Computer Networks 53 (2009) 2967–2984

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Zero config residential gateway experiences for next generation smart homes

Jaime García-Reinoso [a,*], Iván Vidal [a], Francisco Valera [a], Arturo Azcorra [a,b]

[a] Universidad Carlos III de Madrid, Avda. de la Universidad 30, 28911 Leganés, Madrid, Spain
[b] IMDEA Networks, Avda. Mediterraneo 22, 28918 Leganés, Madrid, Spain

ARTICLE INFO

Article history:
Received 30 December 2008
Received in revised form 7 July 2009
Accepted 8 July 2009
Available online 20 July 2009
Responsible Editor: J. Neuman de Souza

Keywords:
Residential gateway
Configuration agents
Signaling

ABSTRACT

Home networks and home environments are developing so fast that a new generation of residential gateways is needed in order to allow emerging services and the huge amount of available bandwidth to take advantage of this evolution. New protocols, applications, devices and services are appearing day after day and in order to properly cope with them, gateways must continuously be evolved. This article presents a novel architecture developed so as to allow the automatic update and configuration of residential gateways. While data flows are treated following a conventional procedure, it is for signaling messages that the architecture proposes an application layer processing. This allows for an easier deployment of new modules capable of understanding the different signaling protocols that are needed to set up the corresponding services. These different modules (Configuration Agents) can be dynamically installed or uninstalled by Service Providers and can also interact with the rest of the layers of the architecture in order to configure the whole platform. This architecture has been validated by means of experiences within a SIP enabled environment to allow the automatic provisioning of QoS guaranteed services to next generation smart homes.

1. Introduction

Homes are becoming smarter, services are becoming more complex, devices more different and numerous and users more demanding. The key to open residential environments to this new set of requirements is located in residential gateways (RGW from now on), responsible for connecting homes with access networks and for providing a framework where services can be properly deployed and configured according to client demands.

Current RGWs (most of them are still in fact more "routers" than "gateways") have already increased their capabilities to include a lot of interfaces (Ethernet, Wireless, PLC, USB, etc.), to provide triple play, to split the reception channels to guarantee the quality of the data streams, etc. This implies a notorious advance compared with common ADSL modem-routers that were, and still are, widely deployed. However, up to now, they are not flexible enough so as to be installed in a multiservice and multiprovider environment. They are typically built as closed solutions for particular providers.

This paper presents a general architecture for configuration and control of RGWs whose main objective is to easily allow different configuration agents (CAs) to be installed in the RGW, in order to set its parameters based on the protocol primitives implemented by that certain CA. One of the most relevant things related to this proposal is that the interface offered by the control layer towards the different CAs is common. No particular technology is imposed on the implementation of these CAs as far as the interface used towards the control layer is respected. A CA may for example be a Java application, a C program, an OSGi bundle

* Corresponding author. Tel.: +34 916248747.
E-mail addresses: jgr@it.uc3m.es (J. García-Reinoso), ividal@it.uc3m.es (I. Vidal), fvalera@it.uc3m.es (F. Valera), azcorra@it.uc3m.es (A. Azcorra).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.07.007

or a web server. It will be responsible for interpreting messages of different protocols like SIP, SNMP, TR-069, RTCP, etc. (snooping protocol messages, acting as a server, etc.) and based on the retrieved information interacting with the communication control layer using the common interface. In this paper we present an exhaustive description of a SIP CA, as a possible automatic enabler for service configuration. The SIP protocol has been adopted to provide session control functionalities in several Next Generation Network architecture proposals (e.g. TISPAN NGN [1]).

The rest of the article is organized as follows. Section 2 presents the motivation of this paper describing the main problems of nowadays home gateways. Section 3 proposes the so called zero config architecture, based on configuration agents implemented at the application level that can be easily installed and can configure the gateway as required. In Section 4, the different layers of the proposed architecture are instantiated and validated by means of diverse tests performed over an RGW prototype in a real scenario. Section 5 describes different examples of configuration agents, some of them considered as signaling agents (they manage signaling protocols) and some other as application agents (capable of running real applications and turning the RGW into a service gateway). Special attention is given in this section to the SIP configuration agent, capable of automatically configuring the Quality of Service (QoS) in the RGW. Section 6 presents some related work and two examples describing how to implement some RGW standards using the architecture presented in this paper. Section 7 concludes with a brief description of the main contributions of this work.

2. Motivation

There are a lot of initiatives to standardize the RGW architecture, using different names and sometimes different expressions to define what an RGW is. This last point is very important, because different vendors, projects or standardization bodies have their own vision about the functionality of an RGW. The basic RGW is just a connection device between the home network and the access network. Several new functionalities can be added to this basic configuration like quality of service (QoS), multicast capabilities, remote and local management, UPnP functionalities, Network Address Translation (NAT), etc. It is also possible to introduce services in the RGW like domotics, media servers (for storing, transcoding, etc.), eCare, web proxy, parental control, etc. promoting the RGW into a service gateway.

It is clear that different RGW standards will define different functionalities and recommendations. Furthermore, new versions of the same standard will require new RGW boxes, which in turn will increase economic expenditures to the companies (also to the customers) and electronic trash.

A relevant fact when designing an architecture is its extensibility which can be defined as the capacity to increase the functionality without changing the main architecture. Although this definition may be too flexible because some devices can increase their functionality, for example with a firmware upgrade, inserting a new card, etc., it is widely used.

Designing an extensible architecture is a problem that has already been addressed and several RGWs devices have this functionality nowadays. For example, an RGW could be extended including new IMS capabilities upgrading its firmware which is a common upgrading mechanism. This behavior has several drawbacks:

- It is usual that just the device manufacturer has the complete RGW hardware and software description, so it is the only one able to create new firmware versions. It is an obvious problem for those customers that want fast adoptions of new functionalities.
- How this upgrade is performed? The user can download a new firmware manually, install it and restart the platform. The network provider can upload it and restart the RGW after the installation process (it may interfere with the normal operation). As the firmware is a monolithic piece of software, the upgrade process typically ends with a reboot sequence.
- Other providers do not have the opportunity to create new modules with certain functionalities for the RGW, so the customer is always forced to use products of its RGW manufacturer. Although nowadays RGWs are small and with a fixed and small set of functionalities, next generation RGWs will be small computers with high capacities, so it should be possible to add and remove modules from different companies.
- Traditional Internet is changing, and due to new laws or necessities, the classical model, where a customer has a single ISP and just a couple of service providers, will probably change [2]. In the near future a customer will choose between several transport providers as his or her connection with the digital world, and then contract services with service providers: Internet access, video on demand, IPTV, VoIP, etc. For this new scenario, a single point of access to the RGW will be no longer valid, because several entities must install and configure their own software [3].

Because of that, a traditional monolithic architecture design does not seem to be recommendable. A multi-layered approach appears to be more adequate to achieve a *dynamically extensible RGW device*. With a layered architecture it would be possible to request, load or unload a module developed to perform a given functionality. The request can be done by the customer using a web interface for example, by another module or by a service or transport provider when a non-existing functionality is necessary.

Furthermore, even with a good method to install new functionalities into the RGW, it is also important to manage all running modules in order to increase the overall performance and to prevent possible problems. With a proper design, it should be possible to share functionalities between modules and to protect restricted access to the gateway core.

Regarding the initiatives described in the previous subsection, up to the author knowledge, none of them proposes the changes explained here to create a dynamically extensible RGW architecture.

2.1. Possible solutions

It is important not to confuse an architecture design with its implementation. An architecture can define a functional block design that will be afterwards implemented (instantiated) using a given technology. This paper proposes a zero config RGW architecture aiming at providing a mechanism to automatize the configuration process. A particular implementation of this architecture will be discussed in the validation section (see Section 4). However, due to the flexibility enabled by this architecture, many other alternative approaches are also possible for the implementation. For instance, the *OSGi* [4] service platform is a Java execution environment for software components, used in a variety of applications like RGW devices. With the execution environment defined, components can be loaded and unloaded without a final reboot.

3. Proposed architecture

The architecture proposed in this article, depicted in Fig. 1, represents the configuration and control layer of the RGW. An overall architecture of the gateway is out of the scope of this paper (for an overall general architecture, please refer to [5]).

The main purpose of this architecture is to be able to update the RGW with new capabilities (or upgrade existing ones) without any intervention of the user (zero configuration). The capabilities are provided by software modules, called Configuration Agents (CA), which are expected to be fully integrated in the gateway and in fact should be able to configure the different parameters of the gateway when needed.

For example, if the RGW does not support the Session Initialization Protocol (SIP) [6] and the user wants to be subscribed to a VoIP service it may be necessary to load a SIP CA. Furthermore, it may happen that the gateway has to provide NAT traversal functionality for the SIP messages by means of an Application Level Gateway, ALG, for example. However, if the terminal is enabled with the typical STUN client (Simple Traversal of User Datagram Protocol [7]) it will allow for a transparent traversal and the ALG should be bypassed. It may also happen that the user wants the gateway to configure the different flows based on SIP information so that the quality of the VoIP call is preserved or that the RTP or RTCP flows are monitored by the gateway in order to react in case of problems.

All these things could be automatically done by the gateway with the proper CA if the provider installs it in the RGW as soon as the user subscribes the service (this installation will be detailed later on).

The architecture has been divided in two layers: the *Configuration Layer* and the *Transport Layer* (being this last one divided in the *Transport Control Sublayer* and the *Transfer Sublayer*).

Following a bottom-up definition, the Transfer Sublayer is responsible for the normal data forwarding mechanisms, also including functionalities like traffic classification, queuing, scheduling, shaping, routing/bridging, Network Address and Port Translation (NAPT), etc. This layer is configured by the Transport Control Sublayer that has direct access to different parameters like queue length, maximum number of allowed flows, etc.

And finally the Configuration Layer is where the *Configuration Agents* (CAs) are managed. This layer is responsible for their installation and un-installation procedures, for creating a framework where CAs can be executed and for relating different CAs into *Configuration Contexts* (CC in the figure) when cooperation between CAs is required.

