Experience of Developing an OpenFlow SDN Prototype for Managing IPTV Networks

Christina Thorpe, Cristian Olariu, Adriana Hava, and Patrick McDonagh

Performance Engineering Lab, School of Computer Science and Informatics, University College Dublin
Email: firstname.lastname@ucd.ie

***Abstract*—IPTV is a method of delivering TV content to end-users that is growing in popularity. It is a paid service, hence the implications of poor video quality may ultimately be a loss of revenue for the provider. Consequently, it is vital to provide service monitoring and reconfiguration mechanisms to ensure quality requirements set out in service level agreements are upheld. This paper describes our experience of building an IPTV Software Defined Network testbed that can be used to develop and validate new approaches for service assurance in IPTV networks. The testbed is modular and many of the concepts detailed in this tutorial may be applied to the management of other end-to-end services.**

Keywords—Software Defined Networking, Monitoring, IPTV

I. Introduction

As the business interests involved in assuring high Quality of Experience (QoE) for real-time, end-to-end services increase, an emphasis is placed on the efficient and effective management of the underpinning network. Traditional network configuration techniques, based on distributed devices making locally optimal decisions, are no longer adequate for these complex environments. New centralised mechanisms are needed to bridge the gap. Software Defined Networks (SDNs) are ideal candidates for managing real-time services: they provide a holistic view of the entire system via a centralised controller that can make smarter automated reconfiguration decisions using monitoring data received from the switches. Multiple controllers can be linked to increase the scalability of the management solution, and federations between multiple domains can be supported (e.g. core and access network domains).

Television is evolving to include less traditional methods of service delivery. Satellite, cable, and terrestrial links have long been the de facto delivery methods, however, providers have recently begun leveraging the Internet Protocol (IP) to facilitate a new TV delivery service called IPTV. IPTV differs significantly to classic Internet video: IPTV is a paid subscription for the live broadcast of many TV channels as apposed to a free download of one video. It is not possible to buffer each channel before playback as subscribers are not willing to wait several seconds every time they switch a channel.

Delivering high QoE is critical to maintaining customer relationships, therefore, monitoring IPTV is a key concern. Typical IPTV networks will deliver video content continuously to thousands of viewers simultaneously, thus, monitoring and reconfiguring such systems is a complex problem. SDN for IPTV network management is a rich research area; there is a lot of potential to contribute via new metrics and mechanisms to detect and overcome video quality issues within the core network. Several proprietary, media-aware, network solutions are available from different network hardware vendors (e.g. Cisco ASR 9000), however, the cost of these systems is very prohibitive - particularly for research purposes. OpenFlow (OF) is an open standard for SDNs that was developed by a research group in Berkeley. It contains many powerful features for network monitoring and reconfiguration and is extensible to support new Key Performance Indicators (KPIs), algorithms, etc. It is open source and can be installed on any linux operating system, offering a vendor-neutral solution. McDonagh et. al. details our initial work on enabling service assurance in IPTV networks using OF [1]; it proposes OF as a good solution for IPTV management but does not include an integrated IPTV SDN. This paper details our experience of developing an OF SDN prototype for managing IPTV networks. Existing metrics such as the Media Delivery Index (MDI) [2] and Media Discontinuity Counter (MDC) were implemented in the SDN to demonstrate the detection of poor video quality in the core network. A new mechanism was developed to locate the root cause of video quality issues. Finally, a new routing algorithm was developed to reconfigure the network to improve video quality.

The remainder of this article is organised as follows: Section II provides some technical details about OF. Section III discusses the various components of the IPTV SDN Prototype built in this work. Section IV focuses on the modifications made to the OF standard to implement the IPTV SDN. Section V details the modifications made to the Floodlight controller to support the IPTV solution. Section VI details an experiment used to validate the prototype. Finally, Section VII concludes this experience paper.

II. OpenFlow (OF)

SDNs facilitate flexible and innovative environments: network functions can be virtualised and new metrics and protocols can be implemented to target specific applications, enhancing the performance of the system and ultimately the QoE for the end users [3]. The SDN paradigm is based on the ability to program a network i.e., the ability to run third party code on a networking device to dictate the behaviour of the control plane [4]. In an SDN, the physical coupling between control and data planes, seen in traditional networks, is removed. The intelligence and decision making is taken from the decentralised data plane and placed in a smart centralised controller, which has a global view of the entire network. Monitoring and state data is collected by the controller and used to make configuration and routing decisions, which are pushed back down to the switches. The logic in the controller can be targeted for specific application data or services traversing the network. For example, different routing algorithms can be implemented in separate, but compatible, modules for voice, video, or data services. OF is a programmable network protocol, in the form of an open standard, that allows researchers and developers to add network functionality and run experiments, without needing to know the internal workings of hardware network devices. OF Switches (OFSs) are composed of one or more flow tables. The OF Controller (OFC) imposes policies on the switch flows. Flows contain a set of packet fields that are used to match flows, and an action (send-out-port, modify-field or drop) used to process the packet.

