Optimization of Low-efficiency Traffic in OpenFlow Software Defined Networks

Jose Saldana[1], Fernando Pascual[2], David de Hoz[1], Julián Fernández-Navajas[1], José Ruiz-Mas[1], Diego R. Lopez[2], David Florez[2], Juan A. Castell[2], Manuel Nuñez[2]

[1]I3A, University of Zaragoza
Ada Byron Building, 50018, Zaragoza, Spain
e-mail: {jsaldana, dhoz, navajas, jruiz}@unizar.es

[2]Telefonica I+D
Don Ramon de la Cruz 82-84, 28006 Madrid, Spain
e-mail: {fpb, diego, dflorez, jacl, mns}@tid.es

***Abstract*— This paper proposes a method for optimizing bandwidth usage in Software Defined Networks (SDNs) based on OpenFlow. Flows of small packets presenting a high overhead, as the ones generated by emerging services, can be identified by the SDN controller, in order to remove header fields that are common to any packet in the flow, only during their way through the SDN. At the same time, several packets can be multiplexed together in the same frame, thus reducing the number of sent frames. Four kinds of small-packet traffic flows are considered (VoIP, UDP and TCP-based online games, and ACKs from TCP flows). Both IPv4 and IPv6 are tested, and significant bandwidth savings (up to 68 % for IPv4 and 78 % for IPv6) can be obtained for the considered kinds of traffic.**

Keywords— Software Defined Networks, multiplexing, traffic optimization, compression

I. INTRODUCTION

Software Defined Networks (SDNs) are a new approach to networking, based on the radical separation of the control and data planes, connected by open interfaces, and including the direct programmability of the control plane. This allows for a logically centralized control of the network as a whole, bringing the possibility of dealing with the network as a single and programmable entity. This is especially interesting in current highly virtualized environments, most notably in cloud computing, because it allows managing network resources in a much more flexible and efficient way, making the network able to provide a QoS level adequate to the nature of each flow. OpenFlow [1] is the most extended and consolidated standard for SDNs.

At the same time, emerging real-time services (e.g. VoIP, online gaming) are becoming more and more popular on the Internet. Their interactivity requirements make them send high rates of small packets (average payloads of tens of bytes). In addition, services using TCP also generate large amounts of ACK packets without payload. For example, up to 56 % of the sent packets are ACKs for certain online games [2]. This also happens when a file is downloaded, and a flow of ACKs is sent to the origin of the communication. As an example, a 3 Mbps file download using packets of 1,500 bytes, may generate 125 ACKs per second, using the typical TCP parameters (e.g. an ACK sent every 2 downloaded packets).

These high rates of tiny packets translate into an inefficient usage of network resources, so there is a need for mechanisms able to reduce the network overhead introduced by these low efficiency flows. Bandwidth savings and packet per second reductions are interesting for network operators, since they may alleviate the traffic load in their networks.

Header compression techniques able to save bandwidth for long-term flows using small packets through the public Internet were developed long ago [3]. They are based on the fact that many header fields are the same for every packet in a flow (NOCHANGE fields). They also reduce the number of bits of increasing fields (e.g. sequence numbers), by sending the difference with the previous value (DELTA). They need to define a *context*, i.e. a set of variables synchronized between the sender and the receiver; and a *context identifier* has to be added to every packet. The desynchronization between sender and receiver may result in a burst of erroneous packets.

The compression and decompression of the headers implies additional processing in the nodes. In addition, the most recent header compression techniques [4] provide a more robust synchronization between the sender and the receiver, including a set of advanced features that imply a higher computation cost [3].

Furthermore, header compression presents another limitation: compressed packets can only traverse a single Layer-3 hop, since they do not include a standard header. One solution is to compress and decompress them at each intermediate node. Another option is to use an end-to-end tunnel, so as to avoid the additional processing caused by compression and decompression, but in this case the tunneling overhead cancels the savings obtained by header compression.

