2014 IEEE 22nd International Conference on Network Protocols

Extending OpenFlow for Service Insertion and Payload Inspection

Robinson Udechukwu* and Rudra Dutta†
Department of Computer Science
North Carolina State University, Raleigh, NC, USA
Email: *rnudechu@ncsu.edu, †rdutta@ncsu.edu

***Abstract*—Software Defined Networking (SDN) offers traffic characterization and resource allocation policies to change dynamically, while avoiding the obsolescence of specialized forwarding equipment. OpenFlow, a SDN standard, is currently the only standard that explicitly focuses on multi-vendor openness. Unfortunately, it only provides for traffic engineering on an integrated basis for L2-L4. The obvious approaches to expand OpenFlow's reach to L7, would be to enhance the datapath flowtable, or to utilize the controller for deep packet inspection, both introduces significant scalability barriers.**

We propose and prototype an enhancement to OpenFlow based on the idea of an External Processing Box (EPB) optionally attached to forwarding engines; however, we use existing protocol extension constructs to control the EPB as an integrated part of the OpenFlow datapath. This provides network operators with the ability to use L7-based policies to control service insertion and traffic steering, without breaking the open paradigm. This novel yet eminently practical augmentation of OpenFlow provides added value critical for realistic networking practice. Retention of multi-vendor openness for such an approach has not been previously reported in literature to the best of our knowledge. We report numerical results from our prototype, characterizing the performance and practicality of this prototype by implementing a video reconditioning application on this platform.

I. Introduction

Internet usage has exploded within the last decade, with smart appliances, such as smartphones and tablets, becoming as ubiquitous as a toaster oven in a consumer's home. These devices offer users rich access to interactive network-centric content such as streaming high-definition video and access to real-time multi-player games. Companies often rely on specialized network appliances, known as middleboxes, in order to manage these new traffic patterns. The term middleboxes covers an umbrella of various devices (deep packet inspection, load balancer, gateway), all providing network managers with the ability to install new functionality into their network.

Traditional network switches and routers only look at a packet's header to determine the packet's route. Network vendors differentiate their products apart through proprietary mechanism for processing this data, creating a fragmented market where vendors market their network equipment based on the equipment's processing performance. Competition between vendors has driven innovation within networks, but the proprietaries and closed nature characteristics for these devices have also impaired network administrators abilities to introduce custom features for their network deployment.

Software defined network (SDN) paradigm, decouples a network device's control and data (forwarding) plane, essentially separating the device's mind from its body, while providing an interface for the two planes to continue communicating. The device's control plane (the mind), lies on a separate commodity hardware, known as a controller, where it delegates the actions for the network device. OpenFlow, a SDN protocol provides an open interface, which defines the structure and mechanics for the intercommunication between the decoupled planes. The data plane is responsible with forwarding/dropping traffic as specified by the controller. SDN encourages rapid development cycles and reduction in capital and operating expenditures (CapEx/OpEx) as network functions shift from purpose-built network elements to commodity hardware [1]. With SDN, companies can have their network managers program new features within commodity hardware rather than wait for network vendors to introduce it. OpenFlow performs traffic engineering using a match-action paradigm based on Layer 2 - 4 header information [2][3]. An OpenFlow match does not look into the Application header, leaving out some potential traffic engineering possibilities. Traffic engineering based on Application domain information would allow network operators the ability to build policies that could expedite traffic based on the content of its URL or shape traffic based on delay experienced on all video traffic. We propose extending OpenFlow using their experimenter action property to control and activate policies for deep packet inspection functionality within a commodity hardware, resolving OpenFlow's limitation in traffic engineering on Application domain information.

Our approach uses an open, reusable extension in an OpenFlow-recommended manner to allow service insertion. By performing service insertion within the OpenFlow framework, no additional software or end-point signaling is necessary for devices to benefit from a value-added network service. A commodity hardware box running Linux provides additional processing at wire speed without the use of any closed proprietary hardware.

We proved the usefulness of this approach by performing video reconditioning based on video signaling properties. This involved performing operations on information gleaned from Layer 5 - 7 data, allowing the network to perform service-aware routing. Service-Aware routing allows traffic steering based on bandwidth capacity, malicious software detection, content-availability, and user-subscribe service and thus has a

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.94

strong influence over user's quality of experience (QoE)[4][5]. QoE describes the user's perceived experience for a provided service. Packet loss, delay, and jitter (variance of delay) can affect the quality of service (QoS) for an application [6]. We evaluated our architecture against a standard OpenFlow network based on these QoS metrics and video quality assessment measurements.

