Differentiated Network Services for Data-intensive Science using Application-aware SDN

Deepak Nadig Anantha, Byrav Ramamurthy, Brian Bockelman and David Swanson
Dept. of Computer Science & Engineering
University of Nebraska-Lincoln, Lincoln, NE 68588, USA
Email: {deepaknadig, byrav, bbockelm, dswanson}@cse.unl.edu

***Abstract*—Data-intensive science projects rely on scalable, high-performance, fault-tolerant protocols for transferring large-volume data over a high-bandwidth, high-delay wide area network (WAN). The commonly used protocol for WAN data distribution is the GridFTP protocol. GridFTP uses encrypted sessions for data transfers and does not exchange any information with the network-layer resulting in reduced flexibility for network management at the site-level. We propose an *application-aware* software-defined networking (SDN) approach for providing differentiated network services for high-energy physics projects such as Compact Muon Solenoid (CMS) and Laser Interferometer Gravitational-Wave Observatory (LIGO). We demonstrate a *policy-driven* approach for differentiating network traffic by exploiting application- and network-layer collaboration to achieve accurate accounting of resources used by each project. We implement two strategies, a 7-3 queuing system, and a 10-3 queuing system, and show that the 10-3 strategy provides an additional capacity improvement of 11.74% over the 7-3 strategy.**

***Index Terms*—Software Defined Networks; Application-awareness; Data-intensive Science, GridFTP**

I. Introduction

High-throughput distributed computing projects such as Compact Muon Solenoid (CMS) [1] and Laser Interferometer Gravitational-Wave Observatory (LIGO) [2] often require high-rate data transfer capabilities and consume significant storage/networking resources. Backbones for Research and Education (R&E) networks such as Internet2 and Energy Sciences Network provide the necessary infrastructure and generalized frameworks for network resource allocation for these scientific projects. Advances in SDN and OpenFlow [3] allows for fine-grained network control policies to be managed by network applications. Although it is possible to improve the *overall* performance of scientific data transfers (i.e. end-to-end) through dynamic resource allocation (e.g. OSCARS [4]) or through software defined networking, problems exist with managing resources and differentiating network services at the experiment/site level.

The GridFTP protocol [5], [6] has become a widely used network protocol for data movement in cluster/grid environments. By overcoming the well-known limitations of transmission control protocol (TCP) for high-latency, high-bandwidth WANs found in R&E networks at the cost of fairness, GridFTP maximizes throughput for bulk data movement. Both CMS and LIGO projects at Supercomputing Center (SCC) heavily rely on high-throughput computing, with both projects utilizing GridFTP for WAN data movement. GridFTP transfers maximize throughput by using multiple parallel TCP sessions, often for the same source and destination endpoints.

Traditional network techniques for traffic management and prioritization are becoming increasingly limited since GridFTP breaks TCP fairness. For instance, a campus network operator may want to apply different priorities on file transfers initiated by users belonging to different project memberships. The ability to reliably differentiate between different experiments, project memberships, and users within a project, is essential to providing a convenient mechanism for the application of appropriate actions and policies. To exacerbate this problem, the GridFTP control channel uses encrypted sessions during the connection establishment phase, which means that no amount of 'sniffing' the control channel flows allows the network to classify traffic on its own. Without reliable traffic classification information, prioritizing traffic based on either project memberships, or based on users within a project is not possible. We propose an application-aware SDN solution to provide differentiated network services at the site-level. Our work focuses on facilitating a policy-driven approach to quality of service (QoS), network resource management and accounting for scientific data projects at the site-level.

The specific contributions of this paper are:

- *Application-aware service differentiation*: Our approach demonstrates how application- and network-layer collaboration can be exploited to utilize application metadata for influencing network-layer control decisions for traffic prioritization. This results in fine-grained control over data-flows of different projects where priorities are applied to individual experiments.
- *Policy-driven resource management*: We implement a policy-driven approach to decision making at the experiment-level. This allows not only the definition of per-flow policies but also policies that can be used to regulate resource usage of opportunistic data transfers from projects such as LIGO.