A solution proposed in [5] is to jointly use multiplexing, header compression and tunneling. Thus, a number of header-compressed packets belonging to different flows can be multiplexed together in the same frame, in order to share the tunnel overhead, which becomes relatively smaller as the

This work has been partially financed by Project TAMA, Government of Aragon; European Social Fund; Project Catedra Telefonica, University of Zaragoza.

number of packets multiplexed in the same frame grows. This combination allows the packets to travel end-to-end through a public network while maintaining a good header reduction rate; as an additional benefit, the amount of packets per second traversing intermediate nodes is significantly reduced, by a factor equivalent to the average number of multiplexed packets.

In order to join a number of packets to be sent together, a period is defined in the device performing the multiplexing process. A multiplexed frame is released at the end of the period, so the longer the period, the higher the number of multiplexed packets and the higher the savings. However, a tradeoff appears, since this multiplexing latency has to be maintained under a threshold in order to grant the delay requirements of the service.

This reduction in the header-to-payload rate of the small-packet flows is also desirable in SDNs. In this context, the contribution of the present paper is the proposal of an optimization method for providing significant bandwidth savings in an Openflow-based SDN, with three main advantages: *i)* the tunneling layer is not necessary, since the SDN provides it in a natural way; *ii)* the avoidance of the use of standard header compression techniques [3] which require a context synchronization between the sender and the receiver; and *iii)* multiplexing reduces the number of frames, so a number of Ethernet fields (header, inter-frame gap) are only sent once instead of being required for each packet. Four kinds of small-packet traffic flows will be considered (VoIP, UDP and TCP-based online games, and ACKs from TCP flows).

The remainder of the article is as follows: the next section summarizes the related work. The proposed method is described in detail in Section III. The saving results are presented in Section IV and the paper ends with the conclusions.

II. Related Work

The combination of multiplexing and compression was first proposed in [5], by the use of Enhanced Compressed RTP [6], PPPMux [7] and L2TPv3 [8] protocols, with the aim of reducing the overhead of VoIP flows using RTP. In [9] another multiplexing method for VoIP was proposed, with the idea of maintaining a good quality level. The extension of [5] for other real-time services not based on RTP has also been proposed recently [10], taking into account that some services (e.g. certain online game genres) present a traffic profile consisting of high rates of small UDP packets [11], very similar to that of VoIP. However, an end-to-end tunnel is required to send compressed packets through the public Internet.

The use of header compression techniques within an OpenFlow SDN was first proposed in [12], with the aim of reducing overhead, and saving the compress-decompress delay at each hop. The controller would play the role of establishing the end-to-end tunnel. In order to avoid decompressing in each hop, L2 information was used in order to route the packet. By means of Openflow, packets compressed with standard techniques were still able to be correctly routed.

As remarked in [13], focused on IPv6 extension headers, Openflow 1.1 introduced the possibility of extensible matches, actions, messages and errors, thus allowing two controllers or switches to agree on different syntaxes when matching flows. Thus, different sets of fields can be selected for matching a flow. This feature is interesting, since additional fields can be included in the *tuple* that Openflow uses for defining a flow, and the value of these fields will be stored on the controller.

The effect of the required additional delay has also been explored: in real-time services it may have an influence on subjective quality. In [14] this effect was explored with VoIP; in [15] a subjective quality estimator for a UDP-based game was used; and in [16] the effect on a TCP-based game was studied. In addition, if a flow of TCP ACKs is multiplexed, the additional delay may have an impact on TCP dynamics, taking into account that this protocol is RTT-based. In [17] the effect of this delay on TCP was explored, showing the conditions in which the throughput obtained by multiplexed flows may be penalized.

The method proposed in the present paper is able to significantly reduce the overhead in an Openflow-based SDN. It multiplexes a number of packets, but it does not require the use of standard header compression techniques based on context synchronization. In contrast, it only removes the fields that are the same for every packet on the flow. Since they are stored in the controller, they can be avoided when sending a compressed packet. Finally, it does not require a tunneling protocol, since the SDN itself is able to provide it.

