CityFlow, enabling quality of service in the Internet: opportunities, challenges, and experimentation

Sachin Sharma[1,5,*], David Palma[2,6], Joao Goncalves[2], Dimitri Staessens[1], Nick Johnson[3], Charaka Palansuriya[3], Ricardo Figueiredo[4], Luis Cordeiro[2], Donal Morris[4], Adam Carter[3], Rob Baxter[3], Didier Colle[1], Mario Pickavet[1]
[1]Ghent University - IMEC, Belgium, [2]OneSource, Portugal, [3]EPCC, Edinburgh, U.K., [4]RedZinc, Ireland
[5]NEC Laboratories Europe, [6]Dept. of Telematics, NTNU, Norwegian University of Science and Technology
Email: [*]Sachin.Sharma@neclab.eu

***Abstract*—In this paper, we propose an OpenFlow enabled Internet infrastructure, using virtual path slicing in an end-to-end path, so that any user connected to an OpenFlow network is dynamically allocated a corresponding *right of way*. This approach allows an interference-free path, from other traffic, between any two endpoints, on multiple autonomous systems, for a given application flow (e.g., WebHD Video Streaming). Additionally, we propose and implement an end-to-end quality of service framework for the Future Internet and extend the virtual path slice engine to support future Internet technologies such as OpenFlow. The proposed framework is evaluated in distinct multiple autonomous scenarios for a city with a population of 1 million inhabitants, emulating xDSL (Digital Subscriber Line), LTE (Long-Term Evolution) and Fibre networking scenarios. The obtained results confirm the suitability of the proposed architecture between multiple autonomous systems, considering both data and control traffic scalability, as well as resilience and failure recovery. Furthermore, challenges and solutions for experimentation in a large-scale testbed are described.**

I. Introduction

In recent years, Internet traffic from content provider companies (e.g., Skype, Google, Netflix, Akamai, Facebook) has been increasing drastically due to the exponential usage of over-the-top applications by users. This traffic is expected to keep increasing, as content providers launch more and more over-the-top applications. However, telecommunication companies, who bear the operational and maintenance cost of the Internet infrastructure, are not interested in investing on additional infrastructure capacity to provide the required bandwidth for these applications without an adequate return on network capital employed.

Several solutions such as IntServ (Integrated Services) [1] and DiffServ (Differentiated Services) [2] have been proposed to solve the problems above. IntServ provides Quality of Service (QoS) based on a per-flow basis and has a scalability issue, whereas Diffserv alleviates this issue by providing QoS on aggregated flows. One of the problems with DiffServ is that QoS requirements (specified in Service Level Agreements i.e., SLA) are defined in the classes of service and the definition of the classes of service is valid only within that domain. Hence, until and unless, all the AS domains in the path towards the destination agree to provide the same service to traffic, it is difficult to provide end-to-end QoS over the Internet.

Since no content provider has created a successful business model for large scale QoS over the Internet [3], it is very difficult for Internet infrastructure owners to get profit from the growing demand for bandwidth and quality. Currently, the Internet works on a best-effort basis and content owners, who obtain revenue for their applications, can inject traffic into the Internet at an originating Autonomous System (AS) and expect it to be carried over the Internet to a destination autonomous system without sharing revenue with infrastructure owners. Enabling end-to-end QoS, instead of QoS specific to an autonomous system, is one possible solution to the above problem. In addition, it is in the interest of content providers and users to open guaranteed pipes over the Internet.

As part of innovation in FP7 and H2020 of the FIRE (Future Internet Research and Experiment community), small and medium Enterprises were encouraged to submit project proposals regarding innovation. For these innovations, the CityFlow project[1], which was centered on the virtual path slice engine[2] (due to its capability of multiple autonomous system innovations and conducting experiments at scale), was approved. In this paper, we describe an end-to-end QoS framework, experimentation, challenges, and solutions proposed in the CityFlow project. We propose a differentiated Internet based on virtual path slicing (VPS) and Software Defined Networking (SDN) technologies, such as OpenFlow. Using SDN [4], [5], the control plane can be separated from the data plane of network devices (such as switches or routers) and can be embedded into one or more external servers called controllers. Using VPS [6], telecommunication companies can enable a *right of way* (end-to-end) for users' traffic over the Internet without interference from best-effort traffic.

For the differentiated Internet, we also propose an operational model for the Internet and give opportunity for infrastructure owners, content providers, and users to benefit from it. We test our model and framework using a wide range of large-scale multi autonomous signaling experiments that are performed on a large scale experimental facility in Europe, i.e., at OFELIA (OpenFlow in Europe Linking Infrastructure and Applications) [7]. One of the experiments is also performed on the public Internet using the Amazon cloud facility. Our experiments mimicked the conditions that would be required for WebHD Video Streaming and HD Video to Video. All the experiments are performed by taking into account the key Internet technologies (4G/xDSL/Fibre) for a mid-sized European city of around 1 million inhabitants.

The contributions of this paper are:

[1]The CityFlow project: https://www.cityflow.eu/
[2]A commercial product from Redzinc, www.redzinc.net

1) The proposal, implementation and experimentation of a scalable end-to-end QoS framework (large-scale) for the SDN enabled future Internet (Section II, Section V);
2) An overview of reference scenarios (emulating real conditions) for a city of $1m$ inhabitants (Section III);
3) An overview of challenges and solutions to perform a large scale experiment on a large-scale experimental facility (such as OFELIA) (Section IV).