A. OF Messages and Processing

The OF protocol consists of several messages used to define the behaviour of the network (see Table I). OF has also an OFPT_ERROR message, which is always raised with the error type and code. An OF

978-3-901882-76-0 @2015 IFIP

Name	Description
OFC → OFS	
OFPT_HELLO	Says 'Hello' with supported OF version
OFPT_FEATURES_REQUEST	Queries available ports
OFPT_SET_CONFIG	Queries flow expirations
OFPT_FLOW_MOD	Requests to add entry in flow table
OFPT_PACKET_OUT	Sends a packet out on port(s)
OFS → OFC	
OFPT_HELLO	Says Hello to the OFC with OF version
OFPT_FEATURES_REPLY	Sends a list of features
OFPT_PACKET_IN	Sends a missed packet to OFC
OFPT_FLOW_EXPIRED	Informs OFS that a flow expired

TABLE I. OF MESSAGES

switch has several flow tables; when a packet is received, it enters the processing pipeline (Figure 1).

If the packet matches an entry in the first flow table, the associated instruction is looked up: (1) The packet can be modified with an 'apply-action' instruction that applies a specific action immediately. (2) The action set (initially empty) can be updated with a 'write-action' or 'clear-action' instruction. The action set can only contain a maximum of one action of a each type. (3) The metadata is updated with a 'write-metadata' instruction. (4) The packet can be sent to a following table with a 'goto' instruction (where the process starts again). (5) If there is no 'goto' instruction, the action set is executed and the packet leaves the switch. (6) If it doesn't match the table, the packet can be sent to the controller or be dropped (depending on the switch configuration).

There are several different OF actions. Some of them are mandatory in all switch implementations: (1) Output: forward to a specified port, it must support the forwarding to reserved virtual ports: (1a) ALL: send the packet out on every port except the ingress port and ports which have a non-forwarding rule. (1b) CONTROLLER: send the packet to the controller. (1c) TABLE: send the packet to the first flow table to be processed. (1d) IN_PORT: send the packet out the ingress port. (2) Drop: drop the packet. (3) Group: process a packet through the specified group. All other actions are optional.

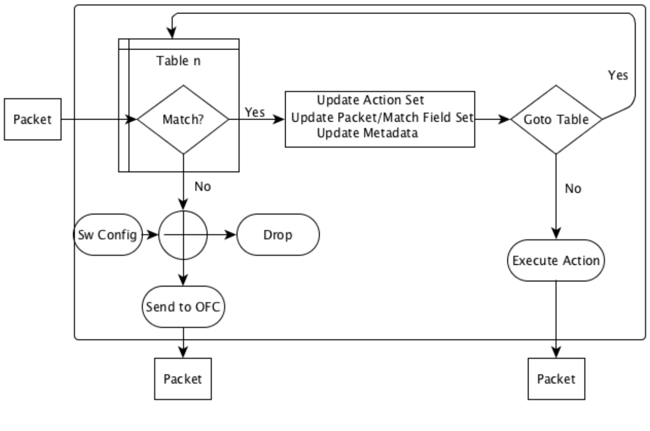

Fig. 1. OF Processing Pipeline

B. OF Controllers

There are many different OFCs available for use; they have different feature sets, different programming languages, and different interfaces. NOX (http://www.noxrepo.org) was the first controller software developed for OF. It is written in C++ and Python and supports OF 1.0. It is targeted at recent Linux distributions (particularly Ubuntu 11.10 and 12.04). POX (http://www.noxrepo/pox) is the brother of NOX, written in Python (requires Python 2.7) with full support for OF 1.0 and partial support for OF 1.1. POX is a command line tool with no native web interface, however, it can be used with third party interface software such as PoxDesk (https://github.com/MurphyMc/poxdesk/wiki/Getting-Started), DjangoFlow (https://github.com/carlio/django-flows). Ryu (http://osrg.github.io/ryu) is also written in Python and requires multiple python libraries to be installed. It uses an Apache 2.0 license and fully supports OF versions 1.0. 1.2, 1.3, 1.4 and Nicira Extensions. Beacon [5] is written in java; it is licensed under GPL v2 and supports OF version 1.3, however, it is incompatible with mininet v2.0. Trema (https://github.com/trema/trema) is written in Ruby and C; it is licensed under GPL v2 and supports OF version 1.0. It has several requirements to install. With Trema network DSL, the network topology can be defined (does not require an external virtualisation environment such as mininet [6]). Opendaylight (http://www.opendaylight.org) is written in java and supports OF versions 1.0 and 1.3. It is more than a simple controller; it is an open platform which includes a fully pluggable controller, interfaces, protocol plug-ins and applications. Floodlight (http://floodlight.openflowhub.org) is an OF controller written in Java that supports OF version 1.0. A Web UI is included in Floodlight, enabled by default, and accessible on http://localhost:8080/ui/index.html The latest version, Floodlight+, supports only OF 1.3, but it has all the features of the previous OF versions. As of now, the Floodlight+ controller does not work with OF 1.0 and OF 1.3 switches at the same time.

