Performance Comparison of SDN Solutions for Switching Dedicated Long-Haul Connections

Nageswara S. V. Rao

Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA
Email: raons@ornl.gov

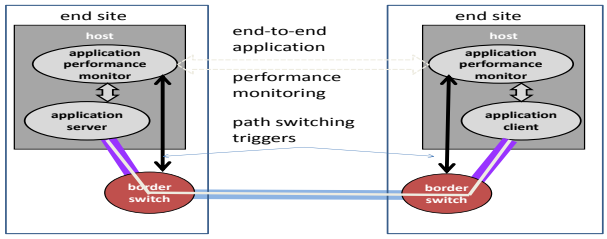
***Abstract*—We consider scenarios with two sites connected over a dedicated, long-haul connection that must quickly fail-over in response to degradations in host-to-host application performance. We present two methods for path fail-over using OpenFlow-enabled switches: (a) a light-weight method that utilizes host scripts to monitor the application performance and dpctl API for switching, and (b) a generic method that uses two OpenDaylight (ODL) controllers and REST interfaces. The restoration dynamics of the application contain significant statistical variations due to the controllers, north interfaces and switches; in addition, the variety of vendor implementations further complicates the choice between different solutions. We present the impulse-response method to estimate the regressions of performance parameters, which enables a rigorous and objective comparison of different solutions. We describe testing results of the two methods, using TCP throughput and connection rtt as main parameters, over a testbed consisting of HP and Cisco switches connected over long-haul connections emulated in hardware by ANUE devices. The combination of analytical and experimental results demonstrates that dpctl method responds seconds faster than ODL method on average, while both methods restore TCP throughput.**

Keywords–Software defined networks; OpenFlow; Opendaylight; controller; long-haul connection; impulse-response; testbed.

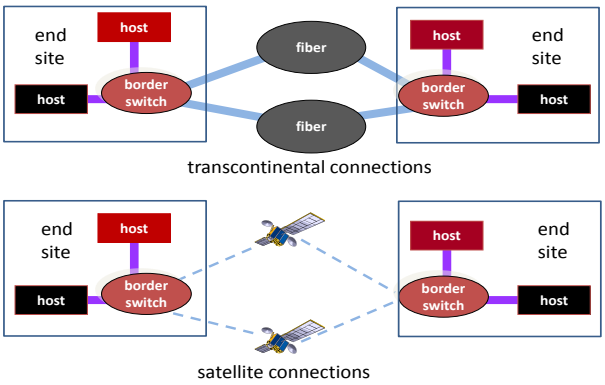
I. INTRODUCTION

We consider scenarios with two sites connected over dedicated long-haul connections such as transcontinental fibers or satellite links, as illustrated in Figure 1. Different client-server application pairs are executed at different times on host systems at the sites, to support data transfers, on-line instrument monitoring, and messaging. The connection quality can degrade due to a variety of factors such as equipment failures, weather conditions, and geographical events, which in turn affect the host-to-host application performance. As a mitigation strategy, a physically diverse and functionally equivalent *standby* path is switched to when the application performance degrades.

The performances of application pairs are monitored on host systems, and the current *primary* path is switched out when needed, for example, by modifying Virtual Local Area Networks (VLAN) and route tables on border switches and routers, respectively. In our use cases, human operators watch the host-level performance monitors, and invoke Command Line Interface (CLI) commands or web-based interfaces of network devices for path switching. Since triggers for path switching are dynamically generated by application pairs, they are not adequately handled by conventional hot-standby layer-2/3 solutions that solely utilize connection parameters. For example, certain losses may be tolerated by messaging applications but not by monitoring and control applications of instruments and sensors. Currently, the design and operation of such application-driven fail-over schemes require a detailed knowledge of host codes, such as Linux scripts, and the specialized interfaces and languages of switches, such as CLI and custom TL1 and CURL scripts (which currently vary significantly among the vendor products). Furthermore, fail-over operations must be coordinated between two physically-separated operations centers located at the end sites.



(a) host-to-host application pairs



(b) two connection modalities

Figure 1. Two sites connected over long-haul connections.

A. SDN Solutions

The rapidly evolving *Software Defined Networks* (SDN) technologies [1], [2] seem particularly well-suited for automating the path switching tasks, when combined with host monitoring codes. In particular, SDN technologies provide two distinct advantages:

(a) trigger modules of new applications can be "dropped in place" since they use generic northbound interfaces, and
(b) switches from different vendors with virtual interfaces can be simply swapped, avoiding the re-work needed for custom interfaces and network operating systems.