II. Architecture Design

Additional processing control within the OpenFlow framework, in support of policies expressed in L5-L7 semantics, can be accommodated in three locations: the controller, the datapath, or a separate additional device. The first two provide obvious possibilities for such enhancement, but are not scalable. Including additional processing within the controller not only breaks the policy/mechanism separation paradigm, but can very quickly overwhelm a controller, which is typically envisaged as being software (for agility) running on commodity hardware, and controlling multiple datapaths. The second choice is expanding the definition of OpenFlow so that a datapath flowtable can represent match fields from higher layers. This would allow the controller to express policies easily, but realizing such a datapath would be prohibitively complex if attempted in hardware, due to the plethora of application protocols.

We follow the third option in our approach: that of an additional "helper box" that can be slaved to the datapath, and used as needed by the datapath to classify traffic using higher layer headers or payload. To the controller, this appears as the capability of the datapath itself. To the datapath, this becomes a problem in simply translating extension matches to the terms understood by the helper box when new flowtable rules are installed, and installing flows to guide traffic requiring such classification to and from the helper box. Scalability is retained, since the helper box can operate independently of the datapath, so that the datapath's processing of other traffic (not requiring such classification) is only minimally affected. The helper box itself can be made as powerful as necessary to ensure the desired performance for the traffic it classifies, and multiple helper boxes can be installed to perform load balancing.

We design such a complete system and discuss the design and architecture issues below. Our helper box contains an open-source Deep Packet Inspection (DPI) engine software. Our OpenFlow extension provides a structure for expressing rules with match fields that can be satisfied by this box. In what follows, we refer to the helper box as the External Processing Box.

The OpenFlow External Processing Experiment's design is split into two core pieces:

- Modifying the OpenFlow protocol
- External Processing Box (EPB)

Installation for the external processing functionality within the OpenFlow protocol, requires registration of an experimenter OpenFlow Action on both the control and data plane endpoints; the controller and the datapath.

Offset	Octet	0								1								2								3							
Octet	Bits	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Type																Length															
4	32	Control Port																Data Port								Library ID							
8	64	Library Options																															
12	96	Library Options																															
16	128	External Processing Search Field Type																															
20	160	External Processing Search Field Value																															
24	192	External Processing Search Field Value																															
28	224	External Processing Search Field Value																															
32	256	External Processing Search Field Value																															
36	288	External Processing Search Field Value																															
40	320	External Processing Search Field Value																															
44	352	External Processing Search Field Value																															
48	384	External Processing Search Field Value																															
52	416	External Processing Search Field Value																															
56	448	External Processing Search Field Value																															

Fig. 1: External Processing Action Message Format

A. Experimenter OpenFlow Action: External Processing

An Experimenter OpenFlow Action has been defined to manage specified traffic flows and issue policy instructions to the EPB. Throughout this literature this Experimenter Action will be called the External Processing Action. An Experimenter Action requires an experimenter ID to be issued to identify the manufacturer for the operation [7]. The External Processing Action is illustrated in Figure 1 and requires the following arguments:

- Control port: the switch port dedicated to transmitting control messages (EPB Policy Message) to the EPB
- Data port: the switch port dedicated to transmitting data to the EPB
- EPB library ID: the policy identifier to instruct the EPB
- EPB library options: library operation for the specified Library ID
- EPB Search Field: defines the construct that the controller wishes for the EPB to perform its instructions on. This construct has three components: type, length, and value

The EPB Search Field is intended to allow comparison between values, much like OpenFlow Match fields are used to match bits within a packet's Layer 2 - 4 headers. EPB Search Field is used to pass search parameters from the controller to the EPB.

B. Experimenter OpenFlow Message: EPB Policy Message

The purpose of EPB Policy Message is to orchestrate policy creation within the EPB. Policy creations involves installing a new rule into the EPB's DPI Engine and instructing the Traffic Shaper how to steer the outgoing packets. The Datapath is responsible for determining when this message should be forwarded to the EPB based on uniqueness of the traffic that triggered the action that is discussed in detail in Section II-D. The EPB Policy Message contains a 32-bit Experimenter ID and a 32-bit Experiment Type [7]. The OpenFlow switch sets Experimenter ID to the EPB ID and uses the action's Library ID for the Experimenter Type.