An experiment was conducted to compare the performance of the various different controllers with a basic L2-switch controller and a linear topology on mininet (with Open Virtual Switch (OVS)-like switches). A linear topology with 2 hosts was created with the following command in mininet:

```
    sudo mn --topo linear,2 --switch ovsk
--controller remote
```

The number of OFSs and hosts in the network was incrementally increased from 2 to 40, and iperf was used to generate network traffic for 10s between the first and last virtual host in the linear topology. The throughput achieved is related to the performance of the OFC, i.e., how quickly it can process 'packets in' and push flow entries back to the OFSs. Figure 3 shows the results of the OFC performance test, graphing the iperf throughput. Opendaylight does not feature in the results because network discovery took too long (e.g., iperf initiation took 15 minutes for 5 switches), and up to 40 switches were tested in this experiment. Ryu crashed with 40 hosts and switches when requesting iperf. Results show that Floodlight is a good option: it achieves the best performance in terms of network throughput (marginally), particularly when the number of switches and hosts is greater than 20; it also has an extensible web UI.

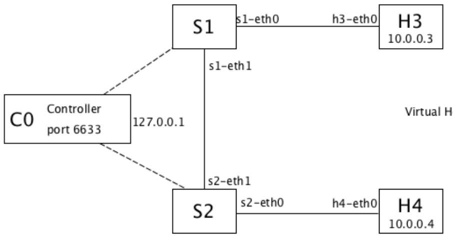

Fig. 2. Controller Performance Experiment Topology

III. OF IPTV PROTOTYPE

A. IPTV SDN Monitoring Architecture

Figure 4 illustrates a typical IPTV delivery network architecture. It is comprised of an IPTV content source (Super-Head End - $SH\#1$), which is the originating point of the IP video stream. Multiple such sources can feed the IPTV content into the Core Network (CN). The CN is not limited to only serving SHs, it can serve content from Video-On-Demand (VOD) servers, local content, or other non-IPTV services. The CN is comprised of two different types of routers: Edge Routers (ER) and Core Routers (CR). One important feature of the CN is the provision of path redundancy for failure

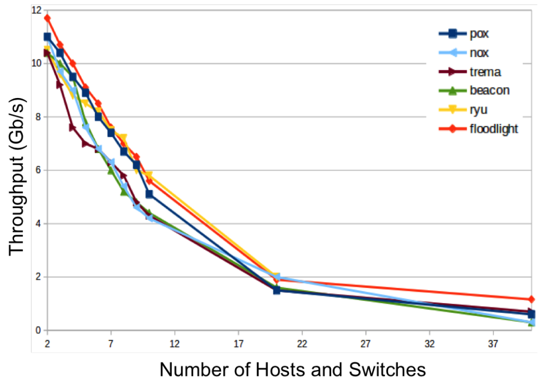

Fig. 3. OFC Performance Experiement Results

scenarios. The subscribers IPTV flows traverse the core network via ERs and are dispatched to Set-Top-Boxes (STBs) through the Digital Subscriber Line Access Multiplexers (DSLAMs) and Home Routers (HRs). QoS metrics specific to video traffic can be obtained at any ER, CR, or HR. For this work, all CRs and ERs are OF enabled (connected to the OFC) and can be media aware; they are built with packet/flow inspection to perform QoS monitoring. The metrics are obtained by inspecting each video packet for its IP, UDP and MPEG-TS headers. Per-flow metrics are recorded and pushed to the OFC, where the IPTV provider can monitor the overall system's performance. OF devices can also monitor other flow, port, queue, and device level statistics.

