Using Software Defined Networking for Call Admission Control and VoIP applications

Deolindo Zanuttini
Centro de Investigación y Desarrollo en Informática
Universidad Nacional de San Martín
Buenos Aires, Argentina
deolindo.zanuttini@gmail.com

Christian Andres
Universidad CAECE
Buenos Aires, Argentina
chrandres@gmail.com

Joaquin Gonzalez
Centro de Investigación y Desarrollo en Informática
Universidad Nacional de San Martín
Buenos Aires, Argentina
joaquin.gonzalez.ar@ieee.org

Aldana Lacapmesure
Centro de Investigación y Desarrollo en Informática
Universidad Nacional de San Martín
Buenos Aires, Argentina
aldanabl87@gmail.com

Daniel Priano
Centro de Investigación y Dearrollo en Informática
Universidad Nacional de San Martín
Buenos Aires, Argentina
d_a_priano@yahoo.com.ar

Nicolás Pucci
Centro de Investigación y Desarrollo en Informática
Universidad Nacional de San Martín
Buenos Aires, Argentina
nicolaspucci1989@gmail.com

Pablo Bullian
Centro de Investigación y Dearrollo en Informática
Universidad Nacional de San Martín
Buenos Aires, Argentina
pablo.bullian@gmail.com

***Abstract*— Call Admission Control (CAC) is a mechanism that allows limiting the number of concurrent sessions established within a VoIP communication platform based on the SIP protocol. Such mechanism is not standardized, and it brings up practical difficulties to deploy that into traditional network environments where different networking devices from multiple vendors interact with each other. Software Defined Networking (SDN) is a model that decouples the control and forwarding logic from the network infrastructure making it programmable and it provides open interfaces to interact with.**

The aim of this paper is to describe the specifications, design, and deployment of an innovative CAC application that takes advantage of the SDN model. In addition, we report experimental results taken from our approach. We conclude with a discussion on how these applications may provide a vendor agnostic solution for networks such as CAC.

Keywords*— *SIP, CAC, IP PBX, API, OpenFlow, SDN, REST

I. INTRODUCTION: CAC AND SDN

Call Admission Control (CAC) [1] is a congestion control mechanism used essentially in networks that support real-time traffic, where the quality of communications must be guaranteed. A typical mechanism of CAC prevents call establishment when processing resources in network devices are not enough or the amount of calls exceeds a pre-established maximum limit. CAC is also used as a complement to other QoS mechanisms, offering a solution to the queue saturation problem caused by traffic prioritization in DiffServ architectures [2].

Despite the existence of a reference framework [3], there is no uniform criterion or standardized CAC mechanism to achieve interaction among domains of different VoIP service providers. The state of the art over VoIP networks indicates that CAC mechanisms are implementations that reside mostly in the IP PBX, using programming logic and proprietary interfaces chosen by the developer of the solution. This is also a problem from the point of view of unifying a CAC mechanism that works in multi-domains environments with IP PBX belonging to different manufacturers.

The aim of this paper is to solve those problems proposing a software application architecture model with open and programmable interfaces (API's) over a Software Defined Networking (SDN) environment.

The SDN architecture of Figure 1 presents a model [4] where the control plane (SDN controller) is decoupled from the forwarding plane (Network Elements), specifying open and programmable interfaces (API) for the control plane both in Northbound and Southbound directions. This allows the development of software applications that can interact with the control plane by managing the forwarding elements through an abstraction layer that favors the programmability of the network. Although there is currently no defined standard, the OpenFlow protocol is usually chosen in practice for the Southbound interface, whereas implementations of API REST interfaces are preferred for Northbound communication.