II. QoS MODEL FOR THE INTERNET

Our QoS framework is derived from previous works (EuQoS [6], NSIS [8] and IPsphere/TMForum [9] methods) on implementing end-to-end QoS over the Internet. To deploy end-to-end QoS, multiple autonomous systems are required to have a consistent view of the classes of services (CoSs). Therefore, in the CityFlow project, we use end-to-end CoSs proposed in the EuQoS project [6], following IETF recommendations. The essential principal of the CityFlow 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 autonomous systems, as determined by BGP (Border Gateway Protocol). By off-line we mean that the resource management is implemented in software off-line from the network elements that are responsible for packet forwarding. Along the path, capacity management is implemented only at choke points which are mostly the interconnection points and the edges.

A. VPS Engine Overview

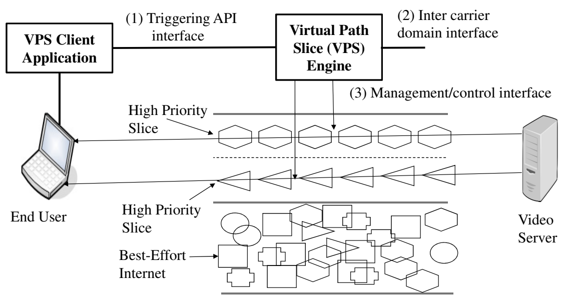
VPS Client Application
(1) Triggering API interface
Virtual Path Slice (VPS) Engine
(2) Inter carrier domain interface
(3) Management/control interface
High Priority Slice
High Priority Slice
Best-Effort Internet
End User
Video Server

Fig. 1: VPS Engine Overview

Our model uses the VPS engine (commercial product from Redzinc) to setup virtual path slices over the Internet. This engine contains three main interfaces (shown in Fig. 1): (1) the first one (the triggering API interface) is to receive requests from users, (2) the second one (the inter-carrier domain interface) is for inter-carrier domain communication and (3) the third one (the management/control interface) is to communicate with infrastructure networks. The first interface is implemented to receive requests from users to reserve *a right of way* in the Internet. The implementation of the inter-carrier domain interface within the VPS engine (i.e., for inter-carrier domain) is largely influenced by the initial work done in the IETF for Next Steps in Signalling (NSIS) [8]. Currently, implementation of managment/control interfaces for adding QoS in SDN networks are also explored in [10], [11]. Our implementation of the management/control interface for the CityFlow project is detailed in the next subsection.

B. Components of the proposed model

The architecture devised by CityFlow was envisaged for future OpenFlow networks, enabling them with the possibility to dynamically configure paths with guaranteed traffic performance. Motivated by the separation between data and control planes followed by the SDN paradigm, additional business intelligence is included on top of the control plane, which in turn enforces the necessary decisions on the data plane. Fig. 2 depicts a high-level perspective of this approach.

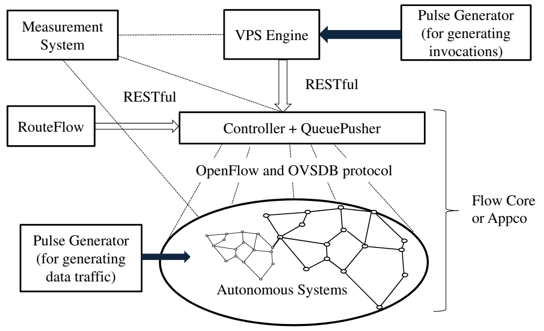
Measurement System
VPS Engine
Pulse Generator (for generating invocations)
RESTful
RESTful
RouteFlow
Controller + QueuePusher
OpenFlow and OVSDB protocol
Flow Core or Appco
Pulse Generator (for generating data traffic)
Autonomous Systems

Fig. 2: CityFlow's architecture and components

In CityFlow, the VPS engine is the entity responsible for business intelligence and manages relationship with the remaining CityFlow components, namely RouteFlow [14] and the controller (extended with the QueuePusher module [12]). RouteFlow is used for running routing protocols in the OpenFlow networks. The QueuePusher module (the source code of QueuePusher is available at: www.cityflow.eu) is used to set up queues for providing high QoS to the personalized flows.

Since the beginning of this project several platforms have been developed for SDN. In fact, even though during the development of this work the Floodlight controller was chosen due to its northbound API, nowadays, other controllers provide similar characteristics (e.g. OpenDaylight). Nonetheless, the conclusions presented by this paper are independent of these factors and could be verified with the latest SDN software.

In the proposed architecture, the communication between the components within the control plane is based on a RESTful interface, so that it can easily ported between different software solutions, while interactions with OpenFlow switches and the controller are supported by the OpenFlow and OVSDB (Open vSwitch Database Management) protocols [13]. Other supporting tools such as Pulse Generator and CityFlow's measurement system are also developed for enabling the scenarios presented in Section III. The pulse generator tool is developed to transmit high rate of control or data traffic to cover 1 million city population. The measuring tool is developed to gather the measurement data from the system.

C. Routing framework

As already explained, our framework relies on the path discovered by routing protocols used in the current Internet (to establish end-to-end QoS). For routing, we assume intra-AS routing protocols (such as Open Shortest Path First, i.e., OSPF and IBGP) to be run within an AS and inter-AS protocols (such as EBGP) to be run between different autonomous systems. For running these routing protocols in an SDN infrastructure, we use the RouteFlow framework [14].

The problem with RouteFlow is that it is not suitable for large-scale experimentation, as its configuration is not automatic. Before running RouteFlow, an administrator needs to devote a significant amount of time in configuration [15]. In the CityFlow work, we proposed a framework which configures RouteFlow automatically with a little manual configuration. In this framework, we run an additional controller module, which discovers the underlying network topology and then forwards configuration information to RouteFlow. The description of this framework is given in [15].

D. Queue Management

1) Controller Queue Management: Since the configuration of queues in OpenFlow Switches is not part of existing OpenFlow Controllers, the QueuePusher (shown in Fig. 2) was created with the intention of providing an interface to be exposed by an OpenFlow controller.