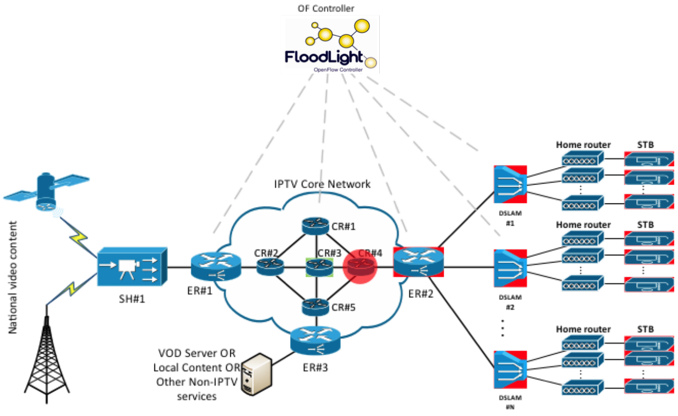

Fig. 4. IPTV SDN Monitoring Architecture

B. OF Detectable Errors

The OF reports, used as a basis for the following detectable errors, are referenced in the OF Specification [7]. Use cases that lead to poor QoE for IPTV subscribers (loss, congestion, failed node) can be broken down into simpler node or transmission line errors. The path between the video content source and end-users is comprised of forwarding nodes and links between them. The following is a list of the typical errors that can be detected using an OF-enabled IPTV delivery system. All these failures can occur more or less simultaneously, but they are at the core of any IPTV system failure:

1) Connection to the OFC Lost/Down: The OFC periodically sends 'keepalive' (using `OFPT_ECHO_REQUEST` and `OFPT_ECHO_REPLY` messages) packets to connected OFSs, in order to maintain connections. If one such connection is down, the OFC should report this event. The OFS should also detect the loss of connectivity and enter a fail-safe mode, pre-established at the installation time, or settings set by the CM/OFC when connectivity existed.

2) One or More OFS's Transmission Line(s) Down: The OFS should detect when an active transmission line (other than the link to OFC) is down. Such an error may occur when a cable was accidentally unplugged, or the transmission line has suffered physical damage. This error can be detected when the value of the field `ofp_port_state`, bit `OFPPS_LINK_DOWN` = 1.

3) One or More Ports Down: The OFS should detect when one of the ports of the physical interface is down. For example, if the port was taken down by the administrator, or a hardware error occurred. The `OFPPC_PORT_DOWN` bit should indicate that a particular port was administratively taken down.

4) Saturated System: The input packet rate per flow can be derived based on the flow table and the input port statistics. The output packet rate per flow can be derived based on the OF queue statistics. The ratio of the two rates should be 1. If the ratio >1, then it needs to be investigated whether the node has internal forwarding problems (system overload) or the output interfaces rate is low.

5) Poor Video Quality: Can be developed as an external module (IPTV Module/Video Quality Device); a copy of each IPTV packet (identified by its source and destination IP) is forwarded to this module for header inspection (e.g. continuity counter), while the original packet is forwarded to the appropriate output port. Using the experimenter module, an OFS can be extended to measure and report via the `OFPST_VENDOR` stats type.

IV. OF MODIFICATIONS

A. OF Node Model

Figure 5 illustrates the OFS node-model defined in this work. Entities like *ports*, *queue*, and *flow table* are natively supported by OF and a specific set of OF metrics exists for each of those. A separate module, IPTV Module, was developed to support new lightweight metrics for IPTV service assurance (e.g., MDI and continuity count). The OFC and OFS were modified to receive reports for event-based reporting. A separate dedicated device (Video Quality Device) was developed to support a new heavy-weight IPTV metric (subject of patent application and therefore not presented in this paper) to perform the calculation off the switch in order to reduce the impact on switch performance.

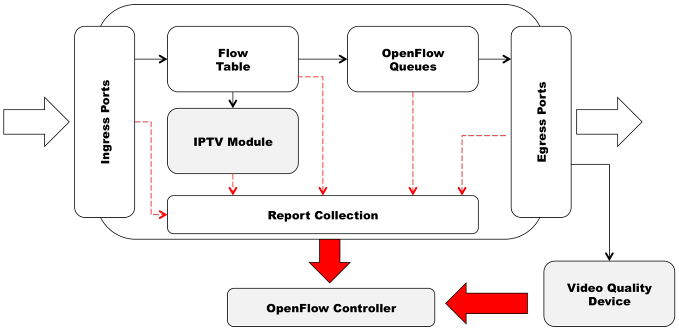

Fig. 5. OF Node Model

B. New Message Types

`OF VENDOR` synchronous messages are used to perform the communication between the OFSs and the OFC e.g., the `STATISTICS` messages with statistics 'type' set to `VENDOR`. Various different vendor type messages are used in the implementation of the IPTV monitoring solution:

1) Set Configuration Message: Sent from the OFC to the OFS; tells the OFS which IP (of the IPTV servers) to monitor.

2) Threshold Alert Message: This is sent from the OFS to the OFC to alert that a metric has exceeded the configured threshold.

When the OFC receives a message of this type, it assesses whether corrective action is possible/necessary.

