GENI Cinema: A SDN-Assisted Scalable Live Video Streaming Service

Qing Wang, Ke Xu, Ryan Izard, Benton Kribbs,
Joseph Porter, Kuang-Ching Wang
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, USA
{qw, kxu, rizard, bkribbs, jvporte, kwang}@clemson.edu

Aditya Prakash, Parmesh Ramanathan
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
Madison, WI, USA
{aprakash6, parmesh}@ece.wisc.edu

***Abstract*— This paper introduces GENI Cinema (GC), a system that provides a scalable live video streaming service based on dynamic traffic steering with software defined networking (SDN) and demand driven instantiation of video relay servers in NSF GENI's distributed cloud environments. While the service can be used to relay a multitude of video content, its initial objective is to support live video streaming of educational content such as lectures and seminars among university campuses. Users on any campus would bootstrap video upload or download via a public web portal and, for scalability, have the video delivered seamlessly across the network over one or multiple paths selected and dynamically controlled by GC. The architecture aims to provide a framework for addressing several well-known limitations of video streaming in today's Internet, where little control is available for controlling forwarding paths of on-demand live video streams. GC utilizes GENI's distributed cloud servers to host on-demand video servers/relays and its OpenFlow SDN to achieve seamless video upload/download and optimization of forwarding paths in the network core. This paper presents the architecture and an early prototype of the basic GC framework, together with some initial performance measurement results.**

Keywords—OpenFlow; video streaming; cloud computing; network architecture;

I. INTRODUCTION

Skype or Google Handout already provide a live video service in today's world but still have the limitation on scalability on-demand [1]. Two key challenges need to be overcome in order to supply stable and reliable live video streaming among different locations: (1) the video service must provide a persistent two-way real-time interaction capability among a possible large amount of users. If a large traffic volume happens, the bottleneck of the video service is the fixed network bandwidth conditions in either the Internet core or campus local networks. This is the motivation behind enhancing the throughput of the live video streaming connection. Decades of research have been done to accomplish this goal and it turns out the appropriate and effective way is to improve the throughput with parallel transports, which means multiple paths could be used to forward the traffic [2]. But the wide-scale deployment of parallel data transfer technologies is still an existing challenge. And (2) either the Internet core or campus network must resolve the congestion problem. Assume the campus network bandwidth is sufficient to support all student connections on the local campus, but within the Internet core, the quantity of video streams is much larger when the number of campuses participating and each campus' equally large number of students is taken into consideration. The way to manage this tremendous number of video streams in the Internet core and to avoid traffic congestion is an active and unsolved problem. From both the user and the traffic management perspective, in general, the service framework needs to be scalable and flexible on-demand for sending each video stream over multiple paths efficiently.

With a traditional non-SDN solution, these challenges above are not easy to overcome due to the limitation of the underlying network architecture's capabilities. A SDN-Assisted live video streaming service is required to overcome those limitations. SDN is a deeply programmable, strongly abstracted, scalable, and logically centralized control network architecture. It can influence one or multiple forwarding paths in the network on-the-fly by interacting with control applications and routing devices in the network. OpenFlow controller is a centralized control point for the entire network. This centralization of control provides scalable and flexible capabilities where network resources can be added, removed and updated in real-time. Due to the centralized control, the controller can learn "global" network information rather than the traditional routing device's "local" network information, which allows the controller to more efficiently forward traffic.

The Global Environment for Network Innovations (GENI) [3] provides an OpenFlow testbed for evaluating SDN applications [4]. One important goal for GENI is to leverage SDN features to create a programmable network environment. Currently, video streaming service is one of the solicitations for researchers to design and deploy using GENI network infrastructure. The initial objective of the GENI Cinema (GC) service is to provide an long-running on-line educational video service among nationwide campuses using GENI's network infrastructure. Users can bootstrap video upload/download via a public web portal. On the back-end, the service proposes a SDN-Assisted framework for providing scalable, seamless, live video streams service over one or multiple paths selected and dynamically controlled by GC. GC utilizes GENI's abundant

978-1-4799-6204-4/14 $31.00 © 2014 IEEE
DOI 10.1109/ICNP.2014.84

distributed cloud resources as video servers/relays. The service overcomes the limitations of traditional live video services using SDN features. The video quality can be improved with OpenFlow, since OpenFlow provides more flexible and efficient control over data forwarding among paths. Multi-path data delivery methods with OpenFlow, such as network coding [5], are an effective way to achieve the throughput requirement of live video streams. And for scalability, the centralized controller can also manage a large number of traffic flows, which can be a benefit when mitigating network congestion.

At this point, the GC service as a whole has not been completed. This paper presents a prototype conceptual experiment that demonstrates the GC framework development. The remainder of this paper is structured as follows: Section II explains the proposed video streaming service architecture and the main components. Section III introduces the experiment design using the GENI testbed, as well as the details of video streaming uploading/downloading procedures. Section IV provides a conclusion and details future work.