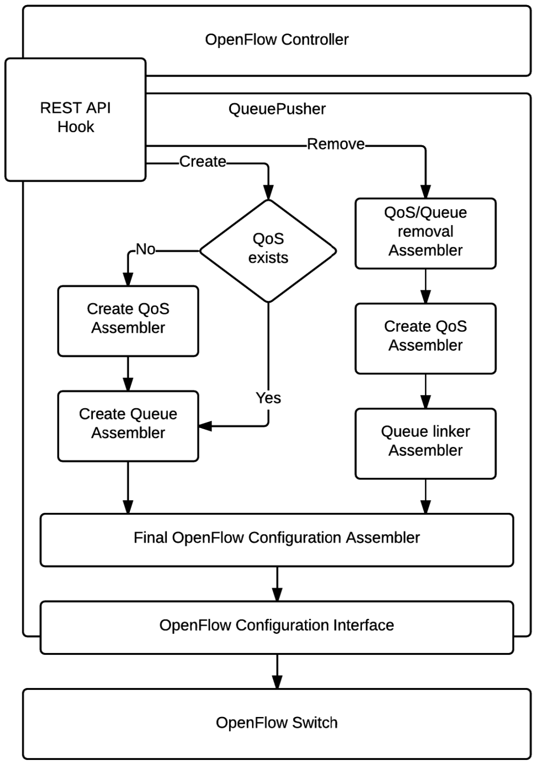

Fig. 3: Queue management from the controller side

The Queue management in the QueuePusher presents the following characteristics:

1) Modular implementation, easy to be embedded in any OpenFlow Controller
2) Able to support OVSDB, OF-Config, or other configuration protocols (currently only OVSDB RFC compliant)
3) Independent state-machine and communication handling

The QueuePusher architecture is based on a client/server model (shown in Fig. 3), where the role of the client is played by OpenFlow controller, whereas the server role is played by an OpenFlow enabled switch that receives the requested commands and configurations. It provides a comprehensive REST API for providing access to the queue management process. In addition to the general logic for queue management, the QueuePusher needs to process each request and assemble appropriate configuration messages, according to the user protocol. The QueuePusher handles the link between installed QoS entries and Queues. The QoS Assembler defines the type of QoS and bandwidth limits per port, and matches queues to their associated QoS entry. The Queue Assembler specifies the characteristics of the Queues to which flows can be associated through OpenFlow. Then, the Final OpenFlow Configuration Assembler validates all configurations before interfacing with OpenFlow switches.

2) VPS engine Queue Management: End-to-end resource management is handled by multiple VPS engines based on the protocols defined by IETF NSIS working group, which includes a signalling protocol for quality of service, the NSIS Signaling Layer Protocol (NSLP) for QoS Signaling [8]. This allows the support of multiple reservation models, being independent of the underlying QoS specification or architecture.

In addition to the QoS NSLP from NSIS, the used VPS engines take also advantage of the principles of the General Internet Signaling Transport (GIST) protocol [16], which can be used together with QoS NSLP to provide functionalities similar to RSVP and further extending it.

E. Operational Model for the Internet

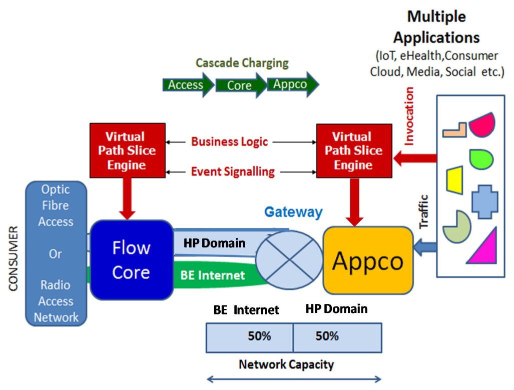

Fig. 4: Conceptual Operational Model for the Internet

A high level view of our proposed operational model is shown in Fig. 4. The essential idea is to segment the network capacity into a Best-Effort (BE) and a high priority (HP) domain. As capacity grows an operator can make a policy decision regarding the proportion to be allocated to the BE or HP domain. Initially, the HP domain might have a low share, but as demand grows, and BE becomes constrained, new capacity could be allocated to the HP domain. This can be implemented by using an aggregated queue in a gateway network element dimensioned for x% (can be 50%) of the capacity for traffic painted as BE using DiffServ code points. The remaining capacity (i.e., HP) can be painted as Expedited Forwarding or Assured Forwarding code points.

Content or application providers (appcos) with multiple applications in the area of Internet of Things, eHealth, Consumer and Cloud can use the slicing mechanism (queuing mechanism shown in Fig. 2)through VPS to obtain a slice of bandwidth in the HP domain across multiple AS domains and to the consumer connected via optic fibre access or 4G/5G radio access network. In exchange for obtaining guaranteed bandwidth to users, applications providers who will be able to drive new business models (e.g., 4K webTV) can expect to receive a charge for conveying the guaranteed traffic. This can be implemented using cascade charging on a wholesale basis, from access-to-core to applications provider.

The concept model in Fig. 4 is based on the separation of control from the data plane and the inclusion of features for business engagement. We add business logic and event signalling between the remote "appco" (application traffic source) and the flow core (at traffic sink) where the consumer is located. A gateway distributes application traffic on the BE or HP domain based on DSCP and/or MPLS EXP marking. The flow core allocates a discrete flow in the distribution and access network onwards to the consumer. While the consumer has a contract relationship (e.g., Netflix contract) with the appco. Cascade charging from the access to the core to the appco enables all infrastructure stakeholders in the traffic pathway share in the economic activity.

