# Implementing Quality of Service for the Software Defined Networking Enabled Future Internet

Sachin Sharma[1], Dimitri Staessens[1], Didier Colle[1], David Palma[2], Joao Goncalves[2], Ricardo Figueiredo[3], Donal Morris[3], Mario Pickavet[1], and Piet Demeester[1]
[1]Ghent University - iMinds, Belgium, [2]OneSource, Portugal, [3]RedZinc, Ireland
Email:[1]{firstName.LastName}@intec.ugent.be,[2]{palma, joagonca}@onesource.pt,[3]{ricardo, dmorris}@redzinc.net

***Abstract*—Achieving ever-growing Quality of Service (QoS) requirements for business customers is a major concern over the current Internet. However, presently, its architecture and infrastructures are inflexible to meet the demand of increased QoS requirements. OpenFlow, OF-Config (OpenFlow Configuration and Management protocol), and OVSDB (Open vSwitch Database Management protocol) protocols are well-known software defined networking (SDN) technologies for the Future Internet, enabling flexibility by decoupling the control plane from networking devices. In this paper, we propose a QoS framework using the SDN technologies and test the framework in failure-conditions using single and multiple autonomous system scenarios of the current Internet. We show that an effectively high QoS can be achieved for business customers using our framework.**

## I. Introduction

Providing high Quality of Service (QoS) for business customers has always been a major concern over the current Internet. To provide high QoS, there is a Service Level Agreement (SLA) between business customers and a service provider. If SLAs are not met, these are compensated for the loss of service. Therefore, in the current Internet, high-priority traffic (traffic from a business customer) should always get a higher precedence over best-effort traffic. For this purpose, two QoS mechanisms – IntServ (Integrated Service) [1] and DiffServ (Differentiated Service) [2] – are being implemented over the Internet. IntServ has a scalability problem as it is based on individual flows and DiffServ alleviates this problem by providing QoS based on aggregated flows. In Diffserv, customers or applications request a bandwidth broker (the broker of a certain DiffServ autonomous system) to allocate a pre-determined amount of bandwidth.

To allocate the requested bandwidth, the bandwidth broker needs to perform three tasks: (1) determine the availability of network resources (e.g. bandwidth), (2) configure respective network resources if they are available, and (3) reconfigure network resources to recover from failures (may be after or before a failure). In the current Internet, all the aforementioned tasks are challenging. The difficulty from the first task resides in the fact that each networking device in the Internet runs its own control software to make the forwarding decisions, lacking the broader picture of available resources in its network. For the second task, the problem is that there is no standard protocol available for configuring networking devices. Currently, the bandwidth brokers perform this task using vendor-specific protocols. For the third task, resilience differentiation frameworks [3] are already proposed to provide QoS to high-priority traffic on failure conditions. However, as configuration protocols are not standardized, resilience differentiation frameworks needs to depend on vendor-specific protocols for reconfiguration. Therefore, the current QoS architectures are not truly successful, and are not globally implemented over the Internet. With Software Defined Networking (SDN) technologies [4], it is practically possible to solve the above issues as it removes software from all networking devices in a network, embeds it into one or more external servers (e.g., controllers), and provides standardized vendor-agnostic interfaces between them. The examples of vendor-agnostic interfaces are OpenFlow [4], OpenFlow Configuration and Management protocol (OF-Config) [5], and Open vswitch Database Management protocol (OVSDB) [6].

In this paper, we implement a QoS framework using vendor-agnostic interfaces of SDN technologies such as OpenFlow and OVSDB protocols. In the framework, we assume that a single autonomous system (AS) of the current Internet is controlled by one SDN controller. We implement a QoS technique in each autonomous system and add a failure recovery mechanism onto it, keeping in mind the QoS requirements of high-priority traffic. Regarding the implementation, we do not focus on fast-failure recovery [7], [8], [9] but instead, we focus on scenarios in which high-priority traffic should always get a higher precedence over best-effort traffic, even after a failure.

We test our implemented framework in a single AS (emulated pan-European topology) and multiple AS scenarios (designed in the CityFlow project [10]) on the OFELIA testbed facility at iMinds [11], and evaluate three failure recovery scenarios. In the first scenario, we assume enough bandwidth so that neither high-priority nor best-effort traffic will be affected after all the traffic being re-routed to a failure-free path. In the second scenario, we restrict the available data path bandwidth so that all priority traffic can be rerouted after recovering all flows. However, some of the best-effort traffic flows experiences packet loss in order to meet the requirements of high-priority traffic. In the third scenario, the capacity is squeezed further so high-priority traffic will also be starved for bandwidth after all traffic is redirected to a failure-free path. In this scenario, there is no best-effort traffic interference. A live demonstration of the single AS scenarios has already been shown in [12] using portable devices.