C. OpenFlow Controller

We have modified a Ryu controller to communicate our External Processing Action to an OpenFlow switch. Using this approach, the controller can provide instructions to the EPB for some matched traffic. At its minimum case, the controller needs to have an OpenFlow Match setup to steer traffic towards the EPB for additional processing.

D. OpenFlow Switch

Open source software switches offer an open programmable platform, a feature which hardware switches can not provide without access to the vendor's firmware. FlowForwarding's Link is Not Closed (LINC) OpenFlow software switch was selected for its ease in developing and testing new OpenFlow features [8]. Along with recognizing the External Processing Action, the OpenFlow switch is responsible for generating and sending an EPB Policy message for the traffic that triggered it. To achieve this the switch must maintain an in-memory database to track whether an EPB Policy message needs to be sent to the EPB via its control interface. The in-memory database's uses the External Processing Action parameters and the identifiers from the header fields of the triggered flow's packet. The in-memory database stores the flow's polling value, and the number of occurrences in which the key has been checked against the database. The polling parameter is used to determine the frequency an EPB Policy message should be issued. Depending on the polling bit's value; an EPB Policy Message is transmitted for the first packet that triggered the action or not at all. The insertion of EPB policies operates reactively to traffic triggering External Processing Action by the Datapath. The polling bit and the uniqueness of the trigger traffic characteristics determine whether EPB message is generated or not.

The EPB Policy generated has the standard OpenFlow structure for a symmetric message with the message type containing the Experimenter class value. The Experimenter message contains a 32-bit Experimenter ID and a 32-bit Experiment Type [7]. The OpenFlow switch sets Experimenter ID to the EPB ID and uses the action's Library ID for the Experimenter Type. The remaining structure has been defined in Section II-B. The EPB Policy Instruction is encapsulated inside a UDP packet. The following list demonstrates the UDP structure sent through the EPB's control interface. A non-routable IP address 0.0.0.0 is used identify control messages between OpenFlow switch and the EPB [9]. The EPB is responsible for checking whether packets received from its control port has the appropriate structure before attempting to dissect the EPB Policy message within the packet, otherwise the packet is dropped.

E. External Processing Box

The External Processing Box is a commodity box that performs additional processing on packets sent to it by the datapath. The EPB can be viewed as a middlebox platform for DPI powered applications. The current model provides only a subset of possible processing components that can be implemented within such a box. The EPB used in this experiment has the following four components: OpenFlow Dissector Agent, EPB Policy Interface, Deep Packet Inspection Engine (DPI Engine), and the Traffic Shaper, also illustrated in Figure 2.

1) OpenFlow Dissector Agent: The OpenFlow Dissector listens to all traffic entering the EPB's control port from the OpenFlow Switch and dissects the traffic to determine whether

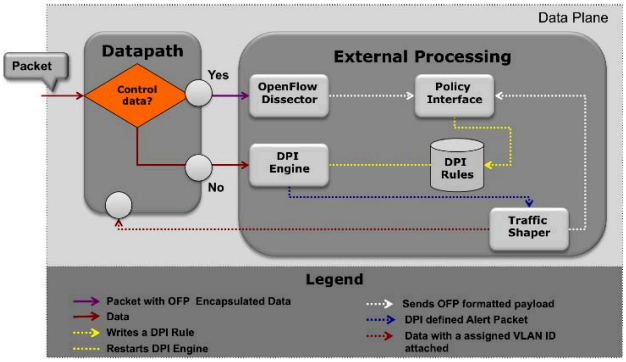

Fig. 2: External Processing Architecture

the traffic has OpenFlow encapsulated message containing an EPB Policy Message. Packets containing the encapsulated message are then parsed by the Dissector and sent to the EPB's Policy Interface via a Unix domain socket. A Unix domain socket provides inter-process communication channel for exchanging data between two processes. If the retrieved packet does not contain an encapsulated EPB Policy message then that packet is dropped.

2) External Processing Box Policy Interface: The EPB Policy Interface is responsible for deconstructing the EPB Policy Messages it receives into policy semantics for the DPI Rules Engine. The configuring of the DPI Rules Engines involves writing policies within a configuration file and restarting DPI service to activate the new policies. The OpenFlow Dissector and the Traffic Shaper communicate to the EPB Library.