Our model works based on the requests from the user application to the VPS Engine as the user request determines the end points themselves. The OpenFlow controller, which in the case of the CityFlow project is Floodlight, tells the VPS Engine (c.f. Fig. 2 and Fig. 4) where those endpoints are connected in the data plane (i.e. the switch). RouteFlow by running BGP determines the output ports of the end points for delivering the user application. The VPS engine then replicates the existing best-effort flow rules and creates a new flow on the output port of the ingress switch. This new flow is 'painted' with the DiffServ code point for expedited forwarding. In parallel with this, a new flow is assigned to a queue that is given a scheduler rate corresponding to the bandwidth for the associated virtual path slice.

From a path point of view, the virtual path slice model follows the path determined by BGP between different autonomous systems and OpenFlow areas. The bandwidth of the slice is determined by the rate in the shaper of policer on the ingress switch. The model is relevant for a mixed topology including legacy IP routers and new OpenFlow switches. Connection Admission Control (also known as RACF – Resource Admission Control Function) is implemented at the ingress. A count is taken of the allocated capacity and a "busy tone" is implemented if a threshold is reached.

F. Billing Logic

Our framework has an internal business logic and which we believe will be ultimately adopted in OpenFlow networks. The recent agreement of the European Commission to accept value added services in the net neutrality debate is a key inflection point on this path [17]. The idea with billing is that the consumer pays wholesale cascade charging. This allows the charge to follow the traffic and ensures that all actors who provide forwarding infrastructure participate economically in the consumer service driving the traffic flow. This is something which does not happen in the Internet today. Retail billing which is focused on the consumer is out of scope. The API towards the application layer received the composite charge as the actor on the the application layer collects retail billing and makes a margin on the value added above the transport charges.

III. Reference Scenarios for experimentation

The CityFlow experimentation was defined, considering a target population of 1 million inhabitants, representing a mid-sized city as a reasonable and practical dimension – not too large and not too small – implementable on the OFELIA testbed. We have analyzed the network infrastructure of Brussels, population 1.1 million, in order to obtain reference scenarios for mobile, xDSL, and Fibre. Unfortunately, Brussels has currently no fibre access network deployed, thus in order to have a more future-proof reference network, we add Fibre-To-The-Home (FTTH) data from other European cities of similar size (e.g., Cologne). Starting from real data gives us a realistic scenario from which we can base our experiments. In reference scenarios, we use the ACG study [18] to design mobile, DSL, and FTTH networks for our experimental city, Flowville.

For mobile networks, we collect data from real sources i.e., BIPT (Belgian Institute for Postal Services and Telecommunications). According to BIPT, there are 958 base stations in Brussels. These stations provide wireless access to all users in the city. In addition, there are three large mobile operators: (1) Base, (2) Mobistar and (3) Proximus. For each operator, we consider a latest radio access technology, e.g., LTE. For each LTE, we place 958 base stations in the access rings of Flowville. According to the ACG Study, a maximum of 25 radio base stations operated over a ring can be connected with a pre-aggregation site. Therefore, there can be a maximum of 39 pre-aggregation sites (958/25=39) for Flowville. In addition, as there can be a maximum of 16 pre-aggregation sites per one aggregation site, there can be a maximum of three aggregation sites (39/16=3) in the aggregation network. Moreover, in order to connect these three aggregation sites to the core, we require two core locations [18].

In xDSL scenarios, DSLAMs (Digital Subscriber Line Access Multiplexers) are used to connect multiple customer Digital Subscriber Lines (DSL) to an aggregation network. Currently, Brussels has only VDSL (Video Digital Subscriber Line) technology and 59.29% of Brussels population use this technology. In our design, we assured that 90% of DSLAMs in Flowville are with 8 line cards and 10% of DSLAMS are with 3 line cards. Therefore, the number of DSLAMs required for Flowville is 1026, as one DSLAM line card can serve a maximum of 48 households [19]. As a DSL-based access/aggregation network is operating over rings, we can use similar architecture as the LTE case. Therefore, there can be a maximum of 42 pre-aggregation sites (1026/25=42) and three aggregation sites (42/16=3) for Brussels.

In FTTH scenarios, we assume that 20% of population will use FTTH connections (e.g Passive Optical Networks, i.e., PONs). PONs consist of Optical Line Terminals (OLTs) and Optical Network Units (ONUs). The OLT resides in the central office and the ONU resides in the customer's premises. For PON, there can be 48 subscriptions per OLT [20]. As there is an average of 2.06 persons per household, Flowville requires 11,518 OLTs to cover the total population. For 20% of population, Brussels requires around 2300 OLTs. As OLTs resides in the central offices and one central office can have up to 500 OLTs, we require 5 central offices to cover 20% of Flowville's population. If we consider 3% growth rate in FTTH adoption, we will require seven central offices. Assuming that one central office is directly connected with one pre-aggregation site, we require seven pre-aggregation sites and one aggregation site for Brussels.

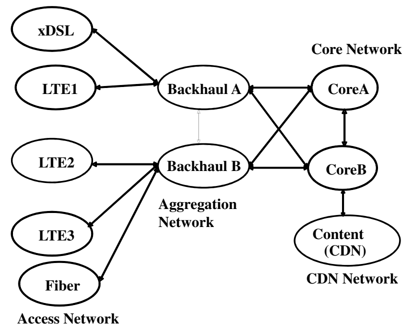

Fig. 5: CityFlow Reference city (Flowville, integrated model)

As a city can contain mobile, DSL, and FTTH networks simultaneously, we also present an integrated model that combines these networks. In Brussels, there are two operators in the Backhaul network: (1) Telenet (Backhaul-A in Fig. 5) and (2) Belgacom (Backhaul-B in Fig. 5). These operators connect the access network to the core network. Therefore, for the integrated model, we connect one LTE and one DSL or Fibre network to one of the Backhaul operators, and the remaining to the second Backhaul operator (Fig. 5). Here, LTE 1, LTE 2 and LTE 3 are the access network scenario from three mobile operators in Flowville. In this design, all core locations are connected with the CDN (content delivery network) servers. The goal of a CDN server is to serve content to end-users with high availability and high performance.

