Multidomain Network Based on Programmable Networks: Security Architecture

Bernardo Alarco, Marifeli Sedano, and Maria Calderon

This paper proposes a generic security architecture designed for a multidomain and multiservice network based on programmable networks. The multiservice network allows users of an IP network to run programmable services using programmable nodes located in the architecture of the network. The programmable nodes execute codes to process active packets, which can carry user data and control information. The multiservice network model defined here considers the more pragmatic trends in programmable networks. In this scenario, new security risks that do not appear in traditional IP networks become visible. These new risks are as a result of the execution of code in the programmable nodes and the processing of the active packets. The proposed security architecture is based on symmetric cryptography in the critical process, combined with an efficient manner of distributing the symmetric keys. Another important contribution has been to scale the security architecture to a multidomain scenario in a single and efficient way.

Keywords: Security networks, programmable networks, authorization, programmable services.

Manuscript received Apr. 8, 2005; revised Nov. 11, 2005.

This work has been funded by CICYT under project AURAS (TIC2001-1650-CO2-01/02).

Bernardo Alarcos (phone: +34 91 885 6628, email: bernardo@aut.uah.es) is with the Department of Automática, Universidad de Alcalá, Madrid, Spain.

Marifeli Sedano (email: marifeli@gsi.dit.upm.es) is with the Department of Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, Madrid, Spain.

Maria Calderon (email: maria@it.uc3m.es) is with the Department of Ingeniería Telemática, Universidad Carlos III de Madrid, Madrid, Spain.

I. Introduction

The enormous growth of the Internet in the last few years has brought about new problems and needs that have triggered the creation of new services on the edge of the network [1]. Some examples of these services are security components introduced into the network such as firewall or intrusion detection systems, caching devices, translation of IP address (NAT), the transcoding of multimedia flows to adapt them to the network, or link requirements.

In this scenario, multiple services can be offered to users. The scientific community has been working on the introduction of new services in a flexible and dynamic manner. In this direction, new technologies based on introducing programmability into the nodes have been proposed, opening the node interfaces, and developing technologies to introduce the services.

A programmable network is made up of programmable nodes within an execution environment to process special packets called *active packets*, to offer programmable services. The active packets are sent by end systems (sources) to other end systems (destinations). The active packets can carry data and control information. The control information is able to configure the behavior of the programmable nodes, allowing the data to be processed (and even modified) by the programmable nodes, before progressing towards the destination. Thus, programmable nodes offer programmable services to users. Depending on the approach towards the programmable network, the code to execute the active packets can be loaded into the execution environment from a code server, or carried in the active packet itself. If the code is downloaded from a code server, the active packets will carry a code reference.

In this article, we define a pragmatic scenario of a multiservice network based on programmable network technologies and propose a security architecture for this scenario.

Focusing on this subject pragmatically, we have considered that the multiservice network will be deployed on the Internet, and that the programmable services will be offered by service providers. In this scenario, the service provider should be capable of obtaining benefits provided by the services; hence, commercial aspects must be taken into account. We must bear in mind that the users of a service must be authorized beforehand, and when they have used the service, they must not be able to deny that they have requested it. In this scenario, the security risk must be analyzed in order to avoid illegal uses of programmable services.

Another important aspect of a pragmatic multiservice network is the multidomain feature. A service would imply the cooperation between different service providers, every one of which has a different administrative domain. The multidomain scenarios introduce scalability challenges into the security architecture.

The security risks in this scenario have been analyzed by focusing on the security from the service provider point of view, looking at how the service provider can be protected from attacks originated by malicious users. In this respect, the main components to be protected in the programmable network are the programmable nodes, and the main security risk would be caused by malicious active packets or malevolent code.

False active packets could cause an attack on the programmable nodes, changing their behavior or consuming resources in an unauthorized manner. The security architecture must verify that the active packets are authentic and authorized before processing them in the programmable nodes. The programmable node must not waste too many resources on the security processing of an active packet because the flow of active packets can be high.

The code that is introduced into the programmable nodes to process the active packets must be controlled to avoid programmable nodes executing malicious or uncontrolled codes.

Therefore, the security architecture must protect the programmable network from malevolent codes and active packets as well as bearing in mind that some active packets could proceed from other domains. The security solution must be scalable.

As we will see later in this paper, the main contributions of this article are the pragmatic vision of the multiservice network model and the efficient manner of introducing security into programmable nodes.

First, we introduce the programmable network technologies. Then, in section III we define a pragmatic multiservice network based on programmable networks as well as analyzing the security requirements of the multiservice network. After that, we describe previous works in programmable network security accomplished by other research groups. Next, in section V, we present the architecture of security. In section VI, we present an analysis of the tests carried out on the implementation of the security architecture. And finally, section VII details our conclusions.

II. Programmable Networks

Programmable networks introduce programmability in the network. There are two different trends in introducing the computation plane inside the nodes: *Opensign*1) and *Active Networks* [2], [3].

The concept of Opensign emerged from the telecommunications companies and standardization organizations in order to introduce programmability into the nodes of the network. The Opensign community advocates that programmability can be achieved by means of defining a series of open network interfaces that represent physical network devices and network services as distributed objects. Opening the interface of the nodes, third party applications can control the resources of them. Some examples of the proposals are a generic framework for providing programmability, IEEE P1520 [4], and the general switch management protocol (GSMP) applied to ATM switches, q-GSMP [5], or to IP routers, e-GSMP [6]. FORces [7] is a working group of the Internet Engineering Task Force that proposes a generic architecture based on the Opensign ideas, applied to Internet routers.

The concept of active networks emerged from discussions within the Defense Advanced Research Projects Agency (DARPA) research community in 1994 and 1995 on the future direction of networking systems. The active network community advocates a dynamic approach through which active packets can offer services on demand as they carry the executable code. In some active network proposals, the active packet carries the user data and executable code that must process them. In other proposals, the active packet carries a reference to the executable code, which is downloaded onto the programmable node separated from the user data.

There are different approaches in active network technologies. In the programmable switch approach, the packets keep the existing format and provide discrete mechanisms for supporting the downloading of the code that processes the packet. In contrast, the capsule approach introduces a new type of packet called an active packet, which

1) Open Signalling Working Group. http://www.comet.columbia.edu/opensig/

can carry user data and the code to process it. In some proposals (in-band) the active packets carry the code, and in others (out-of-band) the active packets carry a reference to the code. In the latter case, the code can be downloaded from another programmable node or from a code server.

DARPA has been the main promoter of the active network technologies[2] [8], developing an experimental network called ABone [9]. Some Information Society Technologies (IST) projects such as Global Communication Architecture and Protocols (GCAP) [10], and Future Active IP Networks (FAIN) [11], have been developed to propose different programmable network technologies. These projects follow some pragmatic ideas: allow the network administrator to control the codes used to process the active packets, define a business model, and create a mix of the Opensign and active network ideas.

III. The Security Problem of Multiservice Networks Based on Programmable Networks

1. A Pragmatic Vision of Multiservice Networks Based on Programmable Networks

In this section, we describe the scenario of a multiservice network based on programmable network technology. We will use pragmatic trends in programmable networks to define the generic features of the multiservice network.

Basically, a programmable network is made up of a number of programmable routers (called programmable or active nodes) inside an IP network. The programmable nodes identify special packets called active packets and load a specific code to process them. Active packets go from an end-system source to an end-system destination, and the programmable nodes in the path between the source and the destination processes the active packets, as shown in Fig. 1, with a specific code.

