Steroid OpenFlow Service: Seamless Network Service Delivery in Software Defined Networks

Aaron Rosen and Kuang-Ching Wang
Holcombe Department of Electrical and Computer Engineering
Clemson University
Clemson, SC
{arosen, kwang}@clemson.edu

Abstract— **In a software defined network (SDN), packet forwarding is controlled by software controllers. In an OpenFlow SDN, a controller can control the forwarding, rewriting, and dropping of packets based on their header attributes. The ability to handle packets in customizable ways in software has significant implications for both network users and operators. Via software, users can convey application specific expectations while operators can deliver application specific services to enhance user experiences. In this paper, we present the Steroid OpenFlow Services (SOS) paradigm for network services delivery. The paradigm enables operators to deliver network services without any setup requirements on user machines. SOS utilizes OpenFlow to *redirect* application specific traffic to application specific *service agents*; SOS also *rewrites* packet headers for a service to remain seamless to users. This paper presents an example SOS service for optimizing large volume TCP download across a large delay-bandwidth-product wide area network. SOS service agents on both ends of the connection seamlessly terminate a user TCP connection, launch a set of parallel TCP connections, and leverage multiple paths when available to maximize throughput. With the NSF GENI future Internet testbed, a prototype implementation achieved up to 320 times throughput enhancement seamless to the end users.**

Keywords- Software Defined Network, OpenFlow, Service

I. INTRODUCTION

Amidst efforts to speed up innovations in future Internet solutions, Software Defined Networking (SDN) emerges as one attractive approach to realize novel Internet switching methods via software controllers. OpenFlow as one SDN approach defines a standardized messaging interface for a software controller to communicate with and control the data plane of an arbitrary set of Ethernet switches [1]. With OpenFlow, network operators can monitor and control the data plane of their packet switched network infrastructure in customizable layer fidelities via one or few centralized software controllers, referred to as OpenFlow controllers. Major industrial proponents including Google, Microsoft, Yahoo, Facebook, Verizon, Deutsch Telecom and others have recently formed the Open Networking Foundation to drive the standardization and realization of OpenFlow [2].

The advent of SDN breathes a paradigm shift in the service delivery architecture in Internet. In the past, the network infrastructure is application-agnostic and offers very few service level options. The majority of applications operate over a best-effort network service, while special applications with such constraints as very high bandwidth, real time latency, and/or security must utilize specialized client software for their data transport to take place over the public Internet or private fiber networks. For example, scientific projects like the Large Hadron Collider (LHC) generate petabytes of data each year. In order to efficiently move such large amounts of data across the network for researcher access today, special software and hardware infrastructure such as the Globus GridFTP [3] (client software and specialized gateways) has been needed. The need of specialized setup and training for its users, however, limits its utilization to only a very specialized user community.

With SDN, such enhanced network services can be tailored for different applications seamless to the end user. This paper presents Steroid OpenFlow Service (SOS). By enabling network operators with the ability to flexibly and seamlessly redirect specific applications' traffic to SOS service agents, these agents can flexibly apply a range of techniques such as using a better transport protocol, reserving resources on behalf of the user, and utilizing multiple paths. SOS provides network operators a tool to decouple applications' protocol choices from the network's data transport mechanism. It represents a major paradigm shift, allowing operators to deliver network services seamlessly as well as to evolve such services perpetually without interruption to users.

SOS is realized across multiple participating sites, based on an OpenFlow controller controlling OpenFlow-enabled switches and service specific software agents at each site. SOS can operate among islands of OpenFlow networks embedded in non-OpenFlow IP networks, while it is beneficial to have OpenFlow-enabled switches in the core to leverage multipath and load balancing opportunities. SOS has been implemented over the NSF Global Environment for Network Innovations (GENI) testbed [4], demonstrating over an "around-the-country" path up to 320 times TCP throughput enhancements over regular TCP without SOS.

The rest of the paper is organized as follows. Section 2 describes the OpenFlow network model, the SOS architecture, and its associated discovery mechanism. Section 3 describes the SOS setup on the NSF GENI testbed across Clemson, SC, Stanford, CA, Seattle, WA, and Cambridge, MA. Section 4 discusses the implication of SOS for various emerging applications for the future Internet and a number of closely related works. The paper concludes in Section 5.