3) Statistic Request/Reply Messages: The request message is sent from the OFC to the OFS to request a statistics reply (based on the flags). The reply message is sent from the OFS to the OFC to report statistics data.

C. Metric Implementation

OF metrics are implemented entirely in the OF Datapath module (responsible for table management, switch flow tables, and switch packet processing). The implementation of new metrics in the IPTV SDN involves the use of two parts of the OF protocol: Statistic message (for reporting metrics) and Packet/Flow (for calculating new metrics). The statistic message is comprised of three parts: (1) *Initialisation* - called when a message for enable statistic is received. (2) *Dump* - called after executing the initialisation. (3) *Done* - called after *dump* (to clean memory). Several modifications were made to the OF protocol to implement IPTV QoE metrics. These include:

Init : a small method was written to parse the body of the message; create a list of requests; and save them for the *dump* function. A call must be made to the *init* function (of the requested metric) to allocate memory for all the variables needed for calculating the metric.

Dump : the *dump* functionality takes the list of requests (from the *init* function) and executes them if it matches the conditions (ip src/dst, type of metric). If they match, the reply buffer is filled with the data (usually in the memory allocated by the *init* function) for this metric. All the metrics for one type can be requested with one request message; in this case, the type is checked to see if it is matching, and the data is dumped.

Done : The *done* functionality releases the memory allocated in the initialisation, when the metric is disabled.

Packet : The *Packet* function checks if the packet matches with the condition, and runs the Deep Packet Inspection (DPI) on this packet to calculate the metric(s). Here, the variable allocated in the *init* can be used to store some additional values.

Three new metrics have been implemented in the IPTV prototype, namely the Media Delivery Index (MDI) [2] Media Loss Rate (MLR) and Delay Factor (DF); and Media Discontinuity Count. When packets traverse a IP network to travel from source to destination, they may be subject to queues, routing, and switching, causing jitter. Buffering at the receiver is used to overcome the network jitter, however, the cost of having buffers is that they introduce delay. Furthermore, buffers are of a finite size, and excessive jitter will cause them to either overflow or underflow. Both of these events will negatively impact on IPTV user QoE [8].

The MDI uses jitter and delay to quantify QoE; it is calculated in two parts: the MLR and the DF. The MLR is simply defined as the number of lost or out-of-order media packets per second. The MLR is calculated by counting the number of RTP packets received in a switch (for each IPTV flow) to determine the difference in the sequence number between the first received packet and the last received. The MLR is the difference between the number of expected packets (defined by the sequence number of the final received packet) and received packets divided by a time interval (1 second), calculated by (1).

$$MLR = \frac{packets\_expected - packets\_received}{interval\_time\_in\_seconds} \qquad (1)$$

A simple experiment was set up to validate the MLR implementation in OF. The core network topology shown in Figure 4 is used, which consists of 3 ERs and 5 CRs. IPTV traffic is multicast from the SHE to a video client, via the core network. The route of the IPTV traffic is: SH#1 → E#1 → C#2 → C#5 → C#4 → E#2 → Client. Figure 6 plots the MLR results when loss is introduced into the network at various points: (1) 10% loss on link E#1 → C#2 (2) 20% loss on link C#1 → C#4 (3) 30% loss on link C#4 → E#2. The MLR increases proportionally for each percentage loss at each OFS in the IPTV route through the core.

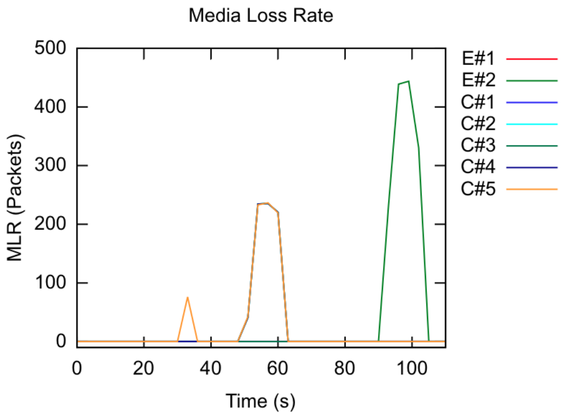

Fig. 6. Media Loss Rate Validation Results

The DF component of the MDI is a time value indicating how many milliseconds worth of data the buffers must be able to contain in order to eliminate jitter. For DF, the difference between the data sent and the data received is calculated at every packet arrival, as in (2):

$$\Delta = data\_sent - data\_received \qquad (2)$$

DF is defined by (3) and is recalculated for the packets received in a time interval of one second.

