*designs*

*Article*

# Session Initiation Protocol Proxy in a Role of a Quality of Service Control Application in Software-Defined Networks

**Dalibor Zeman, Filip Rezac, Miroslav Voznak *, Jan Rozhon**

Department of Telecommunications, Faculty of Electrical Engineering and Computer Science, VSB—Technical University of Ostrava, 17. listopadu 2172/15, 708 33 Ostrava-Poruba, Czech Republic
* Correspondence: miroslav.voznak@vsb.cz; Tel.: +420-596-995-940

**Abstract:** This article deals with quality of service (QoS) in internet protocol (IP) telephony by applying software-defined networking (SDN) tools. The authors develop a new design that deterministically classifies real-time protocol (RTP) streams based on data found in session initiation protocol (SIP) using SIP proxy as a mediator, and the concept making this possible is called SDN. Compared to traditional networks, SDN allows us to approach network configuration differently. SDN networks are programmable through software applications running on top of the SDN controller. One of the technologies that might benefit from this concept is IP telephony, which often needs an additional priority management configuration to ensure consistent quality of its real-time media exchange. Typically, a session protocol for real-time communications is SIP, and as such, its infrastructure may be used to classify the traffic in question and take advantage of the centralized approach of SDN networks to distribute the class information across the switching devices. Different approaches and possible applications are discussed in the conclusion. The contribution of this paper lies in the proposal of SDN-based QoS mechanisms. The entire design of the concept was implemented and validated in a laboratory environment. The results clearly demonstrate the efficiency of the proposed approach.

**Keywords:** SDN; SIP; VoIP; QoS; OpenFlow; traffic classification

check for updates

**Citation:** Zeman, D.; Rezac, F.; Voznak, M.; Rozhon, J. Session Initiation Protocol Proxy in a Role of a Quality of Service Control Application in Software-Defined Networks. *Designs* **2022**, *6*, 123. https://doi.org/10.3390/designs6060123

Academic Editors: Thierry A. Meynard and Jaime W. Zapata

Received: 29 October 2022
Accepted: 30 November 2022
Published: 4 December 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Copyright:** © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## 1. Introduction

In current internet protocol (IP) networks, various types of traffic share the same network resources, and even though the networks' performance has advanced considerably, so has the amount of data and especially real-time media that require particular attention to their qualitative parameters. Those typically include latency, jitter, and packet loss. To maintain the quality of real-time services and, more importantly, to preserve the user experience, these undesired elements must be prevented from exceeding certain levels. In relation to the network, the term quality of service (QoS) is used.

Quality of service is a concept that stands behind some network devices and tools whose functions may be divided into two basic components:

- Local operations—the application of specific QoS tools on each network device such as a router or a switch.
- Source signalization—packet identification making it possible for each network device to deterministically and consistently decide which local operations should be performed.

Two main QoS standards are integrated services (IntServ) described in RFC1663 [1] and differentiated services (DiffServ) described in RFC2475 [2]. IntServ has been developed as a protocol for resource reservation on an end-to-end basis. However, it has never been widely used, although some of its characteristics have been inherited in multiprotocol label switching (MPLS) networks by the resource reservation protocol (RSVP).

The DiffServ model, on the contrary, is intended to differentiate between a number of predetermined traffic classes. Each type of traffic is assigned to a class which is linked to a per-hop behavior (PHB) on each network device. There is no signalization present between neighbors or end points, and PHB implies that the behavior is determined by each device's local configuration. Although this makes DiffServ much more scalable than IntServ, the information available on the data sessions and packets is limited. Upon classification, network devices can forward class information to the headers of a packet, but the success lies in the consistency of the configuration of each network device on the particular data path [3].

Even DiffServ, though, finds its challenges in today's networks. When speaking of QoS, the crucial step is to differentiate and classify different types of traffic. The typical classification method is port-based classification, which expects the packets to use commonly known ports for each type of service [4]. However, the growing use of dynamic ports and tunneling compromises this method. Another option is to inspect the payload and determine the class of service accordingly, and while it may not even be possible in some implementations due to encryption, it raises security concerns as well [5–7].

One of the traffic types that is typically desired to be handled differently is voice traffic typically supported by the session initation protocol (SIP) that exchanges the routing information between the end devices, in other words, signalization. Given that this information can be effectively forwarded to network devices, it should be possible to dynamically prioritize specific media flows without further knowledge of their parameters.

For this purpose, the concept of SDN appears to be attractive [8]. It is a relatively new concept that, in contrast to traditional IP networks, separates the control plane from network devices. The control plane is then represented by an entity called an SDN controller, and the data plane is represented by multiple SDN switches. An SDN controller contains all the network logic and accordingly feeds SDN switches, which are on their own very simple devices, with instructions that are stored in the form of tables. A typical example, as a protocol in charge of communication between an SDN controller and SDN switches, is OpenFlow. Although there are newer versions, the most common version that manufacturers implement in their equipment is 1.3 [9]. As far as performance goes, SDN switches have been proven to be comparable to traditional ones [10,11].

A fundamental characteristic of SDN, considered its main value proposition, is the ability to interact with applications through its application programmable interface (API) [12]. In the position of such an application would be an SIP server which is able to select the desired information from SIP messages and forward it to the SDN controller.