In some programmable networks, the users can introduce their executable codes into the programmable nodes, but this is not a pragmatic solution because of the risk of introducing malicious codes. So, in order to obtain controlled codes from the network administrator, we propose using code servers. Every active packet carries the code identifier that executes the programmable nodes to process the active packet itself. When a programmable node receives an active packet, if it does not have the code to process it, it will download the code from a code server.

The programmable nodes consume resources when the multiservice network offers the services demanded by the users. We will define a service model that forces the users to request a service before using it, so the multiservice network can accept

2) Darpa Active Networks Program, 1996. http://www.sds.lcs.mit.edu/darpa-activenet/

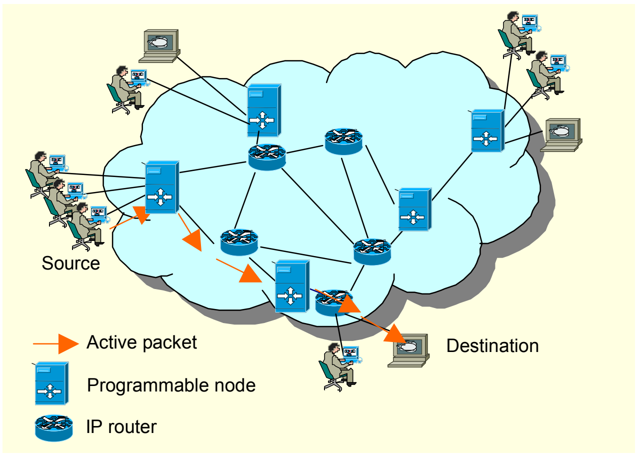

Fig. 1. Scenario of a programmable network.

or refuse the service according to the available resources. These services must be controlled in order to offer just the authorized services. So, the programmable nodes must only process the active packets that belong to an authorized service.

There are some approaches towards a programmable network in which the programmable nodes need to know who its closest programmable nodes are in order to send them the active packets, while in other approaches the programmable nodes that process active packets do not need to know this information (its IP address). In this case, the active packets are sent to the destination and are intercepted by the programmable nodes in the path. We suppose that in a generic scenario of programmable networks, the programmable nodes do not need to know the topology (the other programmable nodes). This supposition allows us to propose a generic security solution, valid for both programmable network technology approaches. In addition, it is a pragmatic requirement that the users (end systems) do not need to know the topology of the programmable network, which means that the users do not need to know the programmable nodes (its IP address) in order to send them the active packets.

A programmable network could experience changes in topology that can be produced by changes in the network routes by new programmable nodes that appear in the network or when a programmable node is down. The changes in topology can cause the changes to take place suddenly, as new programmable nodes start to process the active packets of a programmable service, or when other programmable nodes suspend the processing of active packets. The security architecture must be immune to the changes of topology.

A pragmatic network should be capable of deployment in a network such as the Internet in order to reach the end users. The topology of the Internet network is made up of networks belonging to interconnected ISPs. Every network belonging to an ISP has its own administrative domain that could have programmable network technology. When a multiservice network comprises various programmable networks of different administrative domains, the multiservice network will be a multidomain network.

It is assumed by the scientific community that the programmable nodes will be located on the edge of the network, where the number of flows to be processed is fewer than at the core. Thus, we consider that the programmable nodes will be located in the networks of the ISPs that are on the edge of the Internet, which offer services directly to the users. In Fig. 2, we can see the multidomain scenario of the multiservice network.

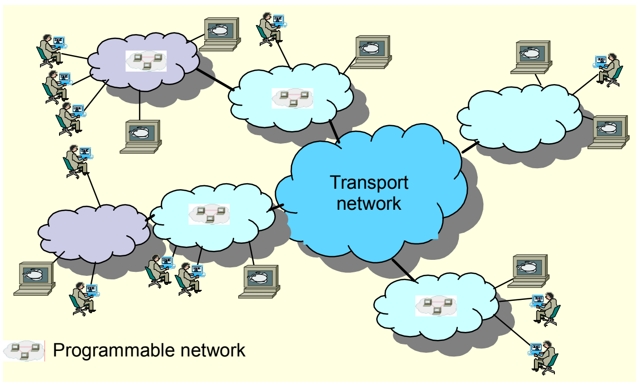

Fig. 2. Multidomain scenario.

2. Security Requirements of the Multiservice Network

The goal of this article is to propose a security architecture to protect the programmable network from malicious users. Another possibility that is not being dealt with here is the protection of users from malicious programmable nodes. This is an interesting point of view, but in real scenarios it is reasonable to trust the programmable nodes of the service provider. In this sense, it is sufficient to guarantee that the programmable nodes that modify the active packets belong to a trusted service provider.

The programmable nodes are especially sensitive to the denegation of service attacks (DoS), since they must carry out a larger process on the packets, than the forwarding process carried out by the traditional routers. Furthermore, the processing of the packets implies the introduction of code into the programmable node, which could be malicious. The active packets carry information that would change the behaviour of the programmable node, so the active packets could be especially dangerous.

The authorization is fundamental in a scenario like this, because of the need to process only the authorized active packets. Therefore, a user must request authorization from the network before sending active packets, and the programmable nodes must verify that the active packets are authorized before processing them.

The user might need to pay for the programmable service received from the multiservice network [12]. To avoid the user denying his responsibility for payment, the authorization request must include the non-repudiation service.

The authentication must be verified in the following situations:

- When a user requests a service from the multiservice network, the multiservice network and the user must both verify that his interlocutor is authentic.
- In the download process of a code from a code server to a programmable node, the code server must verify that the programmable node is authentic. And the programmable node must verify that the received code is authentic.
- When a programmable node receives an active packet, it must verify that the source of the active packet is authentic. The source could be an end system or a programmable node.

The programmable nodes can modify a part of the active packets, called the dynamic part. This situation introduces new security challenges to allow only the modifications made by the authorized programmable nodes. The mechanisms that offer the authentication of the active packets must also offer the integrity service, in order to make sure that the dynamic part of the active packet has not been modified by unauthorized entities. The integrity mechanism must also be applied to the codes exchanged between code servers and programmable nodes, and to the information exchanged to request a service.

An attack that consists of injecting authentic previously stolen active packets would cause an error in the service or simply in the consumption of resources (DoS attack). The authentication and integrity mechanisms do not offer anti-replay services. We must apply anti-replay protection to the active packets in order to avoid degradation of the service.

The definition of the trust relationship in a scenario with programmable nodes is necessary to focus on the security solution properly. The content of the active packets can be modified by the programmable nodes (for example, an audio flow carried by the active packets can be modified by a transcoding process that runs in a programmable node). In addition, a programmable node could generate a new active packet towards the source or destination as the result of processing incoming active packets (that is, if it is multicasting). Programmable nodes must rely on the active packets inserted or modified by other trusted programmable nodes. But we must restrict the scope of the trust relationship between the components of a programmable network. A reasonable limitation includes the programmable nodes and code servers of the same administrative domain. But in the multidomain network, some services require the cooperation between programmable nodes of different domains. Therefore, dynamic trust relationships between programmable nodes of different domains must be established.

The security association between the end system and programmable nodes or between two nearby programmable nodes would be established in a similar way to the security associations between the end systems in IPSec or transport layer security (TLS) protocols. But in the generic and pragmatic scenario of a programmable network, it is not realistic to establish security associations between programmable nodes or between programmable nodes and end systems because of the knowledge implications of the topology.