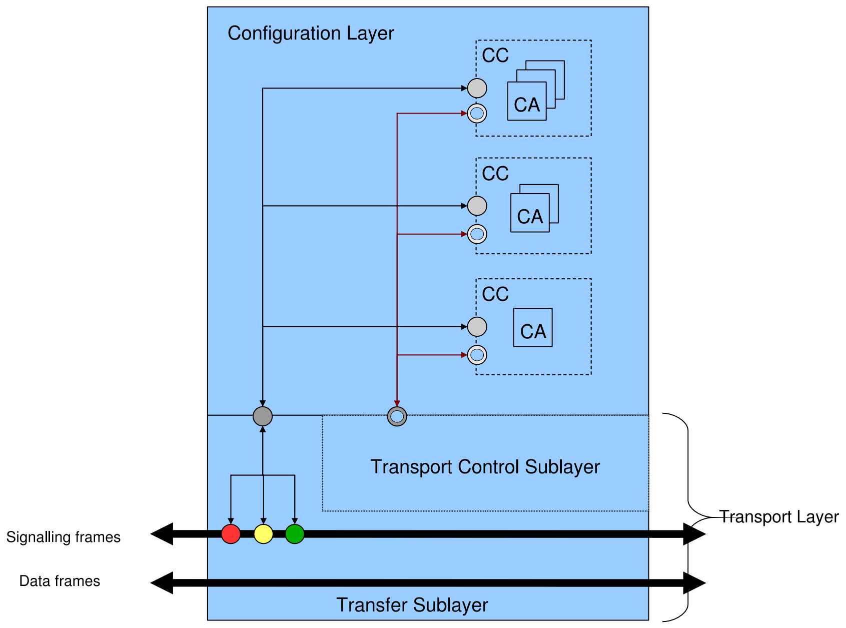

Fig. 1. Proposed architecture.

The Configuration Agents are the entities responsible for contacting the Transport Control Sublayer in order to configure the RGW (or to read the configuration). Although this is one of their main roles, the CAs are able to do many other things since they are in fact independent pieces of software installed in the gateway.

There are two types of CAs defined in this configuration architecture (although in an overall architecture this idea may be expanded to support more types):

- The *signaling Configuration Agents* are responsible for processing messages corresponding to specific protocols. This process may just imply snooping the traffic to retrieve statistics, may imply a reconfiguration of the gateway, may even imply changing the signaling frame, etc. In general there will be one CA per signaling protocol. Following with the SIP example already mentioned, a SIP CA may for example implement NAT traversal functionalities (by means of an ALG) changing the signaling messages as required, may behave as a Back to Back User Agent (B2BUA) to support legacy SIP terminals in an IMS/TISPAN scenario, may deduce (and afterwards configure) the quality of service required from the codec information exchanged in the signaling messages, etc.
- *Application Configuration Agents*. These agents are responsible for supporting applications or services. They will not process protocol frames, but will also be able to configure the RGW on behalf of the related services. An example of an application CA may be an eCare service where the gateway has an attached medical device and it is responsible for configuring the Transfer Layer in order to send the medical information towards the hospital server using a high priority flow (see [8]).

The procedure to install the different CAs is implementation dependent. The idea is that they can be installed as the result of a manual user order, an operator order, a service intervention, automatically when the gateway detects it is needed, etc. The mechanism to download the CA into the gateway and the technology used to develop the CA is left open. It can be developed in any language, be a standalone application, be a Java bundle of an OSGi platform, be an agent running into an intelligent or mobile agent platform, etc. or even several of these options at the same time. In Section 4 a particular implementation will be detailed.

The Transport Control Sublayer is offering a common interface (set of primitives) to the Configuration Layer. This way it does not matter how heterogeneous CA technology is since the configuration of the residential gateway is always the same for them all. Once again the implementation of this interface is open (it may be by means of a Java public interface, an XML definition like in the example described in Section 4.3, etc.). This interface allows the Configuration Layer to set or read a large number of parameters (available bandwidth, NAT bindings, queue sizes, etc.) and perform many actions (reboot, restore configuration, etc.).

The Transport Control Sublayer is closely related with the Transfer Sublayer because it configures the resources used by this last sublayer.

And finally there is also another relation between the Transfer Sublayer and the Configuration Layer: the CAs receive frames from the Transfer Sublayer (and may also send frames to it).

When the CA is downloaded into the gateway an initialization procedure must be performed so that the CA is started and properly registered into the system, providing information about:

- Its unique identifier and its version number.
- A procedure to receive frames from the Transfer Layer.
- A flag indicating if a frame has to be extracted or copied from the Transfer Layer so as to submit it to the proper CA.
- A set of conditions under which the frame will be forwarded up to the CA.
- Where, from the Transfer Layer data flow, will the frame be extracted (or injected). For example, before NAT rules are applied, after a VLAN tag has been added, etc. These interaction points in the data flow are called *hooks* in the architecture.

A final important proposal in this architecture is the decision of running both the Configuration Layer and the Transport Control Sublayer at the application level, only leaving the Transfer Sublayer at the Operating System (OS) kernel (or even hardware) level. Although this is more an implementation than an architectural fact, it is relevant to state it like this, because part of the functional flexibility that the architecture is offering would be lost otherwise.

A possible disadvantage of this approach is the performance. Not really related to processing time, where the difference between processing a frame at the application level or at the OS kernel level will be negligible anyway, but related to the time taken to send frames from the lower layers to the application layer and after their processing, back again to the communication layers (Transfer Sublayer). This is something that will have to be validated once the architecture is instantiated and implemented.

There are however notorious advantages following this approach:

- CAs can be automatically and quickly installed: they can be installed in the gateway or uninstalled, without user intervention (as distinct from typical firmware updates).
- CAs are easier to maintain: the provider can always upgrade the software whenever a new version is available (or even the gateway itself if an automatic update mechanism is developed).
- Software development done at the application level is easier than at the OS kernel level.
- CAs implementations are not OS kernel dependent and so can be reused through different platforms.

Although the flexibility provided by this architecture is helpful when it is integrated into a multiprovider environment (everyone is free to choose its own CA implementation), the implications of this situation from a security, management, etc., viewpoint have been studied in [3] where different solutions are proposed.

4. Architecture validation

4.1. Prototype architecture

Fig. 2 presents an instance of the general architecture introduced in Section 3. A prototype that follows this instance will be presented in the following subsections. A first version of this prototype was used as inspiration for the definition of the architecture presented here and was demonstrated in [9].

As it will be explained later, the Transfer Sublayer was implemented at the OS kernel level using the Click! modular router software [10] while the Transport Control Sublayer and the Configuration Layer were implemented at the application layer using Java. It is important to note that the architecture definition is independent from the technology chosen to implement it.

Fig. 3 shows the testbed used for the validation and the characteristics of the devices involved in the different tests.

4.2. Configuration layer validation

Following the flexible architectural philosophy described in Section 3, the chosen technology used to implement the Configuration Layer is the Java programming language, so each CA will be a Java application. The Configuration Agent Manager (CAM in Fig. 2) is the one responsible for the CA registration procedure (see Section 3) and for installing the different CAs that conform Configuration Contexts. In our implementation, the CAM is a Java application too.

However, as it has been mentioned, this flexible application level approach has the major drawback of the delay that may be introduced due to the transit of messages from the Transfer Sublayer to the Configuration Layer. It is also important to note that this delay highly depends on the final implementation. In this prototype, for example, as the Transfer Sublayer is implemented in Click!, that runs at OS kernel level, it is necessary a mechanism to extract or copy frames from Click! and to send them to a given CA at the application level. This mechanism is based on the so called hooks in Section 3.

To calculate this delay, two experiments were performed using the ping application (any traffic that traverses the RGW both in the upstream and downstream direction is valid for these tests). In the first test (Test1), the ping directly traverses the Transfer Sublayer without further considerations but in the second one (Test2) the message is extracted in the upstream direction and then reinjected down again. Fig. 4 shows the testing scenario and a detailed disaggregation of the different delay components (round-trip delay time).

Let D_1 be the delay obtained after Test1 and D_2 the delay obtained after Test2. Using the nomenclature presented in Fig. 4 it is easy to obtain that $D_1 = T_1 + T_{cl} + T_2 + T_3 + T_4$ and $D_2 = T_1 + T'_{cl} + T_{up} + T_{proc} + T_{down} + T_2 + T_3 + T_4$. Notice that T_{cl} and T'_{cl} (the times necessary to look up in the hook rules table in Test1 and Test2,

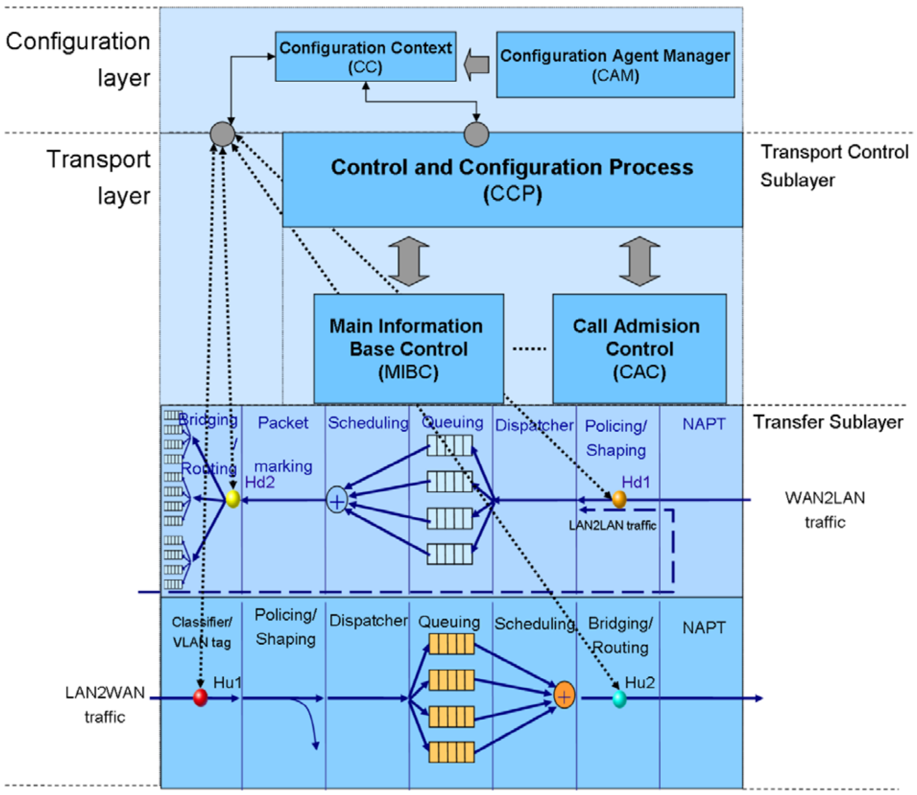

Fig. 2. Implemented architecture.