III. Proposed Method

In this section the method for optimizing the packets is presented. We will use the word "optimized" when referring to compressed packets. Three steps are considered, which are explained subsequently.

A. Removing header fields present in the Openflow tuple

Under Openflow 1.0 [1] all the switches in a management domain are connected to a central controller, and each packet is associated to a flow by means of a 12-field *tuple* (Fig. 1), which is used for assigning the output port at each switch consequently.

When a flow traverses a path within an OpenFlow SDN, the IP and TCP *tuple* fields of all the packets are the same for all the tables of the switches of that path, and also in the controller. Thus, the IP and TCP protocol fields already included in the *tuple* are not necessary for switching decisions but only for matching the packets with a flow. Thus, if we remove these fields and we substitute them by a *flow identifier* (FID), the packet can travel in an optimized manner within the SDN. A new value of the *protocol* field of PPP could be defined, in order to flag the packet as optimized (Fig. 2): it begins with the FID, plus the compressed IP and TCP headers (i.e. the fields not present in the *tuple*), and the payload.

In port	Ethernet	VLAN	IP	TCP
	SA, DA, type	ID, prio	SA,DA,Prot,ToS	Sport, DPort

Fig. 1. Tuple of Openflow 1.0.

This would have some similarities with MPLS or other technologies in which labels are used for identifying a flow across the network. However, the idea of a generic FID would make our solution agnostic of the underlying technology. Another advantage with respect to those technologies is that in an SDN the controller has an overall view, so the ingress and egress points of the tunnel can be dynamically defined according to traffic requirements.

The authors of [12] proposed an end-to-end header compression scheme in an SDN context. However, our proposal does not formally use header compression techniques [3], but it only removes NOCHANGE fields. The compression of DELTA fields is not considered, since it would only provide a marginal increase of the savings, at the cost of the appearance of compressed headers with a variable size, the processing required for the compression and decompression of these fields, and the potential context desynchronization.

With Openflow 1.0, the skipped fields account for 13 bytes per packet for IPv4/TCP. Considering a 3-byte FID, 10 bytes per packet can be saved, i.e. 25 % of the header, which may imply a significant bandwidth reduction for services using small packets.

B. *Removing other fields*

But Openflow 1.1 and subsequent versions also allow switches and controllers to agree on different flow matching syntaxes, in order to avoid a too rigid match structure [13]. Taking advantage of this fact, we consider, as a second step of our proposal, the inclusion in the *tuple* of other NOCHANGE fields of Transport and Network layers. Although these fields are not required for identifying the flow, including them in the *tuple* would make it possible to remove them from all the packets, thus allowing even higher header compression ratios. As a counterpart, we can expect a slight increase of the storage requirements of the ingress and egress switches, and the controller, but it would only mean 40 bytes per flow. Furthermore, fields belonging to well-known application layer protocols can also be included in the *tuple*. As an example, RTP is often used for services based on small packets (VoIP), so removing RTP fields with a constant value, may imply significant savings for these flows.

C. *Multiplexing a number of packets in a single frame*

Finally, taking advantage of its programmability, the SDN controller could be able to match groups of flows sharing a common path segment within the SDN. In this case, packets belonging to different flows could be multiplexed together and sent as a single Eth frame (Fig. 3) in all the hops of the path. This would require the use of a multiplexing protocol between the ingress and egress switches of the common path. PPPMux [7] can be used for multiplexing.

