Video over Software-Defined Networking (VSDN)

Harold Owens II
Indiana University-Purdue University Indianapolis
Dept. of Computer and Information Science
Indianapolis, IN USA
Email: owensh@cs.iupui.edu

Arjan Durresi
Indiana University-Purdue University Indianapolis
Dept. of Computer and Information Science
Indianapolis, IN USA
Email: durresi@cs.iupui.edu

***Abstract*—Supporting end-to-end quality of service (QoS) for video applications requires the network to select optimum path among multiple paths to improve application performance. Multiple network paths from source to destination may be available but due to the current network high coupling design identifying alternate paths is *difficult*. Network architecture, like Integrated services (IntServ), installs a single path from source to destination which may not be the optimum path for the application. Furthermore, it is an arduous task for video application developers to request service from IntServ.**

This paper provides three contributions to research on providing end-to-end QoS for video applications. First, it presents video over software defined networking (VSDN) - an architecture that is capable of making optimum path selection utilizing a global network view. Second, it describes the VSDN protocol used by video application developers to request service from VSDN enabled networks. Third, it presents the results of implementing a prototype of VSDN and quantitatively evaluates its behavior. The prototype illustrates that requesting video service from VSDN is *simple* for the video application developer. The results show that VSDN has a linear-message complexity.

I. Introduction

Integrated Services (IntServ) framework is a flow based quality of service (QoS) architecture that specifies elements (*i.e.*, Sender, Receiver and routers along path) and allows each element to request and receive resource reservations which guarantee end-to-end QoS from network. IntServ framework uses two protocols, flow specifications which is used to describe the traffic pattern and a reservation protocol which is used to transmit the reservations among network elements and to allow reservations to be made by applications (*e.g.*,video and voice) that QoS sensitive tasks (*e.g.*, video and audio compression) require guaranteed bandwidth and bounded delay and jitter.

Supporting end-to-end QoS for video applications requires network to select the optimum path among multiple paths to improve application performance. Today's network control protocols do not always install the optimum QoS path for video applications and is not capable of exploring alternative paths since it relies on routing protocol in selecting path. For example, if there exists two network paths (*i.e.*, `Path1` and `Path2`) from source to destination; `Path1` is 3 hops with congestions and `Path2` is 5 hops with no congestion. If routing protocol uses shortest path then `Path1` with 3 hops will be selected to install reservation. Selecting `Path1` for QoS aware applications like video will result in negative impact on end user's video playback experience because of congestion along `Path1` which data (*i.e.*, video) packets travel.

The path selection process should be adaptive to changing network conditions (*e.g.*, link and node failures) and should be context aware about the state of path (*i.e.*, bandwidth, jitter and delay). Although data-carrying mechanisms like MPLS are capable of multiple QoS path selection, it is not dynamic and network operators must manually configure paths along each router within the network [1]. Today's network control protocols are capable of adapting to network failures but data packets are lost or receive best-effort service between time of failure and time of next PATH refresh messages which can be 30 seconds or more before PATH is updated and resources are allocated to network flow.

Therefore, this paper presents experience and results of identifying a network architecture that is capable of ensuring end-to-end QoS for video applications. The main contributions of this paper are:

- It presents *Video over Software Defined Networks (VSDN)*, which is an architecture and protocol for supporting video over IP networks;
- It presents QoS API used by Sender and Receiver to request video over SDN;
- It presents the results of implementing VSDN as a simulator; and
- It presents an empirical study that evaluates VSDN runtime performance in terms of message complexity

Paper organization. The remainder of this paper is organized as follows: Section II motivates the need for Video over software defined networking (VSDN); Section III discusses the design and implementation of VSDN; Section IV presents results of simulating SVDN in a simulator and interpretation of the results; Section V compares VSDN to related works; and Section VI provides concluding remarks and lessons learned.

This is the author's manuscript of the article published in final edited form as:
Owens II, H., & Durresi, A. (2015). Video over Software-Defined Networking (VSDN). Computer Networks, 92, Part 2, 341–356. http://doi.org/10.1016/j.comnet.2015.09.009

II. Motivation: IntServ QoS Model

Integrated Services (IntServ) architecture uses reservation protocols to signal end-to-end QoS over IP networks. The resources are reserved on a hop by hop basis using two messages. The `PATH` message is used by sender to install reverse routing path on each router along path and convey to receiver the characteristics of traffic. The `RESV` message is used by receiver to request QoS of packets from each router along path. Reservation protocols use `PATH` and `RESV` messages to install soft states along network paths which IP packets traverse. The soft states contain descriptions of traffic characteristics (*e.g.*, rate, queue size and peak rate) of data flow to be received. The reservation protocol works independently of a particular routing protocol and therefore, installs soft states in routers along same path which IP packets traverse from source to destination.