IV. Experimentation Methodology

A. Software used for experimentation

Due to emerging importance of SDN, a large number of OpenFlow switches and controllers are currently available. In particular, software-based switches such as Open vSwitch (OvS), Trafficlab 1.1, Indigo, CpQd could have been used for CityFlow experimentations. This also includes a large number of controllers such as Open DayLight, NOX, POX and Floodlight. As previously mentioned, Floodlight was chosen as the reference controller for the CityFlow project due to its RESTFul API and associated high performance, which was not rivalled by any other controller when this research work was conducted. Moreover, OvS was chosen as the reference OpenFlow switch implementation due to its production quality.

B. Topology setup on OFELIA

The OFELIA testbed has 10 different islands and has hardware OpenFlow switches (from NEC and HP) and other machines to deploy soft OpenFlow switches in a real or virtual machines environment. The iMinds island, which is located in Belgium, has resources to perform large-scale emulations and has the ability to emulate multiple AS experiments and most of the CityFlow experiments are performed on the iMinds island. However, one of the experiments is also performed on different islands in which some of islands worked like autonomous systems of the Internet.

1) Topology setup on the IMEC island: The iMinds island has a limitation that it has only 100 physical nodes with a maximum of 6 interfaces per node. Therefore, for the experiments, we converted the Flowville scenarios to an experimental setup (Fig. 6), which could be implemented in the iMinds island.

The access networks in the island are implemented by nodes USER1, USER2 and USER3. In these nodes, multiple access clients (the numbers are described in Section III) are emulated using virtual interfaces. There are three ASs (AS1, AS2 or AS3) that represent aggregation networks, one AS represents the core network and one AS represents the CDN network, running OvS for forwarding user traffic. Each of these ASs is connected with a separate Floodlight controller. Similarly, OvSs representing the aggregation networks and CDN AS nodes are also connected with the VPS Engine through the connected Floodlight controller. For load sharing, we use three VPS Engines (shown in Fig. 6). In addition, for bidirectional experiments (i.e., user to CDN and CDN to user), we duplicate the access and aggregate networks (Fig. 6).

2) Topology setup in multiple islands of the OFELIA testbed: For multiple islands experiments, we extended the Redzinc lab at Dublin to the OFELIA testbed. Therefore, the Redzinc lab also worked like an additional OFELIA island in our experiments. We configured 4 VMs at each of the following islands: TUB (Berlin), ETH (Zurich) and Dublin islands. Our objective was to setup an OpenFlow environment on the virtual machines of each island and to make virtual machines of each island behave like an autonomous system. For topology creation of the autonomous system, we needed to setup virtual interfaces on top the interfaces of the virtual machines in different islands. For creating virtual interfaces to create the required topology, we used Generic Routing Encapsulation (GRE) network tap (TAP) interfaces on the top of interfaces of virtual machines. Additionally, VPS, Floodlight and OvS are installed on the VMs of the islands.

C. Scale of test platform

Establishing very large topology emulation on OFELIA was a challenging task due to installation and running of software

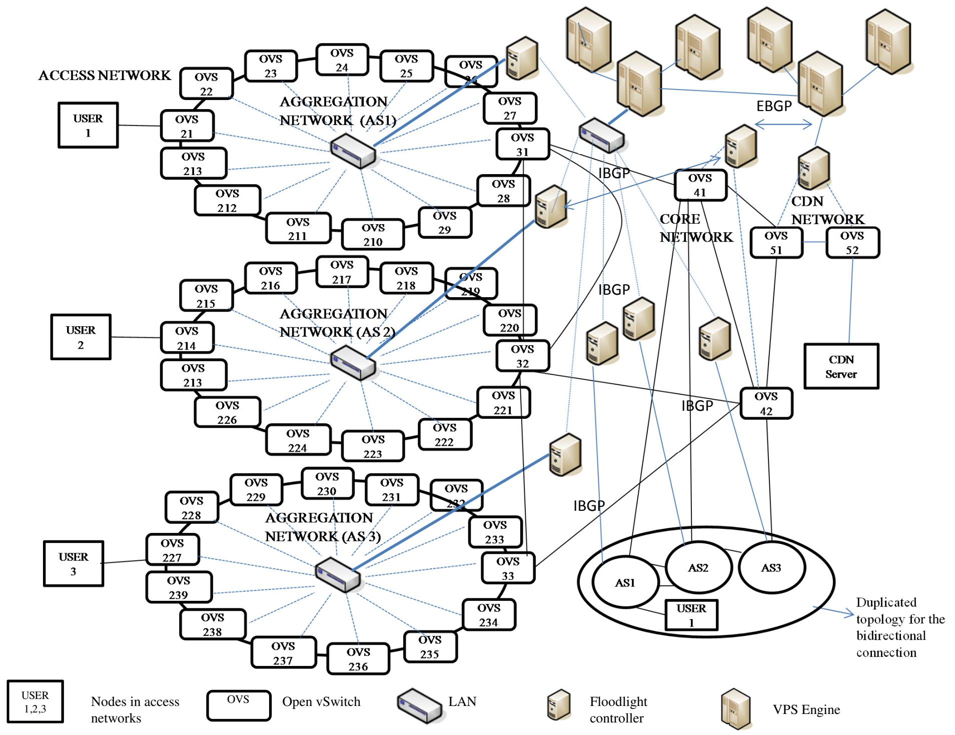

Fig. 6: Flowville for CityFlow experimentation on OFELIA

on many nodes, and collecting and parsing the debugging data. To overcome the challenge, we made Linux images in which OvS, Floodlight, RouteFlow, and VPS are already installed. In addition, we made scripts to automatically run all software in experiments of CityFlow. For collecting and parsing debugging data, we built a measurement system.

