Optimizing VoIP server resources using linear programming model and autoscaling technique: An SDN approach

Ahmadreza Montazerolghaem

Optimizing VoIP Server Resources Using Linear Programming Model and Autoscaling Technique: An SDN Approach

Ahmadreza Montazerolghaem*

Faculty of Computer Engineering, University of Isfahan, Isfahan, Iran.

Abstract

Nowadays, the Voice Over IP (VoIP) technology is an important component of the communications industry as well as a low-cost alternative to Public Switched Telephone Networks (PSTNs). Communication in VoIP networks consists of two main phases, e.g., signaling and media exchange. VoIP servers are responsible for signaling exchange using the Session Initiation Protocol (SIP) as the signaling protocol. The saturation of SIP server resources is one of the issues with the VoIP network, which causes problems such as overload or loss of energy. Resource saturation occurs mainly due to a lack of integrated server resource management. In the traditional VoIP networks, management and routing are distributed among all equipment, including servers. These servers are overloaded during peak times and face energy loss during idle times. Given the importance of this issue, this paper introduces a framework based on Software-Defined Networking (SDN) technology for SIP server resource management. The advantage of this framework is to have a global view of all the server resources. In this framework, the resource allocation optimization problem and resource autoscaling are presented to deal with the problems posed. The goal is to maximize total throughput and minimize energy consumption. In this regard, we seek to strike a balance between efficiency and energy. The proposed framework is implemented in the actual testbed. The results show that the proposed framework has succeeded in achieving these goals.

Keywords: VoIP network, Energy-efficient overload control, Linear programming, Autoscaling

*Corresponding author
Email address: a.montazerolghaem@comp.ui.ac.ir (Ahmadreza Montazerolghaem)

1. Introduction

The prevalence and diversity of services provided by IP networks have led to the integration of various technologies and the integration of different types of access networks and the conversion to the next-generation network. The use of SIP [1] protocol, especially for VoIP-based voice and video calls, has been growing, with SIP practically considered as a signaling protocol in IMS, the proposed signaling platform for next-generation networks. Despite having features such as text-based, IP-based, end-to-end, data-driven independence, and mobility support, this protocol lacks a proper mechanism for dealing with overload conditions. This challenge will cause a wide range of next-generation network users to experience a severe decline in Quality of Service (QoS) [2].

Providing QoS on next-generation networks requires deploying mechanisms to deal with the sudden increase in SIP traffic that causes overload on SIP servers. A list of causes of overload is provided in RFC 5390 [3], which may include sudden reductions in processing capacity, component errors, one-off startups, instant congestion, and DOS attacks. Overload conditions are exacerbated when SIP uses UDP and also intends to increase reliability by resubmitting all unanswered requests. Although using TCP does improve server performance to some extent, it does cause scalability problems, as shown in [4] and [5], as well as delay in TCP calls much higher than UDP. Until now, UDP has been the most common option for SIP, and we assume in this paper that SIP runs on the UDP transport layer protocol.

The main server of the service provider in the SIP protocol is called the server. The capacity of each server to other servers (whether the same operator or other operators) is limited. Therefore, the limited resource saturation of SIP servers at peak times is the main reason for these networks overload.

In addition, stationary resources of SIP servers in idle times consume energy, as all server resources are constant at peak and non-peak times. The main solution is server resource management. For example, we present an energy-efficient framework called GreenVoIP to manage the resources of virtualized cloud VoIP centers. By managing the number of VoIP servers and network equipment, such as switches, this framework not only prevents overload but also supports green computing by saving energy in [6]. GreenVoIP is implemented and evaluated on real platforms, including Floodlight, Open vSwitch, and Kamailio. The results show that the proposed framework can minimize the number of active devices, prevent overloading, and provide service quality requirements.

In the present article, we are also looking for the goals of increasing efficiency and reducing

energy consumption with a new approach: linear mathematical modeling, the autoscaling mechanism, and SDN.

The concepts of Software-Defined Networking (SDN) [7, 8, 9, 10] and Network Functions Virtualization (NFV) [9, 11] can provide an abstract, focused view of network resources and servers. This can be quite helpful in managing resources and addressing the challenges of a variety of networks such as overload, energy consumption, routing, load balance, and quality of service assurance [12, 13, 14, 7, 15]. More precisely, software-based networks allow advanced management by separating data planes and controls from one another, as well as centralized and software control. This will lead to efficient routing and resource management. Furthermore, virtualization technology for network functions can help improve resource management by virtualizing a variety of devices and network functions and autoscaling techniques. This can be done by optimally allocating resources as needed[16]. For example, during peak load times when the overload probability is high, it can be possible to increase network resources such as servers (scaling up) or to reduce the resources of inactive network servers (scaling down).

Given the importance of the challenges presented, the main purpose of this paper is to control overload and energy in SIP networks using SDN based resource management. In this regard, we present an SDN-based framework based on the convex mathematical programming model and autoscaling technique to prevent overloading and loss of energy in the SIP network. This framework increases the throughput of the entire network through optimal resource allocation. In this regard, inspired by the congestion control of the TCP protocol, we present the problem of optimizing SIP resource allocation. We also discuss the automation scaling technique for managing SIP resources for optimal energy consumption.

TCP congestion control uses a network model with a set of L links with a limited capacity (C) that are used jointly by N traffic generator sources (with index s). Each traffic generator uses a series of links, so $S(l) = \{s \in N, l \in L\}$. The $L(s)$ set specifies an $L \times N$ matrix (R_{ls}) which equals 1 if the source s uses the link l; otherwise, it equals 0. Each source s generates $x_s(t)$ traffic at any given moment, and each link l updates a value of $\lambda_t(l)$ at any moment as the link cost. The total traffic generated per link must be less than the capacity of this link. This network can be modeled by the Network Utility Maximization (NUM) optimization problem, which is found in [17, 18] introduced:

$$maximize \sum_{s} U_s(x_s) \tag{1}$$

$$subject\ to \quad Rx \leq c \tag{2}$$

$$minimize_{(\lambda \geq 0)} \quad D(\lambda) = \sum_{s} max(U_s(x_s) - x_s \sum_{l} R_s \lambda_l) + \sum_{l} c_l \lambda_l \tag{3}$$

Considering the optimization problem of Eq. (1) and its dual, it is possible to derive the general model of congestion control algorithms and show that the TCP congestion control methods are interpreted into distributed algorithms, which is the problem of NUM Eq. (1).

Resource allocation and congestion control of the TCP and SIP protocols are theoretically similar in that they are optimized to maximize resource utilization. The biggest difference between TCP congestion control and SIP overload control is that TCP congestion control is a mechanism for maximizing the capacity of links, while SIP overload control is a mechanism for maximizing server capacity utilization while preventing overload. Therefore, the SIP overload problem has the following complexities:

• Each server functions both as a source of traffic generation and a link. Users are actual source of traffic generation; however, since each request is related to an individual user and modeling the overload control of such a scenario is practically impossible, the upstream server is considered as the source of traffic generation (when traffic is sent from users of one server to users from another server, the server is upstream. When traffic is sent from other servers to the desired server, the server is downstream).

• SIP network servers must also bear the cost of dropping overload. In TCP, traffic generation resources must themselves control the forwarded load. However, in SIP, this is not an easy task, and traffic is generated by users, each generating only as much as one session. Overload dropping on servers requires the use of processing resources and should be added to the optimization problem.

• Overload control methods (implicit or explicit) affect the efficiency of overload control. In TCP, the cost of links is generally observed by implicit methods in traffic generation sources. The overhead of the implicit method used in traffic generation resources will not have a role in link efficiency. While in SIP congestion control, the cost of overload control (implicit or explicit) has a significant impact on proxy performance.

On the other hand, network functions virtualization provides resources scalability and autoscaling technique. Autoscaling is the key strategy for dynamic resource supply. This technique should automatically allocate the resources needed by the servers. If a server requires some more resources, they should be allocated to the server without causing any problems with the user service. Moreover, if excess resources are needed, additional resources should be temporarily withdrawn so that they can be reused if required. This saves energy and does not

compromise efficiency.

1.1. Contributions

Some of the main contributions of this article can be summarized as follows:

1. Introducing an SDN-based framework for the softwarization of SIP networks for integrated resource management, including two major components, e.g., overload control and energy consumption control.

2. Mathematical modeling of the problem of optimal allocation of resources to SIP servers. Defining the problem mentioned above in the form of a mathematical model, identifying the types of model constraints according to the requirements, assumptions and constraints and goals defined, as well as a precise definition of the variables, parameters, and inputs of the problem. Hence, in this article, a new approach will be presented to tackle the SIP overload because resource allocation has already been used by researchers to control TCP congestion. This paper introduces the optimal resource allocation to prevent overload from increasing the throughput across the SIP network.

3. Providing a mechanism to scale up and scale down the SIP resources for energy control (Developing a scheme of reducing energy consumption using SDN and autoscaling, which effectively decreases the total energy consumption of servers).