In this paper, we demonstrate an example of ensuring QoS policies achieved by such cooperation of SIP and SDN infrastructures, highlighting the potential of the design. Classification of real-time protocol (RTP) streams will be controlled by SIP signalization of the respective VoIP calls and will therefore not depend on standard classifiers that would in some cases be found to be unreliable. Moreover, the configuration will be performed dynamically and in a centralized manner, which are features that have been missing in traditional networks [13].

## 2. Related Work

Some research has been conducted on SDN and QoS synergy, for example, focusing on QoS techniques that decide which route traffic should take [14,15] or related topics such as speech quality estimation [16]. These works generally take advantage of the SDN controller's ability to oversee the number of SDN switches and the traffic going through them. In [15], Akella and Xiong tested a method that guarantees bandwidth allocation between two points in the network using predefined values. The solution presented in this paper would be able to give such methods dynamic control by taking advantage of SIP that contains not only the routing information of the end points, but also the exact bandwidth needed for the particular session as well as a respective point in time when it is safe to release the reserved resources.

On the subject of traffic classification, some studies propose the use of machine learning algorithms. Nevertheless, there is a trade-off in both setup time and accuracy. It is worth noting that these algorithms were tested for use in smart city networks which are found to be problematic in terms of traffic classification [7].

There was an attempt to use SIP for QoS management in [17]. The authors suggested that SDN switches would listen on port 5060, which is a standard port for the SIP protocol, and forward duplicates of the corresponding traffic to an SDN controller. Then, the SDN controller analyzed the SIP packets and found the desired information. This approach will accomplish the goal, but in a broader picture, it would be rather problematic.

Firstly, unnecessary redundancy is created when duplication occurs for every whole SIP packet. Secondly, if encryption is used, the SDN controller is not able to read the contents. Finally, unless the SDN controller contains the entire logic of an SIP server, the authentication requirements are not met. An analytical experiment was performed in [18], inspecting RTP packets instead of SIP. Further analysis on this matter will be included in the Section 5. Therefore, we propose leaving the logic of QoS parameters provision on an SIP server. SIP servers already perform most of the processes needed for such an application and should be achieved with minimal drawbacks.

The most similar approach was developed in [19], which suggested programming the entire SIP server logic into an SDN controller. This unquestionably solves the drawbacks mentioned above, but, in terms of design, we believe that these entities should remain separate and that SDN controllers should not be incorporated into the infrastructures of other applications. An SDN application that exchanges information with an SIP server was presented in [20]; however, its primary purpose is a call admission control.

## 3. Technologies and Concept Design

The main components of this concept are an SIP server, an SDN controller, and SDN switches. We decided on the following implementations:

- SIP proxy Kamailio;
- SDN controller Ryu;
- Open vSwitch (OvS) SDN switch

One of the features of an SDN network is an option to communicate with external applications through a central control element, an SDN controller. With each SIP dialog, the SIP server should inform an SDN controller about the active voice streams represented by RTP sessions aligned with SIP messages as described in RFC3261 [21]. The system should perform the following operations:

1. Upon successful assembly of a voice session, the SIP server forwards network parameters of its RTP session to the SDN controller. These parameters are contained in requests INVITE and responses 200 OK.
2. An SDN controller ensures that the configuration prioritizing given RTP streams is distributed across the SDN switches.
3. Upon terminating the voice session, that is, upon receiving a request BYE, the SIP server informs the SDN controller that discards given configuration across the SDN switches.

The SDN controller Ryu is remotely programmable via its REST interface. This requires sending hypertext transfer protocol (HTTP) messages of the proper method and syntax accordingly to its documentation described in [22]. Based on these presumptions, it may be deduced that Kamailio needs to be able to extract information from an session description protocol (SDP) body, save the information for later use, and send it via an HTTP configuration message to the SDN controller. Such functions are feasible with Kamailio's modules:

- Sdpops to access an SDP body [23];
- Dialog to keep variables accessible within the entire SIP dialog (by default, Kamailio keeps variables alive only for the duration of the transaction) [24];

- Http_client and http_async_client for sending HTTP requests [25,26].

With these modules available, Kamailio was configured to extract and save an IP address and port values from the media parameter of an SDP body of requests INVITE and responses 200 OK. After processing a response 200 OK this way, these values are sent within an HTTP message POST to the SDN controller, aligning given RTP streams with the preconfigured priority queues on the SDN switches.

In the response, the SDN controller includes an identifier, which Kamailio saves as a dialog variable. This identifier is used at the end of a session when one of the user agents sends out a request BYE which terminates the call. Kamailio then sends an HTTP message DELETE to the SDN controller, discarding the configuration associated with a particular session.

There are many QoS tools that may be used. Typically, it is DiffServ due to its scalability and then some local mechanisms dealing with various traffic classes, such as queues. For purposes of validation, we used simple priority queues, but that part is interchangeable with any other standard method according to the character of the particular network. DiffServ may be used in larger networks as well. However, compared to the best-effort approach, the quality of calls should be preserved under congestion even with no other traffic classifier available.

An SDN network was emulated in the Mininet network emulator running on one Linux system altogether with an SIP server Kamailio and an SDN controller Ryu. SIP user agents were configured separately on other devices connected to the Linux system that served them as a transition network. Aside from the proposed functionalities and basic connection, this topology illustrated in Figure 1 allowed us to simulate a congestion, capture transitioning packets, and analyze them in terms of QoS parameters. The topology is explained further in Sections 4.1 and 4.2; the first focuses on establishing the connection and subsequent dynamic configuration and the second on RTP flows and QoS measurements.