$$DF = max(\Delta) - \frac{min(\Delta)}{media\_rate}, \qquad (3)$$

where *media_rate* is the maximum bit rate allowable by the encoding process. In order to calculate Δ in OF, the formula was modified. Firstly, there is no trigger when the kernel receives a new packet, only when the packet is read from the kernel queue. In order to address this issue, the following variables were defined as in (4):

$$\begin{aligned} data\_sent &= \left\{ \begin{array}{r} Qp - Q + Pp, \text{ if Qp - Q} > 0 \\ Pp, \text{ if Qp - Q} \leq 0 \end{array} \right\} \\ data\_received &= \left\{ \begin{array}{r} Q - Qp + P, \text{ if Q - Qp} > 0 \\ P, \text{ if Q - Qp} \leq 0 \end{array} \right\} \end{aligned} \qquad (4)$$

where *Q* is the queue size in bytes, *Qp* is the previous queue size, *P* is the packet size and *Pp* is the previous packet size. Therefore Δ can be calculated by (5).

$$\Delta = Qp - Q + Pp - P \qquad (5)$$

Figure 7 shows the DF results of a second simple experiment designed to validate the OF DF implementation. The same topology (shown in Figure 4) is used, and the route of the IPTV traffic is: SH#1 → E#1 → C#2 → C#3 → C#4 → E#2 → Client. Loss is introduced into the network at different locations: (1) 20% loss on the link between E#1 → C#2 (2) 20% loss on link between E#1 → C#2 (3) 20% loss on link between C#4 → E#2. Results show that all OFSs on the path of the IPTV flow report an increase in DF when loss is introduced on an upstream link.

V. Floodlight Modifications

FloodLight is implemented as the OFC that manages basic system functions, such as socket connections and connection handshakes. It is extensible by using modules and the powerful REST API. Received packets, a list of all the switches in the network, and a list of different MAC address connected in the network, can all be accessed through the API. It is also possible to send any type of message to the switches using the API.

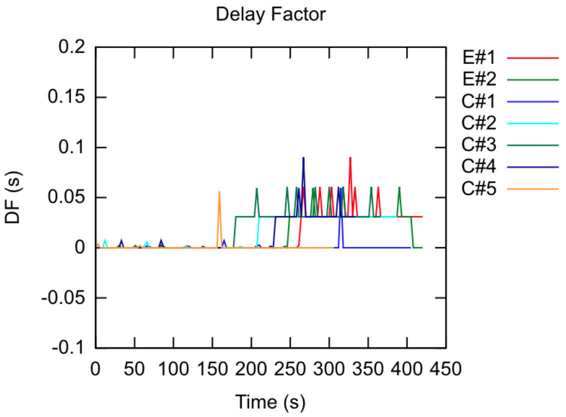

Fig. 7. Delay Factor Validation Results

A. *Statistics Module*

A new module was created to deal with all the statistics processing and to make requests to all the OFSs. The class inherits from the IFloodlightModule class, and implements a `receive()` callback to get OF messages. This class also creates a RequestThread that is responsible for sending `SET_CONFIGURATION` and `STATISTIC_REQUEST` messages to the switches.

B. *Floodlight Messages Implementation*

Several new messages were implemented to support an IPTV SDN: The `Set Configuration Message` is used to enable statistics monitoring on capable devices; it is implemented using 3 classes: (1) `PelVendorHeader` - inherits `OFVendor`, the OF vendor/experimenter message, and adds one integer field to the message so it is possible to have more than one type of vendor message using only one vendor ID. (2) `SetConfigurationMessage` - implements the message itself and holds a list of different `ConfigurationIPData` structures. (3) `ConfigurationIPData` - the structure holding each configuration of enabled statistics.

The `Statistic Request Message` is implemented in two classes: (1) `StatisticRequest` - inherits the OF protocol message `OFStatisticRequest`, and creates the body of the request using an instance of `StatisticRequestBody`. (2) `StatisticRequestBody` - inherits `OFVendorStatistics` and holds a list of `PelStatisticRequest`. Each `PelStatisticRequest` represents an individual request inside the message; it contains the following: (a) The statistic type being requested. (b) A bit to indicate whether the statistic should be computed for all the IPs, for a specific IP, or a range of IPs. (c) The source and destination IP address.

The `Statistic Reply` is implemented using a parser class `StatisticsReplyParser`, which takes a `OFStatisticMessageBase` and puts all the information from this message into a `StatisticReplyBody` structure type. Inside each reply body there is a statistic value that can have different sizes depending on the statistic type received.