3) Deep Packet Inspection Engine: DPI Engine refers to an application with the ability to examine the data (Application layer) of a packet. The DPI Engine provides a sensing mechanism for the EPB by detecting what a packet payload contains and sends a descriptor along with the packet itself to the Traffic Shaper. To achieve this, an open-sourced Network Intrusion Detection System (NIDS), Snort, was selected. Any application with DPI capability could serve as a DPI Engine for this experiment. Several open-source NIDS solutions (Bro, Snort, Suricata), were evaluated for their strength and weaknesses [10][11]. Snort was selected due to its lightweight nature, ease of use, and active community.

Snort is used to detect and classify packets sent through it for the next EPB's process (the Traffic Shaper) to handle. The Traffic Shaper is responsible for listening to incoming traffic. Within Snort's rule options, messages may be logged along with the packet that triggered it. This allows for packet description and/or other messages to be sent to the Traffic Shaper, which it can then be used to categorize the type of traffic and what action should be taken for that packet.

4) Traffic Shaper: Traffic Shaper is responsible for monitoring packets passing through the DPI Engine. The EPB Policy Interface instructs the Traffic Shaper's actions based on a given packet's classification. Based on these actions the Traffic Shaper determines which port that traffic should exit the Datapath. The Traffic Shaper shapes outgoing packets by attaching a VLAN ID tag corresponding to the selected port

before transmitting the packets out of the EPB.

Traffic Shaper maintains three conduits of information: receiving data from the DPI Engine, and transferring data to the EPB Library or back to the Datapath. Data from the DPI Engine is received from the DPI Engine serves. Based on this received data, the Traffic Shaper may transfer control data in the form of an EPB Policy message to the EPB Policy Interface and/or forward the original packet back to Datapath after processing. The egress connection between the EPB and the Datapath serves as a VLAN trunk, allowing any VLAN tagged traffic to travel through it. The Datapath is programmed to pop VLAN tags off incoming traffic from this VLAN trunk port and then forward that traffic out the switch port with the matching VLAN tag value.

III. Proof of Concept Prototype

For our prototype, we evaluated the video quality of Video-On-Demand streams between a media server and a client to assess the effect the EPB has on the network topology. Our test scenarios, include a baseline and three different traffic patterns test scenarios, which will be evaluated based on the delay conditions faced by the client and/or the media server before and/or after the EPB interaction is applied. QoS and video quality assessment metrics will illustrate the overall network performance benefits that the EPB brings for streaming video. While this prototype demonstrates the effect the EPB can have on traffic redirection on Application domain data, this same prototype can be used to perform packet scheduling on Application data with a minor modification.

Test Scenarios

1) Baseline condition, video streams traveling the best-effort route with no Traffic Controller activated delay or EPB intervention
2) Video streams traveling a best-effort route with no EPB intervention
3) Video streams traveling a best-effort route with the EPB involved in the circuit
4) Video streams traveling a high availability route due to EPB intervention

Network Delay Conditions

- Network delay only affecting normal communication route between the client and the OpenFlow Network
- Network delay affecting all normal communication routes.

Our prototype simulates an environment where requests and responses between a client and media server from streaming video must travel through our OpenFlow network. Figure 3 illustrates a logical topology map for this environment. Traffic propagating through the best-effort routes was treated with variable amount of delay. These clouds are simulated by Traffic Controllers, as seen in Figure 4. The Traffic Controllers, introduce outgoing traffic delay with a variance of 100ms $\pm$ 50ms. Delays on networks are rarely uniformly distributed, so to simulate real-life network delay we have programmed our Traffic Controllers to have a normal distribution of variation in delays [12].