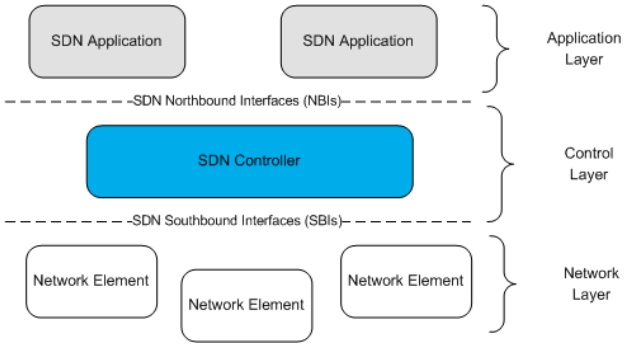

Fig. 1. SDN architecture

978-1-5386-5447-7/18/$31.00 ©2018 IEEE

In the following sections, we describe the operating requirements associated with the CAC mechanism and we define the specifications of Northbound and Southbound interfaces that the application should have. Afterward, we design an application that implements the required features, we describe the development of a prototype to perform tests and obtain experimental results. Finally, we present our conclusions and observations about the modeling of a CAC mechanism in SDN environments.

II. SPECIFICATIONS FOR A CAC APPLICATION

A typical CAC implementation consists of two fundamental functions, one dedicated to allowing or denying a call initialization based on a predefined policy, and another, to assure the quality of current calls.

Considering the client-server approach used in [5], Figure 2 shows a model for our CAC implementation. There, a processing server evaluates the limitations established by the clients to compare it with the active sessions and decides. If a new session is accepted, the processing server changes the state of network elements to set the traffic path from source to destination and set a QoS policy to it. In case the session is denied, the processing server sends the appropriated notification to the clients.

The design of workflows for processing server must take into consideration real-time interaction with clients, because of this swapping of messages between client and server should be prioritized in network elements. Fulfilling these requirements in conventional networks tends to be a real challenge for developers, because of the lack of homogeneous interfaces for network elements. Using SDN allows developers to work with network elements in an abstract way, letting them to modify their behavior and obtain the expected results.

Therefore, we consider that the CAC application must be able to:

1) Interacting with the IP PBX with an open and well-defined interface that permits the exchange of messages and events.
2) Receiving and managing the messages from the IP PBX associated with the following events:
   - Call Start
   - Modification of parameters associated with a current call
   - Call End
3) Denying or allowing a call setup due to the maximum amount of concurrent calls predefined.
4) Setting the needed QoS rules required into the SDN controller.
5) Interacting with an administrator through a web GUI.

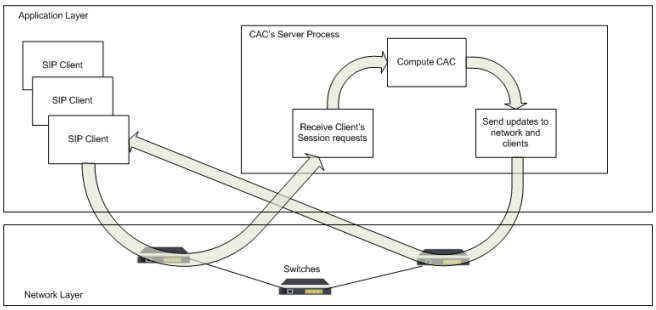

Fig. 2. Client-Server

III. CAC APPLICATION DESIGN

The design of basic architecture for the implementation of the CAC functionality in SDN environments (Figure 3), includes the following components:

a) CAC Server Process: server application that captures the events that IP PBX sends through Northbound API, also sends the necessary information to SDN controller through Southbound API so that the necessary actions are carried out on the network elements.

b) Server: a physical server or VM with the ability to execute CAC server process. The CAC server process could be distributed on multiple servers to mitigate scalability problems.

c) SIP clients: a client application that connects through SIP signaling with the IP PBX for the establishment of VoIP calls.

d) Network elements: programmable SDN switches with a SDN controller using OpenFlow.

e) SDN controller: receives queries coming from northbound API interface. Then, it executes the necessary actions so that QoS policies are applied to data flows established between SIP clients and IP PBX. This is done through the exposed southbound API of the controller that will program network elements using OpenFlow.

f) SDN Module: Integrated within CAC server process, it receives information related to the establishment of sessions and sends directives to SDN controller southbound API.

g) Policy Manager Module: Integrated within CAC server process, it has the ability to manipulate the control policies for CAC process. In addition, it gathers statistics related to resource consumption.

h) CAC Module: Integrated within CAC server process, implements communication with IP PBX through the Northbound API interface. It carries out the monitoring of events generated by IP PBX, controls the establishment of the sessions and sends QoS requirements to SDN module.

i) IP PBX: It establishes communication between SIP clients and manages its sessions. In addition, it sends necessary events and information to CAC module so that the corresponding control is carried out.