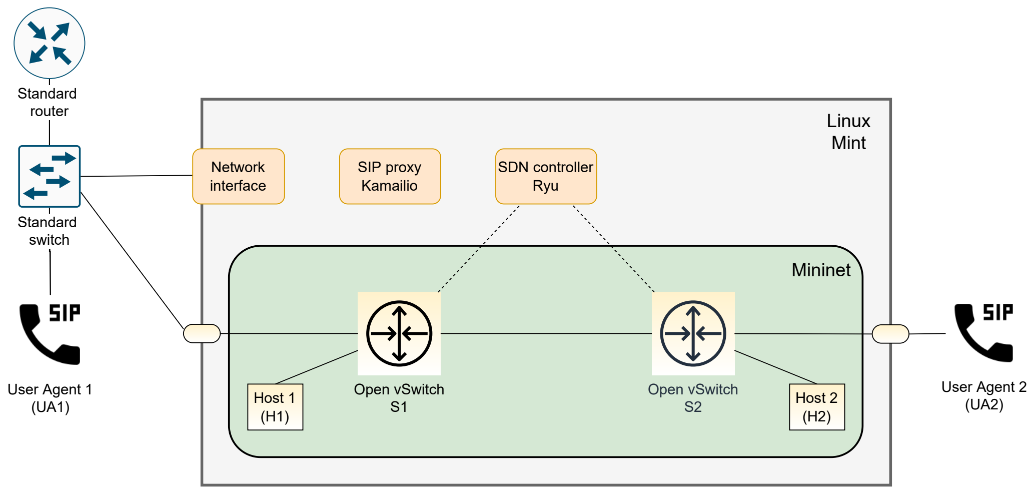

**Figure 1.** Physical topology.

To specify the components used:

- Mininet configured with OvS SDN switches (OpenFlow v1.3);
- OS Linux Mint; i7-1165G7 @ 2.80 GHz; 16 GB RAM;
- 2 standard SIP softphones;
- Ryu controller with OpenFlow v1.3;
- SIP proxy Kamailio v5.4.

## 4. Results

### *4.1. Core Process*

It may be claimed that the QoS configuration was integrated into the SIP dialog, which is captured and shown in Figure 2 as a flowchart generated in Wireshark. Upon successful establishment of a call, Kamailio sends two HTTP requests POST, one for each user agent, to the SDN controller that prioritizes its RTP streams. When the call is terminated, upon receiving a request BYE, the prioritization is revoked in a similar manner. The SDN controller Ryu responds to these requests with HTTP responses of the respective code (200 OK in this case) that may contain metadata regarding respective configuration items.

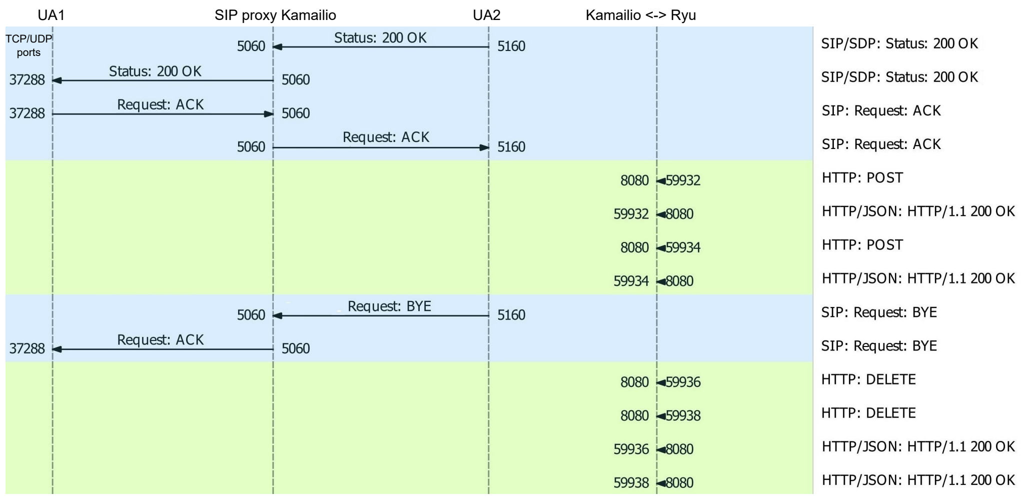

**Figure 2.** Session initation protocol (SIP) flowchart in WireShark.

The following are examples of HTTP configuration messages POST and DELETE and the respective HTTP responses 200 OK. Since there are only two switches in this case, the most simple prioritization technique was chosen. Thus, there is only one configuration item containing an IP address and a port extracted from the SIP dialog as described. The value in a parameter *queue* assigns traffic to the statically preconfigured priority queue on each of the switches.

```
POST /qos/rules/all HTTP/1.1
Host: localhost:8080
User-Agent: kamailio (5.5.0-dev4 (x86_64/linux))
Accept: */*
Content-Length: 103
Content-Type: application/x-www-form-urlencoded
{"match": {"nw_dst": "192.168.50.159", "nw_proto": "UDP",
    "tp_dst": "48004"}, "actions":{"queue": "1"}}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 228
Date: Wed, 21 Apr 2021 16:46:57 GMT
[{"switch_id": "0000000000000002", "command_result":
    [{"result": "success", "details": "QoS added. : qos_id=5"}]},
{"switch_id": "0000000000000001", "command_result":
    [{"result": "success", "details": "QoS added. : qos_id=5"}]}]
```