 ISBN: 978-1-61208-450-3

The execution of this seemingly direct approach, surprisingly however, requires comparing and selecting from among rather complex configurations. Due to the rapid development of SDN technologies [1], there is a wide array of options for controllers and switches, which in turn leads to a large number of their solution combinations [3]. Indeed, their complexity and variety requires systematic analysis and comparison methods to assess their operational effectiveness and performance [4], such as recovery times. In addition, compared to certain data-center and network provisioning scenarios for which SDN technologies have been successfully applied [5], [6], these long-haul scenarios present additional challenges that require newer solutions: (a) single controller solutions that have been extensively used in several applications [1] are not practical for managing the border switches at end sites due to the large latency, and (b) solutions that require a separate control-plane infrastructure, such as DISCO [7], are cost prohibitive since they require additional control plane connections.

B. Outline of Contributions

In this paper, we present automated software solutions for path fail-over by utilizing two controllers, one at each site, that are coordinated over a single connection through measurements. We first describe a light-weight, custom designed *dpctl method* for OpenFlow border switches that uses host Linux bash scripts. This script is about a hundred lines of code, which makes it easier to analyze for its performance and security aspects. We then present a more generic *ODL method* that utilizes two OpenDaylight (ODL) controllers [8] located at the end sites. The executional path of this approach is more complex compared to the dpctl method since it involves communications using both northbound and southbound ODL interfaces and invoking several computing modules within ODL software stack. Thus, a complete performance and security analysis of this method requires a closer examination of a much larger code base that includes both host scripts and corresponding ODL modules, including its embedded http server [9].

We present implementation and experimental results using a testbed consisting of Linux hosts, HP and Cisco border switches, and ANUE long-haul hardware connection emulation devices. We utilize TCP throughput as a primary performance measure for the client-server applications, which is affected by the connection rtt and jitter possibly caused by path switching, and the available path capacity. Experimental results show that both dpctl and ODL methods restore the host-to-host TCP throughput within seconds by switching to the standby connection after the current connection's RTT is degraded (due to external factors). However, the restoration dynamics of TCP throughput show significant statistical variations, primarily as a result of the interactions between the path switching dynamics of the controllers and switches, and the highly non-linear dynamics of TCP congestion control mechanisms [10]–[12]. As a result, direct comparisons of individual TCP throughput traces corresponding to fail-over events are not very instructive in reflecting the overall performance of the two methods.

To objectively compare the performances [4] of these two rather dissimilar methods, we propose the *impulse-response* method that captures the average performance by utilizing measurements collected in response to a train of path degradation events induced externally. We establish a statistical basis for this method using the finite-sample theory [13] by exploiting the underlying monotonic properties of the performance parameters during the degradation and recovery periods. This analysis enables us to objectively conclude that on average the dpctl method restores the TCP throughput several seconds faster than ODL method for these scenarios.

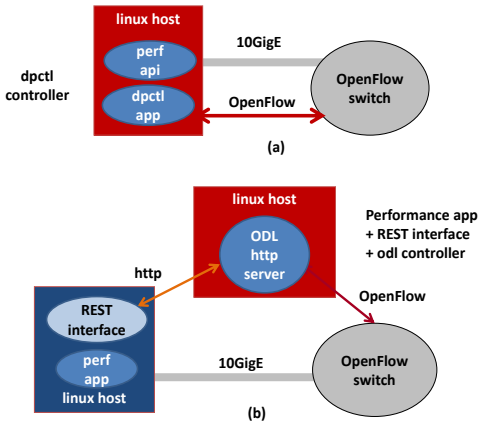


Figure 2. Configurations of dpctl and ODL controllers.

The organization of this paper is as follows. A coordinated controllers approach using dpctl and ODL methods is described in Section II. An experimental testbed is described in Section III-A, and the results of experiments using dpctl and ODL methods using five different configurations are presented in Section III-B. The impulse response method and its statistical basis are presented in Sections IV-A and IV-B, respectively. Conclusions are presented in Section V.

II. Coordinated SDN Controllers

For these long-haul scenarios, single controllers solutions or solutions requiring separate control plane are not effective, although such approaches with stable control-plane connections have been effective in path/flow switching over local area networks using OpenFlow [5], [14] and cross-country networks using customized methods [6], [7], [15]. Since the controller has to be located at a site, when the primary connection degrades, it may not be able to communicate effectively with the remote site. Our approach is to utilize two controllers, one at each site, which are "indirectly" coordinated based on the monitored application-level performance parameters. When path degradation is inferred by a host script, the controller at that site switches to the fail-over path by installing the appropriate flow entries on its border switch. If the primary path degrades, for example, resulting in increased latency or loss rate, its effect is typically detected at both hosts by the monitors, and both border switches fail-over to the standby path at approximately the same time. If the border switch at one site fails-over first, the connection loss will be detected at the other site which in turn triggers the fail-over at the second site. Also, one-way failures lead to path switching at one site first, which will be seen as a connection loss at the other site, thereby leading to path switching at that site as well. Due to recent developments in the SDN technologies, both in open software areas [16] and specific vendor implementations [17], [18], there are many different ways such a solution can be implemented. We only consider OpenFlow solutions which are implemented using open standards and software [5].