II. Background and Related Work

Although work related to application- and network-layer collaboration has been limited, there are numerous works proposing the use of network overlays, middleboxes, and control plane protocols for session adaptation and end-to-end communication. We provide an overview and survey of works that share some common aspects with our application-aware SDN approach.

Phoebus [7], [8], was one of the earliest efforts to provide an infrastructure that uses a session layer to the improve end-to-end performance of high-bandwidth, high-latency networks. Phoebus used strategically placed "Phoebus Gateways" (PGs) to create the ability to dynamically allocate network resources through the use of segment-specific transport protocol adaptations. Unlike Phoebus, which provides end-to-end session adaptation for improving GridFTP data transfer performance, our work focuses on a single, site-specific network flow management and prioritization.

The authors in [9] propose a network overlay architecture for enabling high-throughput, co-ordinated data transfers over a WAN by leveraging Phoebus and OSCARS [4]. By leveraging application overlays that are embedded in the underlying network to create on-demand embedded overlays, applications can use the available network resources more efficiently. However, the solution does not provide fine-grained controls at the end-points to prioritize specific flows but focuses solely on large-scale end-to-end data transfer throughput improvements only. Other approaches such as [10] and [11] rely on data locality in cloud storage services. The authors in [10] utilize common data access pattern observations of file transfers between compute nodes along with data locality and context information to choose an adequate transfer protocol. The work in [12], [13] focuses on exploiting network parallelism and differentiated scheduling of WAN data transfers to achieve performance from (virtually) loss-free dedicated resource provisioning systems such as ESnet and Internet2. The authors in [12] propose the use of multiple paths on the WAN for the same application transfer session. They argue that the use of multiple paths, for example, by either exploiting additional bandwidth resources from existing best-effort WAN links or by utilizing multiple 10G NICs to create multiple paths over a 100G capable WAN link, higher performance can be achieved.

The SDN-managed Network Architecture for GridFTP transfers (SNAG) proposed in [14], enables GridFTP transfers over an SDN. An Application Program Interface (API) exposes application-layer information from the trusted GridFTP process by using the Globus XIO callout module. By extending the GridFTP server to interact with an SDN controller securely over a RESTful API, automated application-layer metadata forwarding is possible. This metadata is exploited by the network layer, and is used to accurately classify all active GridFTP flows. Further, the metadata is also used by the SNAG SDN application to monitor and log all GridFTP flows, including user and project membership information with the monitoring system.

The Globus eXtensible I/O (XIO) [15] provides an extensible I/O library with the Globus Toolkit [16]. The XIO library provides a pluggable architecture and SNAG uses a Globus XIO Callout[1] to interface to the Hadoop Distributed File System (HDFS) [17] plugin for the GridFTP servers. HDFS was designed primarily for use with distributed storage/processing infrastructures. Its fault-tolerant design and built-in scalability features are well suited for serving as the GridFTP servers' storage layer at SCC. The XIO driver interaction with the GridFTP servers is as shown in Figure 1.

[1]https://github.com/bbockelm/xio_callout

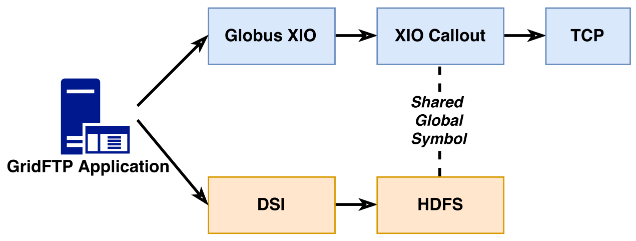

Fig. 1: XIO and GridFTP interaction.