To release the rules that become obsolete when a call is terminated, HTTP message DELETE is sent to the controller. The value in a parameter qos_id saved from the previous answer is an identifier that allows us to access specific configuration items and in this case also associates respective configuration items with an SIP call.

```
DELETE /qos/rules/all HTTP/1.1
Host: localhost:8080
Accept: */*
testHeader: header
Content-Length: 15
Content-Type: application/x-www-form-urlencoded
{"qos_id": "5"}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 226
Date: Wed, 21 Apr 2021 16:47:31 GMT
[{"switch_id": "0000000000000002", "command_result":
    [{"result": "success", "details": " deleted. : QoS ID=5"}]},
{"switch_id": "0000000000000001", "command_result":
    [{"result": "success", "details": " deleted. : QoS ID=5"}]}]
```

### *4.2. Experiment Validation*

The topology in Figure 3 shows the transition SDN network with virtual hosts H1 and H2. Additional load traffic was inserted between these hosts with a network tool iPerf, sharing the same network resources with a testing call between user agents UA1 and UA2. Assuming that the load traffic's bandwidth consumption is high enough, this should create considerable congestion on the shared link between switches, degrading the quality of the testing call unless some QoS technique differentiating and prioritizing VoIP traffic over the other non-priority traffic is used.

Since the SDN switches are emulated on a Linux station, any traffic passing through any of the interfaces can be captured with Tcpdump and, if needed, precisely calculate other parameters, such as latency or jitter, accumulated at a particular interface.

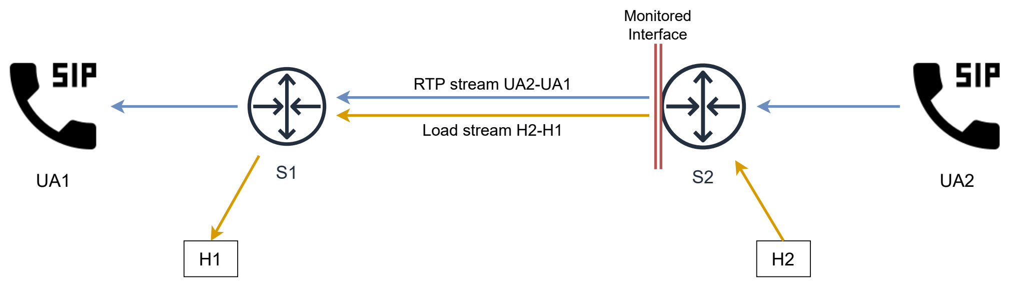

**Figure 3.** Congestion simulation and traffic capture for QoS analysis.

The initiated call was forced to share the network resources with the other traffic, and since the sum of the ingress traffic was greater than the outbound interface's capacity, the resources were distributed according to the configured queue policies. In Figure 4, where the best-effort approach is used, both streams are scaled down equally, suspending the excess traffic into one fair queue.

Now, the main point comes with a crucial twist. Figure 5 shows the same scenario with an RTP stream and load traffic; only the configuration of the SIP server Kamailio was replaced by the proposed one. The result is that the RTP stream transmission is clearly not affected by other non-priority traffic during congestion. In the previous case, the default configuration was used.

This represents the essence of QoS policies in terms of traffic prioritization. One traffic is locally prioritized over the other, limiting its access to the medium less at the expense of limiting the other traffic more. QoS parameters, such as latency, jitter, and packet loss, are then distributed proportional to their importance that is represented by traffic classes. In real networks, it is typically some combination of the presented scenario multiplied across the network.