II. NETWORK MODEL

A. OpenFlow Network

OpenFlow originated as a solution for researchers to experiment novel protocols over an existing Ethernet network [5], while over time it attracted industrial attention as a promising new way to operate future enterprise networks and the Internet [2]. To enable existing Ethernet switches to support OpenFlow with the least effort, OpenFlow defines a standardized and secure interface to access and control a switch's flow table – a common component available on the majority of vendors' existing switches. Over a secure messaging channel, an OpenFlow controller can send messages to Ethernet switches under its control to add or remove flow entries in the flow table. An OpenFlow flow entry can match a packet to an action based on layer 2, 3, and/or 4 fields in addition to the switch port number on which the packet arrives. The action can range from dropping the packet, outputting the packet on specified port(s), and modifying layer 2, 3, and/or 4 header fields.

When a packet comes to an OpenFlow switch, if its flow table has a flow entry matching the packet, the associated action is applied. Otherwise, the packet is sent over the secure channel to the controller, who then decides an action for the packet. The controller can also install a new flow entry on the switch to handle future occurrences of the same type of traffic.

B. SOS Architecture

Figure 1 shows a high level view of the SOS architecture and the six steps for incurring a SOS-based service. For illustration, we assume a file download application such as *wget*. The six steps are:

1. *User application initiation*: Users initiate their applications as usual. In the case of *wget*, an HTTP request is sent towards the intended server.
2. Redirection *to controller*: When the HTTP request arrives at the first OpenFlow-enabled switch, being the first packet of a new flow, it is forwarded to the SOS-enabled OpenFlow controller.
3. *Controller sets up switches and service agents*: Based on the packet type, the controller decides to invoke a specific type of service agent to service this flow. This is based on the switch (alternatively referred to as a datapath in OpenFlow terminology) the packet comes from and its destination IP address. The controller identifies the service agents at both sites and: 1) informs the agents of the incoming flow, the end host to connect to, the number of sockets to use, and a universally unique identifier (UUID) for the connection 2) set up flow table entries for the flow on all OpenFlow-enabled switches along the end-to-end path by sending *flow_mod* messages to these switches. If multiple paths are available, the controller can set up flow entries along all paths simultaneously to utilize them at once if desired.

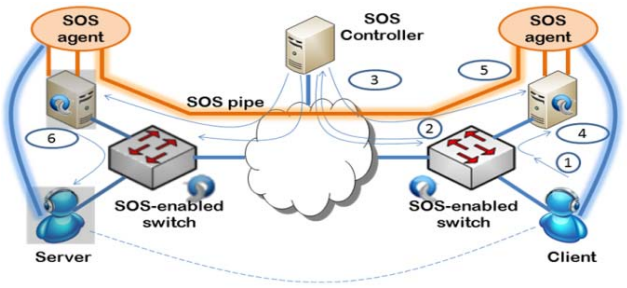

Figure 1. The SOS network architecture.

agents: With Step 3, application traffic begins to be redirected to the service agent on the client site. The switch rewrites the packets' layer 2 and 3 addresses and layer 4 ports in order for the service to be seamless to all parties.

4. *Application traffic sent between service agents*: The service agents incur their own transport protocol of choice to carry the data across the wide area network. In this example, parallel TCP sockets are used. We have also implemented this service using UDT [6], while other transport methods like RDMA [7] would be applicable as well.
5. *Seamless delivery to destination host*: The destination site agent delivers the received payload to the intended end host using the application's original transport protocol.

The entire process is transparent to both application end hosts due to the packet header rewrite performed by the OpenFlow switches at both ends of the connection. Careful observers would have noticed this being analogous to a man-in-the-middle setup, only that the middle man is now a valid agent run by the network operator. Note that OpenFlow's main contribution to SOS is seamless traffic redirect. The example above assumes application packets redirected at the OpenFlow-enabled access switch; nevertheless, even at sites without physical OpenFlow switches, the network provider can manually add static routes to redirect traffic to a server running a software OpenFlow switch (such as the Open Vswitch, as elaborated in the following section) to implement SOS.

C. SOS Service Discovery

The presented work is supported in part by a National Science Foundation grant CNS-0944089For a site (a campus or enterprise network) to support SOS, the site would configure some or all of its OpenFlow-enabled switches to listen to the SOS controller. This controller is shared by multiple sites. In practice and in GENI as of today, this is carried out via network virtualization. Each site will allocate a virtual slice of its switches using network virtualization software such as FlowVisor [8] and delegate the slice to the SOS controller. Thus, users wishing to invoke SOS services can be incrementally moved to the SOS-enabled slice.

