Layer based routing to obtain quality optimization of video streaming in the Future Internet

Sander Vrijders, Piet Smet, Youri Flement, Bruno Chevalier
Ghent University - iMinds, Department of Information Technology,
Gaston Crommenlaan 8 bus 201, 9050 Gent, Belgium

***Abstract*—In TCP/IP networks, there is no inherent support for Quality of Service. Routing is done on the minimum hop count from sender to receiver, and all traffic is best-effort traffic. This is unsatisfactory for applications such as video streaming. This paper presents an OpenFlow controller application which provides Quality of Service (*QoS*) support for video streaming using Scalable Video Coding (*SVC*) in an OpenFlow network. First we route the base layer of SVC encoded videos as a high priority QoS flow while the enhancement layers are routed at a lower QoS or as a best-effort flow. Secondly, we provide failure recovery so the OpenFlow network can converge back to an optimal state when links fail. We show that using different costs to calculate paths for different flows can improve the overall throughput of video streams when network congestion and delay occur.**

I. Introduction

In a TCP/IP-based network, each router or switch contains a flow table containing the next hop for each destination, based on the shortest path calculation. Media applications, such as video streaming, require minimal packet loss and minimal network delay. When these video flows are sent over the same path as other best effort traffic they are subject to network congestion and delay which renders an unsatisfactory outcome. Several efforts have been made to give precedence to certain types of traffic, defined as Quality of Service (QoS). There are solutions available for QoS support in a TCP/IP-based network, such as DiffServ [1] and IntServ [2], but these solutions only work well in managed networks. Tunneling with MPLS offers a partial solution, but it lacks real-time reconfigurability and adaptivity. [3]

Scalable Video Coding (SVC) [4] encodes the stream in a base layer along with several enhancement layers. In order to obtain qualitative video streams, it is important that the base layer has no packet loss and minimal delay, which is hard to accomplish without support for QoS. To solve this issue we need to differentiate between the different flows and route these flows differently than normal best effort traffic.

We use OpenFlow [5] in order to provide QoS support. OpenFlow is a protocol for switches, allowing to control the packet forwarding process. This way, networks are not only configurable, but also programmable. Each OpenFlow Switch has a flow table, which is used for packet lookup and forwarding, and a secure channel to an external controller (see Figure 1). The flow table consists of flow entries. Each flow table entry matches a traffic pattern, by matching the header fields. The header fields that can be matched in OpenFlow 1.0 can be seen in Table I. The controller makes the network programmable by managing the switch over the secure channel using the OpenFlow protocol. The controller installs new entries in the flow table to manage the network traffic. Entries have a hard and a soft timeout. A soft timeout means the entry gets removed if it has not been used for a certain time. A hard timeout means the flow will be removed after the timeout, even if it is still in use. There are many implementations of controllers available for OpenFlow switches. The implementation of the controller we will use is Floodlight [6], because it is open source and widely used.

In Port	Ethernet Source Address	Ethernet Destination Address
Ethernet Type	VLAN ID	VLAN Priority
IP Source	IP Destination	IP Protocol
IP ToS Bits	TCP/UDP Source Port	TCP/UDP Destination Port

TABLE I
Fields that can be matched in OpenFlow 1.0

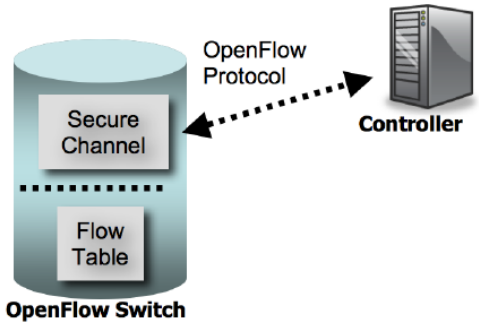

Fig. 1. How OpenFlow works

The rest of this paper is organized as follows. In Section II of this paper we compare related work to our own. In section III we take a closer look at the design considerations for developing an OpenFlow controller application that provides layer-based quality optimization. In section IV we present the general architecture, and in section V we show the algorithms used. Section VI shows experimental results. In Section VII we come to a conclusion.

II. Related work

In our research we identified a need for separate forwarding actions per flow. Wang, R. et al. [7] follow the same principal.

The aim was to apply load-balancing during service discovery of web services when all requests had to be processed by a central component. A problem occurs when each request is forwarded to a centralized component responsible for service discovery, which leads to network congestion. By using OpenFlow they managed to set separate forwarding actions for client requests which match certain criteria. These actions allow forwarding entities to redirect client requests to an appropriate service instance without interference from the centralized component. Even though the use case may be different, the principal is the same as the one applied in our research. When all traffic follows the same path, chances of congestion and packet loss occurring increase significantly. OpenFlow allows us to set up different paths for different flows, which improves the throughput of our QoS flows and also decrease the network load on links used by best effort traffic.