4. Implementing the proposed framework in the actual testbed and evaluating the performance of the proposed approach under different scenarios.

So in this paper, by stating the causes of occurrence and persistence of overload in SIP servers, an extensive study has been done on the problems caused by SIP server overload, the behavior of servers in these conditions, and the parameters affecting server performance in this case. Due to the lack of processing and memory resources, proper resource management in the SIP network is very important as the most destructive factor affecting the performance of SIP servers in case of overload. In this regard, a novel framework is presented. The framework is based on resource optimization and SDN-based network technology. To achieve no overload through resource management, the problem of optimal resource allocation is first modeled. According to the server resources, the proposed resource allocation model tries to maximize the acceptance of calls so that they can be distributed among the servers in the network without any overload. Also, with the virtualization of SIP networks and the capabilities such as resizing the resources it provides, the limitation of hardware resources can be somewhat reduced. This approach provides more dynamic and efficient resource management and provides an abstract

view of all SIP network resources. The framework was also equipped with an automated scaling scheme (at the virtual machine level) to allocate an appropriate amount of resources to both achieve the maximum acceptance rate and minimize resource wastage.

1.2. Organizations

The rest of the article is organized as follows. Section 2 discusses related work. Section 3 provides a comprehensive list of challenges in managing SIP resources. Section 4 presents the proposed framework. In this section, we address the details and components of the proposed framework as well as the solution to the optimization problem with the objective function of maximizing the total throughput. In addition, the details of the proposed autoscaling technique are presented in this section. Section 5 introduces the testbed, test tools, performance evaluation, and results. Finally, Section 6 concludes the work and provides suggestions for future work.

2. Related Work

The most prominent existing approaches to overcoming VoIP overload can be categorized as shown in Fig. 1. In this figure, one way to prevent overload is the distribution of the load between SIP servers based on their capacity [19, 20, 21]. Another categorized mechanism is overload control methods, which are divided into five general categories [1, 22, 23, 24, 25, 26], the most important of which are local and distributed methods.

2.1. Local methods

In local methods, each SIP server monitors its consumable resources independently and rejects additional calls without the need to interact with other servers [1].

This method is based on the basic SIP mechanism for overload control, which rejects additional requests by sending a 503 response message. In local overload control methods, the monitor and operator are both located on the overload proxy. In this method, it is assumed that the cost of rejecting additional calls is lower than accepting them. Therefore, the proxy will be able to serve more calls in total while spending less resources to reject additional calls. On the other hand, because the overloaded proxy itself constantly monitors its consumption resources, it applies its control method without the need to interact with other network proxies. Of course, this method alone can not prevent the harms of proxy exposure to overload. It is necessary to install other methods along with this mechanism in other network components. So

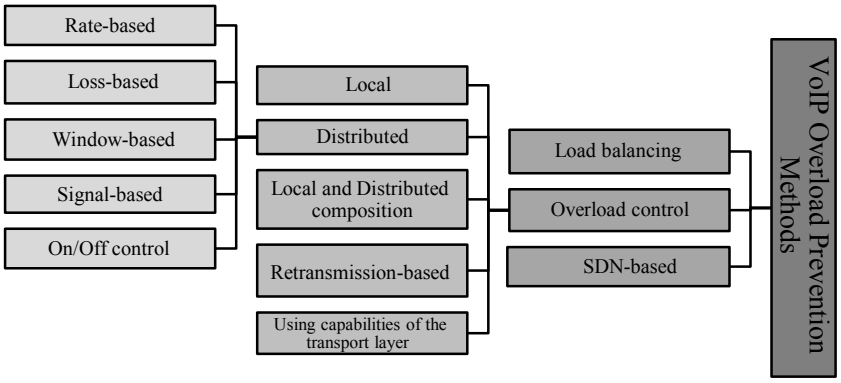

Figure 1: Categorization of papers on VoIP overload

that the local overload control method is the last proxy backup layer. The disadvantage of this method is that the cost of rejecting the call is not insignificant. In the event of a heavy overload, the proxy must devote all its resources to rejecting additional calls. As a result, it will not be able to service all calls and its throughput will be zero. In addition, a "re-try after" header is embedded in this reply message, informing the overloaded proxy of its non-response time to other upstream proxies. During this time, upstream proxies can redirect incoming requests to other proxies on the network. This will cause the load to fluctuate between the proxies in the network if the overload is heavy and continues for a while. Therefore, using this method alone can not completely eliminate the negative effects of overload.

2.2. Distributed methods

In distributed methods, SIP servers prevent overload by interacting with each other and exchanging information. These methods are also divided into five categories, depending on the type of exchanged information[27, 28, 29, 30, 31, 32, 33, 34]. For instance, [31] is a window-based method in which the load is sent to the downstream server only if the upstream server window has empty space. Also, message delays are used to adjust the size of the window.

When a SIP proxy is overloaded by other proxies, the so-called "server-server" overload occurs. In addition to local control mechanisms, distributed overload control methods are used to keep the reference load to the normal load. Distributed overload control methods are also divided into two categories, explicit and implicit: depending on whether the overloaded server informs its status to the upstream servers, or whether the upstream servers themselves are be informed.

In explicit overload control methods with direct feedback, the SIP proxy explicitly uses a signal

(such as a packet or the inclusion of relevant information in a field of a packet) to indicate that an overload is imminent. Upstream proxies that receive this signal adjust their load rate according to the received signal to be optimal for the downstream proxy. The advantage of this method is that each proxy can actively adjust its own load, so that it can use its maximum processing power.

The implicit overload control or indirect feedback method uses the absence of responses, or the loss of packets as a criterion for detecting overload. An upstream proxy that understands such a situation reduces its load rate to its downstream proxy. Given that no explicit feedback is used in this method, it is a robust mechanism, as it is not dependent on proxy reaction under overload. Another advantage of this mechanism is that upstream proxies will be able to avoid overloading the downstream proxy that cannot respond.

In addition, distributed methods are divided into two categories, step-by-step and end-to-end, depending on how the servers on the network interact to control overload. In step-by-step methods, the amount of feedback is transmitted directly between each pair of adjacent proxies. Therefore, independent control policies and parameters can be applied between each pair of proxies. In other words, the operating unit of the overload control structure is located in the upstream proxy, which directly uses the monitoring information located in its downstream proxy to control the overload. But in the end-to-end method, a feedback loop is closed along the path between the primary sender server and the final receiver. Therefore, the control is performed by the primary sender server. In other words, the monitoring unit is located in each of the proxies located along the route, while the operating unit is located at the origin of the sender of the messages. The disadvantage of this method is its high complexity for finding all routes to the desired destination and summarizing the information obtained to decide whether to limit the sending of packets to a particular route.

2.3. Retransmission rate methods

The third approach applied for overload control is based on retransmission rate methods which review retransmission mechanism of SIP by studying servers buffer size [24, 25]. By limiting the dedicated memory of the server, it can be prevented from admitting the over-capacity calls. However, this policy loses its efficiency once the call rate rises, as the server processor is forced to analyze the messages to recognize their content. Therefore, under such conditions, the server reaches saturation, which typically occurs under the higher loads.

As mentioned, by limiting the memory allocated to the server, it can be prevented from

accepting calls beyond its capacity. But this policy loses its effectiveness as the call rate increases. Because in this case, the server processor has to parse messages to know the contents of them. So in this case, too, it will be saturated, although this event will occur under higher loads.

2.4. TCP flow control methods

The next approach to overload avoidance is based on TCP flow control for the purpose of regulating SIP overload [26]. However, problems such as scalability and high delay hinder use of this protocol. Furthermore, the congestion designed for TCP control mechanism occurs due to limited bandwidth, whereas the overload in SIP is caused by the limited processing capacity of servers CPU [24].

As mentioned, overload control can be implemented in any of the data link, network, transmission, and application. Since SIP is an application layer protocol, most overload control methods are application layer. However, methods for controlling overload in the transmission layer when using TCP have also been considered to improve the TCP flow control procedure for controlling SIP overload. In this regard, the effect of using different transmission layer protocols on call delay has been investigated.

Based on the mentioned points, it can be stated that the main disadvantages of the present overload control approaches are: Firstly, their reliance merely over the local call reject reduces their throughput. Secondly, for the majority of explicit feedback-based methods, continuous revision of the status and feedback calculation (which is the function of overloaded server) has complexity and header. The third defect of these methods is the delay in feedback arrival to the upstream, which results in instability of these methods. Nevertheless, these methods are more accurate in overload detection as compared to the implicit methods. Therefore, overload detection, feedback generation, and running the overload control algorithm incurs CPU and memory usage costs and affects throughput of the server.

3. Existing Challenges

This section examines a limited number of challenges in optimizing the allocation of SIP resources, some of which are addressed in this article. These challenges can be helpful for other articles as well. These challenges can be classified into several main phases.

1. Collection of decision-making data and knowledge of open source SIP servers (how and when to measure server resources):

To get the optimal allocation of server resources, the resource allocation algorithm requires some critical information. For example, knowing the number of server resources remaining, the number of call requests per server, the topology, how to connect servers, and so on.