II. GC SDN-ASSISTED VIDEO STREAMING SERVICE

This section first gives an overview of the architecture of the GC SDN-Assisted video streaming service. It then introduces the main components of this architecture and the function of each.

A. Service Architecture Overview

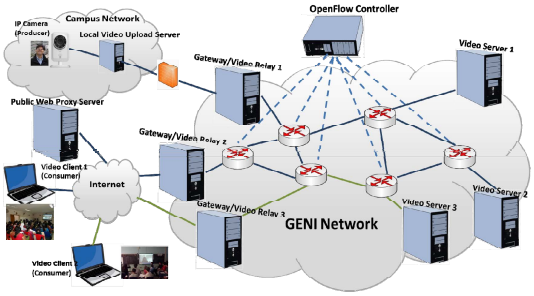

Fig. 1 Service Architecture Overview

The overall system architecture is illustrated as shown in Fig. 1. The architecture consists of the following components: (1) a local video upload server, (2) multiple GC gateways, (3) an OpenFlow controller, (4) multiple video relays, (5) multiple video servers, and (6) a public web proxy server. A typical workflow in this architecture could be decoupled as four parts: (1) gateway selection, (2) video upload, (3) video download, and (4) the forwarding of packets of each video stream over one or multiple paths. Overall, a video producer/consumer inside a campus first accesses the public web proxy server to select an appropriate gateway/video server. After that, the video upload/download process is redirected to that appropriate gateway. For the uploading procedure, videos are first uploaded onto a GC gateway from the video source. After that, the OpenFlow controller executes the forwarding decision between the gateway and the video server(s). For the download procedure, a video request from a client is initially sent to the GC gateway and automatically transmitted to the selected video server by the gateway. After the video server responds to the request, the video is downloaded from the appropriate video server through the GC gateway and is finally sent to the client. The forwarding path decision is still determined by the OpenFlow controller.

The GC architecture could be decoupled as two core subsystems: (1) end-to-end video streaming and (2) the optimal control of efficient data forwarding in the network. The first subsystem consists of an open-source live media streaming server and client framework for video production and consumption. The second subsystem focuses on forwarding traffic under optimal control over GENI network. The centralized SDN controller provides the feasibility and flexibility to control the flow traffic, which greatly increases the scalability of the service. GC service also decouples the traffic forwarding inside the GENI network from user. From the user's perspective, he/she only communicates with the public interface of the gateway and the public web server. Inside the GENI network, one centralized OpenFlow controller can dynamically change the forwarding path based on traffic load or based on network coding efficiency.

B. Functionality of Main Components

This section will mainly describe the functionality of the GC gateway, the public web proxy server, and the video server.

1) Public Web Proxy Server

A public web proxy server is a regular public web server that interacts with video users to convey gateway and video server status information. Since GC gateways and video servers are all across the country, it is necessary for a video user to select the appropriate gateway or video server (e.g. an available and nearby one) in order to decrease the upload/download time and improve the video streaming performance. The gateway or video server status information depends on a few network traffic indicators, such as gateway/video server availability, packet delay between the user and gateway, and the streaming rate in bits per second.

2) Gateway

GC gateways are physical or virtual machines running in multiple geographically distributed GENI racks. They played a key role in GC service architecture. A gateway provides a public interface that interacts with video producers/consumers, and a private interface that interacts with video servers. One main function achieved here is the capability to modify packet headers using OpenFlow so that the video streams can be automatically forwarded to multiple video servers. In the real world, campuses are either OpenFlow-enabled or not OpenFlow-enabled. To satisfy all campuses, regardless of OpenFlow deployments, GC users can simply interact with the nearby GC gateway as the GC gateway has a public interface.

There are three key software components in GC gateway, as shown in Fig. 2. They are: (1) a frontend connect & select (C&S) server, (2) a backend streaming relaying (SR) server, and (3) an OVS bridge. The C&S server acts as public interface. It authenticates, establishes, and terminates the connection between the GC gateway and the video server, for

both video upload and video download tasks. It communicates the user information, (e.g. transport protocols, user's IP address and receiving port number). The SR server is responsible for enabling optimal control of efficient video forwarding from the incoming video source to all video servers over one or multiple paths using OpenFlow. Lastly, the OVS bridge supports controlled traffic forwarding.