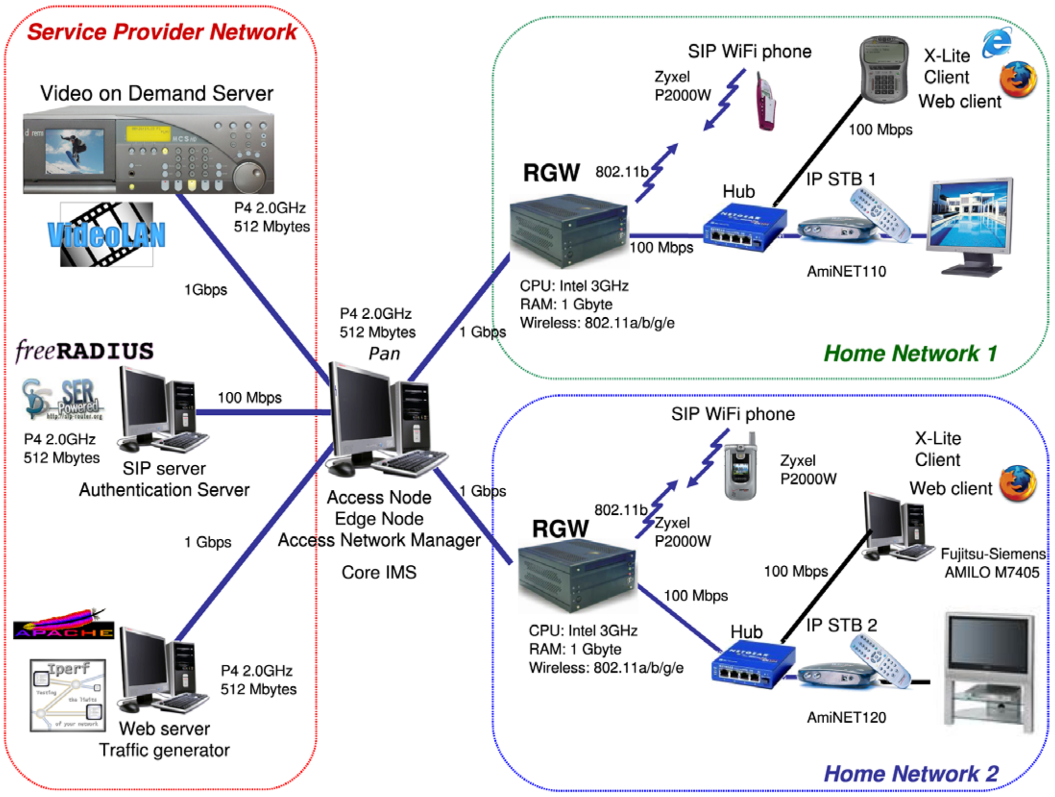

Fig. 3. Testbed used on validation trials.

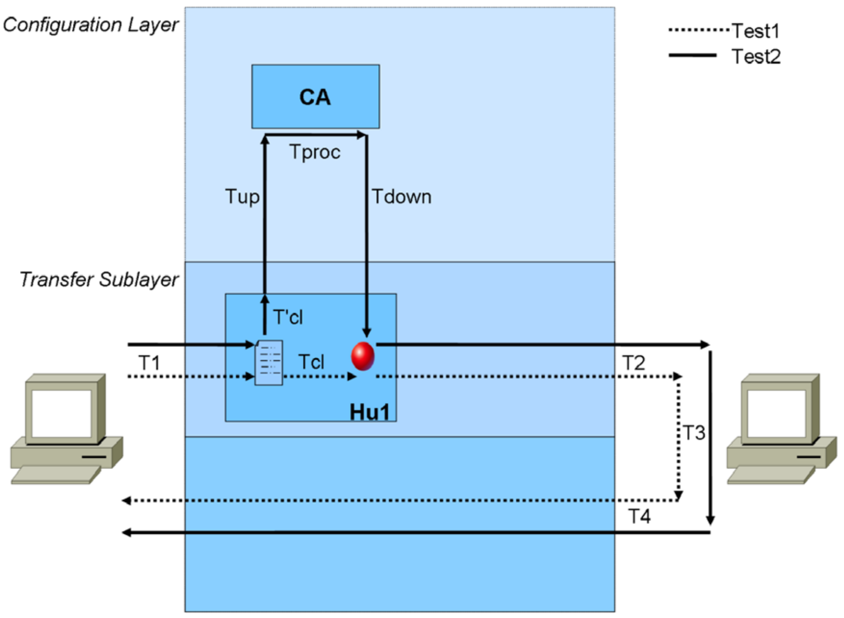

Fig. 4. Delay scenario.

respectively) are different. Therefore, if D_1 is subtracted from D_2, it is possible to obtain the whole delay introduced by this communication mechanism: $D_2 - D_1 = T'_{cl} + T_{up} + T_{proc} + T_{down} - T_{cl}$. Assuming that $T_{cl} \approx T'_{cl}$, and for these experiments T_{proc} is null (no special processing is done in the upper layers), then $D_2 - D_1 \approx T_{up} + T_{down}$ is the extra-time imposed by the proposed architecture. Fig. 5 shows the results (each point represents the

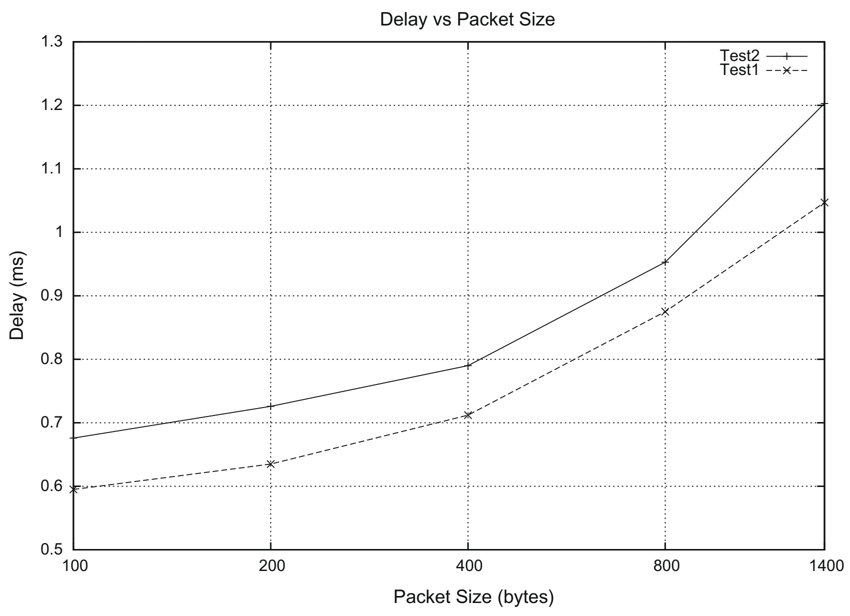

Fig. 5. Delay vs. packet size.

average of 20 different realizations), where it is possible to observe the difference between these two delays (different tests with different ping sizes are shown).

It is important to note that this extra-time is packet size independent and around 100 μs (some considerations about this value will be done after the following validation test).

Another relevant value that must be obtained in order to validate this implementation of the architecture is the maximum throughput available between the Transfer Sublayer and the Configuration Layer. To do so, the system will be stressed using the Iperf [11] application to generate UDP packets with a constant bit rate pattern. This way it will be possible to measure the maximum admissible throughput of the system (using again different packet sizes). For this trial, a CA is created and registered, so UDP traffic with destination port 5001 is extracted from an upstream hook (Hu1 in Fig. 2) and after a certain fixed waiting time (emulating the processing time) the CA reinjects the traffic down towards the same hook (this is somehow similar to Test2 described in the delay tests). To estimate the processing time in a CA, several tests were

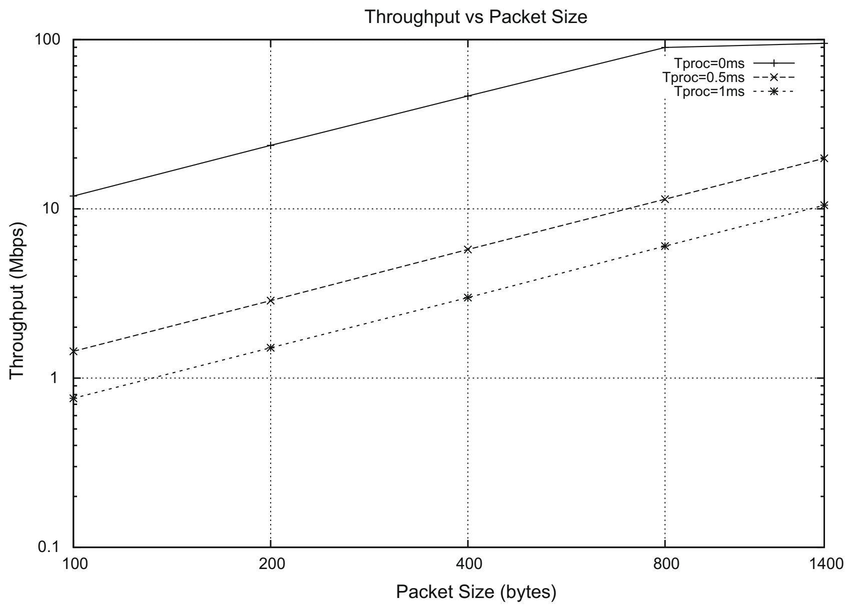

Fig. 6. Maximum throughput vs. packet size.

performed in VoIP testbed in order to obtain the time necessary to process several types of SIP messages (as SIP is a text-based protocol, processing SIP frames is a very time consuming task). These tests gave that $T_{proc}^{min} = 0.5$ ms and $T_{proc}^{max} = 1$ ms.

Fig. 6 gathers the results of these tests. As expected, packet size is a relevant parameter because for small packets the throughput is less than for the biggest ones and in fact the throughput is almost doubled as the packet doubles its size.