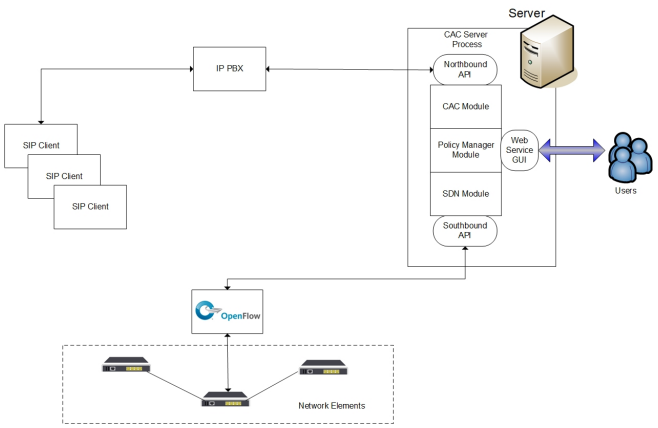

Fig. 3. Basic architecture

IV. PROTOTYPE IMPLEMENTATION

In order to evaluate CAC implementation in SDN environments, a prototype was developed on a completely virtual scenario based on Mininet [6]. Given that the primary objective was to verify the model operation without focusing on metrics related to performance, virtual scenario provides an ideal framework for rapid and low-cost implementations. For prototype development, modular, portable and scalable open source software was preferred.

As a criterion, we chose to remove complexity by focusing on the interaction between CAC application, IP PBX and SDN controller.

The architecture of the application is event-oriented, and it is divided into two main services, front-end and back-end. The front-end has functionalities associated with the interaction with the user, that will display statistics, debug messages and it can input commands to interact with the back-end. The back-end includes the logic of the application and connects to IP PBX, SDN controller, and the front-end, orchestrating the interaction between them.

The front-end was developed on Node.js [7] with the Express.js framework [8], it interacts with the back-end through a web socket interface, allowing a bidirectional and asynchronous real-time dialogue. Into this interface, basic configuration can be modified (in our prototype, the maximum number of simultaneous calls allowed). Another function is the ability to visualize network resources metrics (interface traffic, consumed bandwidth, delay, and latency). Communication between back-end and front-end is done on a process thread instantiated in the back-end, which acts as the client and sends through the web socket necessary data to fill the front-end interface. The back-end integrates communication between an Asterisk-based IP PBX [9], a Ryu-based SDN controller [10] and the network elements based on OpenVSwitch [11]. Asterisk has already a mature REST interface called ARI (Asterisk Rest Interface) [12], which allows communication through a web socket to monitor its events and allows the execution of external applications associated with them through Stasis Applications [13]. These features were used to implement the Northbound API interface of the CAC application.

Figure 4 shows a flow chart that explains the operating logic of the developed CAC application.

When a call is being established, Asterisk transfers the control to CAC application, which monitors events sent by Asterisk. In the event of a call start, the maximum number of calls allowed by the user is compared with the number of calls in progress. If the maximum allowed is exceeded, the application denies and discards the new call.

Figure 5 shows how CAC application controls the call establishment thanks to the Bridge functionality provided by Asterisk [14]. On call initiation, a Bridge is created, and a communication channel is established with the originating endpoint. Then, in case that CAC server allows it, the Stasis App is responsible for routing the call to the destination endpoint and establish a second communication channel against the same Bridge. Both endpoints are now connected through the Bridge. At the same time, Asterisk provides a series of methods and functionalities that allows manipulation of communication channels, which are used to allow or deny the establishment of calls from CAC application.