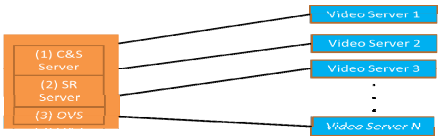

Fig. 2 GENI Cinema Gateway Diagram

3) Video Servers

Video servers are GENI racks as well. The key software components' functionalities, as shown in Fig. 3, are very similar to the GC gateway: (1) a VideoLAN (VLC) server, (2) a SR server and (3) an OVS bridge. Instead of a C&S server in gateway, a VLC server is set up to listen for the request. The SR server is responsible for optimal forwarding, path control. OVS bridge allows for controlled video traffic forwarding.

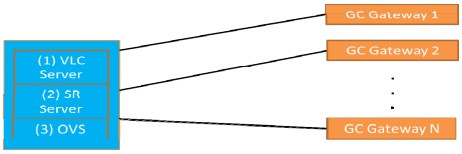

Fig. 3 GENI Cinema Video Server Diagram

III. EXPERIMENTS USING GENI TESTBED

This section first introduces the details of the experiment design using the GENI testbed. It then shows the details of the video upload/download implementation and its results.

A. Experiment Design Using GENI Testbed

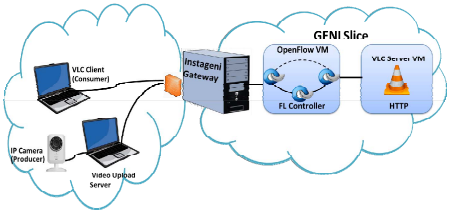

Fig. 4 Demo Experiment Scenario

The experiment scenario implements the base framework of the GC service solution. Within a limited time, an initial prototype with certain simplifications has been designed with GENI networks as shown in Fig. 4. This prototype mainly demonstrates the core concepts for the two subsystems of the GC service: (1) end-to-end video uploading/downloading and (2) OpenFlow forwarding and path control. An IP camera and a wireless router are used in addition to the resources of the testbed. The IP camera can either get a public campus network IP or obtain a private IP from a router. Local video upload server is needed because almost every campus has an active firewall, and GENI racks are usually deployed outside campus firewalls. GC gateway has its own firewall rules with port 30000+ [6] allowed. So video streams cannot be directly fetched from the GC gateway. Local video upload server acts as a video relay that initiates the connection from inside campus firewall to relay the video streams on behalf of the camera to the GC gateway. After that, the video goes into the GENI network. In GENI's network, three InstaGENI aggregates communicate with each other. Their locations are: (1) a GC gateway VM at Boston, (2) an OpenFlow VM at Wisconsin, and (3) a VLC video server VM at Stanford. Since a video stream contains a large amount of data to be transferred, a customized layer 2 network is required to ensure the sufficient bandwidth. Thus, the network topology is configured with GENI Stitcher [7] using an Rspec file.

Between the GC gateway and the video server, the video needs to be automatically forwarded over multiple paths. A programmed OpenFlow scheme is able to accomplish this but requires complex custom code to fully implement it. To prove the key concept as a first step, Linux Network Address Translation (NAT) is configured on both VM to automatically forward video streams from one to another. The OpenFlow-based NAT will be implemented latter. The OpenFlow VM is the place to prove the concept of optimal forwarding path control with OpenFlow. On this VM, there are three OVS bridges connected each other and a local OpenFlow controller installed. The OpenFlow Floodlight controller is chosen. It statically pushes flows with its REST API onto the OVS bridges to select a path: one is the "long path" where traffic goes through three OVS bridges and the other is the "short path" where traffic goes through only two OVS bridges.

B. UDP Video Uploading Procedure

On local video upload server, RTSP video streaming is fetched from IP camera on port 554. It then uses VLC to send incoming video to the IP address of GC gateway with port number 34567 over UDP. Each UDP packet has the source IP address with local server and destination IP address with GC gateway. GC Gateway automatically relays the video stream towards the VLC video HTTP servers via NAT. NAT changes the packet headers' destination IP at the GC to the VLC server. So from the video producer's point of view, the video is just uploaded onto the GC gateway. OF controller executes the routing decision by REST API on whether "long path" or "short path" flows are inserted. The video is on VLC HTTP server at Stanford. On VLC server, a VLC HTTP server is running to listen to the interface with port number 34568.

C. TCP Video Downloading Procedure

Client uses VLC to send a video request to GC gateway with port number 34568. TCP video request packet has the source IP of the client and destination IP of the GC gateway. GC gateway automatically sends video request to the

appropriate video server. NAT changes TCP packet header destination IP from the GC gateway to the VLC server. OF controller executes “long path” or “short path” decision. VLC server receives the video request and sends the video response packet back to the GC gateway. After that the VLC server sends the video to the GC gateway. NAT changes the TCP packet header for both processes. GC gateway receives both the request and video packet from the VLC server and changes the TCP packet header source IP from the video server to the GC gateway. The client receives the TCP video packets from the GC gateway. So from the video consumer’s point of view, the video is originating from the GC Gateway.