In addition to the switch slicing configuration, the SOS

The presented work is supported in part by a National Science Foundation grant CNS-094408.

controller needs to explicitly know the service agents running on each site. Network operators run SOS agents on one or multiple hosts on site, and each agent advertises itself by sending special *discovery packets* to the network periodically. Note that each SOS agent can actually be serving multiple SOS controllers (potentially offering different types of services and/or different service providers). Since each network slice is controlled by one controller, the network operator may create multiple slices that are under different SOS controllers' control. The same service agents can be shared by all the SOS slices, though, as long as the agent discovery packets are included in the slices' flowspace definition (see FlowVisor [8]).

As discovery packets arrive at an OpenFlow switch, the packets are forwarded to all listening SOS controllers. Each controller maintains a table of all agents' type of service, IP address, port number, and current load. The information will be needed for the SOS controller to choose the best pair of agents for each incoming new application flow.

D. SOS Controller

Upon detecting a new application flow, the controller selects the agents, path(s), and packet handling actions for the service. Below describes the controller's key functions:

1. *Agent Selection*: When a packet is sent to the controller due to a missing flow entry (indicating the start of a new flow) the controller checks to see if the packet's header matches a supported type of service. If so, the controller uses the datapath id of the switch the packet comes from to determine if there is a known SOS agent at that site and one near the destination site (based on the packet's destination IP address). If both sites have SOS agents, the controller installs flows for the SOS service.
2. *Path Selection*: The paths used for SOS services are discovered and chosen by the controller. For all controlled OpenFlow switches, the controller sends LLDP packets out of each port containing the switch's datapath id and the switch port number. When these packets are received by any connected OpenFlow switches, they are forwarded to the controller (via *packet_in* messages) for the controller to record the network topology. Thus, for each new flow, the controller selects one or multiple paths per its policies and installs flow entries to the switches on the path(s).
3. *Flow table entries generation*: To forward traffic between end hosts and agents, the flow entries match layer 2, 3, and 4 headers (two entries per client TCP flow). Between agents, due to the large number of parallel TCP flows, special care is needed to avoid an exploding number of flow entries. Specifically, the controller uses an alternative MAC address generated for each path to rewrite the packets, so that the core switches only need two instead of hundreds of flow entries per path.
4. *Multipath support*: When multiple end-to-end paths exist between two end hosts, they can all be utilized to achieve larger throughput. Different policies can be used by the controller for choosing the paths and allocating sockets to respective paths. To achieve optimal performance, appropriate buffering for each socket is needed at the agents as well (see Section 3.3).

E. SOS Agent for Large TCP Fata Transport

SOS agents implement application-specific services. They can be rolled out incrementally as they become available. In this paper, an agent is implemented for enhancing large TCP data transport performance. Below describes the agent implementation, addressing the following:

1. *User TCP seamless termination and restore*: Each end agent binds on a series of ports. One port is for client to connect to and the other ports are used for parallel sockets for agent-to-agent data transfer.
2. *Parallel TCP sockets*: The agents communicate between each other with a series of TCP sockets. When the agent transmits the forwarded TCP data a sequence number is appended to the data to allow for reconstruction at the end agent. Due to limited space details on socket polling, buffering, and handling multiple client connections are not included here.

III. GENI TESTBED IMPLEMENTATION

A. Experiment Topology and Setup

To validate the correct operation and performance of SOS, the NSF GENI testbed provided a most suitable resource with a nation-wide OpenFlow network consisting of multiple campus networks and multiple core network paths provided by National Lambda Rail (NLR) and Internet2. Figure 2 illustrates the experiment topology, involving campus networks at Clemson, SC, Stanford, CA, and Cambridge, MA and transit switches at Seattle, WA, Denver, CO, and Chicago, IL.

For the experiment, a HTTP request was sent from Clemson to Cambridge. The SOS controller identifies two paths, one short (59 ms ping RTT) and one long (164 ms ping RTT). Each path has a 1Gpbs connection from the site to the core and a 10Gbps connection within the core network. The experiment downloads a 1Gb file from Cambridge to Clemson repeatedly over five different configurations: the number of parallel TCP sockets were varied to study their impacts.