III. Problem Setting

Network services differentiation is essential at the site-level to enhance transfer performance at the network edge. With both CMS and LIGO projects using the same network infrastructure, the inability to classify and differentiate services results in low-priority users of one project blocking/preempting the high priority users of the other. Further, the lack of differentiation prevents the site-operator from optimizing data transfers at the experiment-level. Instead of relying on resource reservation for each transfer, our solution focuses on site-level improvements and intelligent decision making at the network edge to enhance transfer performance. Our goal is to provide accurate site- and experiment-level classification of flows to optimize data transfers at the experiment level. By focusing on site-level rather than end-to-end improvements, our solution provides maximum flexibility to the site operator in terms of traffic prioritization and resource usage accounting for individual experiments. To further motivate the problem, we provide below, two important drivers on the need for application- and network-layer collaboration for SDN-managed GridFTP transfers.

A. Application-aware traffic prioritization

The need for application-awareness arises from the necessity for managing both encrypted and unencrypted traffic flows. GridFTP transfers are initiated over an encrypted control channel and uses multiple randomly selected TCP connections for data movement. With the inability to differentiate between low-priority and high-priority users, traffic prioritization (through classification at the network-layer alone) is not possible at the site-level. An example use case is when transfers from two different projects (i.e. CMS and LIGO) use the same source/destination address for data movement. In such cases, application-awareness is imperative for differentiating traffic between the same end-points. Thus application metadata can then be exploited to prioritize data transfers at the experiment-level.

B. Policy-driven service differentiation

Resource usage and accounting have been historically problematic with projects such as CMS and LIGO. With multiple stakeholders involved in operation and use of the network infrastructure, operators find it difficult to monitor resources used by different projects. Therefore, our work focuses on providing a policy-driven mechanism to apply actions and policies appropriately to different experiments. Further, these policies can serve to regulate/meter resource usage of opportunistic data transfers from projects such as LIGO.

IV. Solution Approach

Application-awareness combined with SDN forms the basis our solution for policy-driven management of data-intensive science transfers. First, we begin with reliable and accurate classification of data transfers from different experiments/projects. This objective is accomplished through the exchange of application metadata between the GridFTP application servers and the SDN controller. The SDN controller creates and manages a repository of all ongoing transfers obtained from the classification information. This information is employed to make network-layer and data-plane decisions. A policy framework is used by the network operator to define and apply the appropriate policies to these data transfers. This framework consists of a policy-engine and the associated policy specification language implemented as an SDN application. The policy controls are communicated to the SDN controller, which translates the policy to appropriate data plane decisions. Based on the defined policy, a set of actions (that correspond to the policy strategy) are applied to the data flows.

A. Traffic Classification

Accurate and reliable traffic classification through application-awareness is essential to differentiating data transfers from encrypted sessions. Our system provides application-awareness to the underlying SDN control by communicating GridFTP application metadata over a secure RESTful API. By enabling application-aware traffic classification for the GridFTP protocol, we now have accurate experiment/project-level information for decision making.

B. Policy Framework

The policy framework consists of the following: i) a policy engine that manages user-defined policies, and is implemented as an SDN application communicating with the SDN controller over the north-bound APIs; ii) a policy specification language that uses JavaScript Object Notation (JSON) to specify policies and exposes a RESTful API for network operators to manage policies, and iii) a set of defined actions that is processed by the policy engine's event processor to affect flow treatment actions to appropriate data transfers. In the following, we briefly discuss these components:

1) Policy Engine: The policy engine comprises of a *policy manager*, an *event handler*, a *parser*, and a *policy repository*. The *policy manager* is responsible for the lifecycle management of policies created using the RESTful APIs. After the *parser* validates the policies, the new/updated policy information is stored in the *repository*. The policy engine's *event handler* converts the policy specifications to corresponding OpenFlow rules that are communicated to the switches using the south-bound APIs.