IV. Related Works

We must protect the active packets using an authentication and integrity mechanism. This protection could be implemented using asymmetric or symmetric cryptography. Asymmetric cryptography (digital signature) has better scalability in a multidomain environment because it uses a scalable public key infrastructure to distribute the public keys. However, the asymmetric cryptographic algorithms require more processing and memory than the symmetric cryptographic algorithms (HMAC). On the other hand, symmetric cryptography introduces the challenge of how to obtain an efficient mechanism for key distribution. We now describe the solution adopted by the main security architectures of programmable networks.

The Secure Active Network Environment (SANE) [13] is a security architecture developed by a research group from the University of Pennsylvania. This architecture uses a mechanism to protect the dynamic and static part of the active packets, using a method based on symmetric cryptography (more efficient). However, they propose a complex system of key distribution made up of different keys: one shared key between the end system and every programmable node, and another shared key between the end system and all the programmable nodes. Furthermore, the mechanism to distribute the keys requires the end users and the programmable nodes to know the topology of the network.

The Active Network Security Architecture (ANSA) [14] is a generic security architecture proposed by a group of researchers inside DARPA. This architecture proposes the use of symmetric cryptography to protect the dynamic part of the packets as well as using asymmetric cryptography to protect the static part of the active packets. ANSA has been proposed to offer security in Abone [15]. An interesting implementation of ANSA is SANTS [16], a security architecture for the main proposal for an active network applied in ABone, called ANTS (active node transfer system) [17]. In ANSA, multidomain security is dealt with using various credentials that carry authorization information to the different domains.

This proposal requires a high CPU consumption and large bandwidth because of the use of asymmetric cryptography. In addition, a hop-by-hop security mechanism is proposed, using a shared symmetric key between nearby programmable nodes. This solution requires both the programmable nodes and the end systems to know the network topology.

Another security architecture [18] based on ANSA has been developed within the IST FAIN project. The difference with SANTS is that the communication between the end system and the first programmable node is based on asymmetric cryptography. Therefore, it is not necessary for the user to know the network topology, but the use of asymmetric cryptography is less efficient.

Some security proposals study the security from the user point of view. They define procedures that allow the user to control the nodes that can modify [19] the active packets.

We can conclude that the state-of-art security proposals do not define the multidomain extension of the security in a complete manner, and that the security solutions depend on the knowledge of the topology. In addition, we have seen that the state-of-art proposals use asymmetric cryptography (as well as symmetric) in the protection of the active packet. However, this solution requires a greater consumption of resources by the programmable nodes.

V. Security Architecture

Preliminary ideas on the proposed security architecture have been published in [20]. In this article, we present a number of improvements on the solution, the multidomain extension, and the tests carried out on a real platform in section VI.

The main contributions of this security architecture to the state of the art are as follows: the pragmatic approach of the multiservice network model, the considerations of efficiency in the solution, and the multidomain architecture. The solution is based on the use of a symmetric key in the critical processes, proposing an efficient way of distributing the symmetric key.

First, we describe the security architecture in a scenario made up of one administrative domain. Then, we will describe how to extend this solution to a multidomain scenario.

1. Security in an Administrative Domain

A. Authorization of Preliminary Definitions

The first problem that we must solve is the authorization of the user to access a programmable service. The user must negotiate with the programmable network to request the service. We define a new component, the authorization server, which represents a programmable network domain in the negotiation process. All the service requests in the same domain are centralized in a single authorization server, which is implemented like a high availability server.

When a user requests a service, the authorization server will generate a response. This response can be positive or negative, depending on whether the user is authorized or not.

We define a single set of authorization parameters that define the programmable service. These parameters are as follows:

- C_i: identification of the executable code that must process the active packets to offer the service in the programmable nodes. Because every programmable service is associated with a different executable code, C_i is really a value that identifies the programmable service that the user requests.
- SST: time in which the programmable service starts. The programmable nodes must not process the incoming active packets before the time indicated by SST.
- SET: time in which the programmable service ends. The programmable nodes must not process the incoming active packets after the time indicated by SET.
- S: IP address of the end system that is the source of the active packets.
- D: IP address of the end system that is the destination of the active packets. S and D allow verification that the active packets are sent between the end systems that have been negotiated.
- U: user identifier. This parameter identifies the user who has requested the programmable service, which is responsible for the proper use of the service.

All these parameters define a programmable service requested by a user. By using them, the programmable nodes can verify whether an active packet has authorization to be processed or not.

This single and generic set of parameters can be increased in some cases, with a specific parameter (SP) that depends on the programmable service. The programmable service is responsible for verifying the SP.

The programmable nodes must apply the authorization policy to the incoming active packets. In order to make this possible, the authorization parameters (C_i, SST, SET, S, D, U, and SP) must arrive at the programmable nodes. So we must define an efficient manner to transport the authorization parameters to the programmable nodes.

In similar scenarios, the authorization information is usually transported in credentials carried by the active packets. The credentials are based on public cryptography (signed by the authorization server or a trusted entity), which requires more processing than symmetric cryptography. Furthermore, because of the dynamism of the network topology, it is recommended that the credential travels within all the active packets.

The use of a credential protected by a digital signature in all the active packets implies large bandwidth consumption. Therefore, we must define a procedure to transport the authorization parameters in the active packets not based on the use of public cryptography (digital signature).

B. Authentication and Integrity of Active Packets

The active packets that arrive at the programmable node must be authentic; meaning that the active packets must have been generated by the user or by a programmable node on behalf of the user. In addition, the active packets must be integral, meaning that the active packets must not have been modified by an unauthorized entity. Only the programmable nodes and end systems are authorized to generate and modify the active packets.

To offer the authentication and integrity of the active packets, we can use symmetric cryptography mechanisms (such as HMAC) or asymmetric cryptography mechanisms (such as a digital signature).

Symmetric cryptography requires the use of a secret key shared among all the programmable nodes and the end systems that process the flow of the active packets. The system based on symmetric cryptography usually requires a great effort in key distribution, which is difficult to scale if the number of components sharing the key increases. However, the use of symmetric cryptography (based on HMAC) is a good solution because of the low resource consumption.

The user and programmable nodes can sign the active packet using a digital signature (asymmetric cryptography). If this solution is accompanied by a public key infrastructure that facilitates the public key distribution through certificates, it will have good scalability in large systems because, in this case, a secret key distribution is not needed. However, the verification of the digital signature implies a large consumption of processing and bandwidth. Furthermore, if a programmable node modifies the active packet, it must sign the active packet as well, and the consumption of resources increases.

The most popular solution adopted in the sate of the art consists of the protection of the static part of the active packet (the one that is not modified by the programmable nodes) using a digital signature generated by the user. Therefore, the programmable nodes can identify the user that has generated the active packet. In addition, the dynamic part of the active packet (the one that can be modified by the programmable nodes) is protected using symmetric cryptography (HMAC). The problem of the key distribution is usually resolved by using a different key between every two adjacent programmable nodes, and between the end systems and the adjacent programmable node. This is called a hop-by-hop solution. But this solution requires the programmable nodes and end systems to know the topology, because in order to exchange the symmetric key, adjacent systems must know themselves.

We propose a solution that does not use the digital signature in order to reduce the consumption of resources. The proposal consists of protecting the active packets via symmetric cryptography (HMAC), but using a unique key shared between the programmable nodes and the end systems. The use of a unique shared key facilitates the distribution of the key without knowing the network topology. To solve the problem of the efficient distribution of the key to the end systems and the programmable nodes, we propose a scalable and efficient procedure. This procedure is based on the regeneration of the key in the trusted programmable nodes using an efficient procedure based on hash.