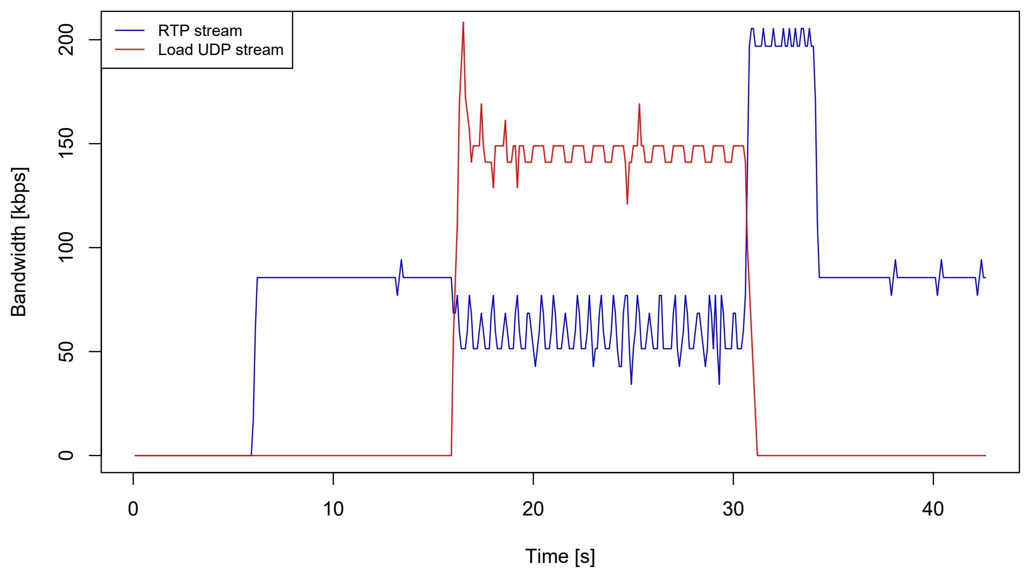

**Figure 4.** Bandwidth distribution with Kamailio in default configuration.

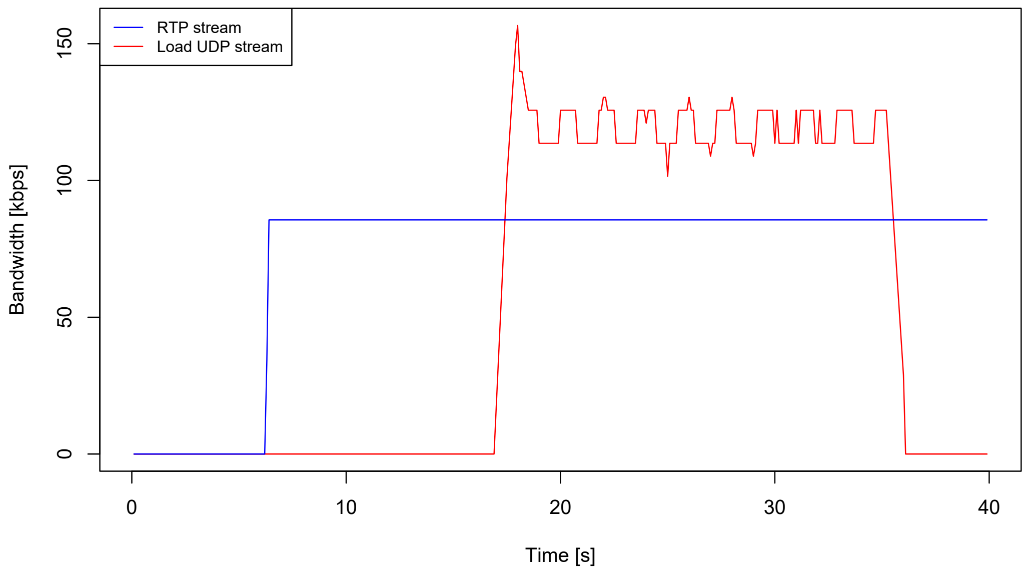

**Figure 5.** Bandwidth distribution with SIP proxy in the proposed configuration.

To clearly demonstrate the impact on QoS parameters, delay and jitter may be calculated. Although it is usually not a trivial matter, it should be assumed that the delay of the first packet is negligible. The expected time of arrival of every packet in the stream may be derived according to the packet rate. For the codec G.711, which is used here, it is 50 packets per second [pps]. The full equation used for the values exported from the captured packets is as follows:

$$t_k = \sum_{n=k_0}^{k} \Delta_n - \frac{1000}{r} \times (k - k_0)[\mathrm{ms}]$$

where $t_k$ = packet delay, $k$ = sequential number of the packet, $k_0$ = sequential number of the first packet in the RTP stream, $\Delta$ = difference in arrival times of two consecutive packets in the RTP stream, and $r$ = packet rate.

Similar to the bandwidth distribution, we have two graphs that compare the delay and jitter of the captured RTP stream before (Figure 6) and after (Figure 7) applying the described configuration of the SIP server. When load traffic is launched, the measured interface is congested, and this congestion manifests itself by putting excess traffic in a buffer. The longer and more robust the congestion, the farther in the buffer the packets are placed and the greater the delay. This constant growth of delay during, in this case, a stable congestion is a deviation that is then manifested in the jitter curve. The jitter and the increase in delay remain stable during congestion.

When the voice traffic is prioritized and its bandwidth is not limited even when congestion occurs, only non-priority traffic is saved into the buffer. Therefore, unlike the visible increase in Figure 6, delay and jitter are essentially negligible in Figure 7, staying relatively stable for the entire duration of the measurement.