 ISBN: 978-1-61208-450-3

A. *dpctl Method*

As a part of OpenFlow implementation, some vendors support dpctl API which enables hosts to communicate with switches to query the status of flows, insert new flows and delete existing flows. It has been a very useful tool primarily for diagnosing flow implementations by using simple host scripts; however, some vendors such as Cisco do not provide dpctl support. We utilize dpctl API in a light-weight host script that constantly monitors the connection rtt and detects when it crosses a threshold and invokes dpctl to implement the fail-over as shown in Figure 2(a). The OpenFlow entries for switching to the standby path are communicated to the switch upon the detection of connection degradation. This script consists of under one hundred lines of code and is flexible in that the current connection monitoring module can be replaced by another one such as TCP throughput monitor using iperf. Compared to methods that use separate OpenFlow controllers, this method compresses both performance monitoring and controller modules into one script, and thereby avoids the northbound interface altogether; for ease of reference, we refer to this host code as the *dpctl controller*.

B. *OpenDaylight Method*

We now utilize two ODL Hydrogen controllers and REST interfaces to implement fail-over functionality using OpenFlow flows as shown in Figure 2(b). ODL is an open source controller [8] that communicates with OpenFlow switches, and is used to query, install and delete flow entries on them using its southbound interface. The applications communicate with ODL controller via the northbound interface to query, install and delete flows. ODL software in our case runs on Linux workstations called the controller workstations, and the application monitoring codes can be executed on the same workstation in the *local* mode or on a different workstation in the *remote* mode.

The same performance monitoring codes of the dpctl method above are used in this case to detect path degradations but are enhanced to invoke python code to communicate new flows for switching paths to ODL controllers via REST interfaces; the content of these flow entries are identical to the previous case. Thus, both the software and executional paths of this method are much more complicated compared to previous case, and also the ODL controllers are required to run constantly on the servers at end sites. Also, this code is much more complex to analyze since it involves not only the REST scripts but also the ODL stack which by itself is a fairly complex software. The executional path is more complex since it involves additional communication over both northbound and southbound interfaces of ODL controllers.

III. Experimental Results

In this section, we first describe the testbed and then describe the experimental results.

A. *Emulation Testbed*

The experimental testbed consists of two site LANs, each consisting of multiple hosts connected via 10GigE NICs to the site's border switch. The border switches are connected to each other via a local fiber connection of a few meters in length, and also via two ANUE devices that emulate long-haul connections in hardware, as shown in Figure 3. Tests use pairs

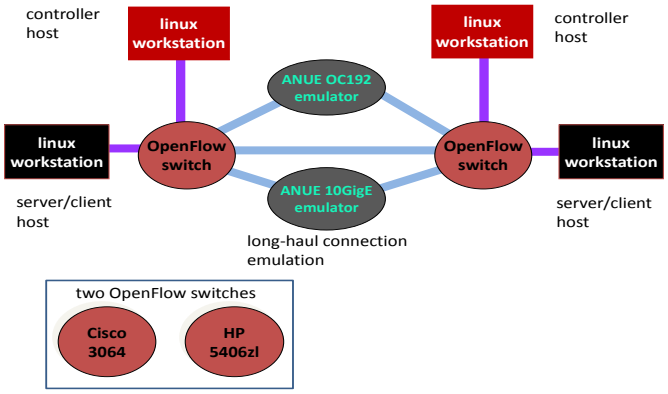


Figure 3. Testbed of two sites connected over local and emulated connections.

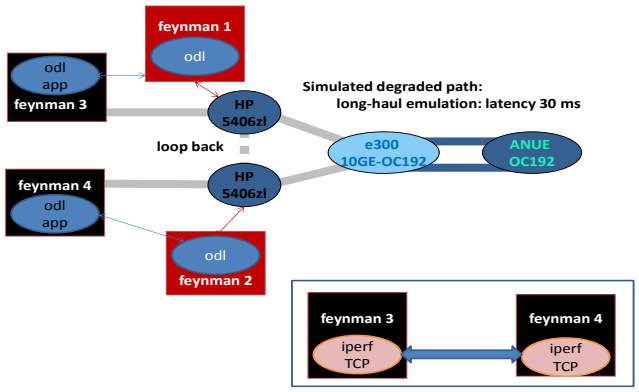
of HP 5064zl and Cisco 3064 devices as border switches, both of which are OpenFlow-enabled but only HP switches support dpctl. The OC192 ANUE devices emulate connections with rtts in the range of [0-800] milliseconds with a peak capacity of 9.6 Gbps. The conversion between 10GigE LAN packets from the border switches and long-haul OC192 ANUE packets is implemented using a Force10 E300 switch, as shown in Figure 4. ANUE devices are utilized primarily to emulate the latencies of long-haul connections, both transcontinental fiber and satellite, to highlight the overall recovery dynamics.