C. Mechanisms of Protection against Processing Illicit Active Packets

The key used to protect the active packet is generated by the authorization server and is sent to the user upon requesting a programmable service. As this key is used only to protect the active packets associated to this programmable service request, it is limited in time by the authorization parameters: service start time (SST) and the service end time (SET). So we will call this key the session key.

The process starts when a user requests a service sending the authorization parameters (C_i, SST, SET, D, S, U, and SP) to the authorization server (AS). When the AS verifies that the user has authorization, it will send him a response allocating the session key (K) and the authorization parameters. The authorization server can modify the value of some authorization parameters (for example, the value of the SET) before generating the response.

The session key is used to send active packets to the destination. The packets are protected by using an algorithm based on HMAC and K; however, this occurs before the user introduces the authorization parameters inside the active packet. Thus, these authorization parameters are sent to the programmable nodes with integrity protection. The programmable nodes verify the active packet integrity by using the same session key (K).

A malicious user would modify the authorization parameters. For example, a user would increase the SET value to send packets within a period of time higher than that which has been negotiated. To avoid a user being able to modify the authorization parameters, the session key will be generated by the authorization server using (1).

$$K = \text{hash}(Kc_i, C_i, \text{SST}, \text{SET}, D, S, U, \text{SP}) \qquad (1)$$

Key Kc_i is a secret key that is only shared among the components of the programmable network (authorization server, code servers, and programmable nodes). Therefore, the user cannot generate a session key because he does not know Kc_i. When an active packet arrives at a programmable node, this programmable node can generate the session key using the authorization parameters carried by the active packet and the value of Kc_i. Thus, if the user modifies the authorization parameters, the session key generated by the programmable nodes will be different to the one generated by the authorization server, and the integrity verification of the active packet will fail.

There is a different Kc_i for every kind of programmable service (C_i value). The Kc_i values are generated by the authorization server and sent to the code servers. Therefore, the programmable nodes can download Kc_i at the same time as they download the code from the code server. The values of Kc_i's are refreshed periodically in order to avoid the excessive use of the cryptographic material. The authorization server is responsible for initiating the refreshment process.

Now, we will describe the process step by step in order to clarify the compression, as shown in Fig. 3:

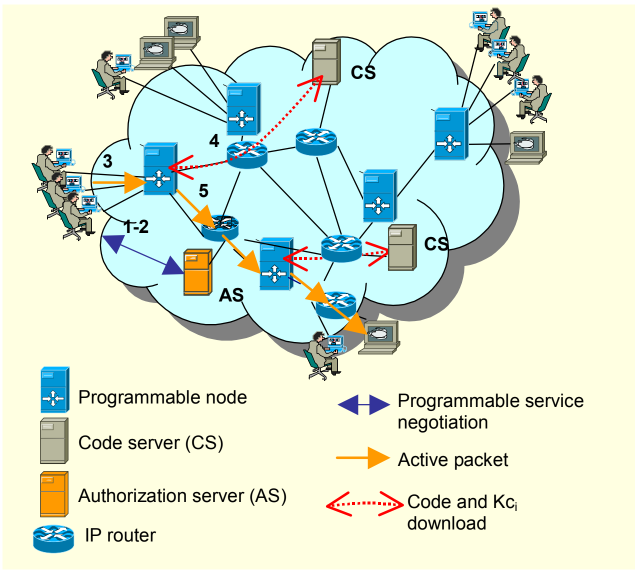

Fig. 3. Security solution.

1. The user requests authorization from the authorization server: sending the authorization parameters C_i, SST, SET, D, S, U, and SP.
2. The authorization server generates the session key and sends it to the user using (1).
3. The user generates an active packet by introducing the

authorization parameters and protecting it (using the session key and HMAC). Finally, the user sends the active packet towards the destination.

4. When a programmable node receives an active packet, if it does not have the executable code identified by C_i and/or the associated Kc_i, it will download it from the code server (usually for the first active packet of a new programmable service). Then, the programmable node generates the session key by using Kc_i and the authorization parameters that carry the active packet, and verifies the integrity and authentication of the active packet. The programmable node also verifies the authorization to process the packet by using the authorization parameters.
5. Once the active packet is processed, if it has been modified, the programmable node protects it by using the session key. Finally, the programmable node sends the active packet towards the destination.

The verification of authorization to process the packet is carried out by the programmable node and consists of the following:

1. The parameters of S and D that correspond to the IP source and IP destination respectively of the active packets are used to verify that the active packet goes between the two end systems.
2. The parameters SST and SET are used to verify that the active packet arrives at the programmable node within a valid period of time.
3. The identifier of the programmable service (C_i) is used to verify that the user does not request a different service to the one negotiated with the authorization server.
4. The parameter U would be used by the programmable node to identify the user that requests the service, for example, for charging purposes.

The programmable nodes can generate the session key, so they are authorized to generate new active packets on behalf of the user, or to modify incoming active packets.

Note that the transport of the authorization parameters to the programmable nodes is single and efficient in resource consumption. In other proposals of the state of the art, the authorization parameters are transported as a credential generated and signed by a trusted authority (for example, the authorization server). The use of a digital signature requires the active packets to transport a larger amount of information and requires too much processing to verify the authenticity of the credential.

In this explanation, we have assumed that the active packet goes from an end system called the source towards an end system called the destination. If the destination needs to send active packets towards the source, it uses the same session key as the source. In general we can say that the end systems (source or destination), from which the user has requested the programmable service, sends the session key to the other end system if necessary.

The authorization parameters S and D are respectively the IP source address and IP destination address of the active packet. To generate the same session key to both directions of the active packet (from S to D, and from D to S), we must generate the session key using the S and D ordered parameters; first, the minor value and then the major value. Therefore, we avoid the use of two session keys, one for each direction.

D. Anti-replay Protection of Active Packets

The HMAC algorithm provides protection against integrity and authentication attacks on the active packet. But an aggressor would provoke denegation of service attacks (DoS), stealing active packets and injecting the packets into the session at a later time. The programmable nodes process the injected active packet because the HMAC verification is right.

Therefore, we must define a mechanism to verify whether an active packet has already been processed by the same programmable node before processing it.

An anti-replay procedure based on IPSec is usually used in IP networks, which consists of identifying every active packet using a different sequence number. As an IP network does not guarantee the ordering of the packets, it is necessary to implement it in the receptors of the active packets' sliding windows. The sliding window will have a constant size and represents a range of sequence numbers.

In a programmable network, the programmable nodes must carry out the anti-replay verification every time a new active packet is received. The mechanism used in IPSec is implemented between two end points. The security proposals in the state of the art define a mechanism based on implementing the IPsec anti-replay procedure between every two neighboring programmable nodes [21]. The proposals that follow this approach are dependent on changes in topology. In addition, the users and programmable nodes need to know the topology of the programmable network. We will propose a variation in the solution that avoids the need to know the topology and that supports changes in topology.

We must consider that a programmable node would insert new active packets into a flow of active packets from a source to a destination, and that the active packets would pass through different programmable nodes when a change of topology is produced.

Every session of a programmable service could have different sources of active packets: the end system (end source) and the programmable nodes (intermediary sources). If we use the same sequence to enumerate all the active packets, situations of inconsistency in the numeration would be created. This situation consists of different sources of active packets of the same programmable service generating new active packets using the same sequence number.