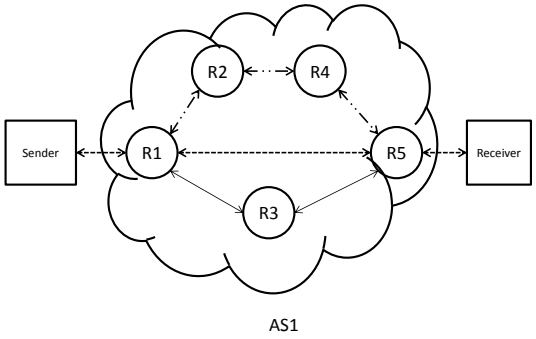

Figure 1. Simple Network Topology with a Sender and a Receiver

As previous stated, where reservation paths are installed depends on the path of IP packets selected by a routing protocol (*e.g.*, RIP, IS-IS or OSPF). Current reservation protocols have advantages that include soft-state adaptive nature flexibility of receiver initiating reservation and possibility to merge reservation requests. IntServ architecture using current reservation protocols also has its disadvantages. For example, in figure 1, let us assume that best path to send video in terms of bandwidth, jitter and latency is `R1 - R2 - R4 - R5`. If the routers within autonomous system one (AS1) is running OSPF and `Receiver` wanted to receive video from `Sender`, the shortest path that would be stored in each routers' link state database would be `R1` to `R5`. Therefore, video packets from `Sender` to `Receiver` will follow path `R1 - R5` which is two hops.

In this case, the reservation protocol would install `PATH` and `RESV` states on `R1` and `R5` for supporting QoS for video application although the best path for video, as stated earlier, is `R1 - R2 - R4 - R5`. With current InServ architecture it is not possible for network to select the best path for video in this context. Furthermore, if a link failure occurs between routers `R1` and `R5`, a new path will need to be found. The network will make the following adjustments: OSPF will detect link failure and update link state database of each router to reflect link `R1 - R5` failure. The next path in terms of hops (cost) is path `R1 - R3 - R5` which video data traverses after failure but the best path for video is `R1 - R2 - R4 - R5`. The network has failed to find the optimum path for video service.

Because relying on current network architecture routing protocols and access control lists to provide end-to-end QoS to video application can result in non-optimum path being selected, there are two issues that must be addressed:

Issue 1: Identify architecture needed to support optimum video path selection. As figure 1 illustrates, selecting the optimum or feasible path requires global knowledge of the network. Hop by hop decision making leads to non-optimum path selection. An architecture needs to be developed that can dynamically make optimum path selections and provide feasible backup paths in case of failure.

Issue 2: Develop protocol that allows video applications to request end-to-end QoS from network. A protocol needs to be developed to allow developers of video applications to request network services from new architecture. The protocol should be simple and only request and maintain states that are necessary to ensure optimum path selection for video.

This papers keeps usability in mind when designing and implementing support for video network services. The rest of this paper discusses how these issues are addressed.

III. Design and Implementation

One key design requirement for addressing - `Issue 1`: Identify architecture needed to support optimum video path selection - is the need to have a global view of network state to make optimum path selection. This requirement leads to use of software defined networking (SDN) [2] and OpenFlow [3] to solve `Issue 1`. Figure 1 has been redesigned to utilize SDN architecture and OpenFlow protocol as illustrated in figure 2. An SDN controller which has global view of network state has been added. The next section briefly discusses SDN.

1) Software Defined Networking (SDN): Figure 2 illustrates SDN. SDN separates the control plane (*i.e.*, routing decision) from data plane (*i.e.*,forwarding decision). The control plane is implemented in software in an element known as the SDN controller. The routers and switches become dumb devices only performing forwarding based on instructions from the SDN controller. OpenFlow is the protocol used to communicate between the control and data plane since in SDN they are distributed and no longer on the same device (i.e., router or switch).

A. Overview of Video Over SDN (VSDN)

Video over SDN is an architecture that allows video applications to request video service from network. `VSDN` design integrates a high level of usability.