Two classes of Linux hosts are connected to the border switches. The controller hosts (feynman1 and feynman2) are utilized to execute ODL controllers, and client and server hosts (feynman3 and feynman4) are used to execute monitoring and trigger codes along with client server codes, for example, iperf clients and servers. Five different configurations are employed in the tests as shown in Table I. The *dpctl* tests utilize only the client and server hosts to execute both monitoring and switching codes. Configuration A corresponds to these tests with the monitoring and dpctl scripts running on server/client hosts, and it uses HP border switches. For *ODL remote mode* tests, the monitoring codes on client/server hosts utilize REST interface to communicate flow message needed for fail-over; both HP and Cisco switches are used in these tests. Configurations C-E implement these tests, which employ ODL controllers running on controller hosts and monitoring scripts running on server/client hosts. In *ODL local mode* tests, the monitoring and client/server codes are executed directly on the control hosts. Configuration B implements these tests, and it is identical to Configuration C except its scripts are executed on controller hosts. The measurements in Configurations B and C are quite similar, and hence we mostly present the results of the latter.

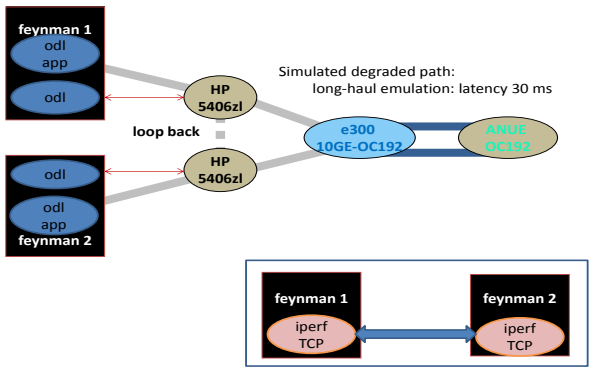
In our experiments, connection degradation events are implemented by external codes using two different methods:

(a) *Path switching using dpctl:* The current physical path with a smaller rtt is switched to a longer emulated path, whose rtt is sufficiently long to trigger the fail-over. This switching is accomplished by using dpctl codes that install OpenFlow entries on the border switches to divert the flow from current path to the longer path. The packets enroute on the current path will be simply dropped and the short-

 ISBN: 978-1-61208-450-3



(a) Configurations C-E: remote



(b) Configuration B: local

Figure 4. Remote and local modes of ODL controller configurations.

term TCP throughput becomes zero. After the fail-over, the path is switched back to the original path, and the TCP flow recovers gradually back to previous levels.

(b) *RTT extension using curl scripts:* The current connection's rtt is increased by changing the setting on ANUE device to a value above the threshold to trigger the fail-over. This is accomplished by using curl scripts that access ANUE http interface. Unlike the previous case, the packets enroute on the current path are not dropped but are delayed; thus, the instantaneous TCP throughput does not always become zero but is reduced significantly. After the fail-over, the original rtt is restored, and TCP throughput recovers gradually to previous levels.

The first degradation method using dpctl to switch the paths is only implemented for configurations with HP border switches in Configurations A - C. The second method is used for both HP and Cisco systems in Configurations D and E, and since the curl scripts are used here to change delay settings on ANUE devices, the border switches are not accessed.

B. Controller Performance

TCP throughput measurements of the connection are constantly monitored using iperf, and the Linux hosts use the default CUBIC congestion control modules [19]. The rtt between end hosts is also constantly monitored using ping, and

test configuration	controller method	path degradation	switch vendor
A	dpctl	path switch	HP
B	ODL local	path switch	HP
C	ODL remote	path switch	HP
D	ODL remote	rtt extension	HP
E	ODL remote	rtt extension	Cisco

Table I. Five test configurations with two controllers, two connection degradation methods and two switch vendors.

path switching is triggered when it crosses a set threshold. The path degradations are implemented as periodic impulses and the responses are assessed using the recovery profiles of TCP throughput captured at one second intervals. Also, the ANUE dynamics in extending the rtt affect the TCP throughput recovery, and we obtain additional baseline measurements by utilizing a direct fiber connection that avoids packets being routed through ANUE devices. Thus, TCP throughout traces in our tests capture the performances of: (a) controllers, namely, dpctl and ODL, in responding to fail-over triggers from monitoring codes, and in modifying the flow entries on switches, typically by deleting the current flows and inserting the ones for the standby path, and (b) border switches in modifying the flows entries in response to controller messages and re-routing the packets as per new flow entries.