2. Determining the importance of SIP servers activity for allocating resources to them:

To avoid wasting resources on SIP servers, authorized operations should be reviewed to allocate resources to them. Depending on whether the caller and the callee are both on the same domain (registered on one server) or registered on different domain servers, each server call requests can be divided into local or extra-domain requests. Thus, categorizing the activities authorized by the servers to allocate resources to them can be considered as local calls or relaying (trunking) extra-domain requests. For example, in Fig.2, each server can spend its resources on making local calls and calls between two users from different domains. An essential component of the overload control mechanism is the optimal adjustment of local and extra-domain call rates due to limited SIP network resources.

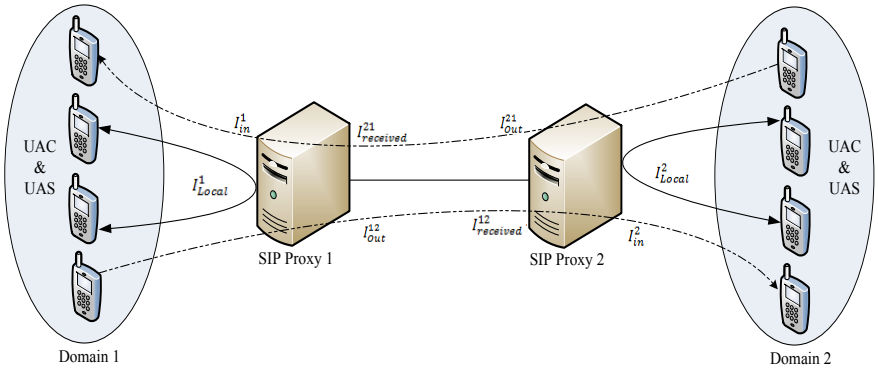

Figure 2: SIP network with two servers

3. Determining the impact factor of server activities on resource consumption:

After determining the authorized activities of the servers to allocate resources to them, the importance coefficient of these activities in resource consumption should be specified as well. For example, making local calls will require more resources than making extra-domain calls to a particular server. Obtaining these coefficients is, in itself, a challenge in allocating server resources. It is also important to note that these coefficients vary for different sources. For example, making a local call affects the processing and memory resources of a server differently.

4. No restrictions on the topology and links between SIP network servers:

As discussed in [35, 36, 31, 37], the SIP server network can have different topologies. The resource allocation algorithm should not limit how the SIP servers are connected or topologized.

As in Fig.3, there is no direct relationship between Server 1 and Server 3. Therefore, to communicate between Server 1 users and Server 3 users, the relevant signaling must pass through Server 2 and reach the destination. Determining this path is also one of the challenges in optimizing the allocation of server resources.

5. Creating a balance between increased throughput and decreased resource consumption of SIP servers:

The proposed resource allocation algorithm should not overlook increasing the total network throughput to maintain server resources. Conversely, it should not increase the likelihood of an overload on SIP network servers to increase the network throughput by allocating total network resources. Creating a balance between increasing the total network throughput and the non-occurrence of overload due to resource constraints is one of the challenges ahead.

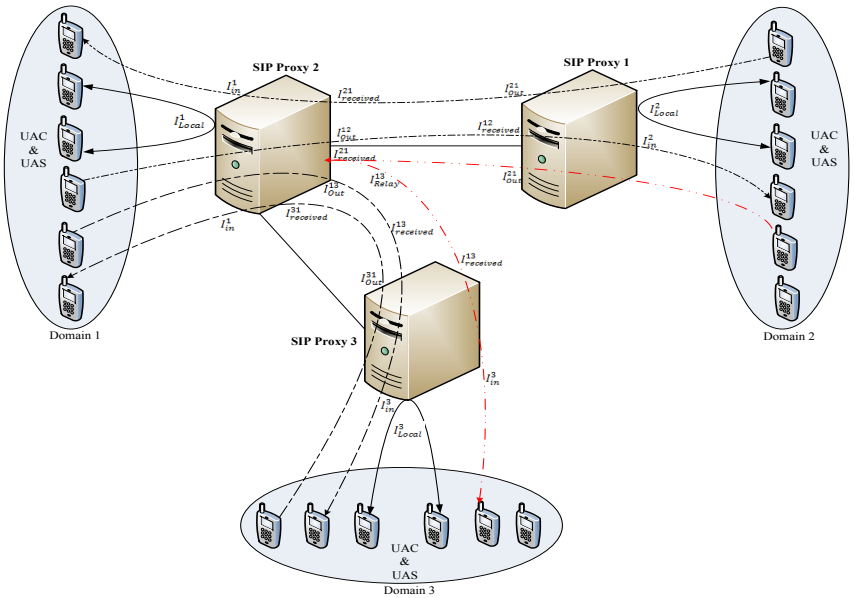

Figure 3: SIP network with three servers and no direct connection between server 1 and server 3

6. Centralized optimal resource allocation:

Gathering server information and deciding how to allocate resources provided that the throughput is increased and overload is not occurred, by a central controller and how to design it is another challenge ahead.

7. The heterogeneity of the network SIP servers.

4. Proposed Approach

As discussed earlier, with the growing use of SIP, the traditional configuration of this type of network is faced with several problems such as ineffective routing, overloaded SIP servers, and lack of optimal management of server resources (including processing and memory resources). An integrated SIP softwarization is required. This section introduces a new approach to upgrading the SIP network framework using new SDN and NVF technologies. SDN allows for advanced management by decoupling data and control panels, as well as centralized and software control. This will lead to efficient routing and resource management. Using SDN, the entire network and its components can be controlled and programmed as an integrated network by software controllers and designed APIs (such as the OpenFlow protocol). SDN architecture and the OpenFlow protocol make the network smarter and more manageable, and the core network infrastructure is separated from applications. NFV can also help SDN by virtualizing a variety of network equipment and functions. This section mainly aims to separate the core functionalities of SIP networks (the mastermind of these networks) from the hardware infrastructure of these networks (which are currently highly complex) and to present as software. The following framework is provided in this regard.

4.1. Proposed Framework

Fig.4 outlines the proposed framework. As can be seen in this figure, a controller tries to maximize the admission of calls so that it can distribute them between servers on the network and save energy at the same time. However, given the limited resources of the servers and the information and status of the servers as well as the knowledge of the topology and paths in the SIP network, this controller seeks to find the optimal solution to the problem by solving the proposed mathematical model.

In addition, due to limited resources of SIP servers (given that hardware resources are constant), full acceptance rates cannot be achieved when the input load is greater than the network capacity. Energy loss occurs when the input load is much lower than the server capacity. Therefore, the virtualization of SIP networks and capabilities such as changes in resource scaling it provides can reduce hardware resource constraints to some extent. This approach leads to more dynamic and efficient resource management. Virtualization techniques can provide an abstraction of physical server resources and allocate a portion of the overall hardware to different operating environments (e.g., different virtual machines), each operating in the form of a separate, independent system. Virtualization also allows for on-demand resource

sharing. As such, computing resources such as CPU and memory as well as storage resources are allocated as needed and as needed. The amount of these resources can be allocated dynamically. As a result, the amount of time spent optimizing, optimizing, and wasting resources for the provider is also minimized. This eliminates static resource allocation (which allocates resources as much as needed and commensurate with its peak usage). In short, virtualization leads to improved resource efficiency, dynamic resource sharing, and better energy management along with improved scalability, accessibility, and reliability of resources.

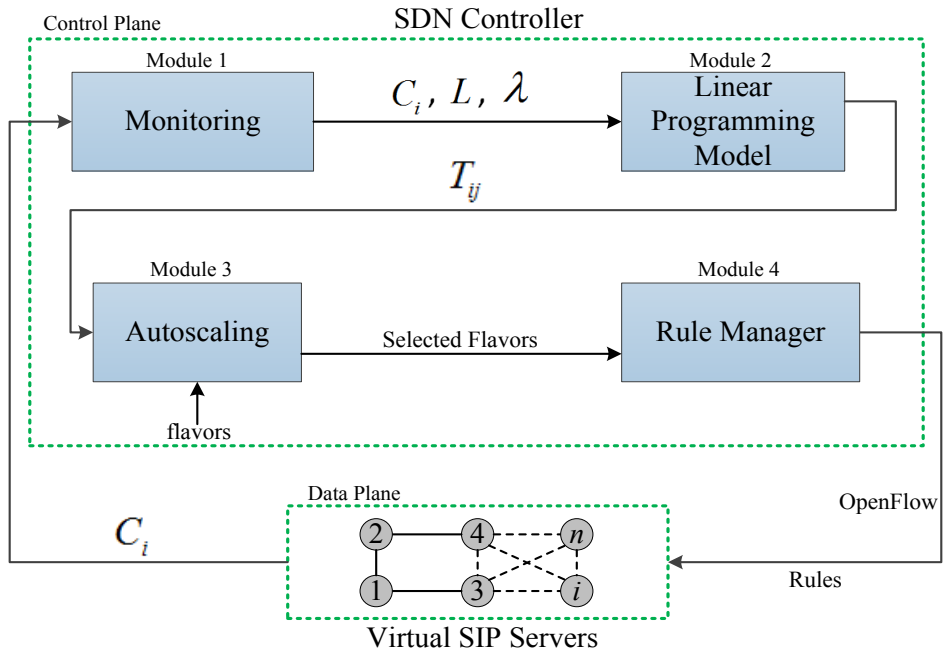

Figure 4: The proposed SIP resource management framework based on the SDN concept

As shown in Fig.4, the data plane includes SIP virtual servers. These servers are equipped with the OpenFlow protocol to communicate with the SDN controller. The controller has four main modules. The monitoring module collects and observes information required from the infrastructure. Given the information collected as well as the capacity of the servers, the second module calculates the maximum possible throughput by solving a linear programming model in a short time. Given the output of module 2, the next module calculates and satisfies the required resources. That is, it determines the size of the virtual machine for each SIP server. Finally, the last module sends the results to the servers in the form of appropriate rules and OpenFlow messages.Fig. 5 shows the proposed controller algorithm described.