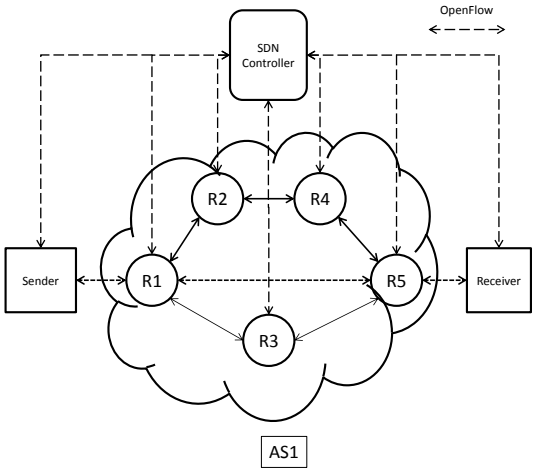

Figure 2. Software Defined Network with a Sender and a Receiver

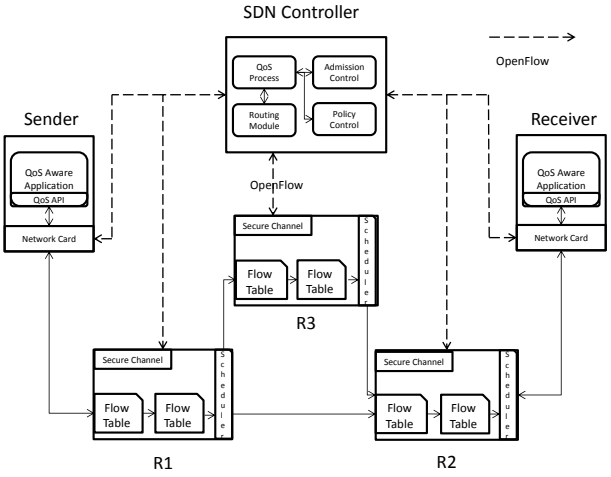

Figure 3. Overview of VSDN. A Sender and Receiver utilizing QoS API and QoS provided by network to transmit and receive video.

Figure 3 illustrates `VSDN` at a high level. The architecture has four system elements: `Sender`, OpenFlow switch, video QoS SDN controller and `Receiver`. The `Sender` and `Receiver` relies on network composed of `R1`, `R2` and `R3` to provide end-to-end QoS.

The `Policy Control` (PC) is separated from the physical device (*i.e.*, router) and is installed on video QoS controller (VQC) to provide policy consistency among system elements. The PC is an important part of VQC. It accepts commands from network administrator about how new traffic should be handled. It is responsible for enforcing constraints imposed by network administrator [4]. A policy translator is used to map policies to network configuration which is stored in policy database. The `resource monitor` (RM) is used to monitor current network resources. It periodically collects counters from physical devices. The RM stores network state information in resource database.

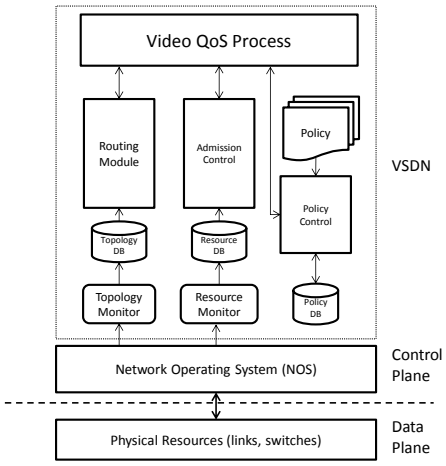

Figure 4. Video Controller receives QoS signaling and determines if network is capable of servicing request.

The `Admission Control` (AC) on VQC occurs when a request attempts to reserve network resources. When a request is received on an interface, admission control will be performed on that interface of provider edge (PE) router. If the resources are available on the interface in which the request was received, VSDN process will attempt to find an optimum or feasible path from PE ingress to PE egress that can service the request. If resources are not available on ingress port, VQC will return an admission error to requester. AC must manage the pool of resources (*e.g.*, number of interfaces, bandwidth, memory, CPU, etc.). Resources are subtracted from the pool when a request is serviced and resources are added back to pool when a requester has finished with the resources.

The `Routing Module` (RM) is used to calculate feasible paths from PE ingress router to PE egress router. Constraint based routing algorithms and implementations have been studied in detail [5]. The requirement for Routing Module is that it returns a list of subgraphs (*i.e.*, paths) that meets QoS request constraints (*e.g.*, bandwidth, jitter and delay). The `Topology Monitor` (TM) updates the network configuration when there is a change (*i.e.*, failure, deletion or addition) to nodes or links. The network topology is stored in a database which is utilized by RM to find feasible paths.