Listing 1: Example JSON policy specification.

```
{
  "name": "string",
  "type": "user|project|experiment",
  "strategy": "strategy_id"
  "action": "start|modify|stop",
  "specs": [value1, value2],
  "time": [start, end]
}
```

2) Specification Language: The policy specification uses JSON resources encapsulated in a REST POST to interact with the policy engine in the framework. The specification language allows us to define/modify the default flow treatment of an experiment/project. An example policy specification is shown in Listing 1. As shown, each policy specifies the type of action to be applied to the data transfers of an user, an experiment, or a project. Each policy specification also provides a mechanism to specify start- and end-times for policy enforcement. The valid actions and the associated service differentiation strategies that can be applied to a data transfer are described next.

3) Actions and Strategies: Actions along with pre-defined strategies are responsible for implementing the desired flow treatment behaviors in the data-plane. A site operator can define different QOS and traffic prioritization/management strategies, and specify the appropriate strategy to be applied using the policy specification. Each strategy is composed of two parts: *definitions* (e.g. queues to use, associated priorities, max/min rate settings etc.) and a *trigger* that initiates the policy enforcement.

4) RESTful APIs: The services implemented by our application-aware SDN solution are exposed using a RESTful API over the SDN controller's north-bound interface. The exposed RESTful APIs are listed in Table I.

TABLE I: Policy Framework REST APIs.

Method	URI	Description
`GET`	/v1/policies/{api_version}	Show details of specific API version
`GET`	/v1/policies/	List all current policies
`POST`	/v1/policies/	Establish a new policy
`GET`	/v1/policies/{policy_id}	List policy by {policy_id}
`DELETE`	/v1/policies/{policy_id}	Delete policy by {policy_id}

These APIs can be used to manage policies over a HTTP(S) protocol. Policies can be created, updated or deleted using the above REST calls. A JSON information model is used to describe the resources associated with these APIs.

C. Solution Architecture

The solution architecture is as shown in Figure 2. The GridFTP server pool oversees data transfers from both CMS and LIGO projects. The GridFTP server uses the XIO callout module to send application metadata to the SNAG application on the SDN controller. SNAG is responsible for providing information regarding new and ongoing data transfers to the site operator. The site operator then uses the transfer statistics to make the policy decisions that are enforced using the policy framework built as an SDN application. The policy framework provides a RESTful API for communication, and policy enforcement can either be performed manually by the site operator or can be automated by the SDN controller.

D. Algorithm Design

Our algorithm focuses on providing a policy-based solution to network services differentiation. We rely on two important principles namely: *application-awareness* and *policy strategy*.

While application-awareness gives us valuable insights into the current state of data transfers from various users/projects/experiments, policy strategy, on the other hand, allows us to apply the right forwarding behaviors to the desired flows. The $AA\_DNS(p, traffic\_stats)$ algorithm is as shown in Algorithm 1. The algorithm initializes policy IDs for each policy and manages them in a *PolicyMap*. The policy engine parses each policy and extracts the specified strategy (*p.strategy*). The policy strategy contains information about QoS requirements, queue specifications and priority definitions for the *target* traffic. This information is used to create the appropriate flow rules by the SDN controller and applied to the corresponding switches in the data plane to change the forwarding behaviors.

E. Implementation

We use the hierarchical token bucket[2] (HTB) for egress traffic shaping. HTB is a egress queuing discipline implementation for Linux kernel packet scheduler user space utilities. We limit our discussion to egress traffic shaping (using queues) and do not use ingress rate limiting algorithms (that employ policing) to provide differentiated services. Ingress rate limiting/policing does not use queues but drops packets beyond a certain rate instead; this is problematic as some protocols react severely to dropped packets. We present two strategies for implementing differentiated network services for GridFTP transfers using egress traffic shaing and queues.