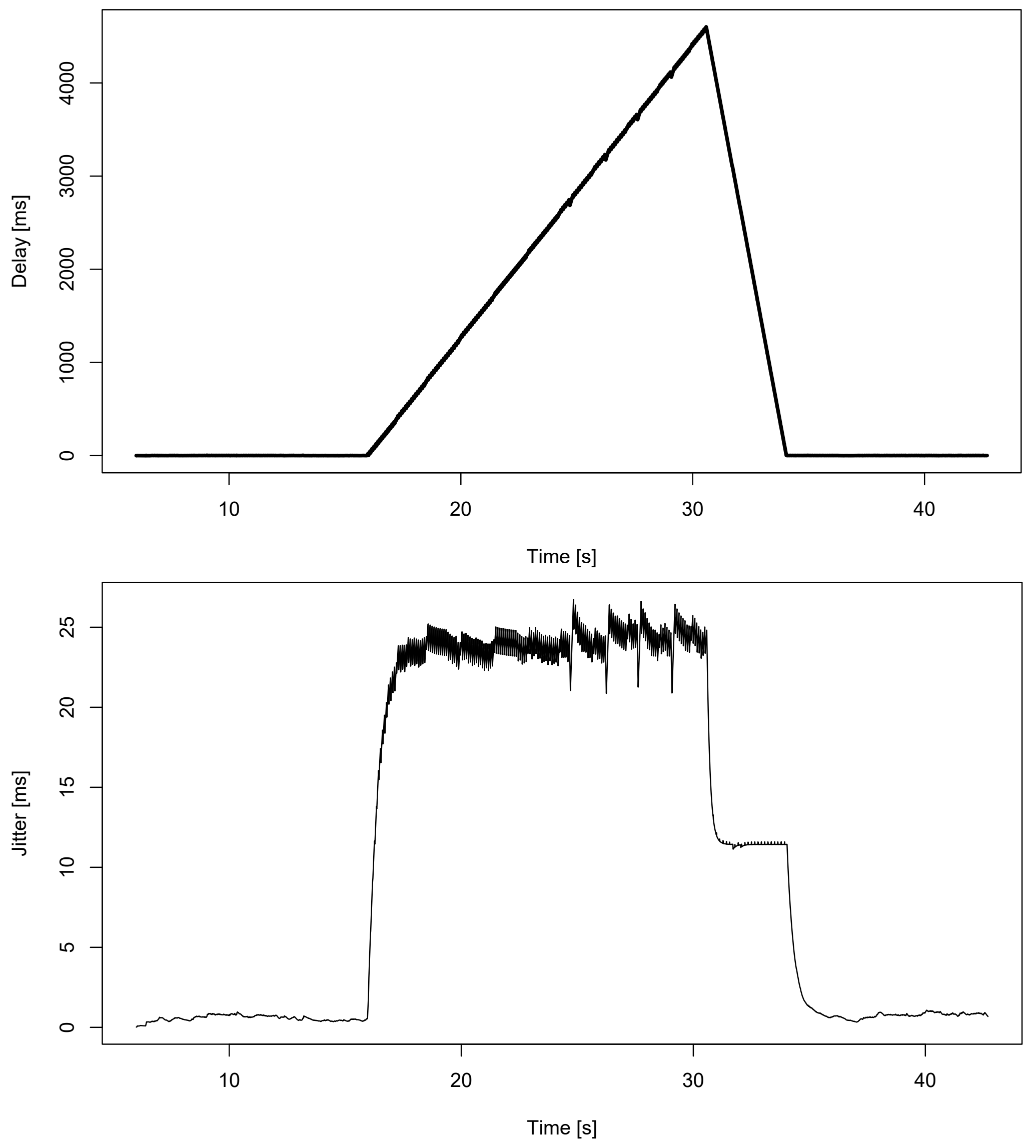

**Figure 6.** RTP delay jitter on the congested interface with the best-effort approach.

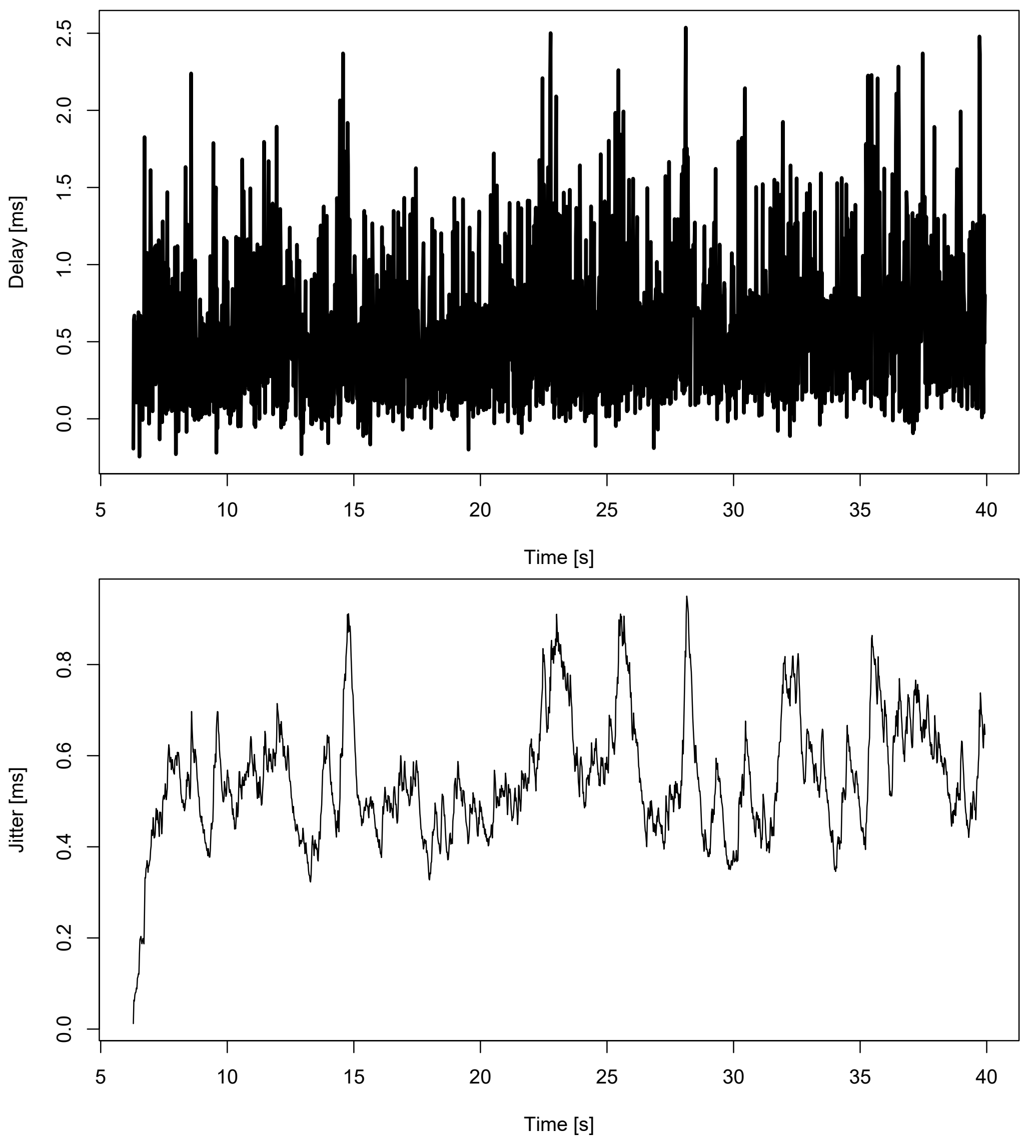

**Figure 7.** RTP delay and jitter on the congested interface with QoS policies applied.

## 5. Comparison

While the impact of the selected QoS tools, which validates their operability, is clearly visible, there is no precedent for deterministically comparing the effectiveness of the classification methods used. If multiple classification methods could be replicated and run in the same environment, it would be possible to perform stress testing and compare the results. The key differences, though, lie in their ability to successfully operate under different conditions, and, as such, they may be directly derived from some of the methods' principles.

As outlined in the Introduction and Related Work, classification methods could be distinguished based on the source of their differentiating information. In relation to recent studies, several approaches that differ substantially in their principles have been chosen, followed by Table 1, comparing how their differences reflect on potential behavior.

To briefly summarize each of the methods by their classification acquisition process:

- Transport inspection—the packets are inspected for an indicator, such as a port or a transport protocol;
- SIP on an SIP server—as we propose, the classification information is extracted directly from an SIP session. The SIP server is configured to forward classification information to an SDN controller separately;

- SIP on an SDN controller—at one or more points of the network, SIP messages are redirected to an SDN controller that performs its own analysis on them;
- Machine learning—based on statistical analysis of the network traffic. Generally, it is very versatile, but it entails high computational difficulty. Various algorithms are available, and their performance might differ. Usually, their execution ranges in units of seconds.

**Table 1.** Comparison of classification methods.

| Classification Source | Redundancy | Works with Encryption [1] | Accuracy | Execution Time |
|---|---|---|---|---|
| Transport inspection | None | No | Limited | Fast [ms] |
| SIP on an SIP server | Low | Yes | Strong | Fast [ms] |
| SIP on an SDN controller [17] | Moderate | No | Strong with limitations | Fast [ms] |
| Machine learning [7] | None | Yes | Moderate | Slow [s] |

[1] Regards the data unit that is inspected.

## 6. Conclusions

This study is not focused on improving the performance of QoS policy tools. Instead, it brings a new approach that can be used to identify desirable VoIP traffic before applying well-known QoS techniques, independent of the form of the RTP headers. The configuration is performed dynamically and in a centralized manner. In addition to the advantages provided by SDN, it was confirmed that it is possible to take advantage of the SIP infrastructure as an alternative to standard traffic classification methods, which come with certain limitations.