The main element of architecture is `Video QoS Process` (VQP) illustrated in figure 4. VQC is used to handle all VSDN messages. It must be able to process Sender, Receiver and error messages. The VQP must maintain the correct state for each session as described by the protocol. If a host does not follow protocol, VQP must generate an error stated why request has failed. A valid request from `Sender` will generate a new session in the session database. The session database contains enough information (*i.e.*, SessionID and destination and source address and

port) to uniquely identify a flow. After a valid request is received from `Sender`, the message will be forwarded to `Receiver`. If `Receiver` wants to accept request or reservation, it will send a receiver request message to VQC. Upon receiving a valid receiver request message, the VQC will perform policy and admission control management to determine if reservation can be made. If reservations are allowed, VQP will request feasible paths from RM. The RM will return a subgraph which is used to configure the physical devices. After devices are configured, the VQC returns a confirmation message to `Receiver` which forwards message to sender. At this point, `Sender` and `Receiver` are able to communicate over the requested QoS path. Once interaction between `Sender` and `Receiver` has finished, either host can send a remove request to network to remove session and release network resources. How the remove message is handle will be covered in detail in next section. Finally, the VQP needs to ensure that sessions are updated or timed-out and removed. Figure 4 shows VQC running on network operating system [6] which provides a uniform and centralized programmatic interface to the entire network.

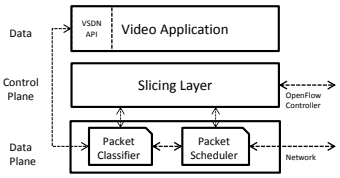

Figure 5. Host system where video QoS Application resides. Application communicates using QoS API provided as a library for developers.

Figure 5 illustrates architecture of hosts (*i.e.*, Sender and Receiver). `Sender` and `Receiver` will be running video application which is VSDN enabled. The Slicing layer [7] is used to allow multiple service providers to share a common infrastructure; and supports many policies and business models for cost sharing. The slicing layer will allow VSDN service providers to control their share of the host's network. The video application will communicate with VSDN enabled network by using VQP API. The VPQ API is described in next section. The packet classifier is responsible for identifying which packet belongs to which flow. The packets are identified using sender and destination address, sender and destination port, and protocol ID. The packet scheduler is responsible for ensuring the packets generated by the application is in specification with agreed service (*e.g.*, rate, bandwidth and queue size). Packet scheduler should ensure that packets are in specification before packets are transmitted to network. In VSDN enabled networks, the network administrator will issue global policy control over network elements (i.e., hosts, switches, routers, applications and etc.).

Figure 6 illustrates architecture of VSDN switch (*i.e.*, `R1`, `R2` and `R3`). VSDN switches are OpenFlow enabled. The VSDN controller, upon receiving and validating QoS service

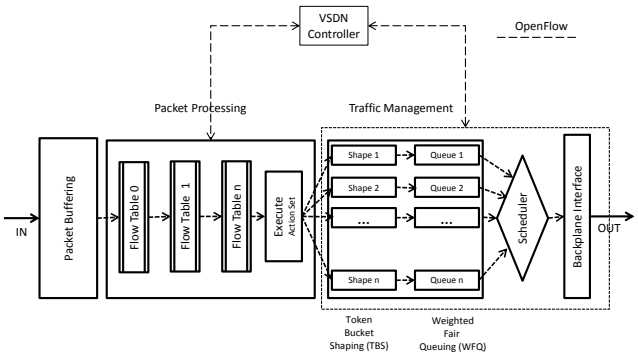

Figure 6. VSDN Switch Architecture

request from host, will issue Set-Queue Action to each VSDN switch in path selected by `RM`. The VSDN controller will issued Flow-Add Action to each VSDN switch in QoS path after sending Set-Queue request. If either Set-Queue or Flow-Add Action request fails VSDN switch must return an error message to VSDN controller. The Set-Queue request will cause VSDN switch to create a per flow weighted fair queue (WFQ) and traffic shaper or regulator based on Traffic SPECification (TSpec) [8] supplied by VSDN controller.

As shown in Figure 6, a packet will enter a VSDN switch and will be queued. It will be forwarded to flow table pipeline where packet based on policy will be sent to VSDN controller, drop or continue through flow table pipeline until there is a matching flow table entry. The packet is eventually forwarded to port where VSDN end-to-end QoS has been configured. Forwarding behavior is dictated by the configuration of the queue and is used to provide basic Quality-of-Service (QoS) support. Full end-to-end QoS support over SDN is not implemented in OpenFlow [3]. Section III-C will discuss necessary changes needed to OpenFlow for VSDN integration.

B. Video over SDN Protocol

This section discusses actions of each element of video over SDN architecture and how `Sender` and `Receiver` interact with network.