To avoid this problem, we will use a different numeration for every source (end source and intermediates sources) that generates active packets in the same session. Thus, we will distinguish whether an active packet has already been processed by a programmable node when it has passed an active packet before with the same following parameters:

1. Session of programmable service (identified by Ci, SST, SET, D, S, U and SP).
2. Source: IP address of the programmable node or end system that has generated the active packet.
3. Sequence number.

Therefore, the active packets must carry the authorization parameters, the IP address of the source that has generated the active packet, and the sequence number generated by this source (end system or programmable node).

We must note that if the source of an active packet is a programmable node, the active packet will carry the IP address of the end system in the IP header. So appending a field to carry the IP address of the programmable node is necessary.

To implement this procedure, the programmable nodes that receive the active packets must implement one sliding window for every source of active packets corresponding to the same session.

Therefore, by using more resources to implement different sliding windows and carrying the IP address of the generator of the active packet we avoid the inconsistency of the number of sequences.

When a programmable node generating active packets reboots, it will reset the value of the sequence number, and the active packets could be rejected by the receivers because the sequence number is reused. To avoid this problem, the programmable nodes save the higher bits of the sequence number in non-volatile memory. When the programmable node reboots, the portion of the sequence number that has been saved is recovered and incremented. Thus, the sequence number will start with a higher value than the last one used before the reboot.

E. Others Security Processes

Now we will briefly describe other less critical processes involved in the security solution.

The programmable nodes download the Kc_i values and programmable service codes from the code server. This procedure requires security services for mutual authentication and confidentiality of the codes and the Kc_i. To provide both services, we propose to carry out the downloading process through a secure TLS [22] connection.

The end systems request services from the authorization server. This process requires security services of mutual authentication and confidentiality for the session key. To make this possible we can also use a TLS connection.

Finally, the refreshment process of the Kc_i's (keys associated to the programmable services) between the authorization server and the code servers requires security services for mutual authentication and confidentiality. In addition, we must take into account that a programmable network would have a considerable number of code servers (for example, 40 code servers), and this fact must not bring about scalability problems in terms of the requirement for processing by the authorization server.

The refreshment of the keys (Kc_i's) is initiated by the authorization server that sends a message, the refreshment request in Fig. 4, to all the code servers of the programmable network. The Kc_i's are valid for a predefined period of time (for example, 24 hours), so the authorization server must initiate this process every period of time equal to the duration of the key.

This message carries a random and secret seed, which is used by the authorization server and the code servers to generate all the new Kc_i's. They make a computation based on a hash function in order to be fast, even if the amount of programmable services (C_i) is high. The procedure is as follows: The authorization server and code servers use the seed to generate a master secret (MS_j) for the refreshment period j using (2). The MS_j value depends on the seed, the period of validity of the keys generated (VP), and the master secret generated in the last refreshment process (MS_{j-1}).

$$MS_j = \text{hash}\,[\text{seed}, VP, MS_{j-1}] \qquad (2)$$

Finally, they use the master secret to generate the Kc_i^j of every programmable service identified by C_i, and for the period of refreshment j, as shown in (3).

$$Kc_i^j = \text{hash}(MS_j, C_i) \qquad (3)$$

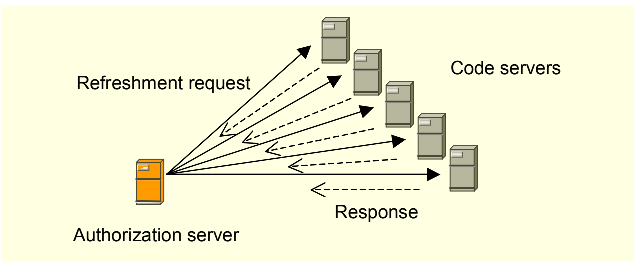

Fig. 4. Kc_i refreshment.

The refreshment request message must be confidential and authentic. To avoid the consumed time to generate the messages depending on the amount of code servers, the protection of confidentiality applied to the seed will be carried out using symmetric cryptography, instead of asymmetric cryptography. For authentication purposes, we use a digital signature.

When sending the response message, *Response* in Fig. 4, the code servers confirm that it has received the refreshment properly. This message must be protected by authentication and integrity procedures. The response message would provoke a processing overload in the authorization server if the number of response messages (code servers) is high. Thus, the authentication mechanism is implemented by using a hash function, which requires less processing than a digital signature.

2. Multidomain Security

A. Introduction

The proposal of the security architecture inside an administrative domain requires a relationship of trust between the components of the same administrative domain of a programmable network. These components are the programmable nodes, the code servers, and the authorization server. The relationship between these components makes it possible for them to share a secret. In this case, the secret is the programmable key associated to the programmable services (Kc_i's). But, it is not reasonable to extend this trust relationship out of a domain. This means that the Kc_i keys must not be shared between components of different domains. So, every administrative domain j will have its own Kc_{ij} values for every programmable service.

Then, if an active packet changes domain, the programmable nodes of the new domain j will use different Kc_{ij} values and thus a different session key (K_j). Therefore, the verification of the packet will fail.

To resolve this problem, it is necessary to establish a dynamic security association between the different domains that take part in a session. These security associations do not require the Kc_i's of every domain to be shared. However, it is possible to share a secret value that has the validity period of the session: these are the dominion session keys (K_j).

In a multidomain scenario there are a lot of domains, the active packets that belong to the same session will cross some of them. The first question that we must answer is which domains will be crossed by the active packets. The user must negotiate with the service over these domains (their authorization servers). Therefore, we give the opportunity to all the domains to decide whether the user is authorized to receive the requested programmable service.

Once it has been decided which domains will offer the programmable service, the domains will exchange a session key. This session key is used by the end system user and the programmable nodes to protect the active packets. We propose to use a unique multidomain session key (K_m) in order to simplify the security processing in the programmable node.

Now, we will explain the security solution in a multidomain programmable network. The following phases will be highlighted in the solution:

1. The process of finding out which domains take part in a session of a programmable service.
2. The negotiation process for the session with the encountered domains.
3. The protection process for the active packets.

The multidomain solution must fulfill the requirements on the topology. This means that the user and the programmable nodes do not need know the topology of the programmable network. Furthermore, the solution must support changes in the topology.

The most important requirement is that the solution must be scalable; this means that the processing carried out by the programmable nodes does not increase when the active packets cross various domains. Additionally, the amount of information related to the security, which is carried by the active packet, must be reasonable as the number of domains increases.

We must take into account that programmable networks must be on the edge of the network, so this technology will be offered usually by the ISPs that give direct service to the users. Therefore, a multidomain session will generally imply two domains, and in some extreme situations, could be up to four domains; that is, when two ISPs that give service to two intranets, the end systems are situated within the intranets that have programmable service technology, as we can see in Fig. 5.

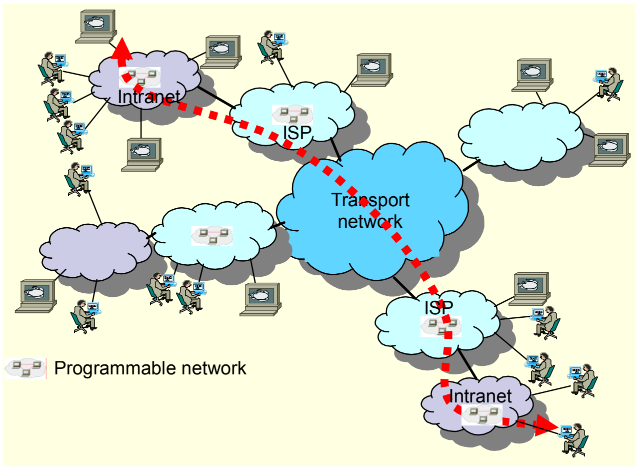

Fig. 5. Multidomain scenario.