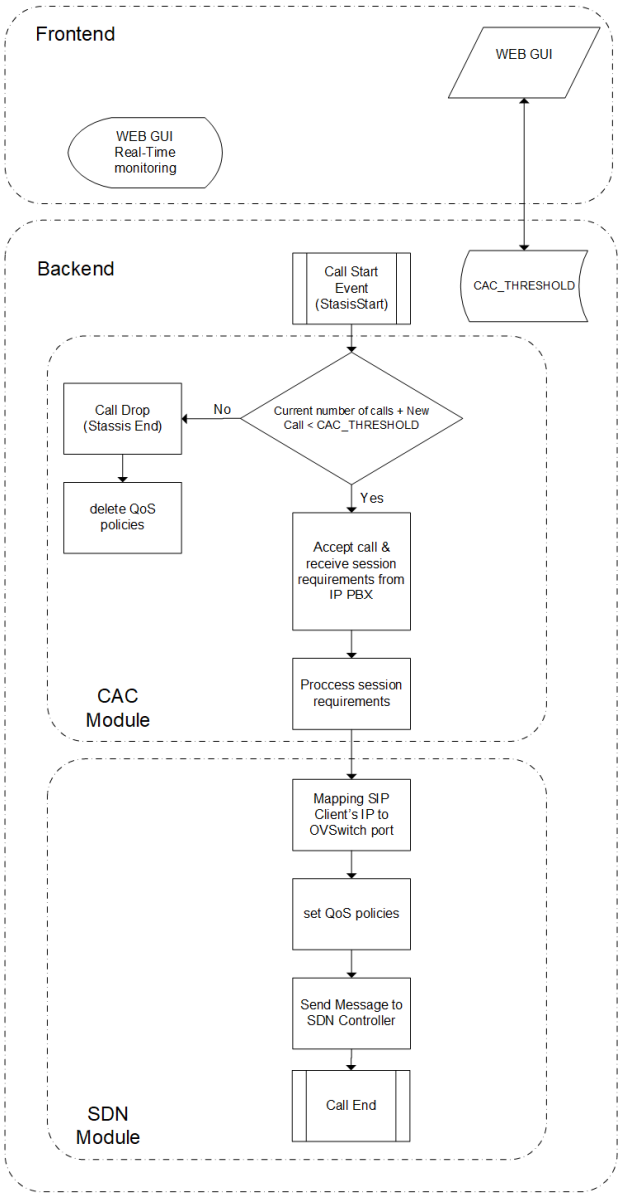

Fig. 4. Flowchart

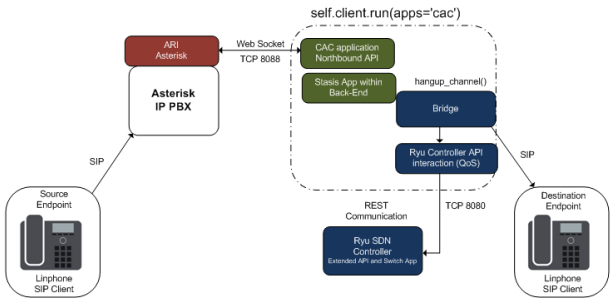

Fig. 5. Stassis App

For example, the *hangup_channel* method cuts a communication channel established between an endpoint and a bridge, which is used in our CAC application to deny a call if the maximum threshold defined by the user is exceeded. Example of a denied call after 2 seconds, once the limit defined for CAC has been exceeded:

```
    if totalChannels >= CAC_THRESHOLD:
        timer = threading.Timer(2,
hangup_channel, [channel, frontClient])
        channel_timers[channel.id] = timer
        timer.start()
```

In case the maximum threshold defined by the user is not exceeded, the call is accepted, and the parameters of SIP session are obtained through ARI for the communication between the endpoints.