## II. Related Work

Since many years, QoS frameworks remain as a feature of paramount importance for the current Internet. In the literature, two QoS frameworks [1], [2] are described for the Internet: IntServ and DiffServ. In IntServ, resources are reserved based on each flow in all networking devices, and in Diffserv, traffic is classified at the network boundary, and assigned then to a behavior aggregate. DiffServ is a framework that introduces a bandwidth broker as its logical resource, network and policy management module. There is one bandwidth broker per Diffserv autonomous system. The bandwidth broker maintains policies and negotiates SLAs with customers and with neighboring bandwidth broker domains.

The introduction of SDN allows us to rethink about QoS frameworks in a standard way. From its first specifications (version 1.0), the OpenFlow protocol, which stands as the de facto standard for SDN, considers QoS as a part of its operations. In its version 1.0, OpenFlow presents an enqueue mechanism with which a certain flow (type of traffic) can be redirected to a particular queue that maintains QoS. However, queues cannot be created using OpenFlow. For queue creation, SDN provides OF-Config and OVSDB protocols. The OF-Config protocol is standardized by Open Networking Foundation and is based on the Network Configuration Protocol (RFC 4721). The OVSDB protocol is standardized by the Internet Engineering Task Force and is based on JavaScript Object Notation-Remote Procedural Calls (JSON-RPCs). The OVSDB protocol is already implemented in one of the OpenFlow switch software (Open vSwitch [13]). The latest versions of OpenFlow (version 1.3 and 1.4) provide per-flow meters that provide simple QoS operations such as rate-limiting, controlling the rate of packets per flow-entry directly, while queues are attached to ports to which the flows are forwarded.

Recently, OpenQoS [14] and QoS methods [15] are proposed for SDN. Using OpenQoS, media flows are dynamically placed on QoS guaranteed routes to meet the QoS requirements and using QoS methods, an approach to implement Quality of Service that is managed and defined by a centralized network is proposed. Our approach in this paper is different from OpenQoS and the already proposed QoS methods. In our approach, we reserve resources for each high-priority flow at the ingress switch but OpenQoS does not use any resource reservation scheme in its method. In addition, we propose a complete framework for implementing QoS in SDN but QoS methods in [15] describes how to implement QoS using a centralized controller.

## III. QoS Framework for SDN

Our framework is based on the research performed in the EuQoS project [16] for end to end quality of service over heterogeneous networks. The essential principal of the research is that bandwidth resources are managed in an on-path off-line manner. By on-path we mean that resource management follows the forwarding path of the IP packets, across multiple AS, as determined by BGP (Border Gateway Protocol) or OSPF (Open Shortest Path). By off-line we mean that the resource management is implemented in software that is off-line from the network elements which are responsible for packet forwarding. Connection admission control is implemented using packet shapers or policers. Along the path, capacity management is implemented only at choke points which are mostly the interconnection points and the edges.

Our proposed QoS framework is shown in Fig. 1. To implement the presented framework, we assume that: (1) a single AS is controlled by a single controller, (2) the controller is able to run both the OpenFlow protocol and the OVSDB protocol, and (3) a bandwidth broker and the controller are able to communicate with each other through a northbound

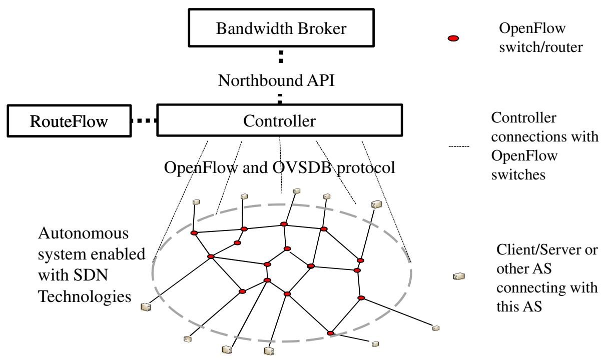
Bandwidth Broker
Northbound API
RouteFlow
Controller
OpenFlow and OVSDB protocol
Autonomous system enabled with SDN Technologies
OpenFlow switch/router
Controller connections with OpenFlow switches
Client/Server or other AS connecting with this AS