Now, we describe the different phases of the multi-domain solution.

B. Domain Discovery

To discover the programmable domains that participate in a session, we will use the programmable network technology itself, sending an active packet called a *scout* from the source to the destination. The *scout* carries the IP address of the authorization server (AS) that belongs to the domain from which it originated. When the *scout* reaches the first programmable node of a new domain, this programmable node will inform its AS of the IP address of the previous AS. Then, a new scout is sent by this programmable node to the destination, carrying the IP address of the last AS. So, at the end of this process every authorization server will know the previous authorization server along the path of the active packets. When an AS knows its nearby AS, it will negotiate with it.

In Fig. 6, we can see the messages exchanged in the discovery and negotiation phases applied to a scenario with three domains represented by AS_1, AS_2 and AS_3.

When S requests the session from its AS, the AS verifies that the destination belongs to a foreign domain, and S will then send a *scout* active packet towards the destination that carries the IP address of the authorization server. When the *scout* packet reaches a programmable node of a new domain, the programmable node detects that the IP address of the authorization server, which carries the *scout* packet, does not correspond to its authorization server.

Then, the programmable node sends a *request of identity* message to its authorization server, indicating the IP address and the IP address of the previous authorization server. The

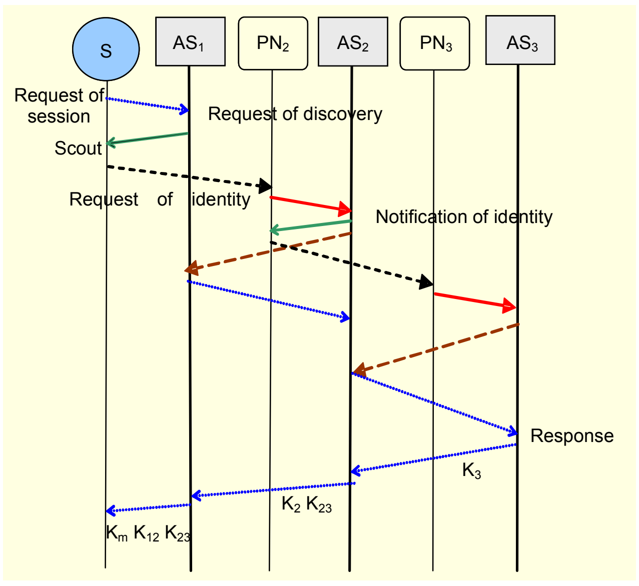

Fig. 6. Discovering and negotiation.

authorization server of the new domain sends a *notification of identity* message to the previous authorization server. Then, the first authorization server knows the IP address of the new one, and can follow the negotiation process of the session.

The new authorization server initiates a repetition, made up of the four messages, request of discovery, scout, request of identity, and notification of identity. But in this case (when it is not the first AS) the first message (request of discovery) is not sent to S; instead, it is sent to the programmable node that had sent him the *request of identity* in the previous iteration. After the repetition, every authorization server will know the IP address of the next authorization server in the path from S to D.

If the last authorization server detects that the destination belongs to its domain, it will finish the search process. If the domain of the destination does not have the programmable network technology, then the authorization server of the last programmable domain will not be able to detect that is the last programmable domain. In this case, the previous authorization server will continue the search process. Then, when the authorization server does not receive a response to the request of the discovery message, it will suppose that it is the last programmable domain (after three attempts).

C. Multidomain Session Negotiation

Once an authorization server receives a *notification of identity* message from the next server, it sends the server a *request of session message*, as shown in Fig. 6. So, this message is extended in a telescopic way up to the last authorization server. The authorization servers would change the authorization parameters in this process. For example, an authorization server would reduce the SET parameter to reduce the service time if the requested time is higher than the one supported by its domain.

Then, the last authorization server sends a *response* message to the previous one. This message carries the final authorization parameters (Ci, SST, SET, D, S, U, SP) and is sent by passing through all the authorization severs from the last to the first. Every authorization server that accepts the session inserts the following into the response message:

- Its IP address. These values are sent back to the source (S).
- Its session key (K_j for the domain j) confidentially. The session key (K_j) for the domain j is generated using (5). This value is sent on to the next authorization server, where Kc_{ij} is the assigned key in the domain j to the programmable service identified by Ci.
- Its interdomain session key (K_{ij}) between the current and previous domain. The interdomain session key (K_{ij}) generated by the authorization server of the domain j is a value generated as shown in (6), where K_j and K_i are the

session keys of domains j and i, respectively. The interdomain session keys (K_{ij}) are public values, so they do not need confidentiality. These values are sent up to the source.

$$K_j = \text{hash}(Kc_{ij}, Ci, SST, SET, D, S, U, SP), \quad (4)$$

$$K_{ij}=K_i \text{ XOR } K_j . \quad (5)$$

Therefore, when the first authorization server (that is, the nearest S) receives the response message, it can generate the multidomain session key (K_m) using the session keys of all the domains, as shown in

$$K_m = \text{hash}(K_j,K_i,...,K_1). \quad (6)$$

However, the first authorization server must know all the session keys ($K_1,\ldots,K_j$) to generate K_m. It generates the session key (K_j) of the other domains using the session key of its domain and the interdomain session keys ($K_{i,j}$), as we can see in

$$\begin{aligned} K_2 &= K_{12} \text{ XOR } K_1, \\ K_3 &= K_{23} \text{ XOR } K_2, \\ K_4 &= K_{34} \text{ XOR } K_3. \end{aligned} \quad (7)$$

Finally, the first authorization server sends a response to S that carries the following information:

- The multidomain session key (K_m)
- The definitive authorization parameters used to generate the session keys (SST, SET, D, S, U, SP)
- The interdomain session keys: $K_{ij},\ldots, K_{12}$
- The IP address of the authorization servers that are along the path of the active packets and that have accepted the session

D. Active Packet Processing

The source (S) sends active packets to the destination (D) protected as we have explained in a domain scenario, but using the multidomain session key (K_m) instead the session key of its domain (K_j).

In addition, the active packets carry a multidomain header with the following information:

- The amount of programmable domains taking part in the session
- IP address of the authorization servers
- Interdomain session keys (remember that these keys are public parameters and therefore do not require confidentiality service)

When a programmable node receives an active packet, it will generate the session key (K_d) of its domain d. Then, the programmable node uses this session key (K_d) and the interdomain keys (K_{ij}) carried in the active packet to generate the other session keys ($K_1, K_2,\ldots, K_{d-1}, K_{d+1},\ldots,K_j$). Then, the programmable node generates the multidomain session key (K_m) using the session keys of all the domains ($K_1,\ldots, K_j$).