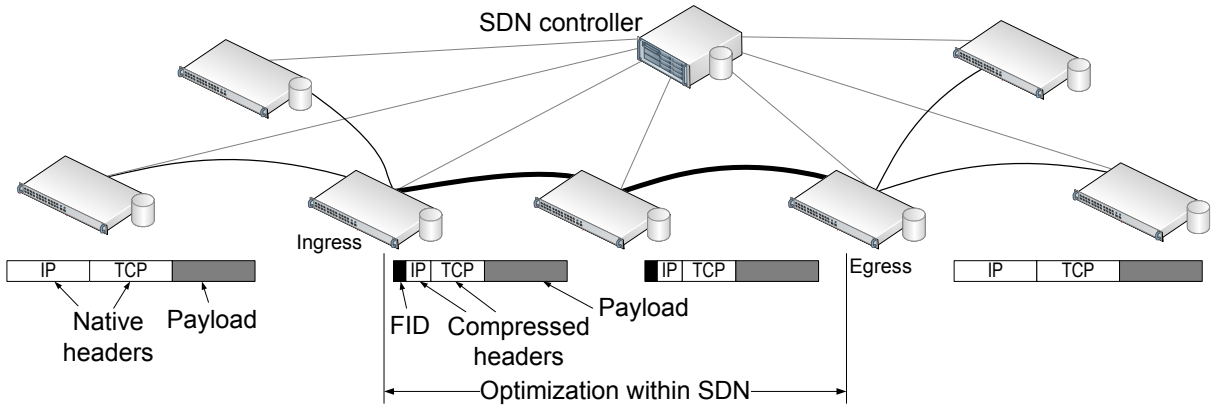


Fig. 2. Scheme of the header compression within the SDN.

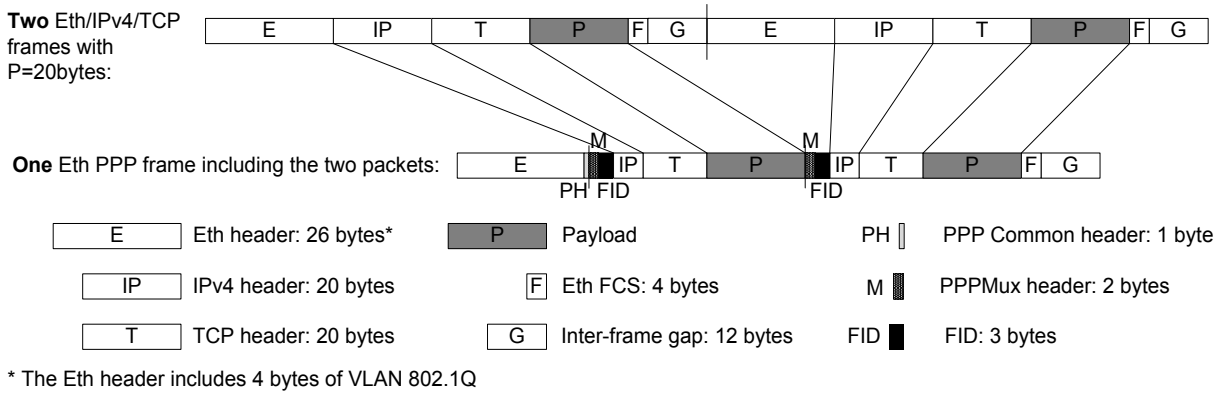


Fig. 3. Scheme of two frames multiplexed together (real scale).

Finally, the egress switch will use the information stored in the controller, in order to get the value of the original fields corresponding to each flow (using the FID). Thus, it will be able to rebuild the packets to their native form and send them as non-compressed individual frames.

IV. RESULTS

In this section, we present the bandwidth savings which can be obtained using this traffic optimization method. The savings are measured as the difference between the number of bytes required at Eth level when using the optimization method with respect to the native Openflow protocol. They are obtained as a function of the number of multiplexed packets *N*. We have to consider the Eth Inter-frame gap in the calculations, since it also limits the throughput of the network.