Fig. 1. Quality of Service Framework for the SDN enabled AS

API (Fig. 1). As the current controllers do not support the OVSDB protocol, one of the controllers (known as Floodlight [17]) is extended to support this feature in order to add queues in SDN networks. For implementing the OVSDB protocol in the controller, we implementing a RPC client and through this RPC client, we Nevertheless, as a northbound API (e.g., REST API [17]) is already implemented in several OpenFlow controller software such as Floodlight or OpenDaylight, we used this API for the implementation of our framework. In our implementation, for routing traffic within or between two ASs in SDN networks, we use RouteFlow [18], and run OSPF and BGP using the RouteFlow automatic configuration framework given in [19]. As our framework runs BGP and OSPF routing that is similar to the routing used in the current Internet infrastructure, our framework using SDN infrastructure can be integrated with the legacy network using BGP or OSPF routing. All the steps of our framework are described below:

1) **Configuration of three default queues**: The controller configures three queues on each port of an OpenFlow router using the OVSDB protocol (Fig. 1). The first queue (which is for control traffic) has the highest priority, followed by the second queue (which is for high-priority traffic), and then by the third queue (which is for best-effort traffic). The traffic is considered as high-priority traffic if its Type of Service (TOS) field is enabled and is considered as best-effort traffic if its TOS field is disabled. This step is performed when the controller establishes an OpenFlow session with an OpenFlow router.
2) **Running of a routing protocol**: The controller runs standard routing protocols (OSPF, BGP) for each OpenFlow router using RouteFlow (Fig. 1). The controller then sends the control traffic of the routing protocol using the first queue configured at Step 1.
3) **Establishment of Flow Entries**: On discovering a new routing entry for an OpenFlow router using RouteFlow, the controller establishes two Flow-Entries in the router. The first Flow-Entry is for high-priority traffic. Hence, the flow of this entry has the same destination address as the destination of the routing entry and has the enabled TOS field. This Flow Entry redirects high-priority traffic to the second queue (configured at Step 1) of the outgoing port, given by the routing entry. The second Flow Entry is for best-effort traffic. Hence, the flow of this entry has the same destination address as the destination of the routing entry and has the disabled TOS field. This

Flow Entry redirects best-effort traffic to the third queue (configured at Step 1) of the outgoing port, given by the routing entry.

4) **Availability of network resources for high-priority traffic**: On request from a business customer to assign a certain amount of bandwidth, the bandwidth broker checks for the availability of network resources or SLA validation by contacting the controller and the neighboring bandwidth broker (multiple AS scenario). The SLA validation between the bandwidth brokers in multiple domains (AS) is done through a signalling protocol [16] that is same as Next Steps in Signalling (NSIS) defined by IETF. If the requested resources are available or SLAs are validated, the next step is performed. Otherwise, an error message can be sent to the customer.
5) **Configuring a rate limiter queue for high-priority traffic**: The controller finds the edge router for high-priority traffic and configures a rate limiter queue (Q) on the edge router to forward high-priority traffic. The rate of the queue has the rate negotiated between the bandwidth broker and a business customer.
6) **Establishing a Flow Entry for high-priority traffic**: After configuration of the rate-limiter queue (Step 5), the controller establishes a Flow Entry on the edge router, which enables the TOS field of high-priority traffic to classify the traffic from the business customer as high-priority traffic and redirects this traffic to the rate limiter queue.
7) **Re-establishing Flow Entries and the rate limiter queues**: On detecting failures due to the routing protocols timeouts (e.g., Router Dead Interval in OSPF), the controller removes the Flow Entries corresponding to the faulty routing entries. When the controller discovers new failure-free routing entries for OpenFlow routers, it establishes the entries (two Flow Entries per routing entry) in the corresponding routers, and for the edge router, it reconfigures the rate limiter queue (corresponding to a new path). This step is performed to recover from a failure.

## IV. Experimentation

In this section, we describe the testbed, emulated topologies and evaluation methodology.

### A. Emulation Testbed and Emulated topologies

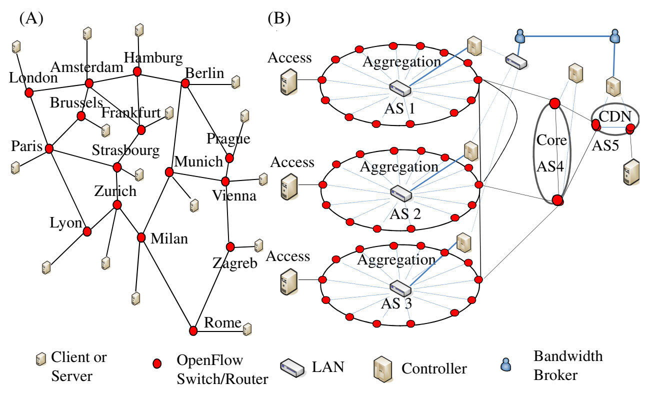

Fig. 2. (A) Single AS Topology (B) Multiple AS Topology

We perform our experiments on the OFELIA testbed facility provided by iMinds [11]. The testbed consists of 100 nodes interconnected by a non-blocking 1.5 Tb/s VLAN Ethernet switch (Force 10). Each node provides the flexibility to change the software handling network traffic and is a dual processor, dual core server with 4GB RAM and 4x 80GB hard disk with 6x1 Gb/s or 4x1 Gb/s interfaces.