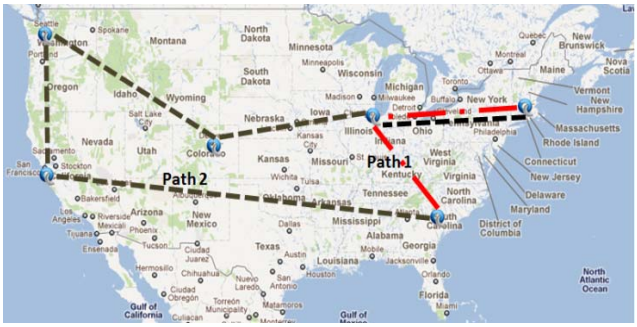

Figure 2. SOS experiment topology on GENI.

Identify applicable sponsor/s here. *(sponsors)*

B. *Performance Measurement*

Table 1 summarizes the measured throughputs. A number of interesting observations can be made:

1. *Long path vs. short path*: With TCP alone, as expected, the long path achieves the lowest throughput. With SOS, both paths achieved similar throughput, much higher than TCP alone.
2. *Throughput efficiency*: *iperf* UDP throughput was measured as a baseline for comparison. Over the 1 Gbps paths, *iperf* UDP achieved 663Mbps on path 1 and 657Mbps on path 2. As seen, SOS achieved nearly 99% of the UDP throughputs. Figure 3 also shows SOS throughput vs. the number of parallel TCP sockets used; it is known that the number of TCP sockets used has an impact on achievable throughput in parallel TCP [9].
3. *Multipath*: The results showed that SOS achieved the highest throughput using both paths simultaneously. It is interesting to note that the achievable multipath throughput depended sensitively on the blocking caused by out of order arrivals from the parallel TCP sockets, and the issue was successfully resolved by the agent with userspace buffering. The multipath results in Table 1 and Figure 3 did not exceed Path 1's throughput much; as the two paths were not fully distinct and shared the same 1 Gbps bottleneck links at both ends.

Table 1 also shows the different results obtained from PlanetLab and Protogeni clients, respectively, on the GENI network. PlanetLab nodes are based on Vservers virtualization architecture while Protogeni nodes are native Ubuntu Linux hosts. For some yet unidentified reason, downloads from the PlanetLab end host were not able to achieve the same end to end throughput as a Protogeni end host. Though, encouragingly, with SOS such differences were no longer an issue and both achieved beyond 600Mbps speeds.

C. *Lessons Learned*

One lesson learned during the development process was that our OpenFlow-enabled hardware switches had a number of limitations that greatly affected performance. These limitations were: 1) the switches were unable to modify layer 2, 3 and 4 header fields of a packet at line rate and 2) installing a large number of flow entries causes the traffic to be processed at the switch's slow path (software processing) and take seconds to be moved to hardware forwarding. The rewrite issue was solved by adopting OpenVswitch (OVS) at each agent machine to perform rewrite before sending packets out to the hardware switches. The large number of flow entries installed was solved by using OVS to rewrite the source MAC address which gave the core switches a field to differentiate on to make use of multiple paths. Doing this greatly reduced the number of *flow_mods* required.

Lastly, the importance of reading/writing strategy and use of socket polling proved to be extremely important. An early implementation of the agent used the SCTP transport protocol which is message based unlike TCP (stream based). This allowed the agent to read and write data in a round robin fashion allowing the order of data to be maintained correct. This approach was easy to implement though lead to performance loss since it required the agent to wait for the next file descriptor to be available to write/read before moving on. Later when TCP was adopted in place of SCTP, the reading and writing strategy was changed to using polling. The agent uses the *epoll()* function to be informed when any file descriptor becomes available for read/write. This avoided any busy socket from blocking the entire flow.

TABLE I. ACHIEVED THROUGHPUT WITH DIFFERENT SCHEMES

Topology	TCP PL (Mbps)	TCP PG (Mbps)	iperf UDP (Mbps)	SOS PL (Mbps)	SOS PG (Mbps)
Path 1 (short)	8	200	663	620	622
Path 2 (long)	2	70	657	615	615
Multi-path				640	639

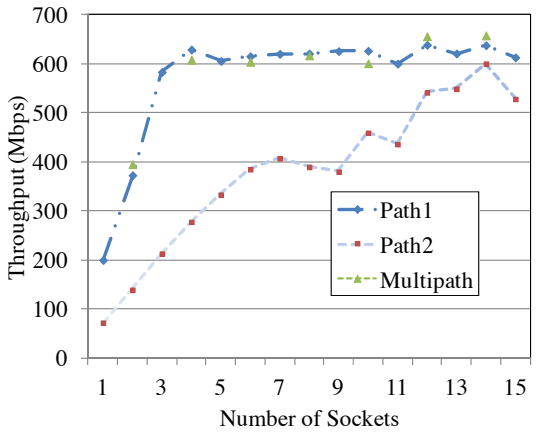

Figure 3. SOS throughput vs. number of sockets.