Since the time for sending the compressed and the native traffic is the same, we can define Bandwidth Savings *(BS)* as:

$$BS = 1 - \frac{Bandwidth_{optimized}}{Bandwidth_{native}} = 1 - \frac{Bytes_{optimized}}{Bytes_{native}} \quad (1)$$

$Bytes_{native}$ is defined as (see Fig. 3) the sum of the sizes of the Eth header *(E)*, the native network and transport headers *(NH),* the expected size of the payload ($E[P]$), and the Eth trailer *(T)* and inter-frame gap *(G)*:

$$Bytes_{native} = N \bullet (E+NH+E[P]+T+G) \quad (2)$$

And the expected size of the multiplexed packet will be the sum of:

- Ethernet header *(E)*.
- Common header: The PPP headers *(PH)*.
- Multiplex header: The size of the PPPMux separator included at the beginning of each multiplexed packet *(N•M)*.
- The *flow identifier* of each packet *(N•FID)*.
- The compressed Network and Transport level headers *(N•CH)*.
- The payload of each packet *(N• E[P])*.
- The Ethernet trailer *(T)*.
- The inter-frame gap *(G)*.

$$Bytes_{optimized} = E+PH+N \bullet (M+FID+CH+E[P])+T+G \quad (3)$$

If we substitute (3) and (2) in (1), we obtain the bandwidth savings, which can be divided into a fixed and a variable term (which depends on the number of multiplexed packets). The fixed term, which is the asymptote of the bandwidth savings, can be expressed as:

$$1 - \frac{M + FID + CH + E[P]}{E + NH + E[P] + T + G} \quad (4)$$

And the term which depends on the number of packets, giving us an idea of how the common header is shared between the multiplexed packets, is:

$$- \frac{1}{N} \frac{E + PH + T + G}{E + NH + E[P] + T + G} \quad (5)$$

Regarding the reduction in the amount of packets per second, the results are similar to those reported in [10], i.e. a reduction by a factor of *N*.

In order to evaluate the performance gains of this approach, four different traffic patterns have been tested:

a) VoIP using IP/UDP/RTP (40 bytes header for IPv4 and 60 for IPv6) and G.729 codec with 2 samples per packet (20 bytes payload) every 20 ms.

b) Client-to-server flows of a UDP-based online game [11] (28 or 48 bytes header), with 24.65 packets per second, and an average payload of 41.09 bytes.

c) Client-to-server flows of a TCP-based online game [2] (40 or 60 bytes header) of 9.51 packets per second with an average payload of 8.74 bytes.

d) IPv4/TCP ACKs of 40 or 60 bytes.

Table I enumerates the fields that present a static behavior for the considered traffic patterns, and can be considered as NOCHANGE. Other fields may also be selected depending on the application and the service (e.g. in VoIP or TCP ACKs using IPv6, the field Payload Length could also be avoided, since it is fixed).

The value of the asymptote (4) for the different traffic patterns is shown in Table II. As a consequence of the compression of the headers and multiplexing, which reduces the total amount of Eth frames, up to 72 % of bandwidth can be saved if IPv4 is used. When using IPv6, this figure rises up to 81%. The savings for all the flows are above 50%. The ACKs flow is the one that obtains the best savings, due to the absence of payload. In the case of the UDP-based game, the header-to-payload ratio is the lowest, so it is the pattern which shows the lowest savings.

TABLE I. FIELDS CONSIDERED AS NOCHANGE FOR THE STUDIED PATTERNS

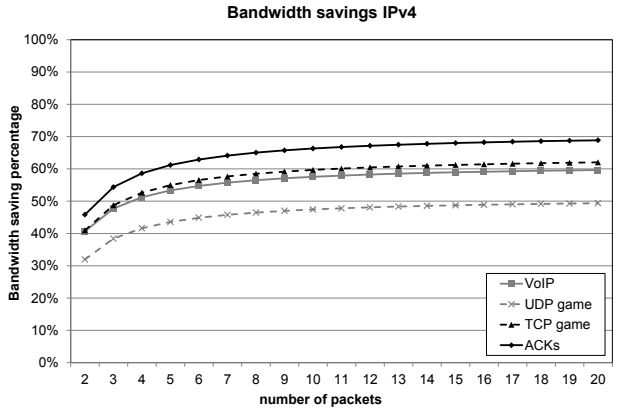
IPv4	***IPv6***	***TCP/UDP***	***RTP***
Version	Version	Source Port	Version
IHL	Traffic Class	Dest. Port	P
DSCP	Flow Label	Data Offset	X
ECN	Next Header	Reserved	CC
Time To Live	Hop Limit	Urgent Pointer	M
Protocol	Source Address		PT
Source Address	Dest. Address		SSRC id
Dest. Address			