Controller Proposed Algorithm

Input:
  Module1: Monitoring
  C, L, λ
Output:
  Module2: Linear Programming Model
    for $i = 1$ **to** number of SIP servers (n)
      calculate T_{ij}
    end for
  Module3: Autoscaling
    for $i = 1$ **to** number of SIP servers (n)
      select flavors
    end for
  Module4: Rule manager
      set rules
end

Figure 5: Controller Proposed Algorithm

4.1.1. Linear Programming Model

Generally, the SIP network consists of L SIP servers with a C_i capacity. Each server divides this capacity into upstream and downstream between other servers. Fig.6 shows the SIP network with four servers ($L = 4$). For example, in this figure, the request rate of Server 1 users to Server 2 users is specified with λ_{12}. In overload mode, Server 1 sends T_{12} to Server 2 and drops the rest.

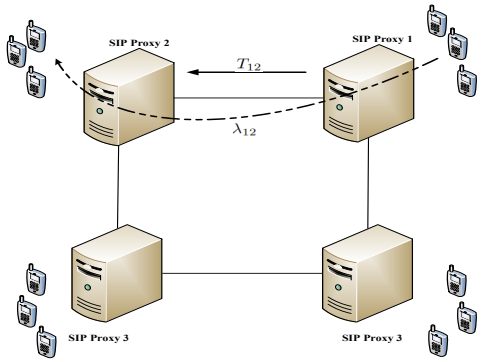

Figure 6: SIP-based network with L = 4

An essential component of the overload control mechanism is to set the transmission rate to each server (T_{ij}). Therefore, in the optimization problem, this parameter must be adjusted to maximize the objective function. The other point is that in the overload mode, the upstream

transmission request rate may be higher than the allocated capacity from one server to another (λ_{ij}). Some requests will be dropped at a w_{rej} cost, and the throughput will be equal T_{ij}. Based on these assumptions, the optimization problem will be defined as following equation:

$$maximize \sum_{i,j \in L} U_{ij}(T_{ij}) \tag{4}$$

$$subject\ to \tag{5}$$

$$\sum_{j \in L} (T_{ij} + W_{rej}(\lambda_{ij} - T_{ij}) + \lambda_{ij} k w_{ret} + T_{ji}) - T_{ii} \leq C_i, i \in L \tag{6}$$

$$0 \leq T_{ij} \leq \lambda_{ij}, i, j \in L \tag{7}$$

The first constraint of this problem is written to divide the server capacity between other servers into two upstream and downstream roles. In this statement, the dedicated capacity of server i must be greater than the sum of the following streams:

- T_{ij} :Transmission server throughput toward other downstreams.
- $(\lambda_{ij} - T_{ij}).W_{rej}$:Overhead of downstream capacity overload dropping.
- $(\lambda_{ij} k).W_{ret}$: The cost of processing downstream traffic redirects coming from users. The third term of this statement is written for cases where the upstream server input load is resent to an average of k times.
- T_{ji}: Throughput received from upstream servers.

It is noteworthy that in the first case of the optimization problem since T_{ii} appears twice in the first two and four terms of sigma, sigma is reduced. The second constraint of the optimization problem determines that all throughputs (T_{ij}) and input loads (λ_{ij}) must be a non-negative value, and none of the throughputs can be higher than their corresponding input traffic.
The objective function of the Eq. (4) is the sum of the productivity of each traffic flow. The productivity of each traffic flow is a function of the transmitted throughput.

4.1.2. Autoscaling

Next, we consider the cost of dropping calls in the linear programming model introduced as zero. This means that dropping calls does not happen because the resources are fixed. The process of matching the number of resources required with server requirements is called horizontal scaling, which can be very challenging. This approach is implemented by two general

methods, e.g., horizontal and vertical. In horizontal scaling, also known as scale in /out, new virtual machines are added to the computational system or platform, and the workload is distributed among all available virtual machines. In vertical scaling, also known as scale-up/down, more resources are added to the same virtual machine to fulfill server requests. For example, adding more RAM, disk space, or CPU to the same machine. As such, we manage load increases. This paper uses vertical scaling assuming that we have f (flavor) sizes with specified sources as flavor $flavor[f] = [P_f, M_f]$ (P_f represents processor and M_f represents virtual machine memory f). The third module selects the closest size for each server virtual machine according to Table 1 so that the throughput is satisfied and minimum resources are used (to reduce power consumption). If total throughput of server i is equal to its capacity, no action is taken; if it is greater than its capacity, scale-up should be done, and if it is lower, scale-down should be done. Resource scale-up occurs when overload is likely. Scale-down also happens to save energy. Fig. 7 shows the autoscaling proposed algorithm.

Table 1: Scaling rules

Actions	Rules
NOP	$\sum T_{ij} = C_i$
Scale Up	$\sum T_{ij} > C_i$
Scale Down	$\sum T_{ij} < C_i$

The third module aims to dynamically adapt the server resources based on throughput and establish a balance between fulfilling the throughput and minimizing resource consumption. It should be noted that concerning the obtained T_{ij}, this paper assumes that routing between OpenFlow switches is performed by the *Dijkstra's* algorithm and in the controller. The rules are also installed on OpenFlow switches by the controller.

Autoscaling Proposed Algorithm

Input:
$flavor\ [f] = [P_f, M_f]$
T_{ij}, C_i
Output:
for $i = 1$ **to** number of SIP servers (n)
  for $j = 1$ **to** number of flavors (f)
    if $\sum T_{ij} = C_i$ **then**
      No Operation
    end if
    if $\sum T_{ij} > C_i$ **then**
      Scale Up $flavor(f)$
    end if
    if $\sum T_{ij} < C_i$ **then**
      Scale Down $flavor(f)$
    end if
  end for
end for

Figure 7: Proposed Algorithm for Autoscaling

5. Performance Evaluation

In this section, open-source Asterisk[1] software is used to implement SIP server, open-source SIPp[2] software for user agent implementation. Moreover, Floodlight is used for controller implementation, and CVX software [38] is used to solve the mathematical model. All servers and controllers have homogeneous hardware, including the INTEL Dual Core 3GHZ processor and 2 GB of RAM using the CentOS 6.3 Linux operating system. Asterisk software reports are also used to measure call status. Oprofile software is used to measure CPU usage and server memory.

The requirements for the implementation of the proposed method is given in Table 2 This

[1]Asterisk is a software implementation of a private branch exchange (PBX). In conjunction with suitable telephony hardware interfaces and network applications, Asterisk is used to establish and control telephone calls between telecommunication endpoints, such as customary telephone sets, destinations on the public switched telephone network (PSTN), and devices or services on voice over Internet Protocol (VoIP) networks. Its name comes from the asterisk (*) symbol for a signal used in dual-tone multi-frequency (DTMF) dialing.

[2]SIPp is a free Open Source test tool / traffic generator for the SIP protocol. It includes a few basic SipStone user agent scenarios (UAC and UAS) and establishes and releases multiple calls with the INVITE and BYE methods. It can also reads custom XML scenario files describing from very simple to complex call flows. It features the dynamic display of statistics about running tests (call rate, round trip delay, and message statistics), periodic CSV statistics dumps, TCP and UDP over multiple sockets or multiplexed with retransmission management and dynamically adjustable call rates.

prototype was implemented in VoIP type approval laboratory whose backbone is 10 Mbps links.