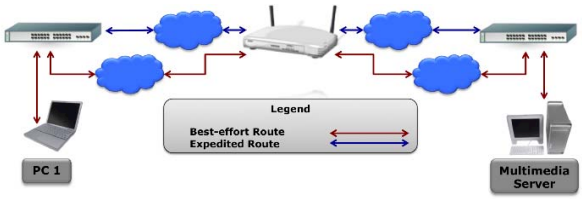

Fig. 3: Logical Topology

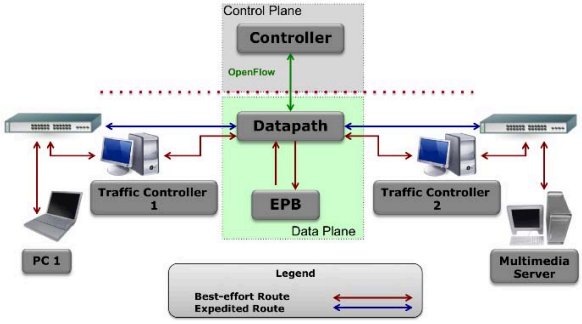

Fig. 4: Physical Topology

A single switch is used to provide the client and the media server with dual access to the OpenFlow network. Each client and the media server's connections were separated into different VLANs. This separation allows the switch to be viewed as two separate logical switches to the end-hosts and thus does not allow intercommunications between them through the device. Connections between the Traffic Controllers to the OpenFlow datapath and Switches are displayed as a bi-directional connection in Figure 4, but are actually dual connections each with a separate directional stream of data. This simplified debugging connectivity issues with a packet analyzer.

A. Testing Tools and Resources

To objectively evaluate video quality, we used Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) to measure the perceived visual quality. We used Video-Tester [13], a video assessment platform that provides SSIM, PSNR measurements, Quality of Service metrics (like a jitter), among other measurement values. Video-Tester comes built with an RTSP server, client, a traffic sniffer, and a HTTP server for platform management messages. In order to use the Video-Tester with our prototype, the code was modified to use a constant port for its internal RTSP server, due to VOD Reconditioning policies requirement for the EPB.

Videos used for evaluating this architecture are among the videos sequences used predominately in video research [14]. Video-Tester requires that their client have the requested video's properties such as the frame per seconds (fps) and the intended bit rate. The video's fps configured matches the original videos frequency, while the bit rate values were subjectively determined based on [15] and the videos content these values are expressed Table I.

TABLE I: Test Video Properties

Video Name	Frames Per Seconds (fps)	Encoding (kbps)
akiyo_cif.264	25	128
football_cif.264	25	282
foreman_cif.264	25	128

IV. Results

For each of the test scenarios, five tests were performed with three videos resulting in 105 total tests. Charts displaying this information have grouped data sets by their video and/or categorized by the test suite it represents. Table II illustrates the test suite's identifier and a short description.

TABLE II: Test Scenarios

Identifier	Description
Test A	Baseline, no Traffic Controller or EPB involved
Test B	EPB with Traffic Controller 1 activated
Test C	EPB with both Traffic Controllers activated
Test D	Only Traffic Controller 1 activated
Test E	Both Traffic Controllers activated
Test F	EPB with Traffic Controller 1 and Expedited Routes activated
Test G	EPB with both Traffic Controllers and Expedited Routes activated

As seen in Figure 5, all videos experienced the same treatment through the network despite their video's varying properties. Each video experienced their lowest jitter during the baseline test. As delay is introduced into the network, each video experienced an increasing amount of jitter, as would be expected. These two figures give us an overall baseline for our entire network before EPB is included into the network. Later figures will compare the effect EPB has on the video quality based on PSNR and SSIM measurements.

PSNR-based Mean Opinion Score (MOS) for best-effort routed video streams maintained a similar score with and without the EPB. Figure 6 shows that there is not a significant difference between these values, thus the inclusion of the EPB into the circuit even when it is not expediting traffic, had little effect on the video quality. The MOS scores were at its highest with baseline traffic, but expedited traffic closely mirrored its average MOS. Figure 7 showed that when both Traffic Controllers are activated, the system without EPB has a slightly higher MOS score than best-effort traffic with the EPB. When expedited service is included, the video scored its highest MOS.