From the point of view of these delay and throughput tests, the prototype has been successfully validated. As it has already been mentioned, the CAs will be responsible for processing signaling frames extracted from the normal traffic flow and based on them reconfigure the RGW. It has been seen that it is possible to maintain a sustained rate of more than 2 Mbps for an average packet size (in the VoIP testbed, the measured average packet size for SIP messages is around 400 bytes) which is far than enough for signaling messages that have an aggregated throughput which is obviously small. This means that it is feasible to send frames up to the application layer, process them (for signaling messages the measured delay is not significant) and send them down to the communication layer to continue their normal transmission. Although it is also possible to use the same idea for data frames, the typical operations for them like transcodification, pattern recognition, quality detection, etc. imply a considerable increase in the processing time and even more important, the aggregated throughput may be really high (of course this all depends on the hardware of the RGW and on the particular operations).

After this performance validation, the last procedure to be defined is the CA registration mechanism, that will be mainly driven by the CAM. The implemented CA registration procedure works as follows:

- The available CAs are stored in web servers (it does not matter if it is only one server or if they are distributed in different servers as far as the CAM is aware of the different URLs of all these servers). In each one of these web servers there is an XML file that provides the information about all the available CAs (and that is in fact the file whose location must be provided to the CAM). This XML file includes for every CA, the URL of the Java bytecode, the CA identifier, the version, and a descriptor with a human readable short text explaining the CA functionality.
- The installation procedure implemented can be both manual or automatic. For the manual procedure, the end user or the provider can install a new CA. In the prototype implementation, the CAM will provide the required information to a Java servlet that performs as a web front end for the installation (the URLs of the XML description files are enough). Once the user (or the provider) selects the CAs to be installed, the servlet transmits this information to the CAM that will proceed with the rest of the installation (in the automatic installation it should be an already installed CA the one that, knowing in advance the identifier of another CA, will ask the CAM for its installation).
- After receiving the installation request, the CAM downloads the CA bytecode file starting it in the Java Virtual Machine, JVM (obtaining as the result of the execution the PID, Process Identifier, from the operating system).
- Once started, the CA asks for its own registration to the CCP (located on the Transport Layer) on a well-known port. The CA includes the following information in the request:
  - The CA identifier (CAid) and the PID (CApid). Every CA is run as a different process (different JVMs).
  - One or several rules. Each rule indicates to the RGW that the CA wants to receive all the packets passing through a particular hook in Click! in a provided port (a hook identifier is also provided as part of the rule). The rule is in fact defining the matching pattern (for example all SIP messages coming from IP address A and interface I). It also indicates if it just wants to receive a copy of the frame or to extract the frame from the information flow for further processing. Note that in general there may be many ports associated with one hook, and these ports do not have to be different for different hooks (many CAs may be listening to the same hook on different ports, one CA may be listening to several hooks on the same port or on different ports, etc.).
- The CCP maintains a table with the ports that are being used by the running CAs (the entries of that table have the format <port, CAid>). The CAs have a configurable port range [20000–21000] pre-assigned, so after receiving the registration request from the CA, the CCP verifies for every port included in the request if they have already been used in the table and in case they are free, they are automatically assigned. Triggered by the registration request, the CCP performs the following actions:
  - It returns a list with the ports that the CA will use to listen for incoming frames on the corresponding hook, using to send this message the same port used by the CA to make the registration request. Note that these ports may be different from the requested ones in case the requested ports are already in use or in case the requested ports are out of the assigned range. The returned ports will be following the same order as the previous set of rules.
  - It stores the information of the CA in its local database: CAid, bytecode name, descriptor, URL, version, CApid.
  - It adds the rules to the rules section of the MIBC (in the same order as they are received).
  - A parser then translates the XML of the MIB to a Click! language file. Click! is instructed to use this new information.

Once all these steps are completed, the CA installation procedure is finished. The RGW is automatically updated with the new functionalities provided by the CA in a straightforward way.

Finally the CA will start its normal execution, listening to incoming frames. Occasionally it may happen that a CA creates one or several threads in order to allow a better treatment of the different frames (to attend in parallel incoming/outgoing frames from different ports/hooks for example). These threads do not have to execute any particular procedure so as to be configured and registered. In case a CA wants to create different threads, during the CA registration process, it will indicate the different ports where the threads will receive the frames. The existence of threads is completely transparent for the rest of the architecture because once the port has been registered by a CA, for the rest of the architectural blocks it is indifferent whether it is implemented by one CA that receives traffic in different ports or by several threads launched by the CA and each one listening in a different port.

The process to uninstall a CA from the RGW would be as follows:

- The user, provider or another CA selects the CA to be uninstalled and sends an unregistration request to the CAM. The CAM kills the CA based on its pid and deletes the CA bytecode (".class" file).
- The CCP deletes the information about the CA (ports, hooks, etc.) and its rules from the MIBC.

In order to validate this Java implementation, it is important to test the scalability of this proposal, in terms of CPU and memory performance. To do this test, different flows were being transmitted from the client and processed at the application layer by a certain CA. Starting with just 1 flow and 1 CA, subsequent tests were done up to 100 simultaneous flows and 100 CAs that were processing them at the same time. This test schema was repeated for three different processing time T_{proc} values and for each T_{proc}, three different aggregated throughputs were used (i.e. the addition of the throughputs of the different flows were 0.5 Mbps, 1.5 Mbps or 2.5 Mbps, which was set as maximum throughput since in the previous tests it was shown that for $T_{proc} = 1$ ms the maximum throughput was around that value). The packet size was set to 400 bytes (average packet size for SIP messages). Fig. 7 shows the results of these tests representing the different CPU usage.

A very interesting result from these tests is that the CPU usage does not depend on the number of CAs but on the aggregated rates and the T_{proc} values. In addition, it was shown, although not represented in the picture, that running one JVM per CA as it has been implemented in the prototype, obviously consumes more memory than running CAs as different threads (although from the point of view of security and management the different JVMs schema offers some advantages). Every new JVM uses around 4 Mbytes while 100 CAs running as different threads require around 11 Mbytes. Each approach has its own advantages and disadvantages and in fact both of them can be implemented since current RGWs should have enough memory (the 100 CAs test can really be considered as a high upper limit, since is not realistic to suppose that an RGW will be running so many CAs).

4.3. Transport control sublayer validation

For the RGW prototype, the Transport Control Sublayer has been implemented using three functional blocks:

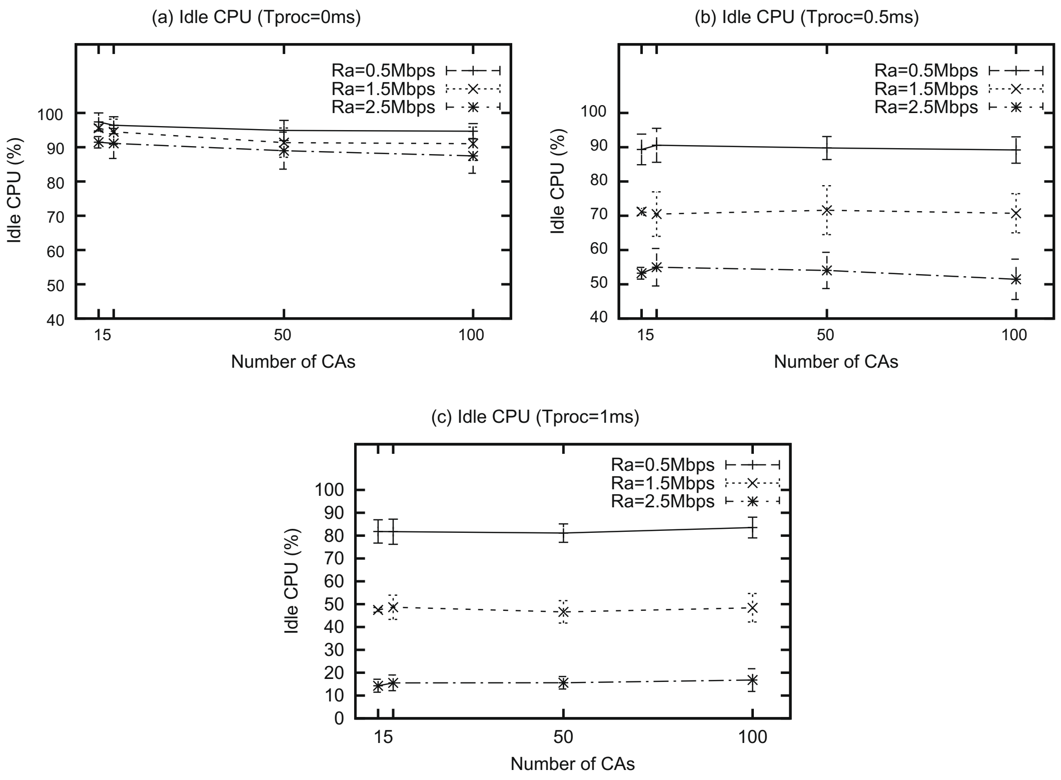

Fig. 7. Idle CPU vs. number of CAs.

- The Control and Configuration Process (CCP), apart from providing a common interface to the Configuration Layer and loading and unloading modules within this sublayer (e.g. the CAC and the MIBC) has other functionalities related with the CAC and the MIBC that will be described next.
- The Main Information Base Control (MIBC) is the only module in the whole architecture that is allowed to change the Transfer Sublayer. Other modules can read, write, modify, register or unregister Transfer Sublayer objects stored in this MIB, but the proper translation between the MIB and the Transfer Sublayer implementation is performed by the MIBC. The MIBC has to take care about the integrity of this MIB and to control multiple accesses. For the prototype implementation, the MIB is an XML document defining all the available objects at the Transfer Sublayer.
- The Call Admission Control (CAC) has a single (but not easy) functionality: to accept or reject a new flow for a given priority. In order to perform its functionality, it will consider the installed flows and when there is a request for a new flow, it will run the CAC algorithm. The output of the algorithm is binary: accepted or not accepted.

The CAC algorithm has to consider the scheduling algorithm, number of queues, queue size, maximum burst size and jitter for a given priority. With these parameters, the CAC algorithm will decide about the new flow based on the priority and the requested bandwidth.

In addition, there are some special cases where the CAC has to be deactivated. This is not a normal behavior and it has to be considered as a critical situation. For example, imagine an emergency call where there are not available resources int the Transfer Layer. In that case, the CAC will reject the call and the user will be forced to close other connections before trying again (the TV, for example). For an emergency call, this is not acceptable at all and other mechanisms have to be enabled.