Table 2: Testbed Characteristics for the Experiments

	VoIP Servers	SDN Controller	OpenFlow Switches	Traffic Generator
Software	Asterisk	Floodlight v1.2	Open vSwitch v2.4.1	SIPp
Quantity	3	1	2	1
CPU	INTEL Dual Core 3GHZ	INTEL Dual Core 3GHZ	INTEL Dual Core 1.8GHz	INTEL Dual Core 1.8GHz
RAM	2 GB	2 GB	1 GB	2 GB
Operating System	CentOS 6.3 Linux	CentOS 6.3 Linux	Linux kernel v3.10	Red Hat v6

5.1. Implementation and Experimental Results

In this section, it is assumed that the network has two SIP servers. Server 1 parameters are $C_1 = 460\ cps$, $\lambda_{11} = 100\ cps$, $\lambda_{12} = 180\ cps$ and Server 2 parameters are $C_2 = 460\ cps$, $\lambda_{22} = 50\ cps$, $\lambda_{21} = 180\ cps$. In addition, $k = 0, W_{ret} = 0.1, W_{rej} = 0.2$.

The considered values are selected with high sensitivity. They are selected from several values. Each value is tested several times and the average of the results is considered to bring the results closer to the actual value. C Indicates the capacity of the server, which depends on the hardware specifications of the server. The C values were considered according to the specifications of the servers mentioned in the previous subsection, as well as the various tests performed on the server. C_1 is equal to C_2 because the hardware of both servers is the same (all servers and controllers have homogeneous hardware, including the INTEL Dual Core 3GHZ processor and 2 GB of RAM using the CentOS 6.3 Linux operating system). λs are percentage of C that servers can handle. So the sum of λ_{11} and λ_{12} and $\lambda_{21}$is less than or equal the C_1.

We define three scenarios, e.g., low-load, medium-load, and high-load. In the low-load scenario, 400 calls per second (cps) is injected into the network. In the medium and high-load scenarios, 600 cps and 800 cps were injected into the network, respectively. For SIP virtual machines, four flavors ($f = 4$) are given in Table 3. Initially, the specifications of all virtual machines are homogeneous, and others are small.

Next, we show the results of network traffic injection. Fig.8 shows the server throughput (T_{ij}) in three low-load, medium-load, and high-load scenarios for the proposed SDN-based mechanism and the traditional mechanism. As can be seen, the throughput in SDN mode is higher than the traditional mode in all three scenarios. Besides, as the load (C_{ij}) increases, the throughput also increases. Nonetheless, increased throughput is higher in SDN mode because, in the high-load scenario, the traditional mode becomes overloaded and fail to handle the load.

Table 3: Available flavor specifications

Flavor	Memory (MB)	vCPUs	Disk (GB)
Small	2048	1	20
Medium	4096	2	40
Big	8192	4	80
Very Big	16384	8	160

Table 4 shows the flavor of virtual machines for different C_{ij}s in both SDN-based and traditional modes. In the traditional mode, the flavor of the virtual machine is the same as the original size. Nevertheless, in SDN mode, the flavor of the machine varies with load. Under heavy loads, the flavor of the machine grows and shrinks under light loads. This is effective in saving energy consumption. From Fig.8 and Table 4, it can be concluded that compared to the traditional mechanism, the SDN-based mechanism has managed to increase throughput (according to the linear programming model) and contribute to a decrease in energy consumption (due to the virtual machine scaling mechanism).

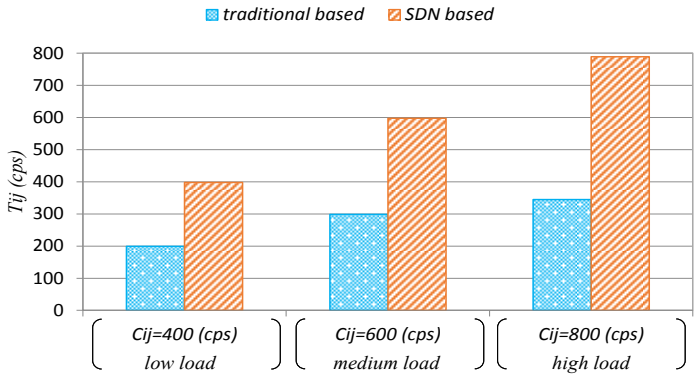

Figure 8: Server throughput in three scenarios

Table 4: The flavor of virtual machines at different loads

C_{ij}	400 (cps)	600 (cps)	800 (cps)
Traditional-based	Small	Small	Small
SDN-based	Small	Medium	Big

Fig.9 shows message retransmission rates. As virtual machine flavors do not change in the traditional method and remain small, overload occurs in scenarios 2 and 3 (medium- and high-

load). This causes resources to be saturated, and the message rate to increase rapidly. However, the SND-based method does not overload as the central controller solves the mathematical model as well as regulates the sources. This does not increase the retransmission rates and the congestion of the network.

An increase in retransmission rates is directly related to the occurrence of overload. As much as this rate can be controlled, overload can be prevented. The reason for the increase in retransmission rates is the inability of servers to respond to requests, which is due to a lack of resources. In other words, if server resources are saturated, then retransmission begins and grows in a short time.

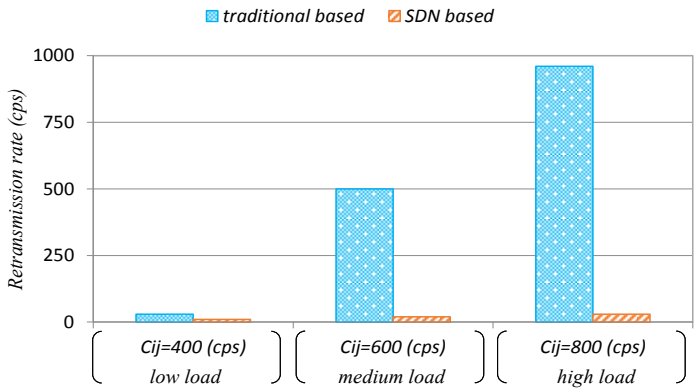

Figure 9: Retransmission rate in three scenarios

Fig.10 shows that resource saturation and exposure to overload increase delay in the traditional mode. However, the SDN-based mechanism has been able to keep the delay below 100 ms even under a high-load scenario. The reason for the high delay in the traditional method is illustrated in Fig.11. In the traditional method and medium- and high-load scenarios, the message rejection rates increase dramatically. The reason for this is the lack of awareness and centralized decision-making in this way. Despite consuming all the resources, the traditional method still failed to handle all the messages properly, resulting in higher rates of rejection and delay.

Delay is toxic to multimedia systems. The main reason for the increase in delay is the growth of the rejection rate. Lack of processing resources and memory of servers makes it impossible to respond to requests in the allotted time and the timer is timed out. From now on, retransmissions will start and the situation will get worse. As the retransmissions rate increases, the shortage of resources gets worse. As the resources become saturated, the rejection rate increases and the delay becomes very large.

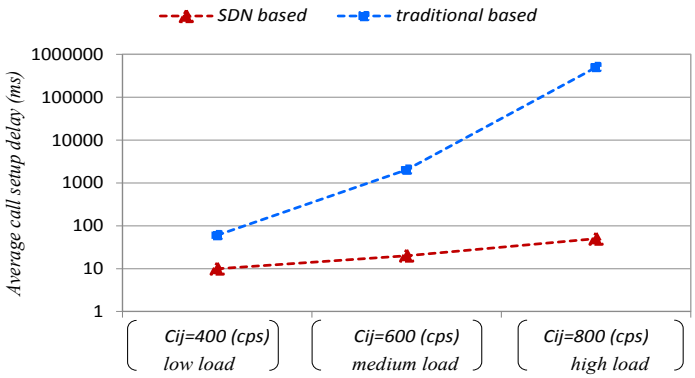

Figure 10: The average delay in three scenarios

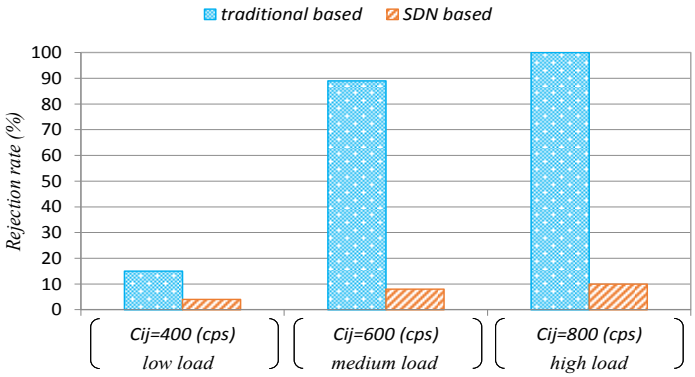

Figure 11: Message rejection rate in three scenarios

In the proposed mechanism, servers will never face resource saturation (Fig.12 and 13). Therefore, even at heavy input loads, the performance of the servers will not decrease, and they will not face the consequences of overload (Fig. 8 to 10). For example, in the high-load scenario, without a sudden increase in the retransmission rate and the average call delay (Fig.9 and 10), it reaches the throughput of 789 successful requests from 800 requests only by consuming an average of 50% CPU and 61% RAM (Fig.12 and 13) yielded (Fig.8). However, in the traditional mechanism and the high-load scenario, a throughput no better than 344 successful requests can be reached with all the server resources consumed (Fig.8).