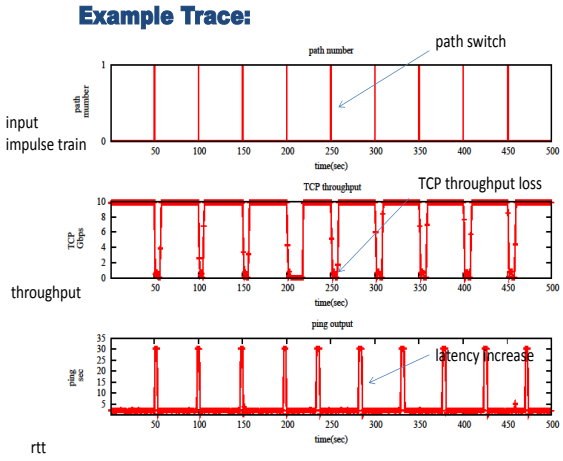


Figure 5. Trace of impulse response of TCP throughput for dpctl method with local primary path and path switching degradation.

An example TCP throughput trace of a test run of Configuration A is shown in Figure 5 for the dpctl method, with fiber connection as the primary path, and using the path switching degradation method. The connection rtt is degraded at a periodicity of 50 seconds by externally switching to the longer ANUE path, and the change is detected, as shown in the bottom plot, which in turn triggers the fail-over. Three different TCP recovery profiles from the tests are shown in Figure 6: (a) *Configuration A*: dpctl method using HP switches with connection degradation by path switching, (b) *Configuration D*: ODL method using HP switches with connection degradation by rtt extension, and (c) *Configuration E*: ODL method using Cisco switches with connection degradation by rtt extension. As seen in these plots, TCP response dynamics contain significant variations for different degradation events of the same configuration as well as between different configurations.

 ISBN: 978-1-61208-450-3

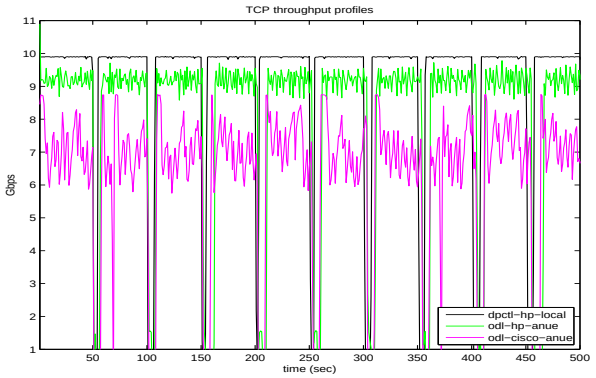


Figure 6. TCP throughput for dpctl method for configuration A and ODL methods for configuration D and E.

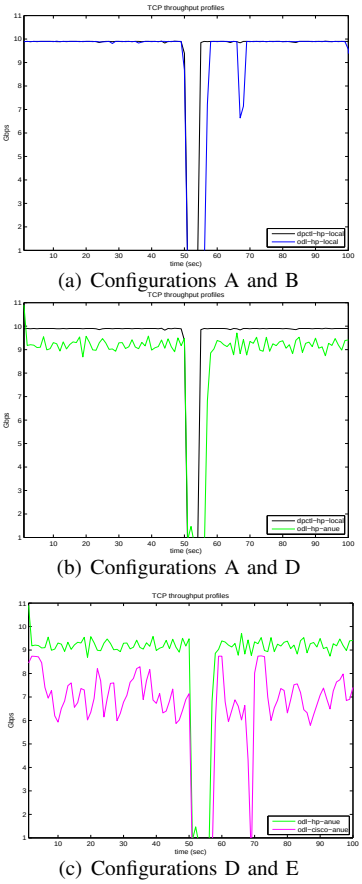
The individual TCP throughput recovery responses to single path degradation events reveal more details of the difference between the configurations as shown in Figure 7. The delayed response of ODL method compared to dpctl method can be seen in Figure 7(a) for Configurations A and B. Since the packets in transit during the switching are simply lost during path switching, the instantaneous TCP throughput rapidly drops to zero for Configuration A. On the other hand, some of the packets in transit when rrt is extended are delivered, and as a result TCP throughput may be non-zero in some cases, as shown in Figure 7(b). Another aspect is that, TCP throughput recovers to 10Gbps when the direct fiber connection is used between the switches, but only peaks around 9 Gbps when packets are sent via ANUE connection with zero delay setting as shown in Figure 7(b). Also, the recovery profiles are different between HP and Cisco switches in otherwise identical Configurations D and E as shown in Figure 7(c). Thus, TCP dynamics depend both on the controller in terms of recovery times, and on the switches in terms of peak throughput achieved and its temporal stability.

C. Switch Performance

TCP performance is effected by the path traversed by the packets between the border switches, in addition to its dependence on dpctl and ODL methods as described in the previous section. In configurations A and B, the primary connection is a few meters of fiber between the switches, and TCP throughput is restored to around 10 Gbps after the fail-over as shown in Figure 7(a). In Configurations D and E, the packets are sent through the emulated connection OC192 ANUE emulator with a peak capacity of 9.6 Gbps, and both peak value and the dynamics of TCP throughput are affected as shown in Figure 7(b). Furthermore, the connection modality affects HP and Cisco switches differently as shown in Figure 7(c) in that the latter reach somewhat lower peak throughput and exhibit larger fluctuations.