1) Sender's Actions: The `Sender` in figure 2 makes a call in QSP API using requestQoS(video service, destination address, destination port). RequestQoS returns a SessionID which is used to remove session from network when applications are done using network resources; developer of video application may request three video services: common interchange format (CIF), enhanced definition and high definition. PE router `R1` forwards message to VQC after receiving message from `Sender` in figure 2. VQC will determine if Sender's request already exists. If session does exists, VQC responds with invalid request if this message is not from a timeout request from `Sender`. If session does not exist and policy allows `Sender` to issue request, VQC will create a new session and respond to `R1` which will forward

packet to Receiver via R2. The Sender waits on confirmation from Receiver. After receiving confirmation from Receiver, the Sender and Receiver begins to communicate over installed path.

2) Receiver's Actions: The Receiver receives request from Sender on an application callback processRequest(SessionID) and calls acceptQoSRequest(SessionID). Application should store SessionID which is used to accept request and to remove session from network. The Receiver makes reservation with network using description of traffic from Sender. PE router R2 will receive Receiver request and forward it to VQC where the controller will determine if session exists. If session doesn't exist, an error is generated by VQC and is sent to Receiver. If session does exist, the VQC contacts policy and admission control and router module to make reservation if Receiver is authorized. The router module returns feasible routes for request. VQC will decide which route to select and issue request to network operating system (NOS) to configure devices if a feasible route exists. The VQC sends Receiver via R2 a confirmation about QoS route being installed and network ability to service request. The Receiver, after receiving confirmation from VQC, sends a confirmation to Sender. If feasible route does not exist, an error is returned to Receiver. The path is considered to be installed when confirmation message from receiver is received by Sender.

3) Removing Reservation: Removing reservation from Sender, Receiver or network can be performed explicitly by host by calling removeQoSRequest(SessionID) or implicitly via timer expiration from the OpenFlow enabled switches and hosts. The remove request message from Sender, Receiver is forwarded by edge switch to VQC to determine validity of message. The VQC, after verifying message, removes session from session database and flow entries from switches and hosts. The VQC returns message back to edge router which forwards remove message to destination. If message is not received by OpenFlow enabled host, the session will be removed.

4) VDSN Controller Actions: The VDSN Controller receives request from receiver. Upon receiving request from receiver, VDSN Controller will perform admission and policy control. Policy control will be used to determine if receiver can make reservation and admission control will be used to determine if resources are available to service request. Assuming that receiver is able to make request, VDSN Controller will first find a constraint path that can service request. After identifying constraint path, VDSN Controller will instruct VDSN switches and hosts to adjust their flow tables and add required QoS queuing to port. The controller will then forward a success message back to receiver. The receiver will respond to sender with a success message.

5) VDSN Switch Actions: The VDSN Switch is a dumb device that takes instructions from the VSDN controller. When a VSDN request message is received by VDSN Switch, it forwards request to VSDN controller. The VSDN controller will perform the necessary logic to determine if a reservation can be made by receiver. The VDSN Switch will be required to add flow to flow table and QoS queuing to port or return error to controller if unable to complete operations.

6) VDSN Host Actions: The VDSN Host is a hybrid device that takes instructions from the VSDN controller. The host configures its flow table as instructed when a VSDN request message is received from VSDN controller. The VDSN host will be required to perform instructions (*i.e.*, add or delete flow and QoS queuing on port) from VSDN controller or return error to controller if unable to complete operations.

C. OpenFlow Changes

An OpenFlow switch provides limited Quality-of-Service support (QoS) through a simple queuing mechanism [3] in version 1.3.1. Therefore, VSDN switches will require changes to OpenFlow queue structures as shown in Listing 1. The queue properties struct $ofp\_queue\_properties$ has been modified to support guaranteed service (GS). A new property has been added to support GS based queuing. $ofp\_queue\_prop\_gs\_rate$ contains required fields for token bucket based traffic shaping. As illustrated in figure 6, VSDN switch will need to create a token bucket shaping queue for each requested flow. The queuing process using GS must be able to regulate traffic per flow based on traffic specification provided by VSDN controller.

```
1
2  /* Description of property related with a queue */
3  enum ofp_queue_properties {
4      /* Minimum datarate guaranteed. */
5      OFPQT_MIN_RATE = 1,
6       /* Maximum datarate. */
7      OFPQT_MAX_RATE = 2,
8      /* Guaranteed service (GS) datarate. */
9      OFPQT_GS_RATE = 3,
10     /* Experimenter defined property. */
11     OFPQT_EXPERIMENTER = 0xffff
12 };
13
14 /* GS-Rate queue property description. */
15 struct ofp_queue_prop_gs_rate {
16     /* prop: OFPQT_MIN, len: 16. */
17     struct ofp_queue_prop_header prop_header;
18     /* Average rate of traffic. */
19     uint32_t token_rate;
20      /* Size of bucket. */
21     uint32_t bucket_size;
22      /* Maximum data rate per second. */
23     uint32_t peak_rate;
24      /* Minimum packet size. */
25     uint16_t min_policed_unit;
26      /* Maximum packet size. */
27     uint16_t max_packet_size;
28      /* Maximum link capacity or peak rate. */
29     uint32_t Rate;
30      /* 64-bit alignment */
31     uint8_t pad[4];
32 };
33
34 OFP_ASSERT(sizeof(struct ofp_queue_prop_gs_rate) == 32);
35 };
```