The idea to provide a minimal quality for SVC encoded videos originates from research described in [8] and [9]. By routing the base layer as a lossless QoS flow, which means there is no packet loss, they can guarantee a minimal quality to the user. Ideally, the enhancement layers will also be delivered without packet loss or delay but when congestion occurs they demote the enhancement layers to best effort traffic and prioritize the base layer.

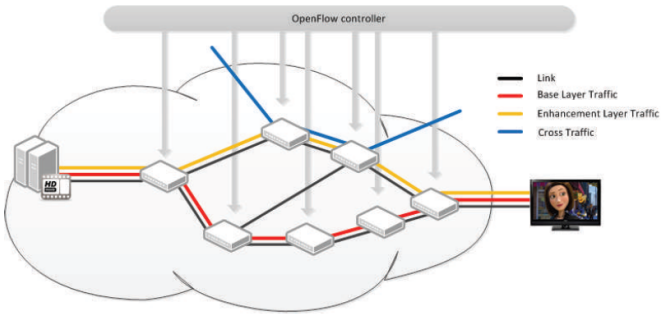

Fig. 2. General problem

In our own research we define the base layer as lossless QoS traffic, which means no packet loss is allowed. The enhancement layers are lossy QoS traffic, which means they may still follow a different route than best-effort traffic, but some packet loss is allowed. These principals are shown in Figure 2. We also enable fast failure recovery when a link fails, as described in [10]. Also, we present experimental results, whereas other research only presents simulations.

III. Design considerations

In designing the OpenFlow application, we considered the following.

- When high priority QoS flows are forwarded on the same link as best effort traffic, there will be packet loss in the best effort flow to prioritize the QoS flow.
- Metrics for QoS flows must be different than those used for best effort traffic. The shortest path and QoS path must be different to obtain good results. If both follow the same path, the best effort traffic will experience more packet loss.
- Calculating a path for a new QoS flow does not change the best effort routing tables. The packet loss of best effort traffic caused by QoS flows must be considered during path selection.
- The length of alternative paths for QoS flows should still be kept as short as possible to optimize transfer time.
- Best effort traffic can be used as a reference to retrieve metrics from a switch or link, such as packet loss or to estimate the amount of traffic on that route.

These observations show that packet loss and path length are important in defining algorithms: the algorithm must find a balance between maintaining low packet loss of the best effort traffic and the delay of QoS flows, caused by using longer paths.

IV. Architecture

To obtain the different SVC video streams, we use HTTP adaptive streaming. This means that the video is split into different segments, and when a new segment starts playing, the following segment is requested using HTTP. If the next segment doesn't arrive in time, it is skipped. In the case of SVC streaming the video stream is first split into different layers (base and enhancement layers). These layers are then further split into segments of different size, but equal playtime. If a segment of a layer does not arrive in time, it is skipped.

Each OpenFlow network contains at least one controller which is responsible for managing all the switches. An OpenFlow switch forwards packets to the controller if the packet does not match any forwarding entry in the forwarding table. Matching is done on the longest prefix. Wildcards are allowed in the forwarding table. A filter in the controller separates QoS flows from other traffic, based on the TCP port, after which the controller can choose a path for the different layers. When the controller calculates a path for a layer, the new forwarding rules are sent to the switches.

The different paths are calculated using Dijkstra. Depending on the current load of the network, a cost is given to each link. The controller must be able to obtain the load any time from any switch in the network before calculating new flow tables. We implemented a monitor module on the controller to gather all network statistics. Besides polling for statistics, a switch also forwards events to the controller when they occur, such as a link failure. These events are also act upon by the controller, by recalculating a different path for every flow.

V. Algorithms

Because the base layer has to be delivered lossless, the weights given to each link depends on the load of the switches connected to these switches. The minimum value a link can get is 1, so the hop count is used when the network experiences no load. The pseudocode can be seen in Algorithm 1. This algorithm is executed on receiving a packet-in event from a switch with TCP port set to 8080.

Algorithm 1 Calculate route for base layer

for all $link$ in $linksInNetwork$ **do**
  $leftSwitch \leftarrow link.getSrc()$
  $rightSwitch \leftarrow link.getDst()$
  $loadLeft \leftarrow monitorModule.getLoad(LeftSwitch)$
  $loadRight \leftarrow monitorModule.getLoad(rightSwitch)$
  $totalLoad \leftarrow loadLeft + loadRight + 1$
  $linkMap.put(link, weight)$
end for$Dijkstra(linkMap)$

For the enhancement layers we followed the same tactic, but we also hold the route that the base layer is following into account. This means we gave a higher cost to the links that the base layer was using, thus ensuring the base layer is delivered lossless. The pseudocode for this algorithm can be seen in Algorithm 2. It is executed on packet-in events with the TCP port of the packet set to 8081,8082 or 8083. The constant extraLoad can be tweaked to render optimal performance. It must be set high enough to ensure the route the base layer is using is not used, unless the traffic on the other links is much higher.