IV. Impulse Response Method

We present the *impulse response method* in this section that captures the overall recovery response by "aggregating" generic (scalar) performance measurements (such as TCP throughput as in previous section) collected in response to periodic connection degradations.



(a) Configurations A and B

(b) Configurations A and D

(c) Configurations D and E

Figure 7. Trace of impulse response of TCP throughput for dpctl method with local primary path and path switching degradation.

A. Response Regression Function

A configuration X that implements the fail-over is specified by its controllers and switches that implement fail-over, and also the monitoring and detection modules that trigger it. Let $\delta(t - iT), i = 0, 1, \ldots, n$, denote the input impulse train that degrades the connection at times $iT + T_D$, where t represents time, T is the period between degradations and $T_D < T$ is the time of degradation event within the period. Let $T^X(t)$ denote the parameter of interest, such as TCP throughput, corresponding to $\delta(t-iT), i = 0, 1, \ldots, n$ as shown in Figure 6 for configurations X=A,D,E. Let $R^X(t) = \mathbf{B} - T^X(t)$ denote the response that captures the "unrealized" portion of peak performance level $\mathbf{B}$, for example, the residual bandwidth of a connection with capacity $\mathbf{B}$. We define the *impulse response function* $R^X(t)$ such that

$$R_i^X(t) = R^X(t - iT), t \in [0, T)$$

is the response to ith degradation event $\delta(t - iT), i = 0, 1, \ldots, n$. An ideal impulse response function is also an impulse train that matches the input, wherein each impulse represents the instantaneous degradation detection, fail-over and complete recovery. But in practice, each $R_i^X(t)$ is a "flattened" impulse function whose shape is indicative of the effectiveness of fail-over. In particular, its leading edge represents the effect of degradation and its trailing edge represents that of recovery, and the narrower this function is the quicker is the recovery.

 ISBN: 978-1-61208-450-3

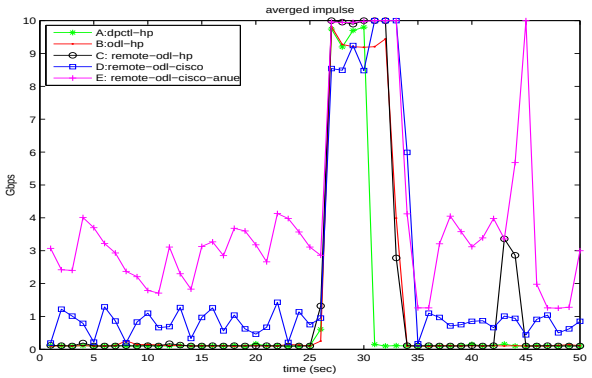


Figure 8. Examples of impulse response functions for Configurations A-E for a single path degradation.

Examples of $R_1^X(.)$ are shown Figure 8 for configurations A-E; these TCP measurements show significant temporal variations that persist across the different degradation events, which make it difficult to objectively compare these single-event time plots.

We define the *response regression* of configuration X as

$$\bar{R}^X(t) = E\left[R_i^X(t)\right] = \int R_i^X(t) d\mathbf{P}_{R_i^X(t)},$$

for $t \in [0, T)$, where the underlying distribution $\mathbf{P}_{R_i^X(t)}$ is quite complex in general since it depends on the dynamics of controllers, switches, end hosts, application stack and monitoring and detection modules that constitute X. It exhibits an overall decreasing profile for $t \in [0, T_D + T_I]$ followed by an increasing profile for $t \in (T_D + T_I, T]$, where T_I is the time needed for the application to react to connection degradation. After the fail-over, TCP measurements exhibit an overall increasing throughput until it reaches its peak as it recovers after becoming nearly zero following the degradation. We consider that a similar overall behavior is exhibited by the general performance parameters of interest.

The *response mean* $\hat{R}_i(t)$ of $\bar{R}_i(t)$ based on discrete measurements at times $t = j\delta$, $j = 0, 1, ..., T/\delta$, is

$$\hat{R}^X(j\delta) = \frac{1}{n}\sum_{i=1}^{n}\left(R_i^X(j\delta)\right)$$

which captures the average profile. Examples of $\hat{R}_i^X(.)$ for TCP throughput are shown Figure 9 for different configurations based on 10 path degradations with $T = 50$ seconds between them, which show the following general trends.

- The dpctl method responds seconds faster than ODL method as indicated by its sharper shape, although their leading edges are aligned as shown in Figure 9(a).
- The connection degradation implemented by the rtt extension has a delayed effect on reducing the throughput compared to the path switching degradation method as indicated by its delayed leading edge in Figure 9(b).
- The dynamic response of regression profiles of HP 5604zl and Cisco 3064 switches are qualitatively quite similar as shown in Figure 9(c), but the latter achieved somewhat lower peak throughput overall.