We emulate extensive QoS experiments using a single AS topology (pan-European topology, Fig. 2A) and using multiple AS topology (Fig. 2B). The single AS topology shows 16 OpenFlow routers connected with each other in a mesh fashion. Each router is also connected with a server or client (shown in Fig. 2A). In addition, each router has provided a dedicated interface to a switched Ethernet LAN (not shown in Fig. 2A), which establishes an out-of-band connection with a single controller. Out-of-band means that there is a separate channel for the control and data planes (i.e. the failure in the data plane does not affect the communication between the switches and the controller). The multiple AS topology, shown in Fig. 2B, is designed in the CityFlow project to emulate future Internet experiments for a city of 1m inhabitants [10]. This topology contains 3 clients in access networks, containing traffic from end users. In addition, it contains 3 AS in the aggregation network (each containing 13 routers), 1 AS in the core network (containing 2 routers), and 1 AS in the content delivery network (CDN) (containing 2 routers). Furthermore, Fig. 2B shows that each AS is controlled by a single controller through a switched Ethernet LAN and a bandwidth broker is directly connected with the aggregation and core network controller. In the OFELIA testbed, we generate the presented topologies. The bandwidth capacity of CDN networks links is 100 Mb/s and all other links are limited to 50 Mb/s. For the single AS experiments, all links are of 50 Mb/s capacity.

### B. Emulation Methodology

In our emulation, we use Open vSwitch [13] for OpenFlow router implementation and use Floodlight (version 0.90) and RouteFlow implementing our QoS framework for controller implementation. Inside an AS, we run the OSPF routing protocol using RouteFlow: the OSPF hello interval is 1 second and the router dead interval is 4 seconds. Exterior to AS, we use BGP. For data traffic (best-effort and high-priority) generation, we used an open source packet generation tool known as DITG (Distributed Internet Traffic Generator) [20].

The emulation methodology of all our experiments including single and multiple AS experiments is same. The difference lies in the amount of high-priority and best-effort traffic sent by each server. Therefore, we describe the methodology in detail only for the single AS experiments Fig. 2A. In the experiments, each server (Fig. 2A) sends traffic – high-priority traffic and best-effort traffic – to all other servers/clients in the topology. The arrival rate of high-priority and best effort traffic are Poisson distributed on an average interval. For the experiments, this interval is varied to increase the total traffic from each server. For high-priority traffic, a server requests the bandwidth broker to assign a certain bandwidth. In the experiment, we fail and repair a link, and show the impact of these operations on high-priority and best-effort traffic.

We now describe the link down and up in our emulation by breaking and repairing the link Zagreb-Vienna in the single AS

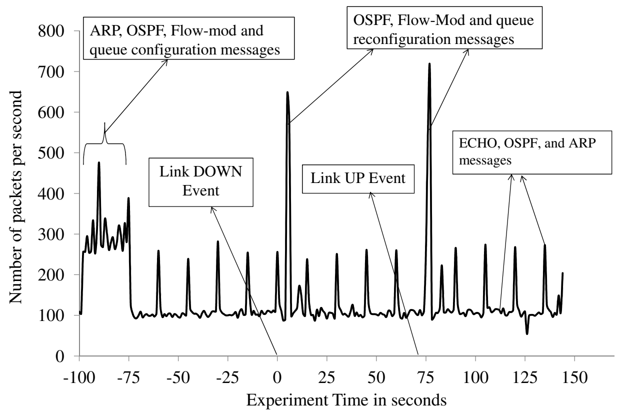

Fig. 3. Controller Traffic Intensity

topology (Fig. 2A). Fig. 3 shows our methodology using the traffic that was captured on the controller using the *tcpdump* utility. At the start of the experiment, during the warm-up time (from −98 to −72 seconds), large spikes are observed. These spikes are due to the Address Resolution Protocol (ARP), OSPF, queue configuration and Flow-Mod configuration messages. ARP messages are sent from each server to all other clients to learn the corresponding Media Access Control (MAC) addresses. Moreover, OSPF messages are sent by the controller to retrieve the connectivity between existing links. Queue configuration messages are sent to configure queues at the edge router to forward high-priority traffic, and finally, Flow-Mod messages are sent to add Flow-Entries at the edge routers to redirect high-priority traffic to a rate limiter queue. Note that traffic to establish other proactive Flow Entries for best-effort and high priority traffic are not shown in Fig. 3.