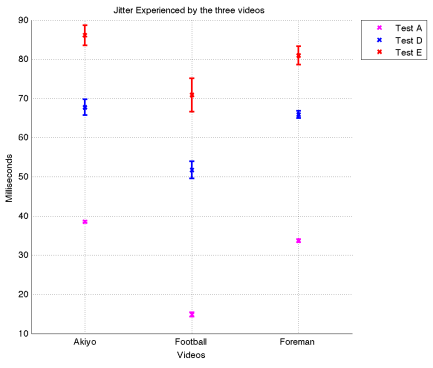

Fig. 5: Jitter experienced for each of the videos

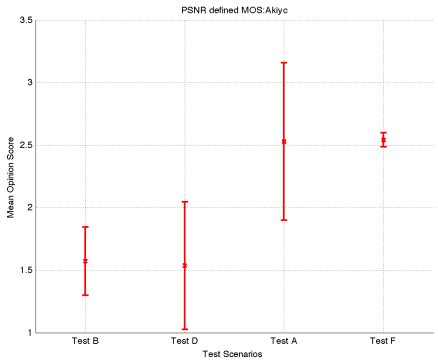

Fig. 6: MOS with Traffic Controller 1 activated versus baseline and expedited

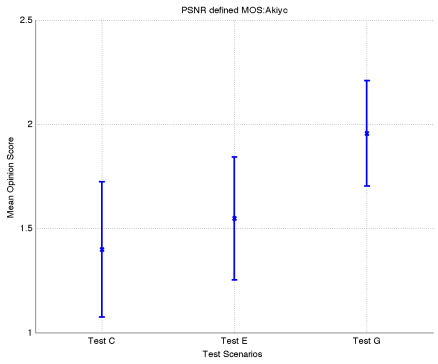

Fig. 7: MOS with both Traffic Controllers activated versus expedited

SSIM measurement for Test B and D (Figure 8) showed a similar assessment as the previous tests with the EPB inclusion within the circuit and not hampering video quality. Unlike our previous tests, comparison between Test A and F shows that the expedited service measured a higher quality than our baseline test. We suspect that Test F has a higher SSIM score, due to Traffic Controller involvement within the baseline test. Baseline tests traveled via best-effort route as described in Figure 4, and thus must travel through the Traffic Controller. While the Traffic Controller is not emulating any delay for the baseline, traffic still endured some propagation time due to traffic traveling up and down the stacks within the Traffic Controller. Further tests will be necessary to validate whether or not that is the case.

Test G's SSIM measurements exceeds the video quality of the other two tests as expected due to expedited traffic reducing the amount of delay traffic faces, this can be witnessed in Figure 9. Test C and E comparative scores, however does not match the predicted trend. Test C has a noticeable higher SSIM value than Test E. It is our belief that this is being caused by the small sample size; further test data would normalize this to fit the common assessment.

As expected our results show expediting traffic towards an high availability route provided a better quality of experience based on the gathered video quality assessment measurements.

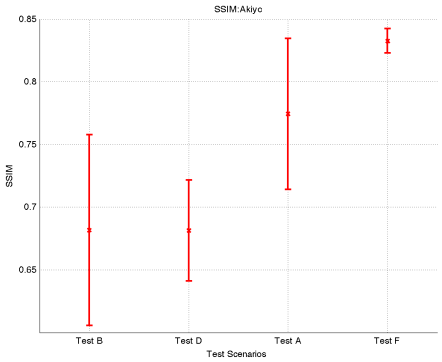

Fig. 8: SSIM with Traffic Controller 1 activated versus baseline and expedited

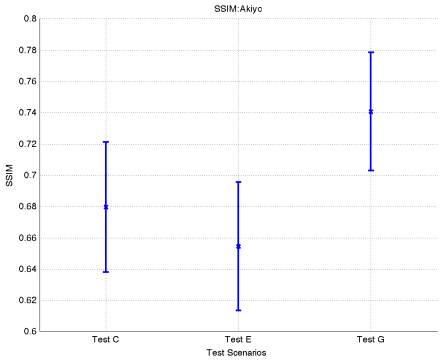

Fig. 9: SSIM with both Traffic Controllers activated versus expedited

Degraded and/or delayed multimedia traffic to our OpenFlow network was shown to have a strong influence on the assessed video quality experienced by the end-user. Despite giving degraded video traffic expedited treatment our test results show that the EPB could not further improve the experienced quality, demonstrating that the EPB is dependent on the quality of arriving network packets. If the end-user's video player has already passed the delayed multimedia traffic, any expedited treatment given to the traffic by the EPB is a waste, as the player will no longer have any use of that particular traffic. On the other hand, expediting traffic away from a delayed network paths has been shown to improve received video quality. The additional propagation time the EPB spends processing and rerouting the packets, has also been shown to have little to no effect in comparison to baseline conditions.

V. Related Work

Including an OpenFlow Extension to OpenFlow's table to accept Application layer metadata has been suggested by industry. Standardization would need to be required in order to facilitate a common App ID and metadata format. Without standardization vendors' devices would not be interchangeable as each device could have inconsistent App ID and metadata thus increasing network complexity, and reducing equipment efficiency [5]. Qosmos ixEngine is a software development kit (SDK) of libraries and tools that can identify applications and extract metadata such as packet loss, latency, codec, mobile ID, and attached documents ran on Intel Data Plane Development Kit (DPDK) [1]. This indicates that there exists industry interest in systems similar to what the EPB can provide.