In view of the faster response of dpctl method, we collected additional measurements in configurations A and C using

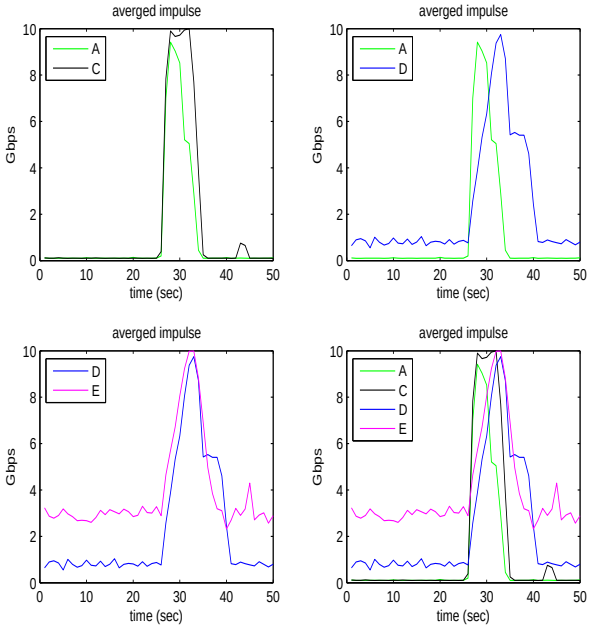
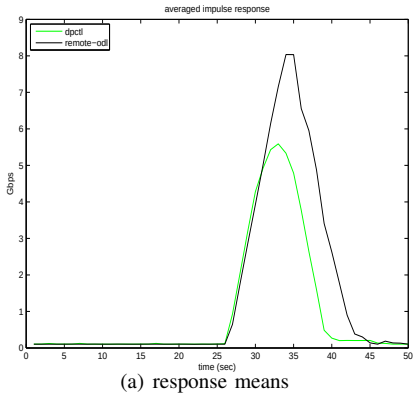
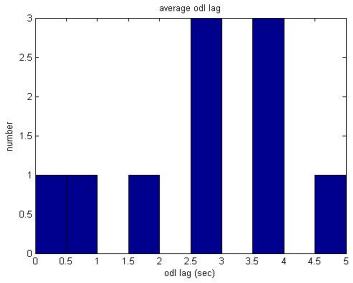


Figure 9. Impulse response regressions for $n = 10$ and $T = 50$ sec: (a) top-left: A and C, (b) top-right: A and D, (c) bottom-left: D and E , and (d) bottom-right: all.



(a) response means



(b) delay histogram

Figure 10. Comparison of response means of dpctl (configuration A) and ODL (configuration C) methods using 100 path degradations with $T = 50$ seconds.

100 path degradations, and the resultant response means are somewhat smoother compared to 10 degradations as shown in Figure 10 (a) for both dpctl and ODL methods. Furthermore, the response mean of ODL method remains consistent with more measurements, and a histogram of the relative delays of ODL method compared to dpctl method is plotted in Figure 10(b), and they are in the range of 2-3 seconds in the majority of cases. These measurements clearly show the faster response of the dpctl method for these scenarios.

B. Statistical Analysis

A generic empirical estimate $\tilde{R}^X(t)$ of $\bar{R}(t)$ based on measurements at times $t = j\delta$, $j = 0, 1, ..., T/\delta$, is

$$\tilde{R}^X(j\delta) = \frac{1}{n}\sum_{i=1}^{n}\left[g\left(R_i^X(j\delta)\right)\right]$$

for an estimator function g. We consider that the function class $\mathcal{M}$ of $\tilde{R}^X(.)$ consists of unimodal functions, each of which consists of degradation and recovery parts when viewed as a function of time. For ease of notation, we also denote $\tilde{R}^X(.)$ by f in this section such that it is composed of a degradation function f_D and a recovery function f_R as follows:

$$f(R_i(t)) = \begin{cases} f_D(R_i(t)) & \text{if } t \in [0, T_D + T_I] \\ f_R(R_i(t)) & \text{if } t \in (T_D + T_I, T] \end{cases} \quad (1)$$

where $f_D \in \mathcal{M}_D$ and $f_R \in \mathcal{M}_R$ correspond to the leading and trailing edges of the response regression. The *expected error* $I(f)$ of the estimator f is given by

$$\begin{aligned} I(f) &= \int [f(t) - R_i^X(t)]^2 d\mathbf{P}_{R_i^X(t),t} \\ &= \int_{[0,T_D+T_I]} [f_D(t) - R_i^X(t)]^2 d\mathbf{P}_{R_i^X(t),t} \\ &\quad + \int_{(T_D+T_I,T]} [f_D(t) - R_i^X(t)]^2 d\mathbf{P}_{R_i^X(t),t} \\ &= I_D(f_D) + I_R(f_R). \end{aligned}$$