Another challenge for test platform was to configure RouteFlow. Before running RouteFlow, an administrator needs to devote significant amount of time in configurations. For a large topology (typically for 100 switches), it may take many hours to configure RouteFlow. To overcome the issue, we proposed and implemented a framework to automatically configure RouteFlow [15] (also discussed in Section II).

In order to perform a realistic experiment we implemented a test harness designed to provide high volume control plane (CP) traffic and to trigger appropriate traffic generators for data plane (DP) traffic. This allowed a stress test of the VPS server alone (by not enabling DP traffic) and of the whole CityFlow stack. We were able to supply the test harness with recipes for different traffic mixes; for example different rates of CP traffic where we could control the duration, inter-arrival time and magnitude of the traffic. We are also able to overlay more than one traffic recipe and be selective about which recipes trigger associated DP traffic.

V. Results

We performed four different experiments in CityFlow: (1) data traffic, (2) control traffic, (3) failure recovery, (4) multiple islands. The first three experiments are performed on the iMinds island, the fourth one is performed on multiple islands of OFELIA. Additionally, one of the control traffic experiment was also performed on the Amazon cloud facility.

A. Data Traffic Experiments

In order to understand how the data plane responds in different situations, using the setup previously presented, data traffic was rate-limited and forwarded through OpenFlow switches dynamically configured by the VPS. In particular, this experiment aims at emulating and analysing the performance of typical video streaming, shaped using VPS.

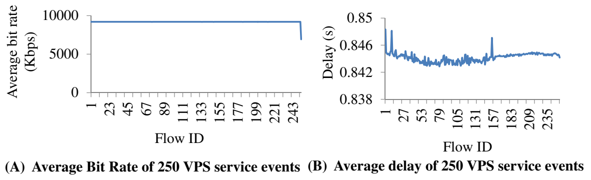

Fig. 7: Data traffic experiments

Having considered two variations of the same setup, the first version of the experiment measures the performance of traffic where 13.75Mbps of UDP packets, resembling a typical HD video stream, are sent from source to destination, being inserted into a queue configured with maximum bandwidth rate at 10Mbps for each of 250 invocations. The route of the flow is same as the route discovered by routing protocols in the current Infrastructure. The chosen limit for the created queue has around 30% less bandwidth, allowing the impact of this queue to be noticeable throughout the experiment. The purpose is to understand how traffic injected in both ways (unidirectional and bidirectional), such as interactive video between source and destination, impacts the overall system performance. The considered traffic pattern consisted the injection of 250 flows with a duration of 45s per flow. The results for measured bit rate and delay are shown in Fig. 7. Another important aspect of this experiment is creation and installation of queues and their associated flows (TABLE I), which imply the creation and deletion of 250 queues per receiving site (500 queues in the bidirectional scenario).

TABLE I: Data traffic experiment results

Average Queue Installation time	Average Queue Removal time	Source speed	Capacity	Throttled speed
48.3 ms	452ms	13.75Mbps	10Mbps	9.2Mbs

The obtained results revealed that the VPS engine was able to cope with the amount of requested invocations, efficiently issuing the respective queue and flow management. Moreover the process of creating or installing queues revealed to be quite efficient, taking on average less than 50 milliseconds. On the other hand, the queue removal or deletion process took unexpectedly more time (approx. 450ms, TABLE 1). After a close analysis we were able to conclude that this was due to internal OvS database verification for consistency, regarding the installed flows and queues for each QoS entry. Regarding the flow management operations there was no significant limitation or variation, presenting a very good performance in data traffic scenarios. These results also showed that expected traffic shaping introduced by create queues successfully limits the amount of transmitted data. The registered average value is around 9.2Mbps, which is lower than the expected 10Mbps.

B. Control traffic experiments

For this experiment, a city of 1 million inhabitants was considered and assumed that the service provider has a penetration of 20% in such a market, giving a possibility of 200,000 users. Two busy periods during the day were considered: a mid-morning period driven by enterprise traffic and an early evening period driven by domestic traffic (2 hours of duration each). In the busy period, it was considered that 75% of the users are active. We consider two cases in this experiment: (1) baseline and (2) expansive. In the baseline case we consider that each customer demands 1 event during the busy hour. This baseline case equates to a requirement to handle 75,000 events during a busy hour. In the expansive case we consider that each customer demands 3 events during a busy hour. Therefore, it needs to handle 225,000 events in a busy hour.

We then consider what engineering headroom is needed for expansion. In a voice network today, service growth is low as voice is a mature service, so the systems operate with a low headroom, but in the Internet with the rapid arrival of new services, growth can be quite fast. So in the expansive case we consider a headroom factor of 2 (i.e., for 450,000 events).

We emulate the baseline case in the iMinds testbed. On the other hand, in the Amazon cloud facility, we increase the number of invocations and show that how many invocations can be handled by the VPS Engine for the expansion case.

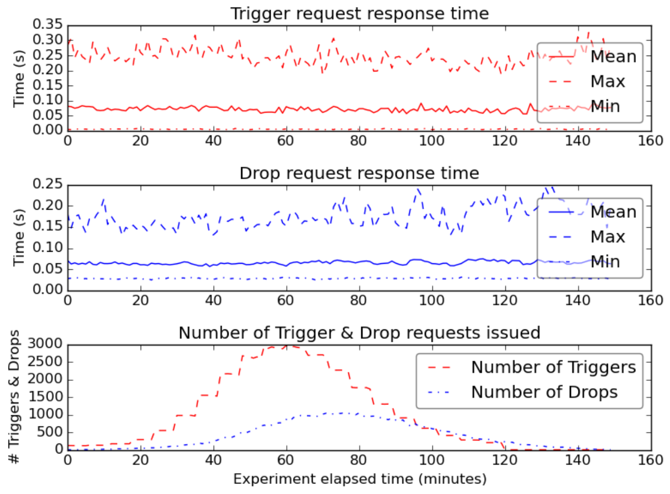

Fig. 8: 75000 Busy Hour Flow Invocations per hour

1) Experiment on the iMinds testbed: With the purpose of assessing the VPS controller, and its associated software stack performance, this experiment submits it to a high-load of signalling requests (i.e., mimics a high number of users). For this purpose a pulse generator was deployed to generate signalling pulses, in a two-hour interval (busy hour), distributed according to a Poisson distribution and with random service duration of between 3 minutes and 30 minutes.