For this implementation, the *unavoidable* flag for a flow rule is defined. The CAC module accepts all unavoidable flow rules despite of their priority or bandwidth requirements. It is known that this kind of flow rules can make the system unstable, as the QoS is not guaranteed anymore while an unavoidable rule exists (the system will not necessarily be unstable after the insertion of an unavoidable rule, but it may be). In other words, if a new flow rule insertion is requested while there are unavoidable rules installed, the CAC will normally process it and will accept it or not depending on the available resources at that moment. The system will only be unstable when a flow rule should not be accepted because there are not enough resources for it, but it has to be accepted because the unavoidable flag is set.

It is then important to state who is allowed to set this flag. As commented before, this must be an exceptional case and only important events can originate this kind of flows. In the example of the implemented SIP CA (see Section 5.2), it can recognize emergency calls (112 is the European emergency call number) so it will automatically set the unavoidable flag for those flows. This functionality was tested and validated in several trials [8], where an RGW with all its resources compromised by video on demand applications received a SIP emergency call. In that case, the call was successfully established.

Sometimes it is not enough to accept unavoidable rules, because it may happen that there are not enough resources for all the flows in the same priority as the unavoidable flow. In this case, it is necessary to stop existing flows in order to leave more resources for the unavoidable flow. When a CA wants to insert an alarm flow, like the emergency call, with a relatively high bandwidth requirement, it will set the unavoidable flag an also the *freeze* one. The algorithm is as follows:

- The CCP receives a flow insertion request with the unavoidable and freeze flags set.
- The CCP sends the flow to the CAC (the freeze tag is not set).
- The CAC accepts the flow.
- The CCP activates the freeze mode and adds the flow to the freeze array.
- The CCP inserts the new flow and stops all the avoidable flows (flows are not removed).
- The CAC does not consider the freeze tag (it is just handled by the CCP).

Pseudocode for requests at the CCP is as follows:

```
IF Flow has to be added THEN
  IF Flow does not have unavoidable
    flag set THEN
    IF FreezeMode is set THEN
      RETURN
    ENDIF
  ELSE
    IF Flow has freeze flag set THEN
      SET avoidable flag in Flow
      IF CAC(Flow) returns accepted THEN
          CALL MIBC.write(Flow)
          RETURN
        ELSE
          SET unavoidable flag in Flow
          SET FreezeMode
          ADD Flow to freeze array
          DEACTIVATE all avoidable Flows
        ENDIF
    ENDIF
  ENDIF
  IF CAC(Flow) returns accepted THEN
    CALL MIBC.write(Flow)
  ENDIF
ELSE IF Flow has to be removed THEN
  IF (Flow has freeze flag set) AND
    (FreezeMode is set) THEN
    REMOVE Flow from freeze array
  IF freeze array is empty
      UNSET FreezeMode
      SET AvoidableFlows
    ENDIF
  ENDIF
  CAC(Flow)
  CALL MIBC.remove(Flow)
ENDIF
```

Another important contribution to the CAC module is the *provisional promotion* behavior. After a flow insertion is accepted, a CA can request a provisional promotion to the CC. The CAC will promote the flow to the next higher priority if there are enough resources, annotating both its new priority and its original accepted priority. There is not a maximum number of promotions. Afterwards, when the CAC receives another flow insertion request, if there are not enough available resources, it will try again to decrease the priority of a previously promoted flow. If there are still not enough resources, the CAC will do it again until there are some free resources for the new flow. If it is not possible to accept the new flow at all, it will be rejected and the previous state will be restored (actually, flows are not changed until the CAC algorithm converges to a stable state).

When the CAC needs to decrease the priority of a promoted flow, it has to select a candidate flow. There are several ways to perform this functionality, but for this prototype implementation, the most promoted flows (the flows with the highest difference between the current priority and its accepted initial one) are the first options. When there is more than one candidate, the chosen one will be the oldest flow (according to its promotion timestamp).

4.4. Transfer sublayer validation

The Transfer Sublayer provides the actual data transmission functionalities within the RGW. These functionalities include but are not limited to QoS control, bridge/routing and network address and port translation (NAPT). As it can be seen from Fig. 2, incoming data flows arriving to the RGW follow separate paths in the upstream and downstream traffic directions. This way, an independent treatment of traffic in each direction is supported, allowing to provide a flexible solution in the design of the Transfer Sublayer. With this approach, new value-added functionalities can be aggregated to the data traffic processing in one specific direction without affecting the treatment of the traffic flowing in the opposite direction.

On the other hand, at this level different types of blocks are defined with specific functionalities. This block-oriented architecture improves the flexibility and scalability of the architectural design, enabling the addition of new functionalities by means of the development of new blocks that can be installed at this level either in the upstream or downstream traffic direction.

These blocks are shown in Fig. 2 and are detailed below for the upstream traffic direction:

- **Classifier**: in this block, the administrator can define specific classification rules to allow or deny certain flows. In addition, if one particular flow is accepted, the Classifier block assigns some meta-information to the flow, specifying how the frames belonging to the flow have to be treated inside the RGW. The time taken by the RGW to process the different classification rules is the so called T_{cl} in Section 4.2.
- **Packet marking**: depending on the meta-information that was previously assigned by the Classifier block, this block configures a specific packet marking for every flow frame. The packet marking function is necessary when QoS is guaranteed in the access network by means of traffic class differentiation mechanisms, e.g. VLAN tagging or DiffServ.
- **Dispatcher**: using the meta-information attached to the frame by the Classifier, this block pushes the frame towards the proper queue.
- **Queuing**: this is a configurable block that allows an administrator to utilize a certain number of queues, fixing the size of each queue in terms of stored packets. Each queue represents a priority level that will be taken into account by a scheduling algorithm so as to prioritize the treatment of the different flows that traverse the RGW in the upstream direction. In addition, the fixed size configured for each queue allows to limit the effects of the jitter introduced by the RGW in the processing of the data flows that arrive from the end user network.
- **Scheduling**: this block implements the algorithm used to extract the packets from the queues. Different algorithms that have been proposed in the literature may be used at this processing point (e.g. round robin, fair queuing, etc.).
- **Bridging/Routing**: this block may implement bridging or IP routing functionalities, depending on the configuration required by the operator of the access network.
- **NAPT**: provides network address and port translation functionalities, mapping the internal IP address and port that identifies the source of each IP packet to a public IP address and port. This public tuple of IP address and port can be globally addressed out of the scope of the end user network.
- **Policing/Shaping**: this block allows to control the rate of data traffic that is sent on the upstream direction (this rate can be configured by the administrator for each data flow), dropping non conformant traffic in case of using a policing mechanism, or buffering it in case of using a shaping mechanism.

A mechanism to extract frames from this sublayer is provided by means of the already mentioned hooks. Next sections describe use cases for this mechanism.

5. Configuration agents use cases

5.1. CA types

In order to test the architecture flexibility two different types of CAs were defined: signaling CAs and application CAs. The signaling CAs are responsible for processing different protocols and for configuring the RGW based on protocol messages. The application CAs are designed to provide support to different applications running in the RGW. Some signaling CAs that were designed and implemented are for example a **NAT-STUN** CA to integrate a STUN server [7] in the RGW itself, to overpass the problems with NAT boxes; a **Web configuration servlet** to remotely configure the most important RGW parameters; a **TR-069** CA to allow service providers to easily configure the RGW and also a **UPnP** CA to allow automatic configuration for in-home devices. However, the two most relevant signaling CAs to allow an automatic configuration of the RGW are the SIP CA and the ALG CA. The rest of this section will detail them both. In addition, some application CAs were also implemented to provide **eCare** and **VoD** services (see [12] for more information of VoD services on an RGW as a CA).

5.2. SIP

The SIP CA processes all the SIP signaling messages exchanged between the end user terminals and the SIP servers that are accessed from the residential environment. The Transfer Sublayer in the RGW will be configured to redirect all the SIP signaling messages to the SIP CA.

During the SIP session establishment procedure, after receiving every SIP message containing a Session Description Protocol (SDP) [13] answer the SIP CA derives certain service information from the SDP answer and its corresponding SDP offer (this offer, according to the Offer/Answer model of SDP [14] must have been received in a previous SIP message). The service information describes the different media streams (e.g. an audio or video media stream) that will be exchanged during the multimedia session. In concrete, for each media stream, the service information contains the parameters defining the flows associated with the stream in the upstream and downstream traffic directions. These parameters include, for each flow, the IP destination address and port, the bandwidth requirements (optional), the media type (e.g audio or video) and the accepted formats for the flow (e.g. accepted codecs in case of an RTP media stream).

With this information, the SIP CA will generate the set of flow rules that must be installed in the RGW so as to support the QoS requirements associated with the multimedia session. These flow rules will be provided to the CCP in the Transport Control Sublayer, that in turn will perform the admission control functionalities that are necessary to verify if the QoS demands for the multimedia session can be provided with the available resources in the residential environment. If so, the CCP will install the flow rules in the RGW, providing a proper configuration of the Transfer Sublayer that guarantees an adequate resource reservation for the multimedia session.

If a new SDP offer and answer exchange takes place, the SIP CA will again derive the service information corresponding to the new SDP offer and answer. If the new service information indicates any change in the flow rules associated with the session, the SIP CA will contact the CCP in order to report the changes that must be performed on the previously installed flow rules. If it is necessary, the CCP will go through the admission control procedures and will apply the modifications provided by the SIP CA to the configuration of the Transfer Sublayer.

Fig. 8 shows an example where a multimedia session is established and the Transfer Sublayer is automatically configured so as to provide the required QoS for the session.

On the other hand, if the admission control procedures state that there are not enough resources in the residential environment to satisfy the QoS demands of the multimedia session, then the session should be terminated. The procedures to terminate the multimedia session will be different depending on whether the session has been established or not. Fig. 9 shows an example of both use cases.

In the first use case (labelled as "case 1" in Fig. 9), the destination terminal accepts the establishment of the multimedia session by means of a SIP OK response for the INVITE request. In this response, the terminal includes an SDP answer for the SDP offer that was previously contained in the INVITE request. In this case, the CCP indicates to the SIP CA that the admission control procedures cannot accept the QoS demands required for the session. Nevertheless, at this point the session has already been established in the destination terminal. Therefore, to terminate the multimedia session, the SIP CA sends to this terminal a BYE request and answers the INVITE request back with a 500 (Server Internal Error) response, which is sent to the initiating terminal.