In SDN module, three operations are performed based on the information received from each SIP session. The first step is to extract the IP addresses, ports and RTP traffic parameters that the SIP clients will use for the call. With this data, the SDN module must determine the location within the network of SIP clients, obtained by sending a query with the appropriate format to SDN controller through the REST interface. A Python wrapper based on PyCurl [15] was used. The second step consists of creating the OpenFlow rules to set prioritization of RTP media among SIP clients on the switch. The flow rules contain a matching field with the information obtained from the network discovery and two actions, tagging traffic of the RTP flow (field DSCP = 46), and sending it to the high priority queue of the corresponding output port. Lastly, the application monitors and catches Call End events that triggers queries to SDN controller to delete previously defined QoS queues.

In order to give greater flexibility in the treatment of flows and improve the efficiency in the exchange of messages between SDN controller and network elements, we opted to use OpenFlow 1.3, which allows us to build a flow treatment pipeline by using multi-tables [16]. As an SDN controller, Ryu was chosen, which implements OpenFlow 1.3 and has modules for queries, flow establishment and creation of service queues on network elements through a REST interface. The existing Ryu framework was expanded by adding the REST methods necessary to interact with Southbound API interface of SDN module in CAC application. Another addition to SDN controller is the functionality that allows the discovery of endpoints within the network, through a mapping that associates the IP addresses of the endpoints with the physical ports of the OVS Switch. As an SDN switch, we use OpenVSwitch which supports

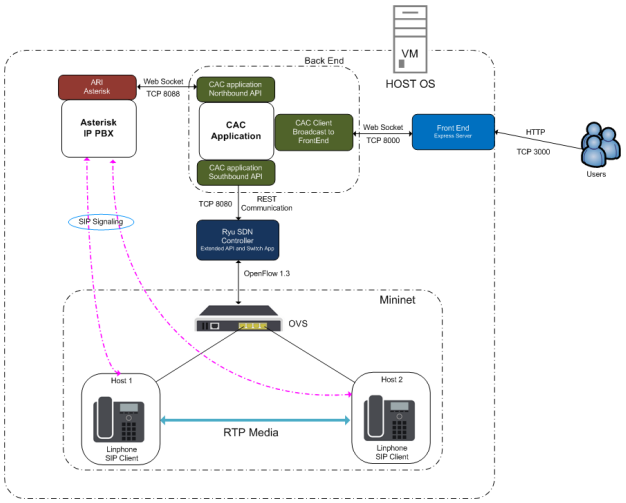

Fig. 6. Proposed prototype

OpenFlow 1.3 as a communication protocol with the SDN controller and allows us to work on Kernel-Space providing more capabilities regarding QoS features. The SIP clients that were used are based on Linphone [17]. In Figure 6, the developed prototype is shown.

V. EVALUATION, TESTS AND RESULTS

The tests carried out consisted of entering through the user interface showed in Figure 7 the maximum number of concurrent calls to 2 (CAC_Threshold). Then a number of calls equal to the CAC_Threshold were established. With each call initiation, a Ryu query was executed to obtain the address mapping corresponding to the hosts (see Figure 8). At this point, it was possible to verify that all calls were established successfully, but when we tried to make a new call, at the start of a second call CAC mechanism is triggered, and the rejection of the call occurs through the *hangup_channel* method of Asterisk. In Figure 9 the closing of the channel established between SIP clients is observed. Due to these results, it is verified that the prototype fulfills the fundamental behavior that a CAC mechanism must perform, preventing or allowing the start of a new call based on a defined maximum threshold.

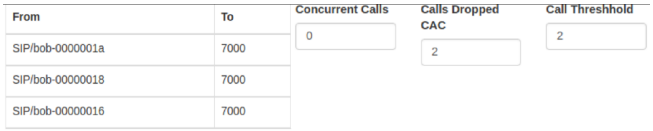

Fig. 7. User interface (Front End)