TABLE II. ASYMPTOTIC SAVINGS FOR THE STUDIED PATTERNS

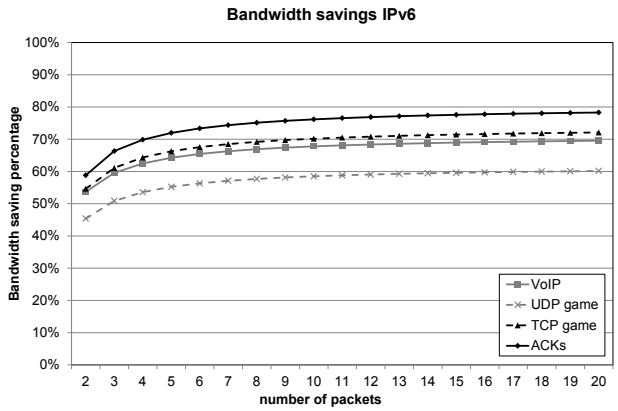
	VoIP	*UDP game*	*TCP game*	*TCP ACKs*
IPv4	62.75%	52.21%	65.02%	72.62%
IPv6	72.13%	62.55%	74.95%	81.37%

The variable term (5) reports the number of packets required for obtaining significant savings. In order to study its influence, we have built Fig. 4. It can be seen that high values of bandwidth savings are obtained not only when 20 packets are multiplexed, but we also obtain similar results for smaller numbers of packets (e.g. 10 packets); and even with 2 packets we can still save 40 % of bandwidth in some cases. These savings are even more significant if IPv6 is used (Fig. 4 b), since the overhead of this protocol is higher than that of its predecessor. In this case, bandwidth savings can reach 78 %.

As a counterpart of savings, additional latency would be added to the multiplexed packets, caused by the retention time required in the *ingress* switch in order to get a number of packets before building the multiplexed frame. The proposed method should avoid the addition of delays which could impair user's experience or may reduce the sending rate of TCP flows, as studied in [16].



(a)



(b)

Fig. 4. Bandwidth savings for each pattern a) using IPv4; b) using IPv6.

The influence of optimization techniques on subjective quality for real-time services has been explored [9], [14], [15] and it has been shown that user's perceived quality can be maintained. The provision of a good quality is possible because of three facts: first, the high rates of the considered traffic patterns: VoIP generates a packet every 20 ms; inter-packet time for the studied games are 40 and 105 ms respectively; and a 100 pps ACK flow can be easily found on the Internet. Second, an upper bound can be put on the added delay, by the use of a suitable value of the multiplexing interval. For this aim, traffic classification based on flows' statistics can be used to establish the interactivity requirements of a given flow [18]. Third, significant bandwidth savings are obtained even if only 2 packets are multiplexed.

Another counterpart would be the increase of the processing capacity required in the ingress and egress switches. However, this increase would be cancelled out over the network, since the number of frames to switch would be reduced in the intermediate elements. As a consequence, the processing capacity could be increased in the edge devices and reduced in the intermediate ones.

V. CONCLUSIONS

A method able to save bandwidth, and to reduce the amount of packets per second, for services using small packets in OpenFlow SDNs has been proposed, based on multiplexing together different flows sharing a common network path, and compressing packet headers. For this aim, the fields that are the same for all the packets in a flow are included in the Openflow tuple, and then avoided in the intermediate hops.

Bandwidth savings up to 68 % for IPv4, and 78 % for IPv6 can be obtained. As a counterpart, latency would be slightly increased, but the additional delay can be kept under tolerable limits for services sending high packet rates.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, J. Turner, "OpenFlow: enabling innovation in campus networks," in ACM SIGCOMM Computer Communication Review, 38(2), pp. 69-74, 2008.