Increasing throughput and reducing energy consumption in the proposed method is based on the global, integrated, and coordinated management of traffic and network resources. Since the proposed controller oversees the entire network and its components, it can make decisions about resource resizing and server acceptance rates in a centralized and accurate manner. This is much more efficient and informed than when decisions were traditionally made and distributed

by switches or servers.

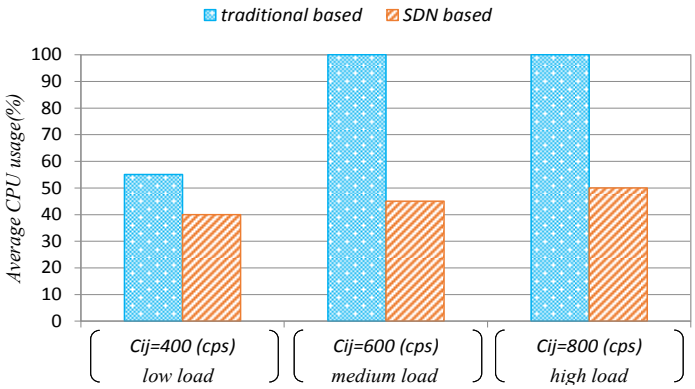

Figure 12: The average CPU usage of servers in three scenarios

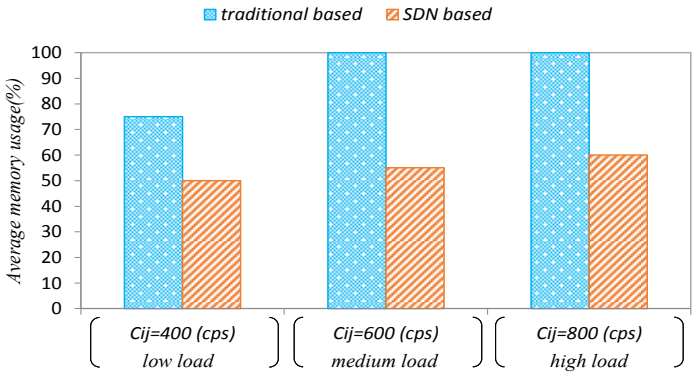

Figure 13: The average memory usage of servers in three scenarios

All of the previous experiments were only in one period; however, in the subsequent experiments, we will examine efficiency over time. Fig.14 shows the cumulative throughput of the servers over time. In this experiment, the time is divided into five 100s intervals, during which low-load, medium-load, and high-load are executed scenarios, respectively. As can be seen, in SDN-based mode, throughput is higher in all scenarios compared to the traditional mode. For example, in the 200-300s interval where the high-load scenario is implemented, the traditional mode failed to achieve a better throughput than 300cps due to being overloaded. However, the SDN-based method has achieved the maximum possible throughput of approximately 1100cps. Note that the T_{ij} fluctuation changes according to the input load to the system.

Fig.15 and 16 show the average resources consumed over time. In these figures, it is evident that resource saturation has taken place in the traditional mode, and this has led to a decline in throughput. However, the SDN-based approach has been able to achieve maximum throughput

with minimal resource consumption by resource scale-up and scale-down. This is also illustrated in Table 5. Due to the input load (C_{ij}), the flavor of the virtual machines on the SIP servers is changed, which is very effective in power consumption. However, in the traditional mode, the flavor of the machines is constant throughout the time interval. In sum, it can be concluded that the SDN approach has managed to create a trade-off between energy and throughput in the VoIP system by solving the proposed linear programming model, as well as the proposed autoscaling technique.

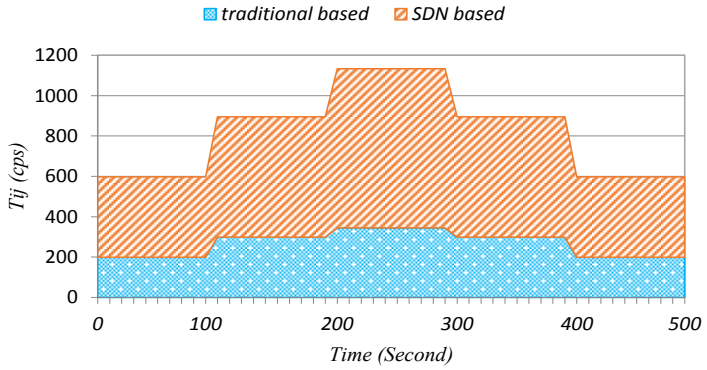

Figure 14: Cumulative throughput of the servers over time

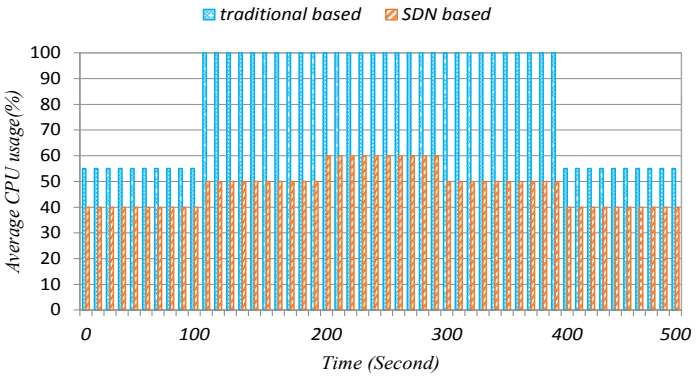

Figure 15: The average CPU usage of servers over time

Fig. 17 shows the details of processor consumption by the proposed controller obtained through Oprofile. As can be seen, the Floodlight Kernel takes up more than half of the controller processor, but the proposed modules consume much less CPU power. This is evidence of the high scalability of the proposed controller. This means that the proposed modules do not impose a high processing overhead on the controller. Among these modules, the Linear Programming Model module consumes more CPU due to the implementation of the mathematical

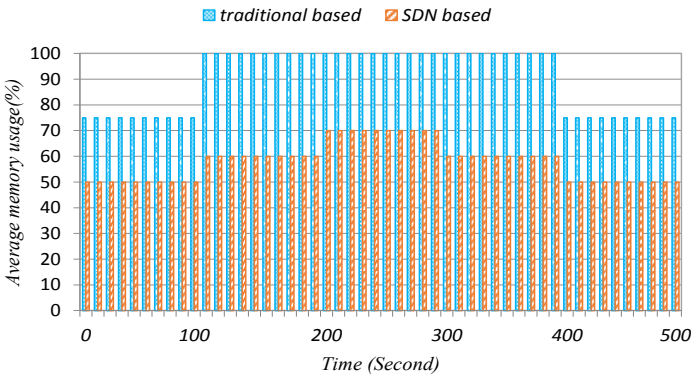

Figure 16: The average server memory usage over time

Table 5: Size of virtual machines at different loads and over time

Time (Second)	0-100	100-200	200-300	300-400	400-500
C_{ij}	400(cps)	600(cps)	800(cps)	600(cps)	800(cps)
Traditional-based	Small	Small	Small	Small	Small
SDN-based	Small	Medium	Big	Medium	Small

method. The Autoscaling module takes the next place in terms of CPU consumption due to the implementation of the flavor increase and size decrease algorithms. This figure also shows that the controller processor is not saturated up to the heavy load of 500 cps.

Also, Fig. 18 shows the details of memory consumption by the proposed controller. As can be seen, the Floodlight Kernel takes up more than half of the controller memory, but the proposed modules consume much less memory. This is evidence of the high scalability of the proposed controller. This means that the proposed modules do not impose a high memory overhead on the controller.

Figure 19 shows that as time goes on and load increases, CPU consumption also increases. But this increase is different for different modules. For example, for the kernel module more than other modules. So proposed modules are not overhead and processing bottlenecks.

This is also shown in Figure 20. As can be seen, with the passage of time and increasing load, memory consumption by different modules also increases. Most of this increase is related to the kernel and the proposed modules are not an additional overhead.

