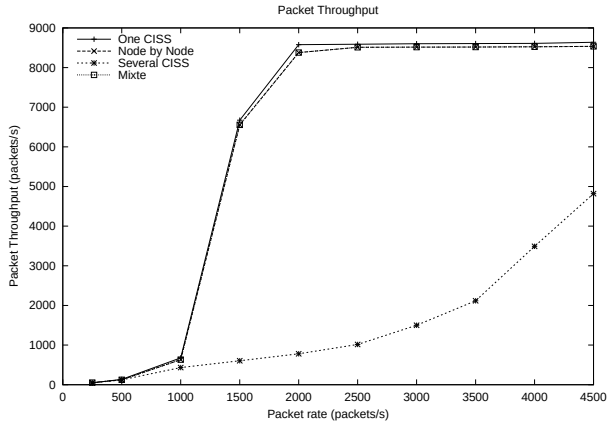
Our tests were performed on Linux machines with a processor "AMD Athlon(tm) XP 1500+" of 1339 MHz, 256 KB of cache memory a 256Mo read-write memory.

4.1 The throughput:

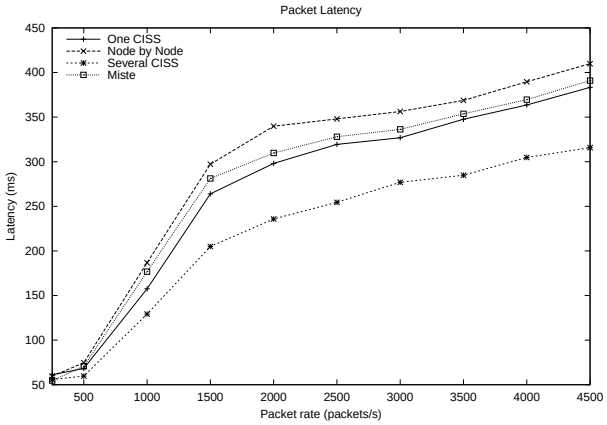
Figure 2(a) shows the throughput of the network for the active packets according to the total arrival rate of the packets (active and passive) for the three techniques. These results show that although the throughput of the *single CISS* technique is slightly higher than the throughput of the technique *hop by hop*, the two throughputs remain very similar. We can note that the two throughputs are stabilised at 8500 $active\ packets/s$ from a

high arrival rate (2000 *packets/s*). It is interesting to note these results because the proportion of active packets arriving at the network is only of 20% i.e. 400 *active packets/s* when the total arrival rate is 2000 *packets/s*. This high throughput compared to the real arrival rate of the active packets is due to the fact that the first data packets of an application arriving at a node will have to wait until the node receives the corresponding code from the CISS (single *CISS*) or from the previous node (*hop by hop*). Once the code received, an immediate processing is carried out on the data packets accumulated in the queue causing a successive sending of the packets treated towards the following node. This phenomenon is repeated for all active nodes forwarding the packets, leading to a throughput which is much higher than the arrival rate. Consequently, along the way of the data, the rhythm of migration of the active packets does not respect necessarily the rhythm of the source.

In addition, we observe that the throughput obtained with the *multi-CISS* approach is significantly lower than those obtained with the two previous approaches, in particular, the *single CISS* approach. This is due to the presence of an additional CISS in this approach. This additional server allows to decrease the load from the first CISS and to absorb the part of the requests for codes sent from the nodes in its immediate neighbourhood. However, the first data packets of an application arriving at a node will wait the arrival of the code to the node less longer than in the *single CISS* approach. The packets will tend to less accumulate in the queues of the nodes. Thus, the throughput is less significant than in the other approaches, although it still higher than the arrival rate. The difference between the throughput of the network for the active packets and the arrival rate of this type of packets remains less significant than for the other approaches. For an arrival rate of 400 *active packets/s*, the throughput is of 800 *packets/*, i.e. 10 times less significant than in the *single CISS approach*. We can note, when the arrival rate increases, the throughput increases too until reaching 5000 *packets/s* for a total arrival rate of 4500 *packets/s* (900 *active packets/s*). As conclusion, the more arrival rate increases, the more both CISS are requested and the data packets wait in the nodes.