The trigger request response time in Fig. 8 is the time to trigger allocation of resources for an invocation request, and the drop request response time is the time to delete the resources of an invocation. The results in Fig. 8 shows the minimum, average, and maximum value of the trigger and drop request time. The obtained results (Fig. 8) revealed that even under high-load, and for prolonged period, the VPS engine is capable of handling triggers in under 400ms for the baseline case (with the arrival of invocations in Poisson distribution). From the results, it is concluded that the VPS Engine can scale to a high volume of flow invocations and terminations, to support a busy hour flow invocation (BHFI) capacity of 75000 events on mid-range servers.

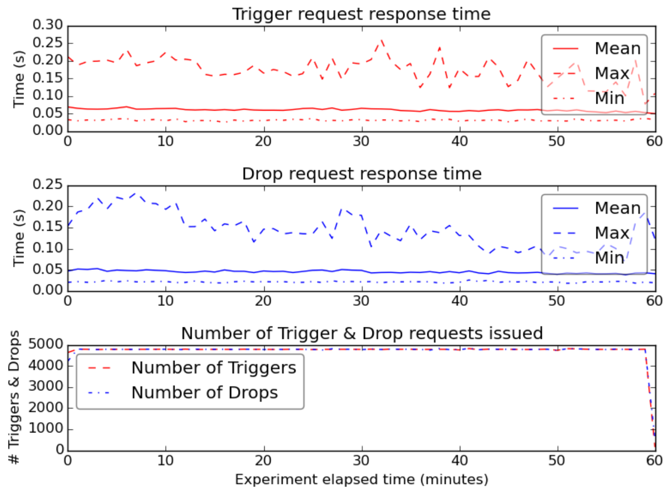

Fig. 9: High Volume Invocations on Amazon

2) *Experiment on the Amazon cloud facility:* The experiment on the Amazon cloud documents the scalability of high volume of invocation. In this experiment, the VPS engine showed the capability to handle up to 4800 requests per minute (see Fig. 9). That is, up to a total of 288000 requests in a one-hour period that would constitute a Busy Hour, well above the estimated 75000 requests in the baseline situation. The results show a BHFI capacity of 288000 on one server, which is 64% to our target of 450,000 in the expansion situation. This number, however ultimately depends on the performance of the associated OpenFlow controller and network hardware.

C. Failure recovery experiments

For failure recovery experiments, we implemented a framework [22] with which high quality of service can also be achieved in failure conditions. In this framework, under failure conditions, the controller reroutes traffic to a failure-free path gained through BGP. Regarding the implementation, we did not focus on fast failure recovery [23], but instead, we focus on scenarios in which high-priority should be provided to high-priority over best-effort users. We conducted emulations on the iMinds islands with different rate of traffic and one of the links between aggregation and core networks is failed (or made down). With results (Fig. 10), we were able to conclude that when there is enough bandwidth of the failure recovery path, neither high-priority nor best-effort traffic gets affected after re-routing traffic to a failure-free path. On the other hand, when there is limited available bandwidth, best-effort traffic experiences packet loss in order to meet the requirements of high-priority traffic. Finally, when the total amount of bandwidth is insufficient even for high-priority traffic, despite registering some losses it still maintains its priority above best-effort traffic, which causes no interference.

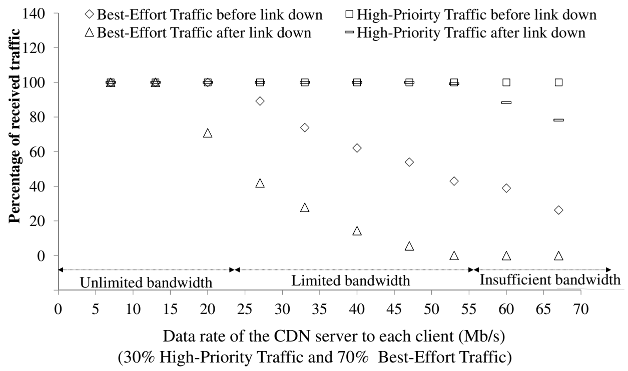

Fig. 10: Failure Recovery Experiment Results

D. Multiple island Experiment

We performed experiments for the multiple island scenarios described in Section IV.

For this, we calculated response time (time taken to response a user request), CPU usage, and memory usage of VPS. All the tests used the ETHZ island as the source, having the services triggered by the test harness from a user machine installed on the aforementioned island. To validate the experiments, two tests were performed: (1) triggers originating from the ETHZ island were performed, having the Dublin island as destination during 1 hour duration; (2) triggers originating from the ETHZ island were performed, having the TUB island as destination during 1 hour.

TABLE II: VPS machine CPU usage and memory usage

Indicator	Dublin island (Average)	TUB island (Average)	ETH Zurich (Average)
CPU usage	33%	40%	35%
Memory usage	21%	15%	41%

TABLE II presents the CPU usage and memory usage of the VPS machine during the span of experiments.