IV. IMPLICATIONS ON FUTURE INTERNET SERVICE ARCHITECTURE

A. *Mobile Computing*

To date, supporting seamless Internet connection for mobile devices translates to significant handover overheads, complex mobile IP agent configurations on base stations, and inferior end-to-end performance. With SOS, agents can be developed and deployed in Internet to support and enhance seamless mobile connection performance leveraging knowledge of the mobile environment and cloud based resources. With SDN and SOS, mobile end users also have the opportunity to convey their mobility preference, projection, and history to proactively provision network forwarding services to achieve zero latency across link and network handovers.

B. Content/Media Delivery

Content/media delivery has already been an important mode of Internet usage for years. Plenty of solutions, such as cache proxies, content distribution networks, and quality of service provisioning techniques have been studied and deployed. Deployment of such services in the network, however, is predominantly a network provider decision. There is not an existing platform for end users to customize their service levels on demand. With the SOS architecture, novel network services are envisioned to be openly deployed as new agents through a standardized procedure. This procedure allows new applications to be released in conjunction with their customized SOS agents that allow user configuration of specific performance requirements. In turn, the agents can optimize the network forwarding strategies accordingly over the SOS paradigm.

C. Related Work

The specific problem of optimizing wide area data transport has been a persistent endeavor by a large community. For example, the Globus GridFTP server and client tools have been leveraged by many for moving large data across the Internet as well as dedicated fiber networks. The Globus approach has a very similar architecture as SOS in that it decouples the user-to-gateway and gateway-to-gateway data transport and handles them separately. Like SOS, Globus as an architecture can also accommodate different transport protocols in the core such as UDT or RDMA. Unlike SOS, Globus users access the Globus network explicitly using special client software.

V. Conclusion and Future Work

In this paper, we have presented SOS as a paradigm for seamless delivery of network services in OpenFlow-enabled networks. The concept was demonstrated with a wide area TCP data transport optimization service, which was implemented for experimentation on the NSF GENI network. The paper serves three purposes:

- Demonstration of prototyping and experimentation on the GENI OpenFlow network;
- Demonstration of a paradigm for university campuses to leverage OpenFlow to serve the wider science research and education purposes;
- Demonstration of a paradigm that is extensible for fast and seamless deployment of novel services.

The SOS paradigm shares similar concepts as various other legacy and emerging data transport services, while we believe the added autonomy and programmability provided by a SDN network would allow it to serve as a unified platform for a wider range of services and applications.

Acknowledgment

The authors would like to express thanks to the technical support provided by the Clemson Computing and Information Technology (CCIT), the GENI campuses and backbone networks, and the GENI Project Office at BBN Technologies.

References

[1] Stanford Clean Slate Program, http://www.openflow.org/, accessed in Nov. 2011.

[2] Open Networking Foundation, https://www.opennetworking.org/, accessed in Nov. 2011.

[3] GENI: Exploring Networks of the Future, http://www.geni.net/, accessed in Nov. 2011.

[4] Globus GridFTP, http://www.globus.org/toolkit/data/gridftp/, accessed in Nov. 2011.

[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, 38(2): 69-74, April 2008.

[6] UDT: UDP based Data Transfer Protocol, http://udt.sourceforge.net/, accessed in Nov. 2011.

[7] E. Kissel, and M. Swany, "Session layer burst switching for high performance data movement," in Proceedings of PFLDNet, Lancaster, PA, pp. 1-6, 2010.

[8] FlowVisor, http://flowvisor.org. Last accessed in Nov. 2011.

[9] T. J. Hacker, B. D. Noble, and B. D. Athey, "Improving throughput and maintaining fairness using parallel TCP," in Proceedings of INFOCOM, pp. 1-10, 2004.

[10] Open Networking Summit, http://opennetsummit.org/, accessed in Nov. 2011.