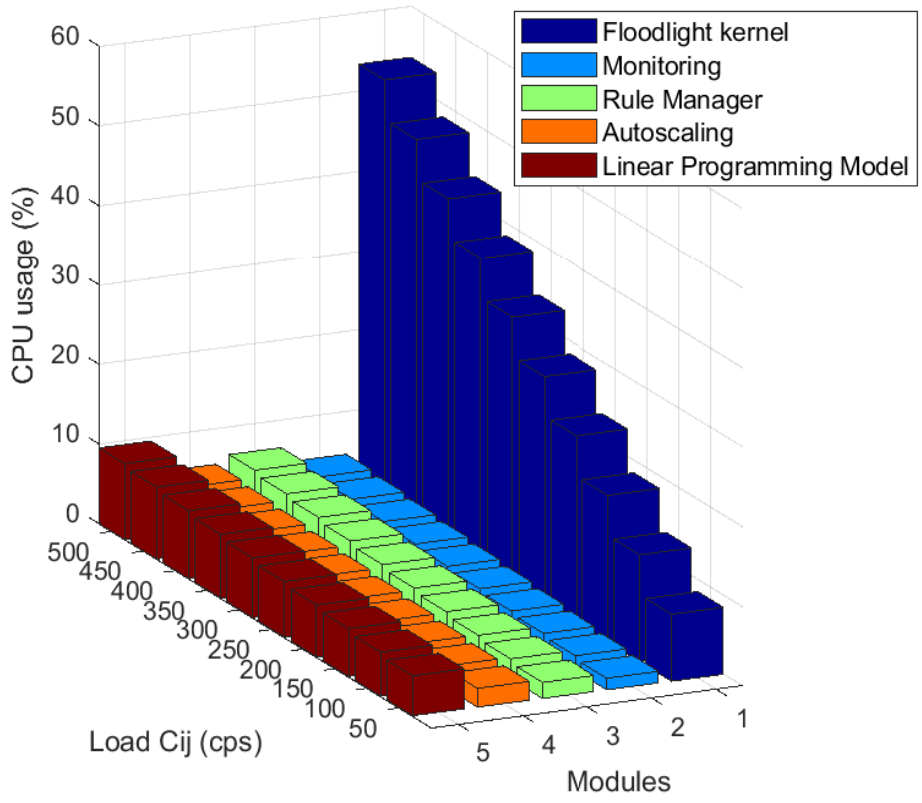

Figure 17: The computational overhead of proposed modules by changing the load

6. Conclusion and Future Work

Saturation or overflow of SIP server resources is one of the issues with the VoIP network, bringing about several problems such as overload or energy loss. These servers are overloaded during peak times and face energy loss in idle times. Given the importance of this topic, this paper introduces a framework based on software-defined networking (SDN) technology for managing SIP server resources. The advantage of this framework is to provide a global view of all the server resources. In this framework, the resource allocation optimization problem, as well as resource autoscaling, are presented to deal with the problems posed. The goal is to maximize total throughputs and minimize energy consumption. In this regard, we seek to strike a balance between efficiency and energy. In this respect, a variety of experiments were carried out under different scenarios in a real test environment. The results show that the proposed SDN-based approach has succeeded both in improving the metrics related to overload control and in improving energy.

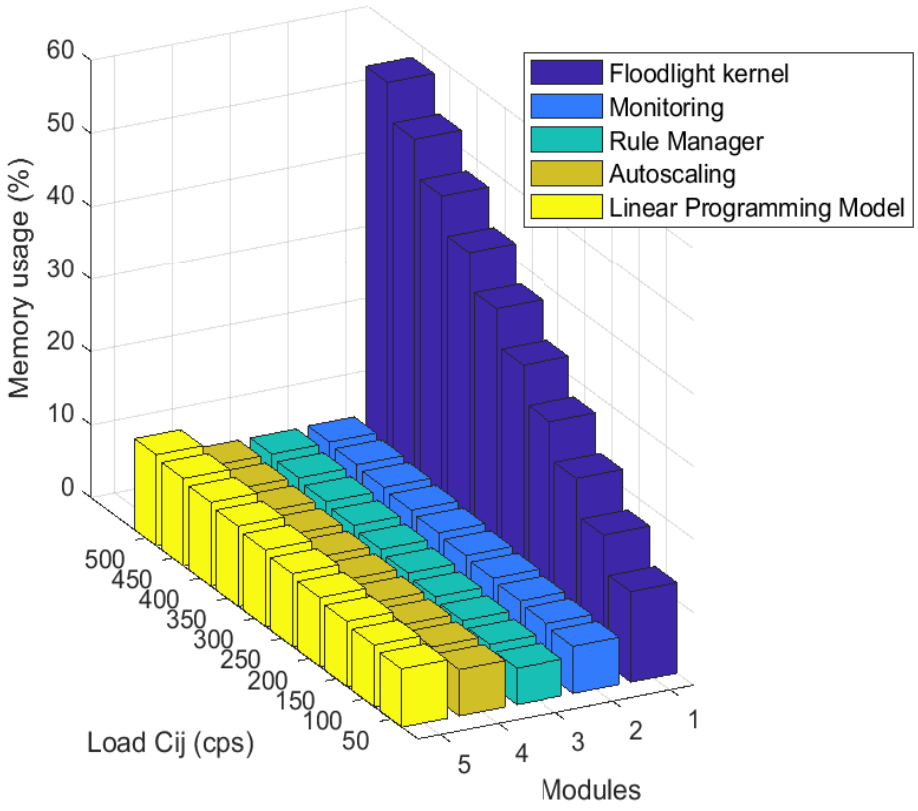

Figure 18: The memory overhead of proposed modules by changing the load

The results show that resource saturation has taken place in the traditional model, which has led to a decline in throughput. However, the proposed SDN-based approach has been able to achieve maximum throughput with minimal resource consumption by resource scale-up and scale-down. In the proposed mechanism, servers will never face resource saturation. Therefore, even at heavy input loads, the performance of the servers will not decrease, and they will not face the consequences of overload. For example, in the high-load scenario, without a sudden increase in the retransmission rate and the average call delay, it reaches the throughput of 789 successful requests from 800 requests only by consuming an average of 50% CPU and 61% RAM. However, in the traditional mechanism and the high-load scenario, a throughput no better than 344 successful requests can be reached with all the server resources consumed. In sum, it can be concluded that the SDN approach has managed to create a trade-off between energy and throughput in the VoIP system by solving the proposed linear programming model, as well as the proposed autoscaling technique.

It is also important to note that the capacity of the entire SIP network in this method is limited

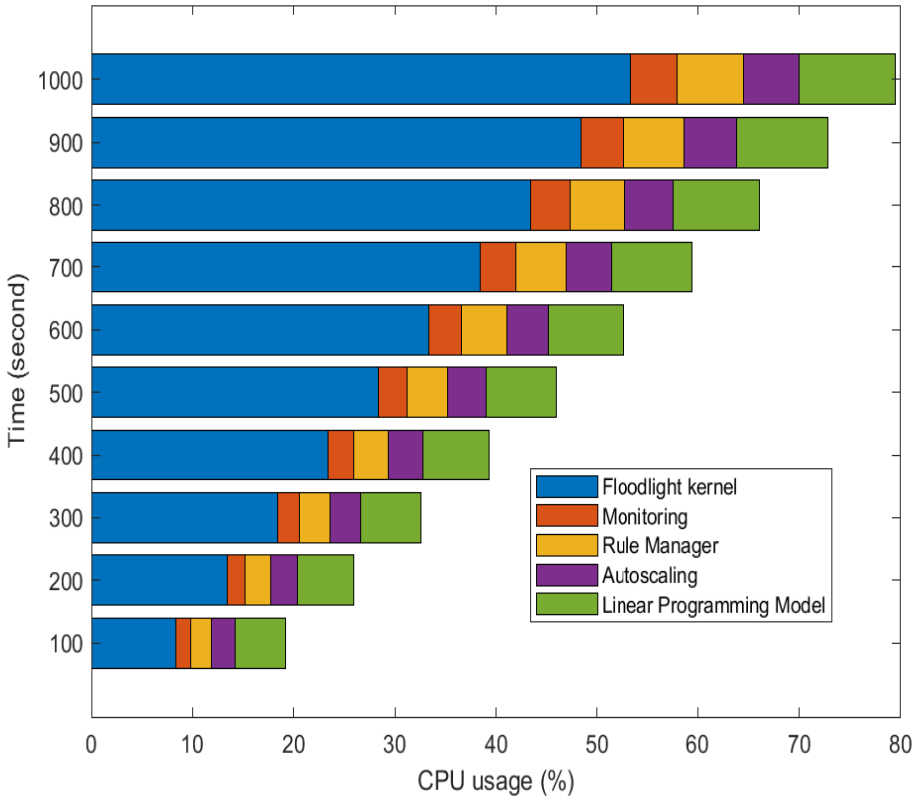

Figure 19: The ~~memory~~ overhead of proposed modules during the time

to the capacity of the controller. Of course, because the controller is logically centralized, multiple controllers can be used. In addition, the designed modules are very simple so as not to add complexity and overhead to the controller. The result is an efficient and agile controller that can manage the network seamlessly. We can also divide the whole SIP network into a number of domains and consider separate controllers to manage each domain. In this case, the task of the controller becomes easier.

For future work, we will generalize and develop mathematical models for optimal allocation of SIP resources as well as taking into account SLA traffic flows between servers. We are also looking to develop autoscaling policies. Using the virtual machine migration technique to manage virtual SIP servers is also a part of the future works of this article.