D. Experiment Results

Couple key service performances and capabilities are evaluated in this section. The lower-bound channel bandwidth requirement of the service is evaluated under a fixed high video resolution condition. A basic video resolution requirement is also found under a fixed bandwidth. A flow entry control structure is lastly shown in order to show the service capability of network traffic control.

A live video snapshot (Fig. 5) with high-quality, persistent, and stable video quality demonstrates the GC service addressed the end-to-end video streaming requirement. The link bandwidth is 300Mbps, and the video resolution is 1080p (1920 x 1080). The left image in Fig. 5 is the video producer and the right one is the video consumer. The delay is about 5 seconds. An on-demand link bandwidth test is then evaluated to show the basic bandwidth requirement of the service. The 1080p video starts to degrade in quality if the channel bandwidth decreases to 2Mbps. With 3Mbps or above, video quality is stable and reliable. It turns out the service does not consume a large link bandwidth, and does potentially have a scalable space to handle a larger number of users. A series of video resolution tests are also evaluated with a 1Mbps channel bandwidth to show the basic video resolution requirement. For this resolution test, the 1080p video frame size is too large for the channel bandwidth. The resolution of 800 x 450 is the highest that could reproduce the video on client end, but with freezing and distortion in video frames (Fig. 6 (a)-(b)). The resolution 176 x 144 is the only one that guarantees good video quality on a 1Mbps link, as shown in Fig. 6 (c)-(d).

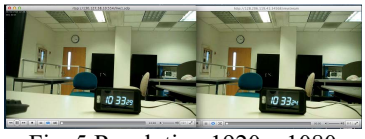

Fig. 5 Resolution 1920 x 1080

(a) (b) (c) (d)

Fig. 6 (a)(b) Resolution 800 x 450 with distortion;
(c)(d) Resolution 176 x 144 without distortion.

The flow control structure of one switch (Fig. 7) shows that an OpenFlow controller can easily manipulate the network traffic flows by changing related fields in this structure. In “short path” or “long path” cases, one switch only needs to install four flow rules in total. For dynamic control, one only needs to customize an OpenFlow controller application to interact with the data plane traffic in–time. This dynamic and feasible traffic control is a critical benefit for GC service since this on-demand network traffic management alleviates the congestion problem in the Internet core.

```
flow1 = {
    'switch':"00:00:00:00:00:00:00:01",
    "name":"br1-arp-flow-1-14",
    "cookie":"0",
    "ether-type":"0x806",
    "priority":"32768",
    "ingress-port":"1",
    "active":"true",
    "actions":"output=14"
    }
```

Fig.7 One Switch Flow entry control structure

IV. Conclusions and Future Work

This paper presents a proposed SDN-Assisted GC service architecture. GC service is a scalable live video streaming service without requiring a high link bandwidth. It deeply leverages SDN features and the abundant resources in the GENI infrastructure. An early prototype conceptual video streaming experiment has been implemented in order to prove the key concepts of the architecture. The live video is successfully uploaded to the gateway and automatically forwarded to the video server. Clients can download the live video from the server via the gateway. The download video quality is stable and reliable. Future work will be conducted on the GENI testbed to further develop, evaluate, and test GC service. First, we need to implement all the functionalities of the GC gateway. Second, video streaming over multiple paths via OpenFlow will be implemented.

References

[1] De Cicco, Luca, Saverio Mascolo, and Vittorio Palmisano. "Skype Video congestion control: An experimental investigation." *Elsevier Comput. Netw.,* vol. 55, no.3, pp. 558-571, 2011

[2] T. J. Hacker, B. D. Noble, and B. D. Athey, “Improving throughput and maintaining fairness using parallel TCP,” in *Proc. IEEE INFOCOM* 2004, Hongkong, China, 2004, pp. 1-10.

[3] GENI. [online]. 2014, http://www.geni.net/ (Accessed: 15 July 2014)

[4] R. Izard, A. Hodges, J. Liu, J. Martin, K.-C. Wang and K. Xu, ‘”An OpenFlow testbed for the evaluation of vertical handover decision algorithms in heterogeneous wireless networks,” in *Proc. EAI TRIDENTCOM* 2014, GuangZhou, China, 2014, pp. 1-10.

[5] K. Xu, S. Sampathkumar, K.-C. Wang and P. Ramanathan, "Network coding for efficient broadband data delivery in infrastructure-based vehicular networks with OpenFlow," in *Proc. Second GREE 2013*, Salt Lake City, UT, USA, pp. 56-60, 2013.

[6] CheckList-ProtoGeni. [online]. 2014, http://www.protogeni.net/wiki/InstaGENI/checklist (Accessed: 15 July 2014)

[7] GENI Stitching. [online]. 2014, http://groups.geni.net/geni/wiki/GeniNetworkStitching (Accessed: 15 July 2014