Listing 1. OpenFlow Changes Needed to Support VSDN

D. Video over SDN Host API

The video of SDN API allows hosts to request QoS from network. VSDN provides user with a simple and consistent interface. The detail of API is not known to user (i.e., developer).

Name	Description
requestQoS(v, d, p)	generate a QoS request
acceptQoSRequest(s)	accept QoS request
removeQoSRequest(s)	remove QoS request
processRequest(s, d)	callback for application

Table I
HOST VIDEO OVER SDN API FOR REQUESTING, ACCEPTING AND RESPONDING TO QOS REQUEST

Video QoS is requested by calling requestQoS(v, d, p). The Sender can request three types of video services: Common Interchange Format (CIF), Enhanced Definition (ED) and High definition (HD).

The `Receiver` accepts request calling acceptQoSRequest(s). `Sender` and `Receiver` may remove session by calling removeQoSRequest(s). The application receives network related events using processRequest(s, d).

E. VSDN QoS Mapping

Name	Description
Token Rate (r)	Average rate or rate which tokens fill bucket
Token Bucket Size (b)	The number of bytes that token bucket can hold before overflow occurs.
Peak Data Rate (p)	The maximum data rate in bytes per second
Minimum Policed Unit (m)	The minimum packet size in bytes
Maximum Packet Size (M)	The maximum allow packet in bytes
Rate (R)	Maximum link capacity or peak rate
Slack Term (s)	Additive end-to-end delay that the sender can tolerate between nodes if a node modifies requested flow specifications

Table II
GUARANTEED SERVICES (GS) PARAMETER

There are two service specifications for real-time tolerant applications like video streaming and real-time intolerant applications like interactive video. The two service specifications are Controlled Load (CL) [9] and Guaranteed Service (GS) [8]. The VSDN framework utilizes GS in this paper. The attributes in table II are used by VSDN protocol to configure VSDN switches' token bucket processes [10].

The Host applications only specifies video type (i.e., CIF, ED and HD) and not flow specification attributes. The VSDN controller understands video types and must convert from video type to TSpec and send request over OpenFlow to VSDN switches. Therefore, a mapping scheme is needed to map between video type request from host to TSPec.

Table III describes the mapping between Video Type and GS specification. The values were derived from available data from Netflix and VUDU on-demand Internet streaming media service providers. The frame size in KB was used to determined the number of packets or bucket size (b) needed on average to deliver a single frame. For example, HD requires 60KB per frame which is forty 1.5KB packets. At 30 fps, video type HD would have a Rate (r) of 1200 (fbs * b) and bucket size of 40 to hold a single frame. The slack term (s) is dependent on service level agreement (SLA) between service provider and customer. Slack term will not be used by VSDN switches because in VSDN architecture, they do not make such decision about computational or communicational resources. The slack term will be used by VSDN controller to make necessary adjustments on network resources when required.

IV. RESULTS

In this section, this paper analyzes the number of and type of messages in system with relationship to number of request from hosts.

Message type	Description
setqueue	used by SDN controller to add flow table to OpenFlow devices
unsetqueue	used by SDN controller to remove flow table to OpenFlow devices
request	used by hosts to request video service from VSDN framework
accept	used by hosts to accept video request from sender
remove	used by hosts to explicitly remove session from network

Table IV
TYPES OF MESSAGES

Performance metrics - To assess the performance of VSDN algorithm, this paper chooses the following performance metric:

- *Message complexity* - measures the number of messages generated by VSDN framework per host's request. The types of messages are shown in table IV .