For example, if a session has four domains, there are four session keys (K_1, K_2, K_3, and K_4) and three interdomain keys K_{12}, K_{23}, and K_{34}. When a programmable node of domain 2 receives an active packet, it generates the session key of its domain using (8).

$$K_2 = \text{hash}(Kc_{i2}, Ci, SST, SET, D, S, U, SP) \quad (8)$$

Then, using the interdomain keys, (K_{ij}) generates the session keys of the rest of the domains (K_j), as we can see in

$$\begin{aligned} K_1 &= K_2 \text{ XOR } K_{12}, \\ K_3 &= K_2 \text{ XOR } K_{23}, \\ K_4 &= K_3 \text{ XOR } K_{34}. \end{aligned} \quad (9)$$

Finally, using the session keys (K_j) of all the domains, the programmable node generates the multidomain session key (K_m) using (10).

$$K_m= \text{hash}(K_1 ,K_2, K_3 ,K_4). \quad (10)$$

We must note that a programmable node can only generate the multidomain session key if it is capable of generating a session key of one of the four domains. Therefore, only a programmable node of one of the domains 1, 2, 3, or 4 can generate the multidomain session key.

We can conclude that the procedure to process the active packets is similar to the one described in a single domain. Here, the procedure only changes to generate the multidomain session key (K_m), which is a bit more complex but does not require many more resources for the programmable nodes.

Finally, we must say that all the messages in the phase of discovering and session negotiation are protected using authentications and anti-replay mechanisms. Furthermore, when it is necessary to send the multidomain session keys (K_m) or a domain session key (K_j), we will use the confidentiality service.

VI. Tests Carried Out

We have implemented the main mechanisms of the proposed security architecture. The implementation has been carried out on the programmable networks platform called Simple Active

Router-Assistant Architecture (SARA) [23], which has been developed in the IST project, GCAP. The SARA implementation is carried out mainly in Java language. So the security implementation has been carried out in Java.

The more critical procedures are those that carry out the programmable node when an active packet is received. We have measured the cost of the different procedures carried out by the programmable node on an active packet.

We have used a single programmable service called NameTrace that copies the node IP address in the active packet. The destination forwards the active packet toward the source. When the active packet arrives at the source, it carries the IP address of all the programmable nodes in the path. Because the programmable nodes must modify the active packet, they must also protect the active packets before forwarding them.

The tests have been carried out using the SARA implementation of programmable networks with an AMD XP 2400+ CPU with 256 MB of RAM and Linux kernel 2.4.21. We have used the security solution JDK 1.4.2. to develop it.

Figure 7 shows the results of the measurements. We can see that the time needed to process an active packet is divided into verification time, programmable service processing time (TraceName in this case), and protection processing time because the active packet has been modified by the programmable node.

We have seen that for one domain, the total processing time of an active packet was 1.0523 ms. The HMAC and authentication verification process of the active packet take up 17.58%, the anti-replay verification process takes up 4.97%, and the protection of the packet takes up 9.09%. The rest of the time (68.35%) is dedicated to the programmable service.

We must note that the percentage of security processing is reduced even using a single programmable service. If we use a more complex service, this percentage will decrease.

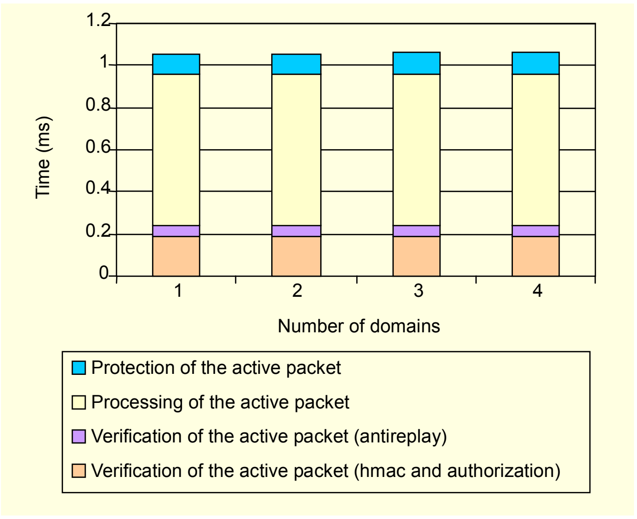

Fig. 7. Time needed to process an active packet.

If the number of domains is 1, 2, 3, or 4, the processing time is 1.052, 1.058, 1.063, and 1.064 ms, respectively. So the increase in the processing time with the number of domains is so insignificant that we can say that the system is scalable. This is because the multidomain solution only introduces changes into the session key generation procedure, which is a procedure with a very low cost.

The measurements have been carried out assuming that the code and the key Kc_i are in the programmable node. Only the first active packet of a session will usually bring about the download of the code and the Kc_i. The download time is great compared to the processing time, but its influence on the process throughout the life of the programmable service will be reduced.

We have evaluated the overload of the active packets that transport the security information: authentication code of the message (mac), authorization parameters, anti-replay information, interdomain keys, and IP address of the authorization servers. By using the implementation of SARA, we can see the overload for different packet sizes, and for a different number of domains between 1 and 4 shown in Fig. 8. So, for an active packet of 500 bytes, for instance, all the security information takes up 15.20% of the packet size for a typical scenario of two domains. In an extreme case of four domains, the overload increase up to 24.80%.

We have compared the solution proposal in this article with a solution based on asymmetric cryptography, like that proposed in ANSA. If the active packet carries a digital signature, in addition to the HMAC authenticator, the processing overload in the programmable node will increase. We have measured the cost of verifying the digital signature in the programmable node using an RSA algorithm and a key of 1024 bits. The time needed to verify the digital signature was 1.4 ms. Thus, if we use a digital signature, the delay in the security process increases from 0,333 to 1,733 ms. In addition, if we transport the digital signature in the active packet, the overload in the security information will be increased. In the case of a domain and for a packet size of 512 bytes, the security information takes up 57% of the active packet, assuming that the digital signature is 250 bytes. Therefore, we

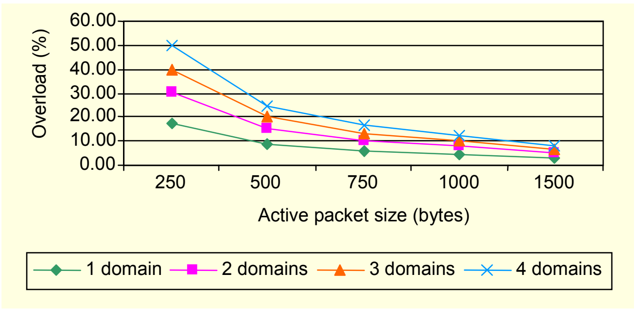

Fig. 8. Overload of the security information.

can see that the solution proposed in this article saves resources with respect to other solutions presented in the state of the art that use a digital signature.

We have estimated the time that a user must wait for a response when requesting a service. The estimations have been carried out by measuring the time needed for the cryptographic procedures and the delay in the messages in a real scenario. In the extreme case of four domains, assuming that the destination does not belong to the last domain (worst case), the estimated time is 2.6 s. This is a reasonable time because the procedure is carried out before starting the service.

Because SARA is implemented in Java, the cryptographic procedures are not optimized in processing consumption. We can obtain an improvement in the implementation using more efficient cryptographic providers [24]. A better improvement may be to create a cryptographic provider based on C language, as it is more efficient, and integrating it into SARA by means of Java native interface.

VII. Conclusion

In this article, we have presented a security proposal applied to a multidomain multiservice network based on programmable network technology. The security architecture presented follows pragmatic ideas in order to be applied in a real network. One contribution in this sense consists of a pragmatic vision of the scenario, treating the requirement of not needing to know the topology.

We have focused our effort on ensuring that the security does not use up a large amount of resources by the programmable nodes when they process the active packets. Therefore, we have proposed a security protection of the active packets based on the symmetric cryptography, as opposed to other proposals of the state of the art that use asymmetric cryptography (digital signature).