1) 7-3 Queues: This strategy creates two queues in the ratio 7:3. We evaluate transfer performance on a 10Gbps link. Therefore, we create two queues $q1$ and $q2$, with ingress traffic shaped to 7Gbps and 3Gbps respectively. The two queues have equal priority and have their priorities set to a value higher than the best-effort link. The strategy places CMS traffic on $q1$ and LIGO traffic on $q2$.

2) 10-3 Queues: Here, we create two queues, $q1$ and $q2$ with rates of 10Gbps and 3Gbps respectively. The priority of $q1$ is greater than that of $q2$, which is an important difference from the previous strategy. In this approach, we utilize a single high priority 10Gbps queue for all data transfers except LIGO.

[2]http://luxik.cdi.cz/ devik/qos/htb/

Algorithm 1 AA-DNS(*p, target*)

```
Require: Policy File ($p$), target
Output: Provisioned Network Flows
    Initialization :
 1: for all $p \in P$ do
 2:     Generate $p\_id(p)$
 3:     if ($p \notin PolicyMap$) then
 4:         Add $(p, p\_id(p))$ to $PolicyMap$
 5:     else
 6:         Replace $PolicyMap(p)$
 7:     end if
 8: end for
    Policy Enforcement :
 9: for all $p \in PolicyMap$ do
10:     if ($p.expired \neq$ TRUE) then
11:         CONFIGURE($queues \in p.strategy$)
12:         $flowrule =$ COMPOSE($p, target$)
13:         $Ruleset \leftarrow flowrule$
14:         for all Switch $s \in S$ do
15:             APPLY $Ruleset(s)$
16:         end for
17:     end if
18: end for
```

LIGO transfers if initiated, are automatically switched to use $q2$. In the following section, we discuss the performance of both strategies.

V. Experiments

Our application-aware SDN solution is used to manage and monitor data transfers over the GridFTP protocol, providing differentiated network services. The solution is tested on the Kingfisher release (1.10.0) of ONOS [18] SDN controller. The policy framework and the QoS/prioritization systems are implemented as an ONOS application. The ONOS application interacts with the GridFTP protocol and obtains application metadata which is used to classify flows based on users, projects or experiments. The application metadata is not only used to obtain real-time data transfer information but also allows for the creation of a GridFTP transfer statistics repository. This repository provides useful information that aids the site operator in designing/choosing the appropriate policy for service differentiation.

A. Network Testbed

We test our solution on a network testbed that is setup to integrate with a U.S. CMS Tier-2 site. The site handles high-volume data flows and performs both high-priority CMS transfers and low-priority opportunistic transfers to Fermilab. While the GridFTP protocol is employed for bulk batch transfer, the XROOTD protocol is used for interactive jobs. The site holds approximately 2PB of data. Our network testbed effectively combines several state-of-art techniques including:

- An SDN controller for intelligent control plane forwarding decisions, associated apps for enabling application-awareness, and a policy framework implementation.

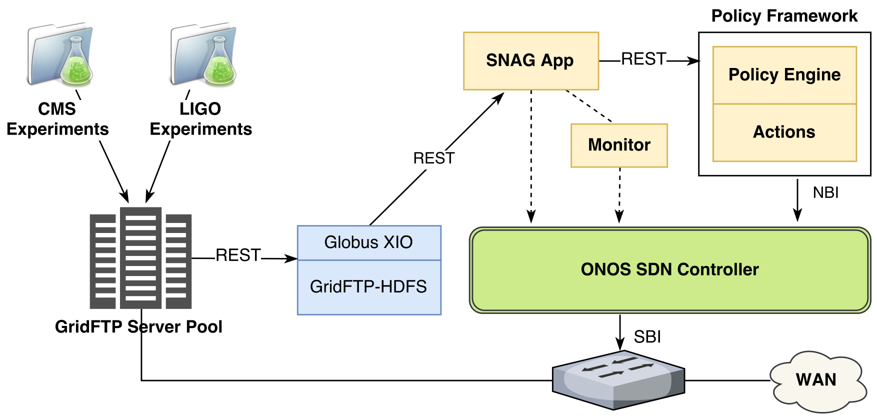

Fig. 2: Solution Architecture.