In the second use case (labelled as "case 2" in Fig. 9), the destination terminal answers back the INVITE request with a provisional response (Session in Progress in the example) that contains the SDP answer. Again, the admission control procedure states that the available resources are not enough to cover the QoS demands for the session. However, in this use case, the session has not been established yet, so the SIP CA sends a SIP CANCEL request towards the des-

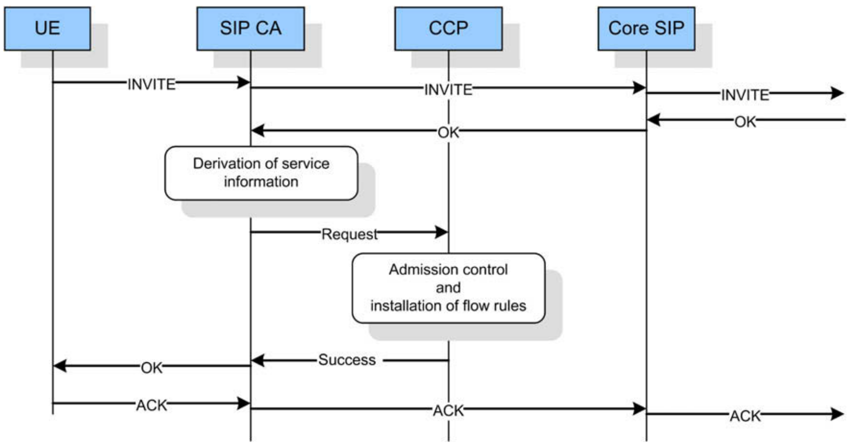
UE
SIP CA
CCP
Core SIP
INVITE
INVITE
INVITE
OK
OK
Derivation of service information
Request
Admission control and installation of flow rules
Success
OK
ACK
ACK
ACK

Fig. 8. Session establishment.

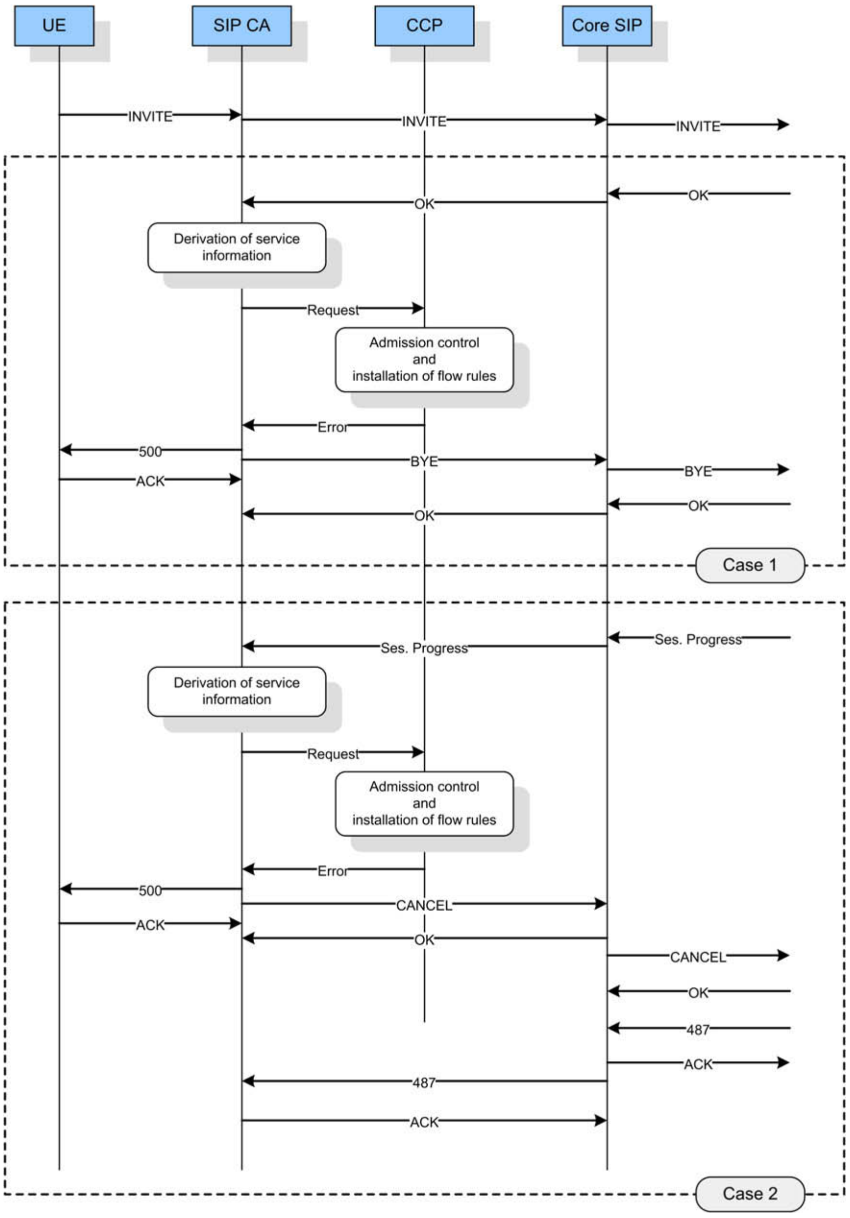
UE
SIP CA
CCP
Core SIP
INVITE
INVITE
INVITE
OK
OK
Derivation of service information
Request
Admission control and installation of flow rules
Error
500
BYE
BYE
ACK
OK
OK
Case 1
Ses. Progress
Ses. Progress
Derivation of service information
Request
Admission control and installation of flow rules
Error
500
CANCEL
ACK
OK
CANCEL
OK
487
ACK
487
ACK
Case 2

Fig. 9. Session cancellation.

tination terminal. In addition, it answers the INVITE request back with a 500 (Server Internal Error) response. When the CANCEL request is received in the destination terminal, as the cancelled request is the INVITE, the destination terminal answers the INVITE request back with a 487 (Request Terminated) response.

The SIP CA can be easily extended to support more functionalities. For example:

- **IMS/TISPAN.** In IMS and TISPAN the session control functionalities are based on SIP, SDP and the Offer/Answer model of SDP. This way, the SIP support introduced by the SIP CA in the RGW, is enough to integrate the RGW in the residential environment of an IMS based next generation network, such as the one that is currently being developed by the ETSI TISPAN group. Nevertheless, in the IMS/TISPAN context, depending on the operator policy, it is possible that early traffic (i.e. data traffic exchanged prior to the completion of the session establishment) is forbidden. In this case, a reserve and commit resource management schema [15] has to be used in the RGW. This schema has been implemented in the following way:
  – When the SIP SP provides the CCP with the flow rules that must be installed in the RGW, it includes with each rule an indication stating that the Classifier block should filter out the frames belonging to the flow.

– When the SIP CA receives a SIP OK response for an INVITE request, it contacts the CCP to commit the resource reservation for the multimedia session that has been established. To do that, the SIP CA provides the CCP with the flow rules that must be allowed in the Classifier block, so gate is opened in the Transfer Sublayer for all the flows of the session.

This way, the SIP SP can be started in a reserve-commit resource management scheme, so as to cover the IMS/TISPAN scenarios where early traffic is prevented from being exchanged on the access network. If early traffic is allowed, the single-stage resource management scheme explained so far for the SIP CA can be used.

- **B2BUA.** In case the RGW is integrated in the residential environment of an IMS based next generation network, it is possible that legacy terminals (non-IMS) are used within the end user network. In this case, the SIP CA should behave as a SIP Back to Back User Agent (B2BUA), adopting the role of a User Agent Server (as it is specified in [6]) from the point of view of the legacy terminal, and the role of a SIP User Agent Client (UAC) with the exceptions and additional capabilities of SIP and SDP described in [16], from the point of view of the core IMS. This way, the SIP CA would be in charge of performing the IMS session control functionalities on behalf of the legacy terminal, as well as contacting the CCP to install the flow rules in the RGW that are necessary to reserve QoS resources in the residential environment for the multimedia sessions being established.
- **Emergency calls.** In case that a SIP emergency call is initiated from the end user network, QoS demands for the call should be granted even if there are not enough resources available in the residential environment. Therefore, when the SIP CA processes the establishment of an emergency call, the service information associated with the call is derived as it has been explained, but the flow rules generated from this information will be marked with an unavoidable flag and possibly a freeze tag, meaning that these flow rules must be installed in the RGW overriding the admission control functionalities.

5.3. Application level gateway

This CA provides a SIP-specific solution to solve the problems related with SIP and NAT traversal. The Application Level Gateway (ALG) CA receives a SIP message and, after examination, requests from the CCP the NAT bindings that are necessary to substitute the internal IP addresses and ports, that are contained in the message, by the public values that will be used after traversing the NAT block in the Transfer Sublayer. This way, internal IP addresses and ports within the SIP messages and the SDP payloads are changed by the bindings that will be assigned by the NAPT block, guaranteeing that the initiator and destination terminals get the information to address data traffic.

The ALG CA will be installed by the CAM in the same Configuration Context as the CAs that will contact it requesting for support. Therefore, if a NAT block is installed in the Transfer Sublayer and SIP is used to provide automatic QoS configuration in the residential environment, an instance of the ALG CA will be installed by the CAM in the same context as the SIP CA.

6. Implementing other RGW standards

This section shows how to implement other RGW standards using our proposed architecture. First of all, some other proposals are reviewed in order to present the related work. Sections 6.2 and 6.3 present a detailed description in order to implement two of the most important standardized RGW architectures using our proposal.

6.1. Related work

Nowadays the scientific and industry communities have a special interest in smart homes and specially in the RGW, as the main device in that environment. There are a lot of initiatives created to standardize the RGW model architecture, focusing on different points but with the same goal. Some important standardization bodies or projects trying to describe the RGW architecture are:

TISPAN. The ETSI TISPAN workgroup is currently standardizing a Next Generation Network using the IMS [17] as the core of the control network and, in turn, SIP as the signaling protocol. In the second release, TISPAN introduces the home network environment in their architecture, considering Customer Network Devices (CND) and also the Customer Network Gateway (CNG) [18] with the same objectives as an RGW. Although the main goal is to interconnect IMS based CND, the TISPAN CGD architecture defines blocks to allow non IMS terminals (both SIP-based and legacy ones). The TISPAN CNG architecture will be explained later in Section 6.2.