Periodical spikes in the controller traffic can also be seen in Fig. 3. These are ECHO, OSPF and ARP messages that were sent to check aliveness of the links of the topology. At second 0, we fail link Zagreb-Vienna by disabling the Ethernet interface at Zagreb and we see a large spike near the 4 seconds of runtime. This is because the controller detects the failure in every 4 seconds, and it sends a new Link State Advertisement (LSA) and reconfigures faulty paths by sending Flow-Mod and queue configuration messages. At about second 71, we make link Zagreb-Vienna up, and see a large spike again at second 75. This large spike is due to LSAs and reconfiguration messages.

## V. Emulation Results

We now show the results of the experiments performed on the OFELIA testbed.

### A. Single AS experiments

Fig. 4 shows the traffic for the client connected to Zagreb (Fig. 2A), when the failure recovery paths have the enough bandwidth to accommodate both best-effort and high-priority traffic. The traffic is captured for the case when link Zegreb-Vienna is failed at second 0 and repaired at second 71. In this experiment, each server sends about 0.560 Mb/s best-effort traffic and 0.240 Mb/s high-priority traffic to each client/server in the topology (Fig. 2A). Fig. 4 shows that the total of 8.40 Mb/s best-effort traffic and the total of 3.60 Mb/s high-priority traffic are received by the client (connected to Zagreb) from all other servers. It also shows that between 0 to 4 second, there is

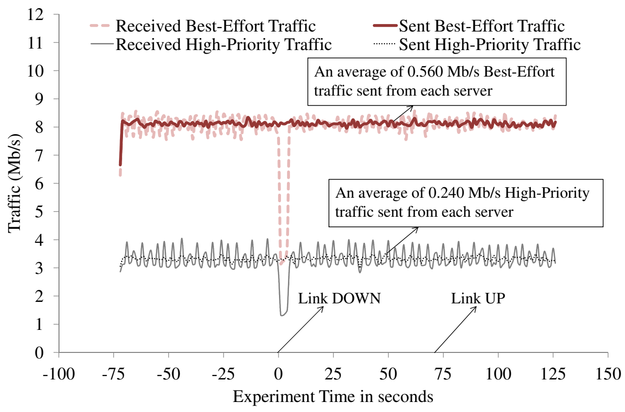

Fig. 4. Traffic for a client (client connected to Zagreb). Best-Effort Traffic is 0.560 Mb/s and high-priority traffic is 0.240 Mb/s from each server

a drop in the received best-effort and high-priority traffic and after second 4, there is no drop in best-effort and high-priority traffic. The dropped traffic is due to the failure that was given on link Zagreb-Vienna. After the failure, the controller has redirected both best-effort and high-priority traffic to failure-free paths. As the failure-free paths has enough bandwidth to accommodate both high-priority and best-effort traffic, we do not see any drop in high-priority and best-effort traffic after failure recovery (second 4) in Fig. 4

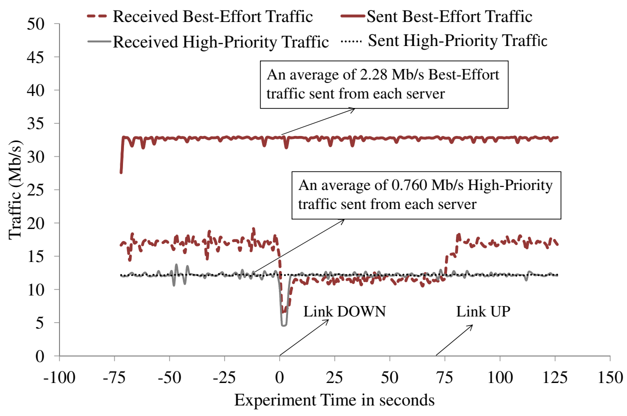

Fig. 5. Traffic for a client (client connected to Zagreb). Best-Effort Traffic is 2.28 Mb/s and High-Priority traffic is 0.760 Mb/s from each server