- A test GridFTP server removed from the production pool and the GridFTP-HDFS plugin for the GridFTP storage layers. A Globus XIO callout module and SNAG interface to provide CMS file transfer information to the SDN controller over a RESTful API.
- 100Gbps connectivity between SCC and the WAN.

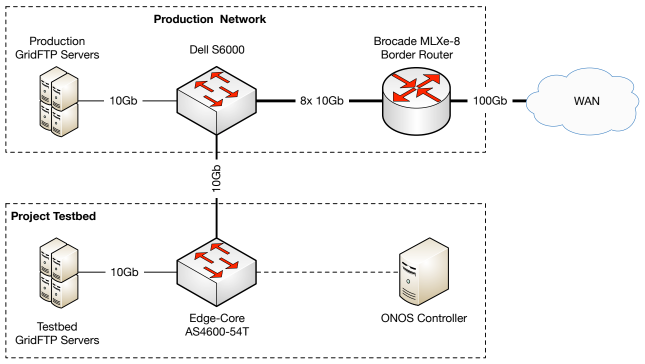

Fig. 3: Network testbed topology and connectivity information.

The testbed network topology is as shown in Figure 3. A Brocade MLXe router at the data center border connects both the production and testbed environments to a 100Gbps WAN. The CMS cluster network core utilizes the Dell S6000 40GbE switch, whereas the testbed network was attached to an OpenFlow enabled Edge-Core AS4600-54T switch.

B. Results and Discussion

First, we present the raw bandwidth of the physical egress port in Figure 4 with a confidence interval (CI) of 95% from 10 measurements over 30 seconds. As the physical interface is shared between multiple servers, we observe significant variations in the available bandwidth. To overcome this limitation and limit the dependence on time varying bandwidth availability, we create a single 10Gbps queue on an Open vSwitch instance connected to this physical interface, and use it for performance testing. Next, in Figure 5, we show the bottleneck bandwidths for creating the 10Gbps queue when both egress traffic shaping and ingress rate limiting are used. It can be seen that the performance of the ingress rate limiting varies with the burst size. Thus, we focus only on egress traffic shaping for our evaluations.

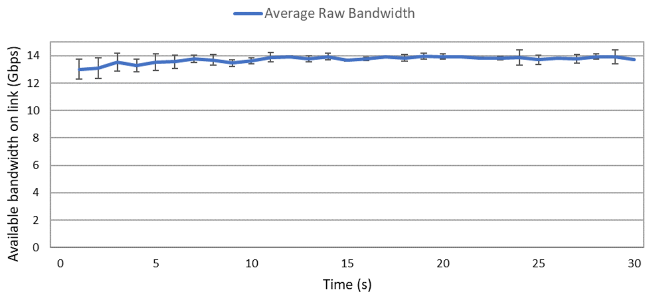

Fig. 4: Average raw bandwidth measured as the achievable throughput of an iperf flow on the link.

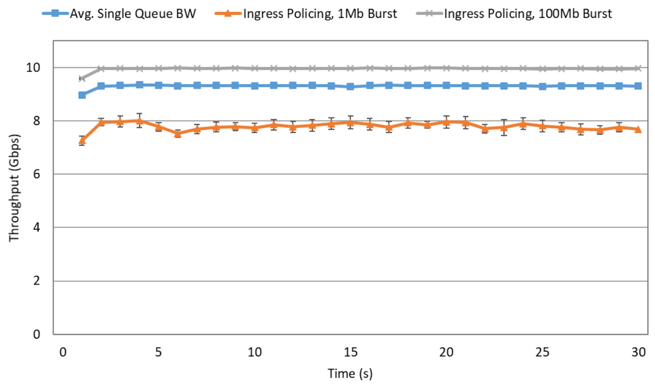

Fig. 5: 10Gbps Bottleneck Bandwidths (Egress Shaping and Ingress Policing).