HGI. "The Home Gateway Initiative is an open forum launched by Telcos in December 2004 with the aim to release specifications of the home gateway." The HGI released a document [5] proposing a very complete and exhaustive RGW architecture. Security, QoS, management, hotspot access and the support for the IP Multimedia Subsystem (IMS) [17] are some of the functionalities considered in the architecture.

Regarding the RGW management, the HGI proposes a Management Abstraction Layer to allow a protocol independent configuration scheme. For example, it could be possible to manage the RGW using the DSL Forum TR-069 specification and another local management protocol at the same time. The HGI home gateway architecture is presented in Section 6.3.

DSL forum. In [19], the DSL forum gathers a huge list of RGW parameters that extends the TR-069 [20] data model to include the specific properties of an RGW (or Internet Gateway Device, using DSL Forum terminology). Although this is a complete object list, the DSL Forum does not specify an RGW architecture.

UPnP. The Universal Plug and Play (UPnP) Forum is a big entity with more than 800 members formed to create standards to allow seamless connections between devices and to simplify network implementation in residential and corporate environments. UPnP defines several steps for addressing, discovery, description, control and eventing between devices to allow a complete and full connection between them. In the home environment, the RGW is an important point for UPnP and [21] describes a data model for this kind of devices, similar to the one proposed by the DSL Forum in [19].

Research projects. The European Union has financed several research projects like MUSE [22], ASTRALS [23] and MEDIANET [24] where RGWs have a special research interest. In the MUSE European project, for example, a complete task force was created to study and propose an RGW architecture. At the end of the project, some of the proposed ideas, use cases and functionalities were shared with the HGI to join forces and achieve a final RGW standard architecture.

6.2. Implementing the CNG architecture from TISPAN

Fig. 10 shows the architecture defined by TISPAN for the Customer Network Gateway (CNG). This architecture can easily be mapped to the flexible RGW architecture that has been defined in this paper. In this respect, the figure represents, by using different tones, the distribution of the functional entities included in the TISPAN CNG into the architectural layers defined in this paper. This distribution is indicated below:

Transfer sublayer. The following functional entities can be located at this level:

- CNG-NFF (*CNG – NAPT and Firewall Function*): provides gate control functionality between the end user network and the NGN.
- CNG-IPTVF (*CNG – IPTV Function*): provides forwarding of incoming multicast packets to those interfaces where subscribed members are connected, and translating of link layer multicast to unicast, in cases when multicast is not supported in the client network.

Transport control sublayer. This sublayer contains the following functional entities:

- CNG-ACF (CNG – *Admission Control Function*): this functional entity exchanges QoS related messages with the CNG – SIP Proxy B2BUA function, being in charge of verifying if there are available resources for each communication, and performing the correspondent resource reservation through the CNG-PCF.

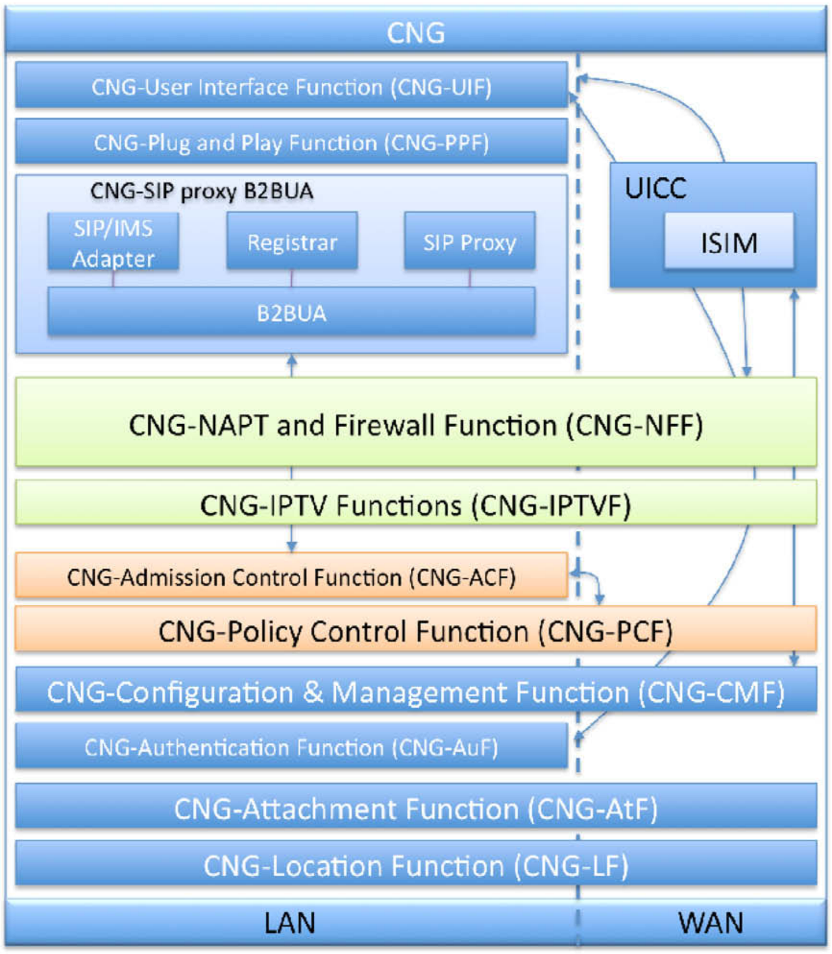
CNG
CNG-User Interface Function (CNG-UIF)
CNG-Plug and Play Function (CNG-PPF)
CNG-SIP proxy B2BUA
SIP/IMS Adapter
Registrar
SIP Proxy
B2BUA
UICC
ISIM
CNG-NAPT and Firewall Function (CNG-NFF)
CNG-IPTV Functions (CNG-IPTVF)
CNG-Admission Control Function (CNG-ACF)
CNG-Policy Control Function (CNG-PCF)
CNG-Configuration & Management Function (CNG-CMF)
CNG-Authentication Function (CNG-AuF)
CNG-Attachment Function (CNG-AtF)
CNG-Location Function (CNG-LF)
LAN
WAN

Fig. 10. CNG architecture (TISPAN).

- CNG-PCF (*CNG – Policy Control Function*): this entity may include a database, containing QoS related parameters that can be configured for applications and terminals in the end user network.

Configuration layer. Finally, this layer includes the remaining functional entities defined by TISPAN for the CNG architecture. These entities can be implemented at this layer by means of Configuration Agents (CAs):

- CNG-AuF (*CNG – Authentication Function*): handles the authentication of client devices that are connected to end user network.
- CNG-CMF (*CNG – Configuration and Management Function*): supports the configuration and firmware upgrade of the CNG. In addition, the CNG-CMF allows the transmission of configuration information to the end user devices.
- CNG-AtF (*CNG – Attachment Function*): enables the configuration of network addresses for the CNG and the devices connected to the client network.
- CNG-LF (*CNG – Location Function*). Enables internal applications to provide location information.
- CNG-PPF (*CNG – Plug and Play Function*): this entity provides functionalities related with retrieving information and allowing the control of client devices, and with supporting a certain degree of communication between client devices (it may also support this communication between the NGN and the client devices).
- CNG-UIF (*CNG – User Interface Function*): allows the user to manage in the CNG the transport layer parameters.
- ISIM module: this is an application related with IMS access that stores IMS-specific subscriber data, such as the private user identity, the public user identity and the necessary elements to support AKA authentication.
- *CNG – SIP Proxy B2BUA Function*: implements a SIP registrar, an outbound SIP proxy and may implement a non-IMS SIP to IMS SIP adaptation module.

6.3. Implementing the HG architecture from HGI

The Home Gateway architecture [25] proposed by the Home Gateway Initiative forum can be easily mapped to our architecture. Fig. 11 shows the HG architecture where boxes are filled with different tones in order to follow the mapping to our architecture previously presented in Fig. 1. Following our two layers scheme, the HG architecture can be implemented as follows:

Transfer Sublayer The following HG blocks can be implemented in this sublayer: PHY, L2-ETH, L3-ETH, IP Switching, ETH Switching and Firewalling (see Fig. 2).

Transport Control Sublayer QoS handling CAC, Access Auth. and the HG-MIM HG blocks can be implemented in this sublayer. These blocks can be configured through our CCP.

Configuration Layer The remaining HG blocks (Management, Control and Data interoperation, all Enablers and i-ATA, PSTN int. and USB Host int. drivers) can be implemented as Configuration Agents. It is important to notice that our proposal fits very well in the HGI proposal, because Data and Control interoperation CAs have to register rules in the Transfer Sublayer in order to accomplish their tasks.

7. Conclusions

In this paper, a generic architecture for a Residential Gateway covering the configuration and the transport layer has been proposed. This architecture allows the automatic update and configuration of the RGW based on flexible Configuration Agents. This capability is supported by the hybrid nature of the architecture, that imposes the execution of CAs at the application level while keeping the communication layer at the OS kernel or hardware level. It is one of the most important architectural decisions because it would be certainly easy to implement and run CAs at the OS kernel level, probably increasing the execution

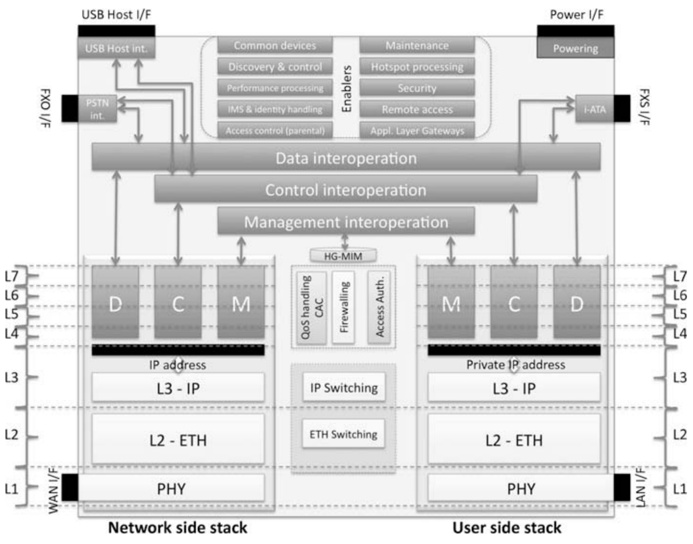

Fig. 11. HG architecture (HGI).

efficiency without having to send the frames up to the application level. However this approach would lose the main advantages of the architecture: flexibility, reusability, upgrading facilities, extensibility, etc. As a general conclusion, it can be stated that if flexibility in the configuration is pursued in the RGW, the Configuration Layer and Transport Control Sublayer should be implemented at the application level while the Transfer Sublayer functionality should be performed at the OS kernel or hardware level.