C. *Floodlight REST API*

When started, Floodlight creates a webserver on the controller port 8080 and (using Restlets) creates services that let the user access information about the entire OF network. It also includes methods for flow programming, setting static routes and many other options [9]. Various new methods were implemented inside Floodlights rest API for configuring the switches, and extracting data for use in the user interface:

Method 1 URL : http://server/wm/iptv/switch/{idSwitch}/{enable}/type/{metricType}/opt/{ipSrc}/{ipDst}/{threshold}/json
Function : Send a `Set Configuration` message to the switch identified by {idSwitch}, enabling if {enable} = enable or disabling if {enable} = disable the statistics collection of the statistics type {metricType} between the {ipSrc} and {ipDst}.Set the threshold for this statistic as an integer value defined by {threshold}

Method 2 URL : http://server/wm/iptv/switch/{switchID}/stats/{statType}/json
Function : List of statistic values for the switch {switchID} of the type {statType} if {statType} = 0 then request all statistic values.

Method 3 URL : http://server/wm/iptv/switch/{idSwitch}/{enable}/type/{metricType}/opt/{threshold}/json
Function : Send a `Set Configuration` message to the switch identified by {idSwitch}, enabling if {enable} = enable or disabling if {enable} = disable the statistics collection of the statistics type{metricType} between all the IPs (wildcard on source and destination). Also set the threshold limit for this statistic as an integer value defined by {threshold}.

D. *New Reconfiguration Algorithms*

The three steps involved in a reconfiguration algorithm are: (1) problem detection, (2) problem location, and (3) corrective action:

Problem Detection : The three video metrics implemented in this work include MDC, MDI:DF, and MDI:MLR. Each metric can be enabled in the OFC for any IPTV flow, by specifying the source/destination IP and a metric threshold. The OFC sends a `Set Configuration` message to the OFS; as the IPTV flow enters each OF switch, all enabled metrics are calculated.

ProblemLocation : The testbed supports both periodic and event based reporting of metrics. The decision algorithm utilises event based reporting. Each metric is compared to the stored threshold. If the value exceeds the configured threshold, a `Statistics Reply Message` is constructed on the OFS and transmitted to the OFC. Reports are transmitted out-of-band and therefore, it is assumed that the probability of a lost report is negligible. When the OFC receives the `Statistics Reply Message` from an OFS, it must identify the problematic link/device. When poor quality is detected in a OFS, all OFSs downstream will also detect and report the poor video quality. It is assumed that all reports are received by the controller (no loss). Therefore, the ingress link to the switch with the shortest distance (least number of hops) to the video server is identified as the problematic component.

CorrectiveAction : The key purpose of monitoring the OFSs is to implement intelligent corrective actions to ensure high QoE for end-to-end services. A routing algorithm was developed in this work and is used to reconfigure the network; it is described in the next section.

E. *Routing Algorithm*

When an OFS receives a packet whose source, destination, and port information is unknown, it sends a ’packet-in’ to the OFC. The controller then calculates the best route using Djikstras Algorithm (DA) [10], and the flows are configured in all the OFSs. DA is a shortest-path/lowest cost routing algorithm that assigns a cost/weight on each edge. The shortest path is then calculated from each vertex (node) to all other vertices in the graph. The mechanism of DA can be utilised to determine the optimal route/path through a network in terms of video quality. The idea is that when a link in a network becomes problematic (causes poor video quality), the cost of the corresponding edge on the graph can be increased in the attempt to find a new route that avoids the problematic link (cost function is currently the subject of a patent application process).

In order to implement this behaviour, the native forwarding module in Floodlight was disabled, and another module was created to represent of the network as a graph and to calculate the best path.

The reactive routing algorithm monitors `LINK_DOWN` events in the network. Every time an OFS receives this notification for one of its ports, all the flows for this OFC, and flow entries with same match (IP source, IP destination, source and destination port) in the entire network are deleted. This is to ensure that when the next packet arrives, the new optimal route is calculated. If the optimal route does not include the OFS that generated the 'packet-in', the data contained in this packet is injected as a 'packet-out' in the last OFC of the path, preventing any loss.

In the case where no route is valid, a packet flood is created so the OFS that sent the packet-in to the OFC will send the same packet to all output ports (except the one where the packet was received). This is a solution for scenarios where the network topology was changed and the OFC did not have enough time to learn the new topology: the message is sent to the neighbours to see if the new topology was learnt. The drawback to this mechanism is that it can eventually cause packet duplicates, but it will suppress packet loss.

When an OFS sends a threshold report to the OFC, the OFC stores it in a HashMap. If more than two threshold messages have been received from an OFS within 5 seconds, the OFC will send `DELETE_FLOW` messages to all the matching flows that are passing through the OFS that sent the threshold message. This will force the next packets with those matches to be rerouted in the network using DA with modified weights.