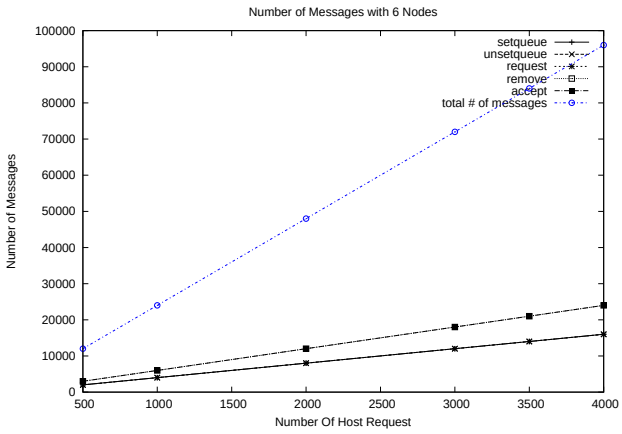

Figure 7. VSDN number of messages in system based on number of request from host(s)

Video Type	Bandwidth (BW)	Rate (r)	Bucket Size (b)	Peak Rate (p)	Minimum Policed Unit (m)	Maximum Packet Size (M)	Rate (R)	Slack Term (s)	Frames Per Second (FPS)
CIF	1.0Mbps	168	7	168	74	1522	168	SLA	24 (FPS)
ED	1.5Mbps	500	20	500	74	1522	500	SLA	25 (FPS)
HD	3.0Mbps	1200	40	1200	74	1522	1200	SLA	30 (FPS)
HDx	6.0Mbps	1620	54	1620	74	1522	1620	SLA	60 (FPS)

Table III
VSDN VIDEO TYPE TO SERVICE SPECIFICATION

Figure 7 illustrates the number of messages in system with a topology of 6 nodes. At 500 request, the number of setqueue and unsetqueue messages is 2000. The number of request messages is 2000. The number of remove and accept message is 3000 because each remove and accept messages must traverse the control plane of controller for resource management. At 1000 requests, the number of setqueue and unsetqueue messages are 4000. The number of request messages is 4000. The number of remove and accept messages is 6000. Each message type increases linearly from 500 requests to 4000 requests. The total number of messages in system for 500 requests is 12000. At 2000 requests, the total number of messages is 48000. At 4000 requests, the total number of messages in system is 96000. From the results, it can be argued that total number of messages in the system at a given time is 24 times the number of host requests. This paper did not introduce network errors during test runs which would affect the number of messages in the systems (i.e., number of error messages which is 0 in this experiment).

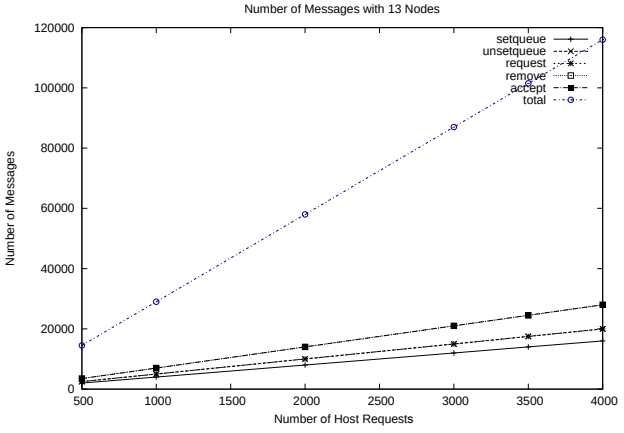

Figure 8. VSDN number of messages in system based on number of request from host(s)

Figure 8 illustrates the number of messages in system with a topology of 13 nodes. At 500 requests, the number of setqueue and unsetqueue messages is 2500 . The number of remove and accept messages is 3500. At 1000 requests, the number of setqueue and unsetqueue is 5000. The number of remove and accept messages is 7000. At 2000 requests, the number of setqueue and unsetqueue messages is 10000. From the results, each message type increases linearly from 500 host requests to 4000 host requests. The total number of messages at 500 requests is 14500. At 1000 requests, the total number of messages is 29000. At 2000 requests, the total number of messages in system is 58000. At 3000 requests, the total number of messages in system is 87000. Finally, at 4000 requests, the total number of messages in system is 116000. From these results, it can be argued that the total number of messages in the system at a given time is 29 times the number of request into system. In these test runs, this paper did not introduce network errors which would effect the number of messages in the systems (i.e., number of error messages which were 0 in this experiment).

Between figure 7 and figure 8, the total number of messages in system at a given time is 24 and 29 times number of host requests respectively. In figure 8, the total number of messages is 5 times more than total messages in figure 7 due to longer path length from source to destination with a 13 node topology. Each additional node along path will need to be configured with VSDN messages. Therefore, the number of messages will be affected by number of nodes in path.

V. RELATED WORKS

There are mainly three architectures that deals with providing end-to-end quality of services in IP network.