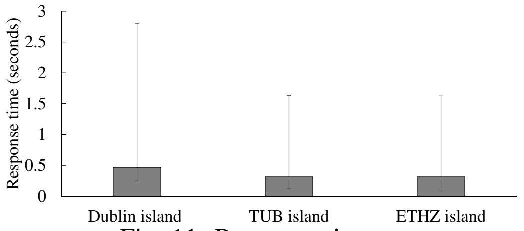

Fig. 11: Response time

Fig. 11 shows the response time in different islands. It shows that the response time in these islands is significantly longer than the iMinds islands (see results in previous subsections). Due to the fact that the machines present on these OFELIA islands, as well as the Dublin island, are underpowered (i.e., virtualized) when compared with the machines on iMinds island could justify the increase in response time. Additionally, the results obtained against TUB and ETH islands show a small difference between them.

VI. Conclusion

In this paper we have presented the CityFlow project's proposal of dynamic OpenFlow-capable public networks, capable of addressing the challenge of the current and future Internet data-traffic growth. Large-scale experiments for a city of one million inhabitants were performed in order to demonstrate the feasibility of this proposal, considering scenarios involving multiple autonomous systems. These experiments were undertaken on multiple islands of the OFELIA testbed emulating different technology networks.

Based on the obtained results, we have confirmed that an OpenFlow network with VPS coordination between multiple autonomous systems is feasible and is able to achieve all of the proposed objectives. These results motivate future research (e.g., on other testbeds such as FIRE [24]), upgrading the used network emulation platform, based on OpenFlow software-switches, into a a production network for validating the proposed architecture with OpenFlow-enabled hardware switches. Moreover, this conclusion further reveals that support of quality of service queues on commercial OpenFlow switches would be required in the future.

Acknowledgement

This work was partially funded by the FP7 CityFlow project (agreement no. 317576) and CleanSky (no. 6607584).

References

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

[2] S. Blake, S. Blake, F. Baker, and D. Black, "An Architecture for Differentiated Services," Internet Engineering Task Force, RFC 2475, 1998

[3] A. Meddeb, "Internet QoS: Pieces of the Puzzle," IEEE Communications Magazine, vol. 48 (1), pp. 86-94, 2010.

[4] N. McKeown, T. Andershnan, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," ACM Computer Communication Review, Vol. 38(2), pp. 69-74, 2008

[5] OpenFlow specification: https://www.opennetworking.org/

[6] T. Braun, M. Diaz, J. E. Gabeiras, T. Staub,"End-to-End Quality of Service Over Heterogeneous Networks," Springer,

[7] M. Sune, L. Bergesio, H. Woesner et. al., "Design and implementation of the OFELIA FP7 facility: the European OpenFlow testbed," Computer Networks, Vol. 61, pp. 132-150, 2014

[8] R. Hancock, G. Karagiannis, J. Loughney, S. Van den Bosch, "Next Steps in Signaling (NSIS): Framework", RFC 4080, IETF, 2005

[9] TMForum: https://www.tmforum.org/

[10] C. Sieber, A. Blenk, D. Hock, M. Scheib, T. Hhn, S. Khler, W. Kellerer, "Network configuration with quality of service abstractions for SDN and legacy networks," IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 1135-1136, 2015

[11] C. Thorpe, C. Olariu, A. Hava and P. McDonagh, "Experience of developing an openflow SDN prototype for managing IPTV networks," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, 2015, pp. 966-971.

[12] D. Palma, J. Goncalves, B. Sousa, L. Cordeiro, P. Simoes, S. Sharma, D. Staessens, "The QueuePusher: Enabling Queue Management in OpenFlow," EWSDN, 125-126, 2014

[13] B. Pfaff and B. Davie, "The Open vSwitch Database Management Protocol," RFC 7047, 2013

[14] C. Esteve Rothenberg, Marcelo R. Nascimento et.al, "Revisiting Routing Control Platforms with the Eyes and Muscles of Software-Defined Networking," HotSDN, Helsinki, Finland, Aug 2011

[15] S. Sharma, D. Staessens, D. Colle, M. Pickavet, P. Demeester, "Automatic configuration of routing control platforms in OpenFlow networks," ACM SIGCOMM Computer Communication Review, Vol. 43(4), pp. 491-492, 2014

[16] H. Schulzrinne and R. Hancock, "GIST: General Internet Signalling Transport", RFC 5291, 2010

[17] N. Tocci, "Imagining Europe: Towards a More United and Effective EU," Edizioni Nuova Cultura - Roma, 2014

[18] M. Kennedy, "A TCO Analysis of Ericsson's Virtual Network System Concept Applied to Mobile Backhaul," ACG Research Inc., 2012

[19] J. P. Pereira, "Telecommunication Policies for Broadband Access Networks", in Proceedings of the 37th Research Conference on Communication, Information and Internet Policy, pp. 1-13, 2009.

[20] M. V. Wee, K. Casier, K. Bauters, S. Verbrugge, D. Colle, M. Pickavet, "A modular and hierarchically structured echno-economic model for FTTH deployments," ONDM, pp. 1-6, 2012

[21] B. Sonkoly and A. Gulys and F. Nmeth and J. Czentye and K. Kurucz and B. Novk and G. Vaszkun, "On QoS Support to Ofelia and OpenFlow", EWSDN, pp. 109–113, 2012

[22] S. Sharma, D. Staessens, D. Colle, D. Palma, J. Goncalves, R. Figueiredo, D. Morris, M. Pickavet, P. Demeester, "Implementing quality of service for the software defined networking enabled future internet," EWSDN, pp. 49–54, 2014

[23] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and Piet Demeester, "In-band control, queuing, and failure recovery functionalities for OpenFlow," IEEE Network, Vol. 30(1), pp. 106-112, 2016

[24] M. Berman, P. Demeester, J. Woo Lee, K. Nagaraja, et. al., "Future internets escape the simulator", Communications of the ACM, Vol. 58(6), pp. 78–89, 2015