This generic architecture has been instantiated into an RGW prototype allowing its validation by means of the different tests that have been performed. It has been shown that implementing Configuration Agents at the application level does not impose a severe impact on the bandwidth or on the delay since all the traffic that is going to be processed by the CAs is signaling traffic. In addition, a complete mechanism that supports the automatic and manual installation of CAs at the configuration level has been shown. Regarding the Transport Control Sublayer, a Call Admission Control mechanism has been implemented including several original value added enhancements to enable the coexistence of the QoS mechanism with the preemptive requirements of alarms or emergency calls. For the Transfer Sublayer, a block-oriented architecture has been described allowing to implement QoS control functionalities based on traffic class differentiation and/or policing/shaping mechanisms.

Finally, several use cases have been included in order to show the possibilities offered by the proposed architecture. Special emphasis has been set on the automatic configuration based on the SIP protocol. It has been detailed how the RGW can configure the different services without any user intervention by means of this SIP interception performed by the SIP CA, which is capable of inferring the quality required by the flows following the session negotiation and of reserving and releasing resources accommodating the different session demands.

Regarding the different research issues that still remain open in this architecture, probably one of the most interesting ones is the possibility of extending the application layer processing to data flows. In general it is on the signaling messages (this is particularly true for SIP) where the information about the requirements for the connection resides. However, by data packet flow inspection (all the packets or certain samples) it may be possible to obtain information about the quality received by the platform and finally delivered to the user, or to estimate traffic patterns that can be useful to make occupation predictions, etc.

It should also be interesting to test other solutions to implement this architecture in order to go further in the validation. The authors consider that the OSGi platform is a technological alternative that could naturally fit into this architecture, since OSGi bundles can play the role of CAs and the OSGi platform is already providing some mechanisms for bundle management or inter-bundle communications. Finally, there are different concerns like the performance (the purpose of the implementation was just to validate the feasibility of the proposed architecture and not to evaluate its performance) or the security (which is out of the scope of this paper) that should also be considered and studied.

Acknowledgement

This article has been partially granted by the Spanish MEC through the CONPARTE project (TEC2007-67966-C03-03/TCM) and by the Madrid Community through the BIOGRIDNET project (S-0505/TIC-0101). The authors would like to thank the whole "MUSE implementation team" for their efforts along these last four years and particularly David Díez for his support in this article.

References

[1] ETSI-TISPAN, <http://www.etsi.org/tispan/TISPAN> (Telecoms & Internet converged Services & Protocols for Advanced Network). URL <http://www.etsi.org/tispan/>.
[2] T. Monath, Business role models for bb access, in: BB Europe, 2004, Brugge, Belgium.
[3] S. Royon, Y. Frenot, Multiservice home gateways: business model execution environment management infrastructure communications magazine, IEEE 45 (10) (2007) 122–128.
[4] OSGI, OSGi Service Platform Release 4, <http://www2.osgi.org/Release4/Download> (October 2007).
[5] H.G.I. HGI, Home gateway requirements: Residential profile, 2007.
[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: Session Initiation Protocol, RFC 3261 (Proposed Standard), updated by RFCs 3265, 3853, 4320, June 2002.
[7] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, STUN – Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs), RFC 3489 (Proposed Standard) (March 2003). URL <http://www.ietf.org/rfc/rfc3489.txt>.
[8] V. Ribeiro, V. Pinto, J. Wellen, W. Hellenthal, W. van Willigenburg, F. Valera, I. Vidal, J. Garcia, M.I. nez, A European high speed Access Platform and Residential Gateway, in: BB Europe, Antwerp, Belgium, 2007.
[9] F. Valera, J. Garcia, C. Guerrero, V.M. Ribeiro, V. Pinto, Demo of triple play services with QoS in a broadband access residential gateway, in: IEEE Infocom, Barcelona, Spain, 2006.
[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, M.F. Kaashoek, The Click Modular Router Project, Internet, <http://www.read.cs.ucla.edu/click/>, May 2006.
[11] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, K. Gibbs, Iperf, <http://dast.nlanr.net/Projects/Iperf/>.
[12] J. Garcia, F. Valera, I. Vidal, A. Azcorra, A broadcasting enabled residential gateway for next generation networks, in: Second IEEE International Workshop on Broadband Convergence Networks (BcN 2007), Munich, Germany, 2007.
[13] M. Handley, V. Jacobson, C. Perkins, SDP: Session Description Protocol, RFC 4566 (Proposed Standard), July 2006.
[14] J. Rosenberg, H. Schulzrinne, An Offer/Answer Model with Session Description Protocol (SDP), RFC 3264 (Proposed Standard), June 2002.
[15] TISPAN, ETSI ES 282 003 V1.1.1: Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Resource and Admission Control Sub-system (RACS); Functional Architecture, March 2006.
[16] TISPAN, ETSI ES 283 003 V1.1.1: Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); IP Multimedia Call Control Protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Stage 3, July 2006.
[17] 3GPP, 3GPP TS 23.228 v8.4.0: Digital cellular telecommunications system (Phase 2+); Universal Mobile Telecommunications System (UMTS); IP Multimedia Subsystem (IMS); Stage 2, 2008.
[18] 3GPP, 3GPP TS 185.003 V2.1.1: TISPAN Customer Network Gateway (CNG) Architecture and Reference Points, July 2008.
[19] DSLForum, TR-098 Amendment 1: Internet Gateway Device Data Model for TR-069, December 2006.
[20] D. Forum, TR-069 Amendment 1. CPE WAN Management Protocol, 2006.
[21] UPnP, Internet Gateway Device (IGD) Standardized Device Control Protocol V 1.0., 2001.
[22] MUSE, Multi Service Access Everywhere., Internet, <http://www.ist-muse.org/>, May 2006.
[23] ASTRALS, Audio–visual STreaming plAtform for domestic Leisure and Security, Internet, <http://www.ist-astrals.org/>, May 2009.

[24] MEDIANET, MultiMedia Networking., Internet, <http://www.ist-ipmedianet.org/>, May 2009.
[25] H.I.G. forum, Home Gateway Technical Requirements: Residential Profile. Version 1.0, Internet, <http://www.homegatewayinitiative.org/publis/HGI_V1.01_Residential.pdf, April 2008.

Jaime García-Reinoso received the Telecommunications Engineering degree in 2000 from the University of Vigo, Spain and the Ph.D. in Telecommunications in 2003 from the University Carlos III of Madrid, Spain. He is currently an associate professor at Univ. Carlos III of Madrid having joined in 2002 and he has published over 25 papers in the field of broadband computer networks, peer-to-peer IPTV and Next Generation Networks in magazines and congresses.

He has been involved in several international and national projects related with protocol design, user localization, broadband access and signaling protocols like the EU IST MUSE, the EU NoE CONTENT project and the BIOGRIDNET project funded by the Madrid Community.

Iván Vidal received the Telecommunication Engineering degree in 2001, from the University of Vigo, Spain, and the Master degree on Telematic Engineering in 2007 and the Ph.D. in Telecommunications in 2008, both from the University Carlos III of Madrid, Spain. He is a research and teaching assistant in Telematics Engineering at University Carlos III of Madrid since 2002. He has been involved in several national and international research projects, including the IST MUSE and E-PHOTON/ONE. His current research interests are focused on multi-service broadband access networks and on network-level multicast transport services within IMS based Next Generation Networks.

Francisco Valera was born in Ciudad Real, Spain in 1974 and received the Telecommunication Engineering degree in 1998 from the Technical University of Madrid, Spain (UPM) and the Ph.D. in Telecommunications in 2002 from the University Carlos III of Madrid, Spain (UC3M). He is currently a tenured associate professor in the UC3M and he has published over 50 papers in the field of advanced communications in magazines and congresses. He has been involved in several international research projects related with protocol design, protocol engineering, network management, advanced networks and multimedia systems. Some of the recent research projects funded by the European Commission in which he has participated are: TRILOGY, MUSE, E-NEXT or E-NET. He has also has participated in the scientific committee, organization and technical review in different national and international conferences (like IEEE Networks, IEEE Communication, Elsevier Computer Communications, IEEE Infocom or IEEE Globecom).

Arturo Azcorra received the Telecommunication Engineering degree in 1986, and the Ph.D. in Telecommunications in 1989, both from the Technical University of Madrid (UPM), Spain. On 1993 he obtained a Master in Business Administration from Instituto de Empresa, Madrid. He is currently a full professor at Univ. Carlos III de Madrid, having joined it on 1998. He has been involved in several international research projects related with protocol design, protocol engineering, advanced networks and multimedia systems. Some of the recent research projects in which he has participated are: MUSE (IST), DAIDALOS (IST), GCAP (IST), LONG (IST), MobyDick (IST), BTI (AC 362), NICE (AC 110), FORMAT (ESPRIT III), BRAIN (RACE), DAMS (ESPRIT II), SDE (ESA-ESTEC), MEHARI (CICYT), SIMM (PLANBA) and SIGER (PASO). He has published over 60 papers in the field of advanced communications in technical books, magazines and congresses. Some of these publication fora are IEEE Communications Magazine, IEEE Network Magazine, European Transactions on Telecommunications, Computer Networks and ISDN Systems, ATM Forum Newsletter, Networld + Interop, IEEE-PROMS-MMNET, PSTV and FORTE. Professor Azcorra has been invited speaker in tutorials, conferences and panels, and has participated in the program committee, organization and technical review in different national and international conferences. These conferences include ACTS Workshop, IEEE-BAC, IEEE-INFOCOM, ABC, IEEE-PROMS-MMNET, JITEL, FORTE and PSTV.