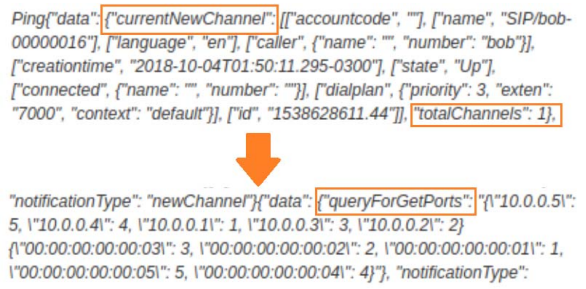

Fig. 8. NewChannel event generates a queryForGetPorts to Ryu

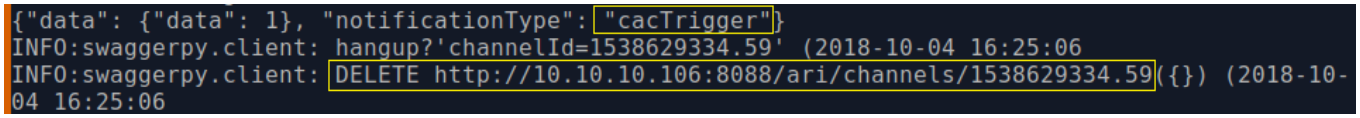

Fig. 9. CAC triggered action and *hangup_channel*

VI. CONCLUSIONS

We focused our contribution on the specification of a model for the development of CAC applications in SDN environments, based on the use of API interfaces and open source tools. In addition, we also described an implementation of a functional application prototype.

It should be noted that although there is not yet a formal standardized framework for the interaction between CAC and IP PBXs from different manufacturers, the presented model provides a solution to this problem, that can be achieved with widely used tools. The implementation of well-specified interfaces (API) by IP PBX manufacturers allows us to decouple the CAC mechanism and adopt a model which is totally independent of VoIP platforms.

The use of SDN environments offers multiple advantages from administrative and operational standpoint of the network in contrast to conventional network devices. For example, it allows us the possibility to interact dynamically with network elements without the intervention of an operator to react to QoS requirements of the calls that are established.

In our first evaluation, we have found results that encourage us to think about the use of the presented model for the development of new applications in SDN environments. For example, one possible application can be the implementation of an adaptive CAC mechanism, where the maximum number of admitted calls is dynamically adjusted to the restrictions of the network such as the available bandwidth on links or the delay variation.

These and other improvements can be developed over the model adopted in this work.

REFERENCES

[1] VoIP Call Admission Control. Available at https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/voip_solutions/CAC.html October, 2018.

[2] IETF RFC2475 (1998), "An Architecture for Differentiated Services," Available at link . October, 2018

[3] IETF RFC2753, "A Framework for Policy-based Admission Control," 2000.

[4] Open Networking Foundation, “SDN architecture 1.0 - ONF TR-502,” 2014.

[5] Tim Humernbrum, Frank Glinka, Sergei Gorlatch, "Using Software-Defined Networking for Real-Time Internet Applications," Proceedings of the International MultiConference of Engineers and Computer Scientists 2014 Vol I, IMECS 2014, March 12 - 14, 2014.

[6] Mininet. Available at http://mininet.org/ October, 2018.

[7] Node.js. Available at https://nodejs.org/en/ October, 2018.

[8] Express.js. Available at https://expressjs.com/ October, 2018.

[9] Asterisk. Available at https://www.asterisk.org/ October, 2018.

[10] Ryu SDN Controller. Available at https://osrg.github.io/ryu/ October, 2018.

[11] OpenVSwitch. Available at https://www.openvswitch.org/ October, 2018.

[12] Asterisk Rest Interface (ARI). Available at https://wiki.asterisk.org/wiki/pages/viewpage.action?pageId=29395573 October, 2018..

[13] Stasis Applications. Available at https://wiki.asterisk.org/wiki/display/AST/Asterisk+13+Application_Stasis October, 2018.

[14] Asterisk Bridges. Available at https://wiki.asterisk.org/wiki/display/AST/Bridges October, 2018.

[15] PyCurl. Available at http://pycurl.io/docs/latest/index.html October, 2018.

[16] Open Networking Foundation, "OpenFlow Switch Specification Version 1.3.0 - ONF TS-006," 2012.

[17] Linphone. Available at http://www.linphone.org/ October, 2018.