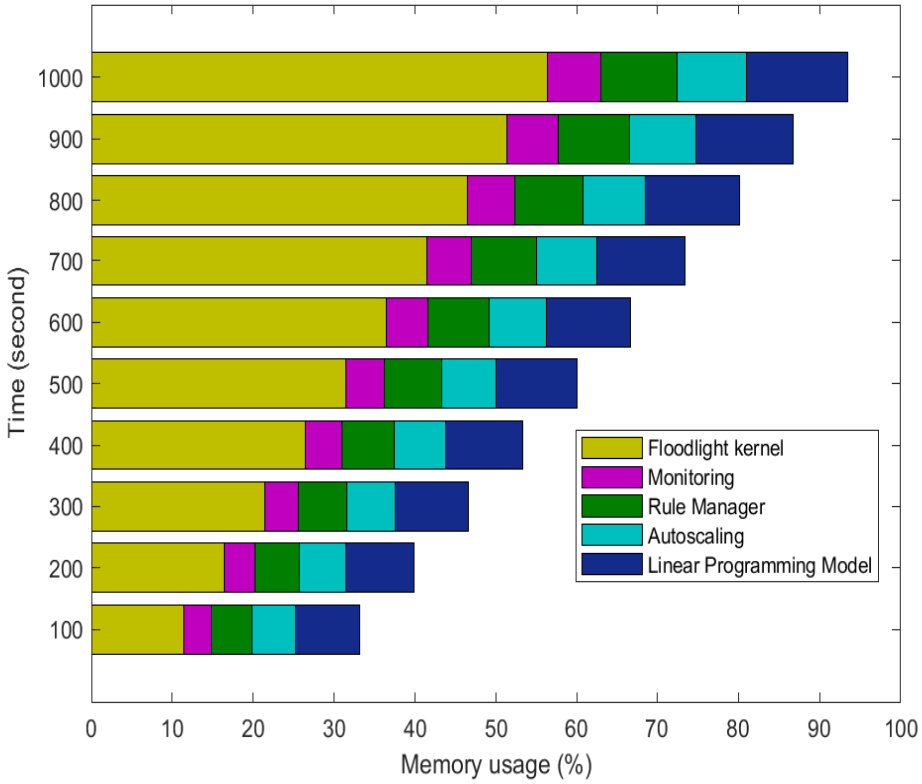

Figure 20: The memory overhead of proposed modules during the time

References

[1] L. De Cicco, G. Cofano, S. Mascolo, Local sip overload control: Controller design and optimization by extremum seeking, IEEE Transactions on Control of Network Systems 2 (3) (2015) 267–277.

[2] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, Sip: session initiation protocol, Tech. rep. (2002).

[3] J. Rosenberg, Requirements for management of overload in the session initiation protocol, rfc 5390.

[4] V. K. Gurbani, R. Jain, Transport protocol considerations for session initiation protocol networks, Bell Labs Technical Journal 9 (1) (2004) 83–97.

[5] M. Ohta, Performance comparisons of transport protocols for session initiation protocol

signaling, in: 2008 4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks, IEEE, 2008, pp. 148–153.

[6] A. Montazerolghaem, M. H. Yaghmaee, A. Leon-Garcia, Green cloud multimedia networking: Nfv/sdn based energy-efficient resource allocation, IEEE Transactions on Green Communications and Networking 4 (3) (2020) 873–889. doi:10.1109/TGCN.2020.2982821.

[7] V. S. N. Amulothu, A. Kapur, K. Khani, V. Shukla, Adaptive software defined networking controller, uS Patent App. 10/257,073 (Apr. 9 2019).

[8] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76.

[9] Y. Li, M. Chen, Software-defined network function virtualization: A survey, IEEE Access 3 (2015) 2542–2553.

[10] Y. Zhang, L. Cui, W. Wang, Y. Zhang, A survey on software defined networking with multiple controllers, Journal of Network and Computer Applications 103 (2018) 101–118.

[11] D. Cotroneo, L. De Simone, R. Natella, Nfv-bench: A dependability benchmark for network function virtualization systems, IEEE Transactions on Network and Service Management 14 (4) (2017) 934–948.

[12] R. Amin, M. Reisslein, N. Shah, Hybrid sdn networks: A survey of existing approaches, IEEE Communications Surveys & Tutorials 20 (4) (2018) 3259–3306.

[13] F. Bannour, S. Souihi, A. Mellouk, Distributed sdn control: Survey, taxonomy, and challenges, IEEE Communications Surveys & Tutorials 20 (1) (2018) 333–354.

[14] K. Xie, X. Huang, S. Hao, M. Ma, P. Zhang, D. Hu, E^3 mc: Improving energy efficiency via elastic multi-controller sdn in data center networks, IEEE Access 4 (2016) 6780–6791. doi:10.1109/ACCESS.2016.2617871.

[15] C. Pham, N. H. Tran, S. Ren, W. Saad, C. S. Hong, Traffic-aware and energy-efficient vnf placement for service chaining: Joint sampling and matching approach, IEEE Transactions on Services Computing.

[16] A. Montazerolghaem, M. H. Yaghmaee, A. Leon-Garcia, M. Naghibzadeh, F. Tashtarian, A load-balanced call admission controller for ims cloud computing, IEEE Transactions on Network and Service Management 13 (4) (2016) 806–822.

[17] S. H. Low, A duality model of tcp and queue management algorithms, IEEE/ACM Transactions on Networking (ToN) 11 (4) (2003) 525–536.

[18] M. Chiang, S. H. Low, A. R. Calderbank, J. C. Doyle, Layering as optimization decomposition: A mathematical theory of network architectures, Proceedings of the IEEE 95 (1) (2007) 255–312.

[19] A. Montazerolghaem, S. Shekofteh, M. Yaghmaee, M. Naghibzadeh, et al., A load scheduler for sip proxy servers: design, implementation and evaluation of a history weighted window approach, International Journal of Communication Systems 30 (3).

[20] H. Jiang, A. Iyengar, E. Nahum, W. Segmuller, A. N. Tantawi, C. P. Wright, Design, implementation, and performance of a load balancer for sip server clusters, IEEE/ACM transactions on networking 20 (4) (2012) 1190–1202.

[21] K. Singh, H. Schulzrinne, Failover, load sharing and server architecture in sip telephony, Computer Communications 30 (5) (2007) 927–942.

[22] J. Wang, J. Liao, T. Li, J. Wang, J. Wang, Q. Qi, Probe-based end-to-end overload control for networks of sip servers, Journal of Network and Computer Applications 41 (2014) 114–125.

[23] R. G. Garroppo, S. Giordano, S. Niccolini, S. Spagna, A prediction-based overload control algorithm for sip servers, IEEE transactions on network and service management 8 (1) (2011) 39–51.

[24] Y. Hong, C. Huang, J. Yan, Modeling and simulation of sip tandem server with finite buffer, ACM Transactions on Modeling and Computer Simulation (TOMACS) 21 (2) (2011) 11.

[25] Y. Hong, C. Huang, Yan, Mitigating sip overload using a control-theoretic approach, in: Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, IEEE, 2010, pp. 1–5.

[26] C. Shen, H. Schulzrinne, On tcp-based sip server overload control, in: Principles, Systems and Applications of IP Telecommunications, ACM, 2010, pp. 71–83.

[27] J. Liao, J. Wang, T. Li, J. Wang, J. Wang, X. Zhu, A distributed end-to-end overload control mechanism for networks of sip servers, Computer Networks 56 (12) (2012) 2847–2868.

[28] G. Mishra, S. Dharmaraja, S. Kar, Reducing session establishment delay using timed out packets in sip signaling network, International Journal of Communication Systems 29 (2) (2016) 262–276.

[29] C. Shen, H. Schulzrinne, E. Nahum, Session initiation protocol (sip) server overload control: Design and evaluation, in: Principles, systems and applications of IP telecommunications. Services and security for next generation networks, Springer, 2008, pp. 149–173.

[30] M. Homayouni, H. Nemati, V. Azhari, A. Akbari, Controlling overload in sip proxies: An adaptive window based approach using no explicit feedback, in: Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE, IEEE, 2010, pp. 1–5.

[31] S. V. Azhari, M. Homayouni, H. Nemati, J. Enayatizadeh, A. Akbari, Overload control in sip networks using no explicit feedback: A window based approach, Computer Communications 35 (12) (2012) 1472–1483.

[32] A. Abdelal, W. Matragi, Signal-based overload control for sip servers, in: Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, IEEE, 2010, pp. 1–7.

[33] E. Noel, C. R. Johnson, Novel overload controls for sip networks, in: Teletraffic Congress, 2009. ITC 21 2009. 21st International, IEEE, 2009, pp. 1–8.

[34] A. Montazerolghaem, M. H. Y. Moghaddam, F. Tashtarian, Overload control in sip networks: A heuristic approach based on mathematical optimization, in: Global Communications Conference (GLOBECOM), 2015 IEEE, IEEE, 2015, pp. 1–6.

[35] M. Brandenburg, Growth opportunities in the voip access and sip trunking services market (2017).
URL `http://www.frost.com/`

[36] A. Rakity, Ovum telecom research (2018).
URL `https://ovum.informa.com/`

[37] A. Montazerolghaem, M. H. Y. Moghaddam, A. Leon-Garcia, Opensip: Toward software-defined sip networking, IEEE Transactions on Network and Service Management 15 (1) (2018) 184–199.

[38] M. Grant, S. Boyd, Y. Ye, Cvx: Matlab software for disciplined convex programming (2008).