Figure 6 shows both the individual and aggregate bottleneck bandwidths for 7-3 queues. The performance of 7-3 queues for 10 random transfers is shown in Figure 7. It can be seen that both queues exhibit linear correlation with a decrease in $q2$ traffic resulting in a corresponding increase in $q1$ traffic as shown by the trend lines. Since both queues are set to use the same priorities, they behave as two rate-limited best-effort queues. Figure 8 shows the performance of strategy 2, where LIGO traffic is switched to a lower priority queue. It can be seen that the during LIGO transfers, the higher priority queue $q1$ shapes its traffic to accommodate LIGO flows. Note that $q2$ uses a portion (max 3Gbps) of the larger 10Gbps queue i.e. $q2$ does not represent a separate queue from $q1$. Comparing the aggregate throughputs of both strategies from Figures 6 and 8, we see that strategy 2 provides an additional capacity improvement of 11.74%. This is because CMS flows can achieve higher throughputs as fewer traffic shaping requirements free more bandwidth.

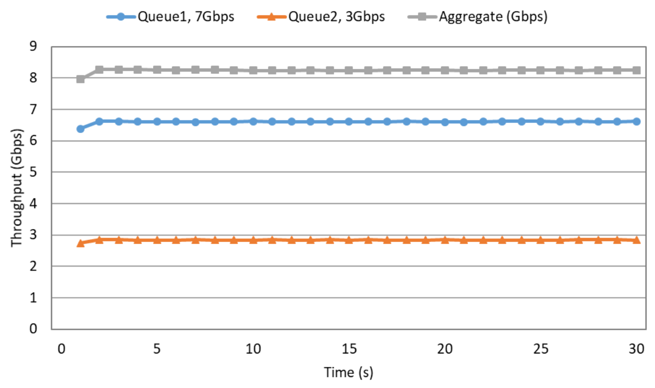

Fig. 6: Two Queues with Traffic Shaping, Ratio: 7-3.

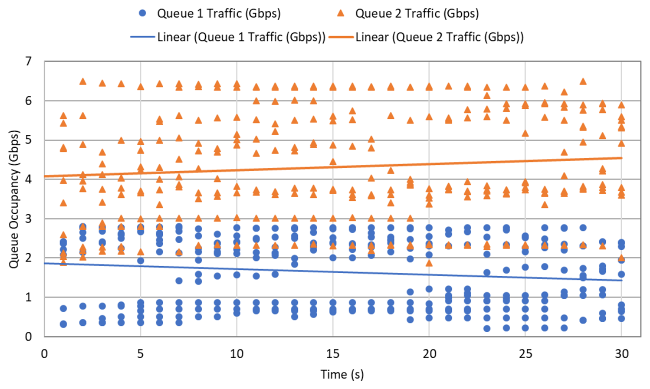

Fig. 7: 7-3 Queue performance, Random Traffic, 10 Samples.

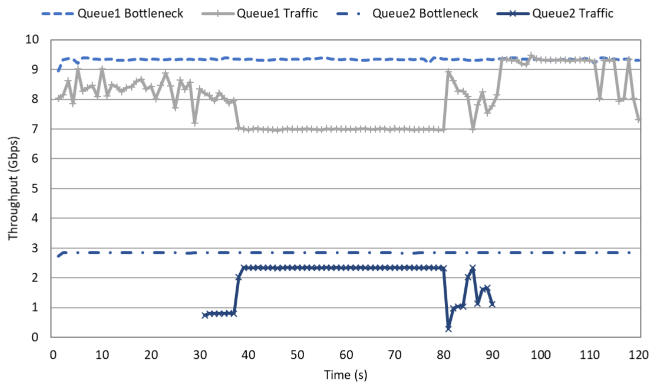

Fig. 8: 10-3 Switched Queue performance with LIGO Traffic.