IntServ architecture [11] is a flow based concept that uses reservation protocols to signal end-to-end QoS between sender, network and receiver. VSDN is similar to IntServ's reservation protocols because to provide end-to-end quality of service, resources are reserved explicitly along the network path between sender and receiver. Unlike IntServ, VSDN does not require refresh messages to refresh the soft-states per flow at routers. Flooding of soft-state refresh is one of the major disadvantages of IntServ which affects scalability [12]. VSDN is capable of selecting optimal path

based on requirement of video application. IntServ is not capable of exploring alternate paths and install or select path based on IP routing protocol.

Differentiated Services(DiffServ) [12] uses flow aggregation and a hop by hop decision making process to address scalability issues that were associated with IntServ. DiffServ applies a network-wide set of traffic classes. Flows from sender and receiver are classified in a predefined manner by network operator. A router or switch, upon receiving a packet marked with DiffServ value, will apply scheduling and shaping techniques based on class. The classification is based on Type of Service (TOS) header field in IP header. Unlike VSDN, DiffServ can not guarantee QoS to application because each router is configured independently of the other and network wide policing is difficult because there is no global network view.

MPLS is a layer 2.5 label switching technique that inserts a label for a network prefix to allow routers to perform a quick look up of label instead of using longest prefix matching [13]. This technique allows MPLS to perform faster packet classifications and forwarding. As with DiffServ a number of flows can be aggregated or classified along a path. VSDN works on a per flow basis but could aggregate flows from a single user for prefix to further optimize VSDN. Unlike VSDN, MPLS does not have the real-time path configuration ability. VSDN can make real-time configuration changes due to network conditions (*e.g.*, link errors and network congestion) without prior knowledge about network traffic. Aggregation of flows from same host or prefix by MPLS is a technique that VSDN could utilize.

VI. Conclusion and Future Work

This paper presented *Video over Software Defined Networking (VSDN)*. VSDN is capable of selecting the optimum path for video applications which can improve the QoS of video applications. A VSDN API allows application developers to request service from VSDN enabled networks. A prototype was developed to illustrate VSDN core functionality. The experiment with prototype suggests that message complexity of VSDN is linear.

After conceptualizing, designing and implementing VSDN in a simulator, the following lessons where learned.

- **Development of a network service.** Developing a network service requires deep thought about usability.
- **Using a single SDN controller.** Using a single network controller for large scale networks can lead to performance issues.
- **Separation of control and data plane.** Separating the control and data plane allows a great deal of flexibility in designing VSDN.

References

[1] H. Egilmez, B. Gorkemli, A. Tekalp, and S. Civanlar, "Scalable video streaming over openflow networks: An optimization framework for qos routing," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*. Brussels, Belguim: IEEE, sept. 2011, pp. 2241 –2244.

[2] O. N. F. (ONF), "Software-defined networking: The new norm for networks," https://www.opennetworking.org/images/stories/downloads/white-papers/wp-sdn-newnorm.pdf, 2012.

[3] OpenFlow, "Openflow switch specification, version 1.3.0," https://www.opennetworking.org/images/stories/downloads/specification/openflow-spec-v1.3.0.pdf, 2012.

[4] E. Al-Shaer and S. Al-Haj, "Flowchecker: configuration analysis and verification of federated openflow infrastructures," in *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, ser. SafeConfig '10. New York, NY, USA: ACM, 2010, pp. 37–44.

[5] O. Younis and S. Fahmy, "Constraint-based routing in the internet: Basic principles and recent research," *IEEE Communications Surveys and Tutorials*, vol. 5, pp. 2–13, 2003.

[6] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, July 2008.

[7] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proceedings of the 2nd ACM SIGCOMM workshop on Home networks*, ser. HomeNets '11. New York, NY, USA: ACM, 2011, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2018567.2018569

[8] S. Shenker and C. Partridge, "Specification of guaranteed quality of service," http://www.ietf.org/rfc/rfc2212.txt, 1997, rFC 2212.

[9] J. Wroclawski, "Specification of the controlled-load network element service," http://www.ietf.org/rfc/rfc2211.txt, 1997, rFC 2211.

[10] P. P. Tang and T.-Y. C. Tai, "Network traffic characterization using token bucket model," in *In Proceedings of IEEE Infocom99*, 1999, pp. 51–62.

[11] X. Xiao and L. M. Ni, "Internet qos: A big picture," *IEEE Network*, vol. 13, pp. 8–18, 1999.

[12] R. Chakravorty, S. Kar, and P. Farjami, "End-to-end internet quality of service (qos): An overview of issues, architectures and frameworks," 2000.

[13] J. Ruela and M. Ricardo, "MPLS - Multiprotocol Label Switching," in *The Industrial Information Technology Handbook*, 2005, pp. 1–9.