Furthermore, taking into account some of the recent studies revolving around the SDN and VoIP concepts, there is a potential for further development. Using the same concept, an SDN network can utilize various session parameters that usually remain unknown to it and use them to adjust QoS policies or perform other network operations on a similar basis. Some of the examples using SDN also include routing path selection and bandwidth allocation.

In future research, we could experiment with the scalability and deployment of the proposed concept in large-scale networks. Moreover, it seems convenient to experiment with the combination of this and the other concepts mentioned in the Related Work. We plan to explore newer SDN technologies, such as P4 and ONOS controller, which promise to cover the shortcomings that have been holding OpenFlow back. Furthermore, there is a demand for SDN-controlled QoS mechanisms in 5G mobile networks, the most notable of which is network slicing, as quite a few recent studies imply. We would like to incorporate the knowledge acquired in this study into one of these directions after sufficient theoretical research is carried out.

**Author Contributions:** Conceptualization, D.Z.; methodology, D.Z., F.R., and J.R.; validation, D.Z. and J.R. writing—original draft preparation, D.Z. and F.R.; writing—review and editing, M.V.; All authors have read and agreed to the published version of the manuscript.

**Funding:** The research leading to these results was supported by the Czech Ministry of Education, Youth and Sports within project reg. no. SP2022/5 and also partially within the Large Infrastructures for Research, Experimental Development and Innovations project "e-Infrastructure CZ" reg. no. LM2018140, both projects were conducted by VSB-Technical university of Ostrava.

**Data Availability Statement:** The data presented in this study are openly available in FigShare at https://doi.org/10.6084/m9.figshare.21667145.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| API | Application Programmable Interface |
| DiffServ | Differentiated Services |
| HTTP | Hypertext Transfer Protocol |
| IntServ | Integrated Services |
| IP | Internet Protocol |
| MPLS | Multiprotocol Label Switching |
| ONOS | Open Network Operating System |
| OvS | Open vSwitch |
| PHB | Per-Hop Behavior |
| QoS | Quality of Service |
| RSVP | Resource Reservation Protocol |
| RTP | Real-time Transport Protocol |
| SDN | Software-Defined Networks |
| SDP | Session Description Protocol |
| SIP | Session Initation Protocol |