VI. Conclusions and Future Work

We present an application-aware SDN approach to network services differentiation. Our solution allows the application of per-experiment policies for science data transfers at the site-level which was not previously possible. We present two strategies, 7-3 and 10-3 queues, for applying QoS to CMS and LIGO flows. We show that the 10-3 strategy performs better for handling opportunistic transfers from the LIGO experiment. Our future work will focus on creating adaptive strategies on-the-fly, so that policies can be applied in an automated fashion.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant Number OAC-1541442. This work was completed using the Holland Computing Center of the University of Nebraska, which receives support from the Nebraska Research Initiative. The authors would like to thank Garhan Attebury, Holland Computing Center at UNL for his valuable support.

References

[1] S. Chatrchyan *et al.*, “The CMS experiment at the CERN LHC,” *JINST*, vol. 3, p. S08004, 2008.
[2] B. P. Abbott *et al.*, “LIGO: the Laser Interferometer Gravitational-Wave Observatory,” *Reports on Progress in Physics*, vol. 72, no. 7, p. 076901, 2009.
[3] N. McKeown *et al.*, “OpenFlow: Enabling Innovation in Campus Networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.
[4] C. Guok, E. N. Engineer, and D. Robertson, “ESnet On-demand Secure Circuits and Advance Reservation System (OSCARS),” *Internet2 Joint Techs*, 2006.
[5] W. Allcock *et al.*, “The Globus Striped GridFTP Framework and Server,” in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, Nov 2005, pp. 54–54.
[6] W. Allcock *et al.*, “GridFTP: Protocol Extensions to FTP for the Grid. 2003,” in *Global Grid Forum.*
[7] E. Kissel, M. Swany, and A. Brown, “Improving GridFTP performance using the Phoebus session layer,” in *Conference on High Performance Computing Networking, Storage and Analysis*, Nov 2009, pp. 1–10.
[8] E. Kissel, M. Swany, and A. Brown, “Phoebus: A System for High Throughput Data Movement,” *J. Parallel Distrib. Comput.*, vol. 71, no. 2, pp. 266–279, Feb. 2011.
[9] L. Ramakrishnan *et al.*, “On-demand overlay networks for large scientific data transfers,” in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, May 2010, pp. 359–367.
[10] R. Tudoran *et al.*, “Adaptive file management for scientific workflows on the azure cloud,” in *2013 IEEE Intl. Conference on Big Data*, Oct 2013, pp. 273–281.
[11] R. Tudoran, A. Costan, and G. Antoniu, “Overflow: Multi-site aware big data management for scientific workflows on clouds,” *IEEE Transactions on Cloud Computing*, vol. 4, no. 1, pp. 76–89, Jan 2016.
[12] D. Gunter *et al.*, “Exploiting network parallelism for improving data transfer performance,” in *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, 2012.
[13] R. Kettimuthu *et al.*, “Differentiated scheduling of response-critical and best-effort wide-area data transfers,” in *2016 IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS)*, May 2016, pp. 1113–1122.
[14] D. N. Anantha *et al.*, “SNAG: SDN-managed Network Architecture for GridFTP Transfers,” in *INDIS ’16*, SLC, Utah, November 2016.
[15] W. Allcock *et al.*, “The Globus eXtensible Input/Output System (XIO): A protocol independent IO system for the Grid,” in *19th IEEE Intl. Parallel and Distributed Processing Symposium.*
[16] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems.* Springer Berlin Heidelberg, 2005, pp. 2–13.
[17] K. Shvachko *et al.*, “The Hadoop Distributed File System,” in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST ’10, 2010, pp. 1–10.
[18] P. Berde *et al.*, “ONOS: Towards an Open, Distributed SDN OS,” in *3rd Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN ’14. New York, NY, USA: ACM, 2014, pp. 1–6.