2(a) *Throughput*



2(b) *The latency*

4.2 Latency

In this part, we study the end-to-end latency experienced the active packets for the three approaches. In Figure 2(b), although the nodes must request the code from the CISS which can be some routers away, the latency for the active packets is better in the *single CISS* approach than in the *hop by hop* approach. This can be explained by the fact that the active nodes in the second technique have to manage also the code requests emanating from other nodes. This additional task has an impact on the latency. This figure shows also that the introduction of an additional CISS (muti-CISS approach) allows to improve end-to-end latency of active packets. As explained previously for the throughput, the second CISS takes in charge a part of the code requests, those coming from the active nodes in its neighbourhood. According to these results, it is clear

that the management of the code by one or more dedicated entities allows us to have better performances that when this task is achieved by the nodes themselves.

5 Related Work

Beyond the two main techniques, integrated and discrete there is not many works in code distribution area. We can divide the current contributions into two main categories according to the use or not of a *code server*.

The first category gathers the works in which a code server is used for an out-band code deployment (discrete approach). The fist technique introducing Code Server notion for the storage and the publication of active programs is Distributed Code Caching for Active Networks (DAN) in [6], which has been proposed for high performance active routers. Nevertheless, this work is neither motivated by an universal active code identification nor by proposing safety and security verification mechanisms.

Future Active IP Networks project (FAIN) [12] is a component based approach. The components are stored in a code server and compose a service. The service is identified as a concatenation of the service provider name and the service name. Note that with this identification method the service stays strongly linked to his provider which must guarantee the service name unicity. The user data are oriented to the corresponding service by a channel installed at the NodeOS level without the need of the service reference. This manner limits the service users to whom belonging to the service provider customers. The main difference between our code distribution system and FAIN is that this one is dedicated to component based architecture rather than packet based one.

Code Distribution Scheme for active networks (CDS), proposed in [21], is a set of recommendations for the design of Code Distribution Protocol (CDP). The authors recommend out-band code transport rather than in-band transport arguing with the large capsule size. They suggest also the store of AA with its code using the code server. Considering naming and name resolution method of an AA as a basic function of a CDP, the authors suggest that the name of an AA should be universal.

The projects of the second category are considered mainly in-band code deployment. The basic idea is the use of a *unique key* [19] hold by the code developer and provided by a trust certificate authority to create a unique reference. With the key, the developer defines a one way MD5 hash code with 128 bits length on code content. This approach can be useful in order to limit the number of code developers authorised to deploy their codes in the network. However, this approach makes hard the share and the re-use of the code by other users because the code reference is strongly linked to the code and its owner.

The security must rely on how the code is distributed and the elements which take part in the process of injecting the code into the network (the developer and the user). Several works, prompted by the security workgroup [7], are leaning towards secure execution rather than secure deployment. Projects like SANE [1] and SANTS [14] provide a secure framework for active nodes. Another technique adopts a different method to secure the execution of codes by restrictive active languages like PLAN [8] and SafetyNet [18].

ROSA [11] defines a user authorisation scheme based on the symmetric-key generation. The generated key corresponds to a key related to the code itself, shared between the code server and the authentication authority, and the session time and duration. The task cannot be performed only if the user has the grant to deploy the code. However, ROSA does not define identification and publication mechanisms and does not secure the developer side. We have adapted and improved this technique to be a part of our global security scheme.

6 Conclusion

The processes of identification and deployment of code define respectively how the code is identified by the different entities of the network (nodes and users) and how it reaches the nodes. In this paper, we developed a new approach based on the classification of the developers and the users according to their rights in the network. Two mains categories can be distinguished. The first one can be the application level related particularly to the user data. The second category concerns the network level which is more restrictive and reserved to the network manager. We developed also a new mechanism for active code distribution based

on the use of a code server (CISS) which is the central point of a three poles security architecture. The CISS attributes a unique identifier and checks the developer grants and code safety before its publication. Moreover, this server allows the user authentication and authorisation by the generation of a symmetric-key which involves code key, user key and a valid session time. This technique avoids to the nodes to perform a hard task of verification which arises a lots of computations and communications. The CISS mixes the advantages of the two approaches of code distribution, the approach based on the pre-programming of certain active nodes and the *hop by hop* which uses capsules of code. We compared the CISS performances to the *hop by hop* in terms of latency and throughput. The results obtained showed that the use of the code server gives better performances. Although our approach was applied in the context of our framework ARFANet [3], it can be used for any kind of active applications and in other existing platforms. The use of the *Publication web site* allows an easy access to all published applications and their composition.