FRESCO[16], an OpenFlow security application framework allowed the integration of legacy security application, like Snort, to interface with FRESCO's Event Management to generate flow rule logic when an alert is generated. The key difference between this approach and the one we have suggested is the fact that FRESCO sits on control plane while ours sits within the data plane alongside with the Datapath.

VI. Conclusions

We have proposed an architectural extension to OpenFlow to allow management and granular control over traffic through a datapath that match desired L7 characteristics. The value of our extension lies in its seamless integration into the open multi-vendor nature of OpenFlow, while providing flexibility and scalability. Our architecture exhibits an effective programming model for performing service-aware routing using commodity hardware. We have realized a practical prototype that demonstrates the realistic nature of the solution, and its potential feasibility in practice. Quantitative results from a study of our prototype show that this approach can provide the desired traffic engineering results while imposing a low overhead that can be easily tolerated, for the challenging task of in-flight video reconditioning without explicit cooperation from end-systems.

Our architecture is not limited to video or other real-time traffic, or to traffic scheduling or re-routing applications. Our approach can be adopted to provide a variety of other value-added network services inside the network; such as security services to detect or eliminate malware embedded in unwary user traffic, context-sensitive services for fast-breaking situations in dissemination of news or public service information, or even as pure network analytics to detect if there are any statistically significant unexpected trends in delay characteristics of specific flows of traffic, indicating the possibility of attacks or violation of neutrality by some carrier. As industries continue to move their on-premise infrastructure to cheaper cloud computing resources, SDN powered clouds offer network managers the ability to maintain elastic services and to strategically steer resources for high performing applications and users. Our EPB approach would enable orchestration of traffic engineering, or content- and resource-management in such scenarios. In brief, we believe that our work in demonstrating the practicality of such an approach opens the door to many fruitful areas of research.

References

[1] Intel and Qosmos, "Service-Aware Network Architecture Based on SDN, NFV, and Network Intelligence," 2014.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, pp. 69–74, Mar. 2008.

[3] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (SDN): A reference architecture and open APIs," in *ICT Convergence (ICTC), 2012 International Conference on*, pp. 360–361, 2012.
[4] Aricent, "Application-Aware Routing in Software-Defined Networks," 2013.
[5] Qosmos and H. Reading, "The Role of DPI in an SDN World," 2012.
[6] L. Sun, I.-H. Mkwawa, E. Jammeh, and E. Ifeachor, *Guide to Voice and Video over IP: For Fixed and Mobile Networks*. Springer Publishing Company, Incorporated, 2013.
[7] Open Networking Foundation, "OpenFlow Switch Specification 1.3.1," 2012.
[8] "Linc - openflow software switch." http://flowforwarding.github.io/LINC-Switch/. Accessed: 2014-06-14.
[9] R. Braden, "Requirements for Internet Hosts – Communication Layers," RFC 1122, October 1989.
[10] E. Albin and N. Rowe, "A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems," in *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pp. 122–127, 2012.
[11] P. Mehra, "A brief study and comparison of Snort and Bro Open Source Network Intrusion Detection Systems," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 6, pp. 383–386, 2012.
[12] "Linux foundation - netem." http://www.linuxfoundation.org/collaborate/workgroups/networking/netem#Emulating_wide_area_network_delays. Accessed: 2014-03-01.
[13] I. Ucar, J. Navarro-Ortiz, P. Ameigeiras, and J. Lopez-Soler, "Video Tester - A multiple-metric framework for video quality assessment over IP networks," in *Broadband Multimedia Systems and Broadcasting (BMSB), 2012 IEEE International Symposium on*, pp. 1–5, June 2012.
[14] "YUV CIF reference videos (lossless H.264 encoded)." http://www2.tkn.tu-berlin.de/research/evalvid/cif.html. Accessed: 2014-03-20.
[15] "NetroMedia - How do I choose my bitrate / bit rate?." http://login.netromedia.com/solutions/View.aspx?ID=1bcc5cd5-3da0-4937-8be0-6b87c7e97084. Accessed: 2014-03-19.
[16] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "FRESCO: Modular Composable Security Services for Software-Defined Networks," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS'13)*, February 2013.