Fig. 5 shows traffic for the Zegreb client when failure recovery paths have enough bandwidth to accommodate only high-priority traffic and have not enough bandwidth to accommodate all the best-effort traffic. In this experiment, each server sends about 2.28 Mb/s best-effort traffic and 0.760 Mb/s high priority traffic to each client of the topology (Fig. 2). Fig. 5 shows that about 12 Mb/s high-priority traffic is sent to the client connected to Zagreb and all the high-priority traffic is received by the client. It also shows that the total of 34 Mb/s best-effort traffic is sent to the client connecting Zagreb, but until second 0, it receives only about 16 Mb/s. This is because some of links in the working paths (shortest paths from each server) to Zagreb do not have enough bandwidth to forward all best-effort traffic to Zagreb. Therefore, these links first forward all the high-priority traffic and then if bandwidth left, these forward best-effort traffic. During second 0 to 4 (Fig. 5), there is a drop in the received best-effort and high priority traffic. The dropped traffic is due to the failure that was given on link Zagreb-Vienna. At about second 4, the controller has redirected all best-effort and high-priority traffic to failure-free paths. Therefore, after this failure recovery, we see that all the sent high-priority traffic is received by the client. However, after failure recovery, the received best-effort traffic is again decreased to about 12 Mb/s. This traffic is decreased to accommodate all high-priority traffic in failure-free paths. At second 71, we restore the link Zagreb-Vienna and see that after second 75, best-effort and high-priority traffic become equal to the traffic before the failure.

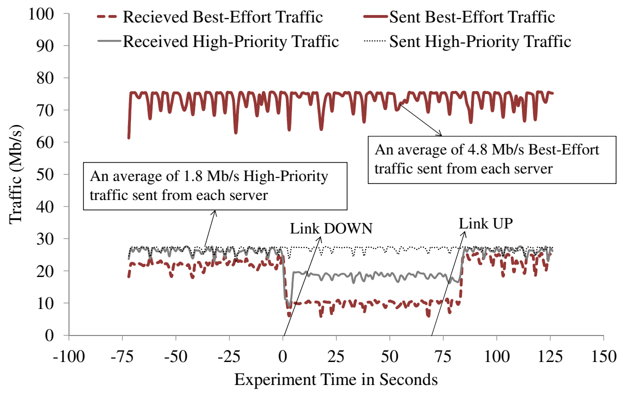

Fig. 6. Traffic for a client (client connected to Zagreb). Best-Effort Traffic is about 4.8 Mb/s and High-Priority traffic is about 1.8 Mb/s from each server

Fig. 6 shows best-effort and high-priority traffic for client Zagreb, when each server sends about 4.8 Mb/s best-effort traffic and 1.8 Mb/s high-priority traffic to each client in the topology. Fig. 6 shows that the total of 72 Mb/s best-effort traffic is sent to the client connecting Zagreb, but until second 0, it receives only about 21 Mb/s. This received best-effort is greater than the received best-effort in the previous figure. This is because as we increase traffic, average bandwidth utilization for links increases, which leads to an increase in the received best-effort traffic.

There is about 27 Mb/s high-priority traffic sent to client Zagreb, and all the high-priority traffic is received by the client until second 0 (Fig. 6). At second 0 in Fig. 6, we see a drop in the high-priority traffic and best-effort traffic. At about second 4, the controller has redirected all the traffic to failure-free paths. After this failure recovery, both high-priority traffic and best-effort traffic are affected. Fig. 6 shows that the received high-priority traffic is decreased to about 19 Mb/s (from second 4 to 75) and the received best-effort traffic is decreased to about 9 Mb/s (from second 4 to 75). This is because some of the links in the paths (calculated by OSPF) to Zagreb have not enough bandwidth to accommodate all the high-priority traffic. This has happened because all the 50 Mb/s of bandwidth in these links is already taken by high-priority traffic and additional high-priority traffic needs to be dropped. As high-priority traffic is given precedence over best-effort traffic, there should not be any best-effort traffic on these links. One of such links in our experiment is link Milan-Munich. The traffic on this link is shown in Fig. 7.

Fig. 7 shows that the total high-priority traffic on link Milan-Munich is about 42 Mb/s (from second −75 to 0). After the link goes down (second 4 to 75), the total high-priority traffic becomes equal to about 50 Mb/s and after the link goes up (second 75 to 125), it again becomes equal to about 42 Mb/s. In the case of the best-effort traffic, the total best-effort