## References

1. Rand, D. *RFC 1663: PPP Reliable Transmission*; RFC Editor: Marina del Rey, CA, USA, 1994.
2. Blake, S.; Black, D.L.; Carlson, M.A.; Davies, E.; Wang, Z.; Weiss, W. An Architecture for Differentiated Services. RFC 2475, RFC Editor, 1998. Available online: http://www.rfc-editor.org/rfc/rfc2475.txt (accessed on 31 March 2022).
3. Barreiros, M.; Lundqvist, P. *QOS-Enabled Networks: Tools and Foundations*, 2nd ed.; John Wiley & Sons: Hoboken, NJ, USA, 2016.
4. Nguyen, T.T.; Armitage, G. A survey of techniques for internet traffic classification using machine learning. *IEEE Commun. Surv. Tutor.* **2008**, *10*, 56–76. [CrossRef]
5. Li, J.; Zhang, S.; Lu, Y.; Yan, J. Real-Time P2P Traffic Identification. In Proceedings of the IEEE GLOBECOM 2008—2008 IEEE Global Telecommunications Conference, New Orleans, LA, USA, 30 November 2008–4 December 2008; pp. 1–5. [CrossRef]
6. Pacheco, F.; Exposito, E.; Gineste, M.; Baudoin, C.; Aguilar, J. Towards the Deployment of Machine Learning Solutions in Network Traffic Classification: A Systematic Survey. *IEEE Commun. Surv. Tutor.* **2019**, *21*, 1988–2014. [CrossRef]
7. AlZoman, R.M.; Alenazi, M.J.F. A Comparative Study of Traffic Classification Techniques for Smart City Networks. *Sensors* **2021**, *21*, 4677. [CrossRef] [PubMed]
8. Karakus, M.; Durresi, A. Quality of Service (QoS) in Software Defined Networking (SDN): A survey. *J. Netw. Comput. Appl.* **2017**, *80*, 200–218. [CrossRef]
9. OpenFlow Switch Specification. 2012. Available online: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf (accessed on 31 March 2022).
10. Hegr, T.; Voznak, M.; Kozak, M.; Bohac, L. Measurement of switching latency in high data rate Ethernet networks. *Elektron. Elektrotechnika* **2015**, *21*, 73–78. [CrossRef]
11. Hegr, T.; Bohac, L.; Kocur, Z.; Voznak, M.; Chlumsky, P. Methodology of the direct measurement of the switching latency. *Prz. Elektrotechniczny* **2013**, *89*, 59–63.
12. Kreutz, D.; Ramos, F.M.V.; Veríssimo, P.E.; Rothenberg, C.E.; Azodolmolky, S.; Uhlig, S. Software-Defined Networking: A Comprehensive Survey. *Proc. IEEE* **2015**, *103*, 14–76. [CrossRef]
13. Bannour, F.; Souihi, S.; Mellouk, A. Distributed SDN Control: Survey, Taxonomy, and Challenges. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 333–354. [CrossRef]
14. Jia, W.K.; Chou, Y.Y.; Chen, Y.C. QoS Improvement of VoIP over SDN. In Proceedings of the 2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC), Las Vegas, NV, USA, 10–13 January 2020; pp. 1–6. [CrossRef]
15. Akella, A.V.; Xiong, K. Quality of Service (QoS)-Guaranteed Network Resource Allocation via Software Defined Networking (SDN). In Proceedings of the 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing, Dalian, China, 24–27 August 2014; pp. 7–13. [CrossRef]
16. Rozhon, J.; Rezac, F.; Jalowiczor, J.; Behan, L. Augmenting Speech Quality Estimation in Software-Defined Networking Using Machine Learning Algorithms. *Sensors* **2021**, *21*, 3477. [CrossRef] [PubMed]
17. Page, J.; Hubain, C.; Dricot, J.M. Dynamic QoS on SIP Sessions Using OpenFlow. In Proceedings of the Eighth International Conference on Emerging Networks and Systems Intelligence, Venice, Italy, 9–13 October 2016; pp. 45–49.
18. Olariu, C.; Zuber, M.; Thorpe, C. Delay-based priority queueing for VoIP over Software Defined Networks. In Proceedings of the 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), Lisbon, Portugal, 8–12 May 2017; pp. 652–655. [CrossRef]
19. Gandotra, R.; Perigo, L. SDVoIP—A Software-Defined VoIP Framework For SIP and Dynamic QoS. *Comput. J.* **2020**, *64*, 254–263.

20. Zanuttini, D.; Andres, C.; Gonzalez, J.; Lacapmesure, A.; Priano, D.; Pucci, N.; Bullian, P. Using Software Defined Networking for Call Admission Control and VoIP applications. In Proceedings of the 2018 Congreso Argentino de Ciencias de la Informática y Desarrollos de Investigación (CACIDI), Buenos Aires, Argentina, 28–30 November 2018; pp. 1–5. [CrossRef]
21. Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; Johnston, A.; Peterson, J.; Sparks, R.; Handley, M.; Schooler, E. SIP: Session Initiation Protocol. RFC 3261, RFC Editor. 2002. Available online: http://www.rfc-editor.org/rfc/rfc3261.txt (accessed on 31 March 2022).
22. QoS—Ryubook 1.0 Documentation. Available online: https://osrg.github.io/ryu-book/en/html/rest_qos.html (accessed on 31 March 2022).
23. Merla, D.C. SDPOPS Module. Available online: https://kamailio.org/docs/modules/4.1.x/modules/sdpops.html (accessed on 31 March 2022).
24. Iancu, B.A. Dialog Module. Available online: https://kamailio.org/docs/modules/5.1.x/modules/dialog.html (accessed on 31 March 2022).
25. Johansson, O.E. HTTP_Client Module. Available online: https://www.kamailio.org/docs/modules/devel/modules/http_client.html (accessed on 31 March 2022).
26. Cabiddu, F. HTTP_Async_Client Module. Available online: https://www.kamailio.org/docs/modules/devel/modules/http_async_client.html (accessed on 31 March 2022).