VI. Network Reconfiguration Experiment

A third experiment was conducted to validate the new reconfiguration mechanism in OF. The scenario considered is: a customer subscribed to an IPTV service is experiencing a low QoE. The route of the IPTV flow is: SH#1 → ER#1 → CR#2 → CR#3 → CR#4 → ER#2 → DSLAM[1-3] → Client (see Figure 4). The IPTV SDN Testbed was developed using three physical machines, including a powerful server and two laptops (Figure 8). All machines are running ubuntu 12.04.2, and the server has a multiport NIC. The core SDN is emulated on the server using mininet and the Floodlight OFC. The SHE is implemented on a laptop running live555 (http://www.live555.com), and the IPTV client is on a laptop running VLC (see Figure 8).

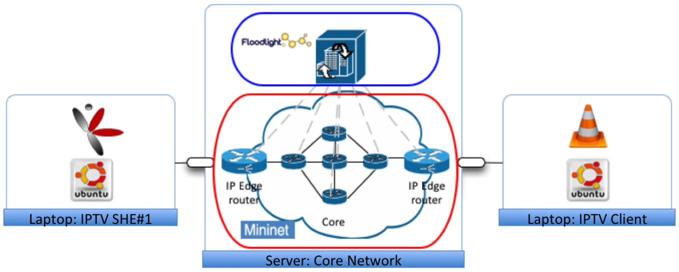

Fig. 8. Experimental Testbed Setup

Congestion is introduced to the core network. CR#4 reports that there has been loss/damage to a video flow (MLR value has exceeded configured threshold illustrated by in red in Figure 4). This is due to congestion between the core nodes CR#3 and CR#4. On receipt of the alert, the OFC updates the cost of each edge in the graph and recalculates the routes by running DA. The new optimal route is: SH#1 → ER#1 → CR#2 → CR#1 → CR#4 → ER#2 → DSLAM[1-3] → Client. The OFSs in the path of the IPTV flow are instructed to delete the relevant entries from their flow tables. The next time a packet from the flow arrives on the CR#2, it queries the new path by sending a 'Packet In' to the OFC. In this way, the IPTV flow is rerouted around the congested link and the video quality is restored (MLR value returns to 0, see Figure 9).

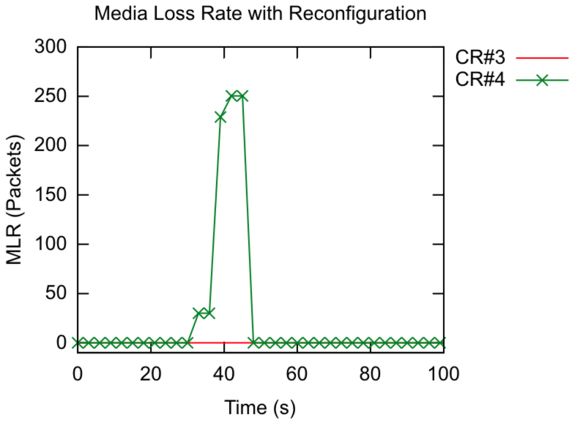

Fig. 9. Reconfiguration Validation Results

VII. Conclusion

This paper details the development of an OF IPTV SDN. It discusses the various technologies used to create a cost-effective prototype for research purposes and includes comprehensive description of new metrics and reconfiguration mechanisms implemented in OF and Floodlight. To date, several different novel mechanisms have been developed and validated on the testbed, and are currently the subject of patent applications.

Acknowledgment

This research is supported, in part, by Science Foundation Ireland (SFI) via grant 08/SRC/I1403 FAME SRC (Federated, Autonomic Management of End-to-End Communications Services - Strategic Research Cluster, and in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie). The contribution of the following software developer interns is gratefully acknowledged: Emmanuel Rieg, Filipe Caldas, Julien Langlois, and Alexis Dumas.

References

[1] P. McDonagh, C. Olariu, A. Hava, and C. Thorpe, "Enabling iptv service assurance using openflow," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*. IEEE, 2013, pp. 1456–1460.

[2] J. Welch and J. Clark, "Rfc 4445: A proposed media delivery index," *Massachusetts: Network Working Group, IETF*, 2006.

[3] C. Monsanto, J. Reich, N. Foster, J. Rexford, D. Walker *et al.*, "Composing software-defined networks," in *USENIX NSDI*, 2013, p. 2.

[4] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 2, pp. 909–928, 2013.

[5] D. Erickson, "The beacon openflow controller," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013, pp. 13–18.

[6] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.

[7] "Openflow specification simulation tool." [Online]. Available: http://archive.openflow.org/documents/openflow-spec-v1.1.0.pdf

[8] "Qoe iptv, understanding and interpreting mdi values," *Agilent Technologies*, 2006.

[9] "Floodlight rest api." [Online]. Available: http://www.openflowhub.org/display/floodlightcontroller/Floodlight+REST+API

[10] S. Skiena, "Dijkstra's algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225–227, 1990.