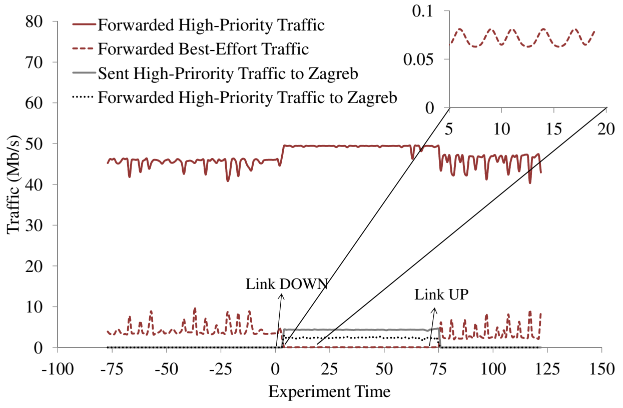

Fig. 7. Traffic on link Milan-Munich in the case of the scenario shown in Fig. 6. For better visualization, the inset shows forwarded best-effort traffic after failure recovery (second 5 to 20)

traffic is about 8 Mb/s before the link down (second −75 to 0). After the link down, it is about 0 Mb/s (≈ 0.08 Mb/s), and after the link up best-effort traffic again becomes equal to 8 Mb/s. Note that the total best-effort traffic on link Milan-Munich is not completely 0 (shown by the inset in Fig. 7) after failure recovery (second 4 to second 75). This is because of the traffic pattern sent on the link which follows a Poisson distribution.

There is no high-priority traffic sent to Zagreb on link Milan-Munich (Fig. 7) before the failure on Zagreb-Vienna. However, after failure recovery (after second 4), about 6 Mb/s of high-priority traffic is sent to Zagreb on the link Milan-Munich. However, this link is only able to forward about 4 Mb/s of this high-priority traffic and the remaining traffic (about 2 Mb/s) is dropped. This is dropped because this link has not enough bandwidth to forward additional high-priority traffic. All 50 Mb/s of bandwidth is already taken by all the high-priority traffic.

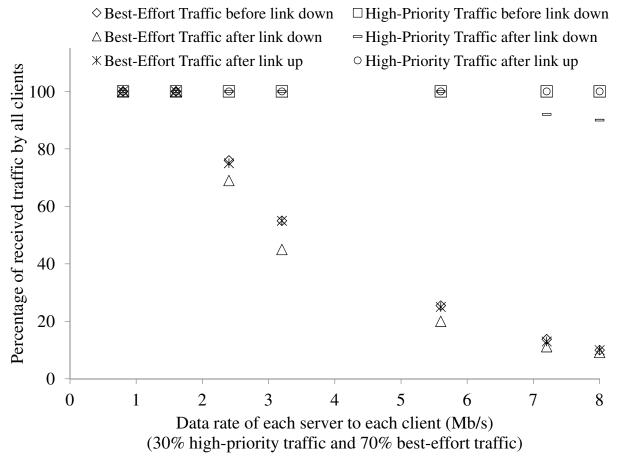

Fig. 8. Total traffic received by all clients (Single AS experiment)

Fig. 8 shows the results of all the experiments performed on the single AS. In these experiments, we increase the data rate of each server to each client in the topology. Each server sends – 30 % of traffic as high-priority traffic and 70 % of traffic as best-effort traffic – to all other servers/clients, and the percentage of total best-effort and high-priority traffic received by all clients are calculated. The percentage of the traffic is calculated before the link down, after the link down, and after the link up event. The traffic before the link down is calculated from second $-72$ to 0 (depicted time scale in Fig. 3), the traffic after the link down is calculated from second 4 to 74 (i.e. after failure recovery), and the traffic after the link up is calculated from second 75 to 120.

Three scenarios can be seen in Fig. 8 at: (1) low data rate (data rate $<$ 2.4 Mb/s), (2) medium data rate (2.4 Mb/s $\leq$data rate $<$ 7 Mb/s), and (3) high data rate (data rate $>$ 7 Mb/s). In case of low data rate scenarios, all the best-effort and high-priority traffic are received before the link down, after the link down, and after the link up. In case of medium data rate scenarios, all the high-priority traffic is received but some of best-effort traffic is dropped to accommodate the high-priority traffic. In case of the high data rate scenarios, after the link down, some of high-priority traffic is also dropped. This is because on high data rate some of links in the paths to clients do not have enough bandwidth to guarantee the delivery of all the high-priority traffic.

### B. Multiple AS experiments

We now present the results of multiple AS experiments performed on the emulated topology shown in Fig. 2B. In this experiment, the server in the CDN network sends high-priority and best-effort traffic to each client in the access network, the link between two autonomous systems is failed and repaired, and the percentage of traffic delivered at access networks is calculated.