Algorithm 2 Calculate route for base layer

for all $link$ in $linksInNetwork$ **do**
  $leftSwitch \leftarrow link.getSrc()$
  $rightSwitch \leftarrow link.getDst()$
  $loadLeft \leftarrow monitorModule.getLoad(LeftSwitch)$
  $loadRight \leftarrow monitorModule.getLoad(rightSwitch)$
  $totalLoad \leftarrow loadLeft + loadRight + 1$
  if $linksRouteBaseLayer.contains(link)$ **then**
    $totalLoad \leftarrow totalLoad + Constants.extraLoad()$
  end if
  $linkMap.put(link, weight)$
end for$Dijkstra(linkMap)$

VI. Experimental results

All of the experiments are performed on the iLab.t Virtual Wall at iMinds. The Virtual Wall is derived from Emulab, developed at Utah university. The Virtual Wall facility is a generic test environment for advanced network, distributed software and service evaluation, and supports scalability research.

The Virtual Wall facilities consist of 100 nodes (dual processor, dual core servers, 4x1 or 6x1 Gb/s interfaces per node) interconnected via a non-blocking 1.5 Tb/s VLAN Ethernet switch enabling flexible creation of experiment topologies, and a display wall (20 monitors) for experiment visualization. Each server is connected with 4 or 6 gigabit Ethernet links to the switch. The Virtual Wall nodes can be assigned different functionalities ranging from terminal, server, network node, and impairment node. Each node is fully dedicated and under full control of one experiment, so the users can test all kind of operating system images and configurations [11].

We created the topology shown in Figure 3, on which we performed the experiments. Each link has a 1 Gb connection. The switches are running OpenVSwitch instead of the reference user-space switch, because Floodlight won't work with the reference user-space switch.

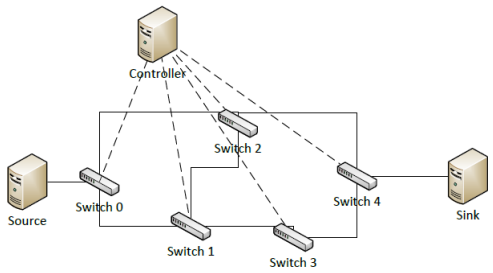

Fig. 3. Topology used

To measure the quality of the video we chose the PSNR value of the received video compared to the raw footage. The higher the PSNR value, the better the frames were preserved during transfer. A base layer has a low PSNR value, and every extra enhancement layer that is also received, increases this value. Unfortunately, we did not have enough time to obtain results.

VII. Conclusion

In this paper we described a way to perform quality optimization of video streaming. We do this by using HTTP adaptive streaming of an SVC encoded video stream, which is composed of a base layer and three enhancement layers. Each layer is routed differently, depending on the required Quality of Service. The base layer is delivered as a lossless video stream. This means it is always delivered. It has the highest QoS priority. The enhancement layers are considered lossy video streams, which means they may not always arrive, depending on the network load. We also implemented fast failure recovery. This means if a link fails, the routes get recalculated very fast, rendering an optimal performance.

In future work we will try to optimize the weights given to each link in the algorithm. This will give an even higher quality optimization.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," December 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2475.txt

[2] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," June 1994. [Online]. Available: http://tools.ietf.org/html/rfc1633

[3] E. Rosen and Y. Rekhter, "Bgp/mpls vpns," March 1999. [Online]. Available: http://tools.ietf.org/html/rfc2547

[4] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the h. 264/avc standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 9, pp. 1103–1120, 2007.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[6] "Floodlight controller for openflow," 2011. [Online]. Available: http://floodlight.openflowhub.org/

[7] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*. USENIX Association, 2011, pp. 12–12.

[8] 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*. IEEE, 2011, pp. 2241–2244.

[9] S. Civanlar, M. Parlakisik, A. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem, "A qos-enabled openflow environment for scalable video streaming," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 351–356.

[10] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Enabling fast failure recovery in openflow networks," in *Design of Reliable Communication Networks (DRCN), 2011 8th International Workshop on the*. IEEE, 2011, pp. 164–171.

[11] "ilab.t virtual wall," 2012. [Online]. Available: http://www.iminds.be/en/develop-test/ilab-t/virtual-wall