[2] P. Svoboda, W. Karner, M. Rupp, "Traffic Analysis and Modeling for World of Warcraft," in Proc. IEEE International Conference on Communications, ICC, Urbana-Champaign, IL, USA, 2007.

[3] E. Ertekin, C. Christou, "Internet protocol header compression, robust header compression, and their applicability in the global information grid," IEEE Communications Magazine, vol. 42, pp. 106-116, Nov. 2004.

[4] K. Sandlund, G. Pelletier, L-E. Jonsson, "The RObust Header Compression (ROHC) Framework," RFC 5795, 2010.

[5] B. Thompson, D. Wing, T. Koren, "Tunneling Multiplexed Compressed RTP (TCRTP)," RFC4170. 2005.

[6] T. Koren, S. Casner, J. Geevarghese, B. Thompson, P. Ruddy, "Enhanced Compressed RTP (CRTP) for Links with High Delay, Packet Loss and Reordering," RFC 3545, 2003.

[7] R. Pazhyannur, I. Ali, C. Fox, "PPP Multiplexing," RFC 3153, 2001.

[8] J. Lau, M. Townsley, I. Goyret, "Layer Two Tunneling Protocol - Version 3 (L2TPv3)," RFC 3931, 2005.

[9] R. M. Pereira, L.M. Tarouco, "Adaptive Multiplexing Based on E-model for Reducing Network Overhead in Voice over IP Security Ensuring Conversation Quality," in Proc. Fourth international Conference on Digital Telecommunications, Washington, DC, pp. 53–58, Jul. 2009.

[10] J. Saldana, J. Fernandez-Navajas, J. Ruiz-Mas, D. Wing, M. Perumal, M. Ramalho, G. Camarillo, F. Pascual, D. R Lopez, M. Nunez, D. Florez, J.A. Castell, T. de Cola, M. Berioli, "Emerging real-time services: optimizing traffic by smart cooperation in the network," Communications Magazine, IEEE, vol.51, no.11, pp.127,136, Nov 2013.

[11] W. Feng, F. Chang, W. Feng, J. Walpole, "A traffic characterization of popular on-line games," IEEE/ACM Trans. Networking, vol 13, 3, pp. 488-500, Jun 2005.

[12] S. Jivorasetkul, M. Shimamura, K. Iida, "End-to-End Header Compression over Software-Defined Networks: A Low Latency Network Architecture," in Proc. Int. Conf. on Intelligent Networking and Collaborative Systems, Washington DC, USA, pp. 493-494, 2012.

[13] R.R. Denicol, E.L. Fernandes, C.E. Rothenberg, Z.L. Kis, "On IPv6 support in OpenFlow via Flexible Match Structures," Nov 2011.

[14] J. Saldana, J. Fernandez-Navajas, J. Ruiz-Mas, J. Murillo, E. Viruete, J. I. Aznar, "Evaluating the Influence of Multiplexing Schemes and Buffer Implementation on Perceived VoIP Conversation Quality," Computer Networks (Elsevier), vol 56, Issue 7, pp. 1893-1919, May 2012.

[15] J. Saldana, J. Fernandez-Navajas, J. Ruiz-Mas, E. Viruete Navarro, L. Casadesus, "Online FPS Games: Effect of Router Buffer and Multiplexing Techniques on Subjective Quality Estimators," Multimedia Tools and Applications, Springer, 2012.

[16] J. Saldana, "The Effect of Multiplexing Delay on MMORPG TCP Traffic Flows," Consumer Communications and Networking Conference, CCNC 2014. Las Vegas, pp 447-452, Jan 2014.

[17] J. Saldana, J. Fernandez-Navajas, J. Ruiz-Mas, "Can We Multiplex ACKs without Harming the Performance of TCP?," Consumer Communications and Networking Conference, CCNC 2014. Las Vegas, pp 921-922, Jan 2014.

[18] T. T. Nguyen, G. Armitage, P. Branch, S. Zander, "Timely and continuous machine-learning-based classification for interactive IP traffic," IEEE/ACM Trans. on Networking, 20(6), pp 1880-1894, 2012.