We have resolved the problem of distribution of the symmetric key (session key) to the programmable nodes in an efficient and scalable way. The solution defines a single way to distribute the session key, to check the authorization of the programmable services, and to distribute the authorization parameters to the programmable nodes.

In a multidomain scenario, the programmable nodes follow using symmetric cryptography to verify and protect the active packets. Furthermore, we have presented an original procedure to regenerate a multidomain session key in the programmable nodes. The asymmetric cryptography is used in the negotiation procedure of the programmable service between the domains to establish trust relationships between domains. Therefore, the solution is pragmatic and efficient.

We have proposed an anti-replay protection of active packets that deal with the problems derived from the requirement of not needing to know the topology. These particular problems are not dealt with in other previous works.

The tests carried out demonstrate that the security proposal presented in this article is scalable according to the number of domains, and is more efficient than the more representative ideas presented in the state of the art.

References

[1] D. Wetherall, U. Legedza, and J. Guttag, "Introducing New Internet Services: Why and How," *IEEE Network*, *Special Issue on Active and Programmable Networks*, vol. 12, no. 3, May/June 1998, pp. 12-19.

[2] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden, "A Survey of Active Network Research," *IEEE Commun. Magazine*, Jan. 1997, pp. 80-86.

[3] Jonathan T. Moore and Scott M. Nettles, *Towards Practical Programmable Packets*, Technical Report MS-CIS-00-12, Dept. of Computer and Information Science, University of Pennsylvania, May 2000.

[4] J. Biswas, A. Lazar, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang, and S. Weinstein, "The IEEE P1520 Standards Initiative for Programmable Network Interfaces," *IEEE Commun. Magazine*, *Special Issue on Programmable Networks*, Oct. 1998, pp. 72-78.

[5] Constantin M. Adam, Aurel A. Lazar, and Mahesan Nandikesan, *QoS Extensions to GSMP*, OPENSIG draft, COMET Group, Columbia University, Apr. 1997.

[6] S. Hariri et al, "Quality of Service Resource Management Using Enhanced General Switch Management Protocol," *Int'l Software Eng. (ISE) Conf.*, July 2002.

[7] IETF ForCES Working Group, draft-ietf-forces-requirements-08.txt, Jan. 2003.

[8] AN Working Group, *Architectural Framework for Active Networks*, Version 1.0, Active Networks Working Group, July 1999.

[9] Steve Berson, "A Gentle Introduction to the ABone," *OPENSIG 2000 Workshop*, Oct. 2000, pp. 11-12.

[10] IST 1999-10504-GCAP project, *Global Communication Architecture and Protocols for New QoS Services over IPv6*, http://www.laas.fr/GCAP/

[11] Spyros Denazis, Alex Galis, eds, *D14—Overview FAIN Programmable Network and Management Architecture–Draft*, Fain Project Deliverable, May 2003.

[12] Marcelo Bagnulo, Bernardo Alarcos, María Calderón, and Marifeli Sedano, "Providing Authentication & Authorization Mechanisms for Active Service Charging," *Proc. QofIS/ICQT'02*, 2002, pp. 337-346.

[13] D. Scott Alexander, W. Arbaugh, A. Keromytis, and J. Smith, "A

Secure Active Network Architecture: Realization in the SwitchWare," *IEEE Network*, *Special Issue on Active and Programmable Networks*, vol. 12, no. 3, May/June 1998, pp. 37-45.

[14] AN Security Working Group, Security Architecture for Active Nets, 2001.

[15] T. Faber, B. Braden, B. Lindell, S. Berson, and K. Bhaskar, *Active Network Security for the ABone*, Nov. 2001, *Document of the IST ARP Project*, http://www.isi.edu/active-signal/ARP/DOCUMENTS/secarch.pdf

[16] S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee, "Strong Security for Active Networks," *Proc. OPENARCH'01*, Apr. 2001, pp. 63-70.

[17] David J. Wetheral, John Guttag, and David Tennenhouse, "ANTS: A Toolkit for Building and Dynamically Deploying Networks Protocols," *Proc. OPENARCH'98*, 1998.

[18] Arso Savanovic, Dušan Gabrijelcic, Borka Jerman Blažic, and Stamatis Karnouskos, "An Active Networks Security Architecture," *Informatica Int'l J. of Computing and Informatics*, vol. 26, no. 2, 2002, pp. 211-221.

[19] Jun Li, Mark Yarvis, and Peter Reiher, "Securing Distributed Adaptation, Computer Networks," *Special Issue on Programmable Networks*, vol. 38, no. 3, Feb. 2002, pp. 347-371.

[20] Marcelo Bagnulo, Bernardo Alarcos, María Calderón, and Marifeli Sedano, "ROSA: Realistic Open Security Architecture for Active Networks," *Proc. IWAN'02*, 2002, pp. 204-215.

[21] Bob Lindell, *Active Networks Protocol Specification for Hop-By-Hop Message Authentication and Integrity*, ABone Draft: draft-nodeos-security-00.txt, Dec. 1999.

[22] T. Dierks, and C. Allen, *The TLS Protocol Specification*, version 1.0, IETF Std. RFC2246, Jan. 1999.

[23] D. Larrabeiti, M. Calderón, A. Azcorra, and M. Urueña, "A Practical Approach to Network-Based Processing," *IEEE 4th Int'l Workshop on Active Middleware Services*, 2002, pp. 3-10.

[24] B. Alarcos, E. de la Hoz, M. Sedano, and M. Calderón, "Performance Analysis of a Security Architecture for Active Networks in Java," *Proc. CSN'03*, 2003, pp. 471-476.

Bernardo Alarcos is an Associate Professor in the Automatic Department of the University of Alcalá, Spain. He received the telecommunication engineering degree in 1997 at the Technical University of Madrid (UPM), and the PhD in telecommunications in 2004 at the Alcalá University. He has published over 18 papers in the fields of broadband networks, programmable networks, ad-hoc networks, and network security. The more recent research projects are in the subject of programmable networks (CICYT TEL1999-0988-C02-02, MCYT TIC2001-1650-C02-01) and ad-hoc networks (UAH2005/082).

Marifeli Sedano is an Associate Professor in the Telematic Systems Engineering Department of the Technical University of Madrid (UPM), Spain. She received the computer science engineering degree in 1987 from the University of Deusto, Spain and the PhD degree in computer science in 1999 from the Technical University of Madrid (UPM), Spain. She has published over 16 papers in the fields of advanced communications, reliable multicast protocols, programmable networks, and network security in outstanding magazines and conferences, such as IEEE Networks Magazine, European Transactions on Telecommunications Magazine, IWAN, and IEEE-PROMS-MMNET. The more recent research projects in which she has participated are in the subjects of programmable networks and network security (CICYT TEL1999-0988-C02-02, MCYT TIC2001-1650-C02-01).

Maria Calderon is an Associate Professor in the Telematic Engineering Department of the University Carlos III of Madrid, Spain. She received the computer science engineering degree in 1991, and the PhD degree in computer science in 1996, both from the Technical University of Madrid (UPM), Spain. She has published over 20 papers in the fields of advanced communications, reliable multicast protocols, programmable networks and IPv6 mobility, in outstanding magazines and conferences, such as IEEE Networks Magazine, IWAN, and IEEE-PROMS-MMNET. Some of the recent European research projects in which she has participated are LONG (IST-1999), GCAP (IST-1999-10504), DAIDALOS (FP6-2002-IST-1-506997), COST-263 and E-NEXT (FP6 Network of Excellence on Emerging Networking Experiments and Technologies).