An extension to this work would consist in analysing the different management techniques for distributed databases for a possible application within the case of several and distributed CISS maintaining an harmonious and reliable security protocol. In particular, we are interested in the number of CISS, their location in the network and the distributed code base management.

References

[1] D. Alexander, W. Arbaugh, A. Keromytis, and J. Smith. Safety and security of programmable network infrastructures, 1998.

[2] D. S. Alexander, W. A. Arbaugh, M. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. Nettles, and J. M Smith. The switchware active network architecture. pages ,, Juillet 1998. Departement of Computer and Information Science University of Pennsylvania.

[3] M. Bouzeghoub, L. Kloul, and A. Mokhtari. A new active network framework based on active rules. Rapport de Recherche 21, PRiSM, 2002.

[4] R. Braden, B. Lindell, S. Berson, and T. Faber. The asp ee: An active network execution environment. Ieee cs press, Proceedings of DARPA Active Networks Conference and Exposition (DANCE'02), 2002.

[5] K. Calvert. Architectural framework for active networks. Rapport de recherche, AN Architecture Working Group, Juillet 1998.

[6] D. Decasper and B. Plattner. Dan - distributed code caching for active networks. In *IEEE INFOCOM*, San Francisco, Avril 1998.

[7] AN Security Working Group. Security architecture for active nets, 2001.

[8] M. Hicks, P. Kakkar, J.T. Moore, C.A. Gunter, and S. Nettles. Plan : A packet language for active networks. pages ,, Juillet 1998. Departement of Computer and Information Science University of Pennsylvania.

[9] J. Hillston, L. Kloul, and A. Mokhtari. Towards a feasible active networking scenario. *Telecommunication Systems*, 27(2-4):413,438, October - December 2004.

[10] S Kuchinskas. A decade of e-commerce. Site web, http://www.internetnews.com/ec-news/article.php/3422901, October 2004.

[11] Maria Calderon Marcelo Bagnulo, Bernardo Alarcos and Marifeli Sedano. Provinding authentication & authorization mechanisms for active service charging. *Lecture Notes in Computer Science*, Zurich, Switzerland(2511):337,346, October 2002.

[12] Hideki Otsuki Matthias Bossardt, Takashi Egawa and Bernhard Plattner. Integrated service deployment for active networks. *In Proceedings of Fourth Annual International Working Conference in Active Networks (IWAN 2002), Zürich, Switzerland*, December 2002.

[13] W. Miao. Worldwide ip traffic patterns and network evolutions. http://www.atmforum.com/meetings/bbx-july01.html, Advanced Consumer Service Research-Probe Research, 2001.

[14] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee. Strong security for active networks, 2001.

[15] G. C. Necula and P. Lee. Research on proof-carrying code for untrusted-code security. In *In Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, 1997.

[16] S. Paltridge. Internet traffic exchange and the development of end-to-end-international telecommunication competition. Technical report, Organisation for Economic Co-operation and Development, Mars 2002.

[17] UNCTAD. E-commerce and development report 2003. Technical report, UNITED NATIONS-New York and Geneva, 2003.

[18] Ian Wakeman, Alan Jeffrey, Tim Owen, and Damyan Pepper. Safetynet: A language-based approach to programmable networks. *Computer Networks*, 36(1):101–114, 2001.

[19] D. et al. Wetherall. Ants : A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH*, San Francisco, Avril 1998.

[20] D.J. Wetherall. Active network vision and reality: lessons from a capsule-based system. *Operating Systems Review*, 17th ACM Symposium on Operating Systems Principles (SOPS'99)(34):64, 79, December 1999.

[21] Y. Zhou, Y. Zhang, and J. Lu. Cds: a code distribution scheme for active networks. *Computer Communications*, 27(3):315,321(7), 15 February 2004.