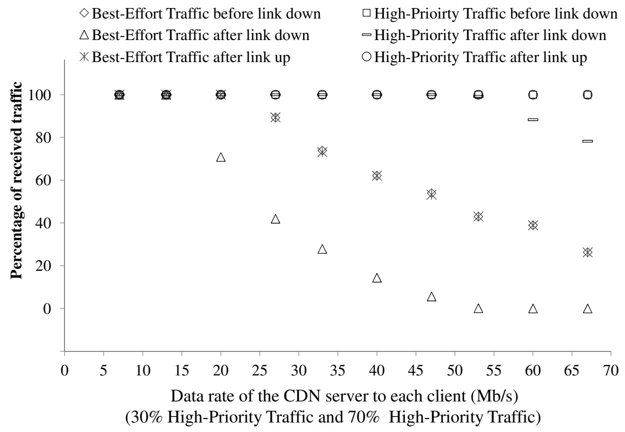

Fig. 9. Total traffic received by the clients located at the access network

Fig. 9 shows the results when a link between the core and access network is failed and repaired as explained in the previous section. As discussed in the single AS experiment, this experiment also shows three different scenarios of high-priority traffic. In the first scenario, when the data rate is low ($<$ 15Mb/s), all the traffic (high-priority and best-effort) is received by the clients. However, in the medium data rate scenario (20 Mb/s $\leq$data rate $<$ 55 Mb/s), some best-effort traffic is dropped to meet the packet delivery requirement of high-priority traffic. Furthermore, in the case of high-data rate scenario (data rate $>$ 55 Mb/s), almost all the best-effort traffic is dropped to meet the requirement of high-priority traffic. In this case, as there is only one link remained connected between CDN and the core network after the link down and all the capacity is already utilized by high-priority traffic, some of high-priority traffic needs to be dropped. Therefore, we see decrease in the delivered high-priority traffic in Fig. 9. Furthermore, we do not see the competition between high-priority and best-effort traffic as different queues are maintained to serve different traffic.

## VI. Conclusions

In this paper, we have implemented a resilient differentiation framework for OpenFlow networks and have tested the framework for single AS and multiple AS scenarios. The single AS scenarios are tested using an emulated pan-European topology, and the multiple AS scenarios are tested using the multiple AS topology developed in the CityFlow project. The results show that high-priority traffic can get precedence over best-effort traffic even on failure conditions. The obtained results revealed that the proposed framework is suitable for the Future Internet, using SDN compliant operations. Our assessment further proved that our framework is in fact resilient to failures and is able to maintain the desired QoS performance, adapting to the available links and providing the necessary configurations in real-time.

## Acknowledgment

This research has received funding from the EU FP7 under agreement $n^o$ 317576 (CityFlow), and $n^o$ 258365 (OFELIA).

## References

[1] R. Braden et al., Integrated Services in the Internet Architecture: An Overview, Internet Engineering Task Force, RFC 1633, 1994

[2] S. Blake et al., An Architecture for Differentiated Services, Internet Engineering Task Force, RFC 2475, 1998

[3] A. Autenrieth et al., Engineering end-to-end IP resilience using resilience-differentiated QoS. IEEE Communications Magazine, 2002

[4] N. McKeown et al., Openflow: Enabling innovation in campus networks, ACM Computer Communication Review, Vol. 38(2), 2008.

[5] OF-CONFIG: https://www.opennetworking.org/sdn-resources/onf-specifications/openflow-config

[6] B. Pfaff et al., The Open vSwitch Database Management Protocol, IETF, 2013

[7] S. Sharma et al., Enabling fast failure recovery in OpenFlow networks, DRCN, pp. 164–171, 2011

[8] S. Sharma et al., A demonstration of fast failure recovery in software defined networking, TridentCom, pp. 411-414, 2012.

[9] S. Sharma et al., OpenFlow: Meeting carrier-grade recovery requirements, Computer Communications, Vol. 36(6), pp. 656-665, 2013

[10] CityFlow [Online]: http://www.cityflow.eu/

[11] OFELIA [Online]: http://www.fp7-ofelia.eu/

[12] S. Sharma et al., Demonstrating resilient quality of service in Software Defined Networking, INFOCOM WKSHPS, pp. 133–134, 2014

[13] Open vSwitch [Online]: http://openvswitch.org/

[14] H. E. Egilmez et al., OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks, APSIPA ASC, pp. 1–8, 2012

[15] R. Wallner, et al., An SDN Approach: Quality of Service using Big Switchs Floodlight Open-source Controller, APAN, pp. 14–19, 2013

[16] T. Braun, et al., End-to-End Quality of Service Over Heterogeneous Networks, Springer, 2008.

[17] Floodlight Controller [Online]: http://www.projectfloodlight.org/floodlight

[18] C. Esteve Rothenberg et al. Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking, HotSDN, 2012.

[19] S. Sharma et al., Automatic configuration of routing control platforms for OpenFlow networks, ACM SIGCOMM Computer Communication Review, Volume 43(4), pp. 491-492, 2013.

[20] A. Botta, et al., A tool for the generation of realistic network workload for emerging networking scenarios", Computer Networks, 2012.