The *best expected estimator* $f^* = (f_D^*, f_R^*) \in \mathcal{M}$ minimizes the expected error $I(.)$, that is

$$I(f^*) = \min_{f \in \mathcal{M}} I(f).$$

The *empirical error* of an estimator f is given by

$$\hat{I}(f) = \frac{\delta}{Tn}\sum_{i=1}^{n}\sum_{j=1}^{T/\delta}\left[f(j\delta) - \left(R_i^X(j\delta)\right)\right]^2.$$

The *best empirical estimator* $\hat{f} = (\hat{f}_D, \hat{f}_R) \in \mathcal{M}$ minimizes the empirical error $\hat{I}(.)$, that is,

$$\hat{I}(\hat{f}) = \min_{f \in \mathcal{M}} \hat{I}(f).$$

Since the response mean $\hat{R}(t)$ is the mean at each observation time $j\delta$, it achieves zero mean error, which in turn leads to zero empirical error, that is, $\hat{I}\left(\hat{R}\right) = 0$; thus, it is a best empirical estimator. By ignoring the minor variations for the smaller values of n, we assume that $\hat{R}$ is composed of a non-decreasing function $\hat{R}_D$ followed by a non-increasing function $\hat{R}_R$ that correspond to decreasing and increasing parts of the performance parameter (such as TCP throughput), respectively. This assumption is valid for the response means of dpctl and ODL methods in Configurations A and C, respectively, shown in Figure 10. In both cases, the response mean is composed of an increasing part followed by a decreasing part once the small variations in the tail of ODL method are ignored.

We will now show that Vapnik-Chervonenkis theory [13] guarantees that the response mean $\hat{R}(t)$ is a good approximation of the response regression $\bar{R}(t)$, and furthermore its performance improves with more measurements from connection degradation events. Such performance guarantee is a direct consequence of the monotone nature of the underlying f_D and f_R functions. Furthermore, this performance guarantee is distribution-free, that is, independent of the underlying distributions due to controllers and switches, and is valid under very general conditions [13] on the variations of performance (such as TCP throughput) measurements. We now provide an outline of the proof of this result. Let $\hat{R} = \left(\hat{R}_D, \hat{R}_R\right)$ such that the estimator is decomposed into two monotone parts, namely non-decreasing $\hat{R}_D$ and non-increasing $\hat{R}_R$ such that $\hat{I}(\hat{R}) = \hat{I}_D(\hat{R}_D) + \hat{I}_R(\hat{R}_R)$. By using Vapnik-Chervonenkis theory [13] we have

$$\begin{aligned} &\mathbf{P}\left\{I\left(\hat{R}\right) - I(f^*) > \epsilon\right\} \\ &\leq \mathbf{P}\left\{I\left(\hat{R}_D\right) - I(f_D^*) > \epsilon/2\right\} \\ &\quad + \mathbf{P}\left\{I\left(\hat{R}_R\right) - I(f_R^*) > \epsilon/2\right\} \\ &\leq \mathbf{P}\left\{\max_{h \in \mathcal{M}_D} |I_D(h) - \hat{I}_D(h)| > \epsilon/4\right\} \\ &\quad + \mathbf{P}\left\{\max_{h \in \mathcal{M}_R} |I_R(h) - \hat{I}_R(h)| > \epsilon/4\right\} \\ &\leq 16\mathcal{N}_\infty\left(\frac{\epsilon}{2\mathbf{B}}, \mathcal{M}_D\right) n e^{-\epsilon^2 n/(8\mathbf{B})^2} \\ &\quad + 16\mathcal{N}_\infty\left(\frac{\epsilon}{2\mathbf{B}}, \mathcal{M}_R\right) n e^{-\epsilon^2 n/(8\mathbf{B})^2} \end{aligned}$$

where $\mathcal{N}_\infty(\epsilon, \mathcal{A})$ is the ϵ-cover of function class $\mathcal{A}$ under L_∞ norm. In the above derivation, the first inequality follows since the negation of the condition in either right term implies the negation of the condition in left term. The second inequality follows from the uniform convergence property applied to each term [13], and the third inequality follows by applying the uniform bound for each term ([20], p. 143). Due to the monotonicity of functions in $\mathcal{M}_D$ and $\mathcal{M}_R$, their total variation is upper bounded by $\mathbf{B}$, which provides us the following upper bound ([20], p. 175): for $\mathcal{A} = \mathcal{M}_D, \mathcal{M}_R$, we have

$$\mathcal{N}_\infty\left(\frac{\epsilon}{2\mathbf{B}}, \mathcal{A}\right) < 2\left(\frac{4n}{\epsilon^2}\right)^{(1+2\mathbf{B}/\epsilon)\log_2(2\epsilon/\mathbf{B})}.$$

By using this bound, we obtain

$$\begin{aligned} &\mathbf{P}\left\{I\left(\hat{R}_i\right) - I(f^*) > \epsilon\right\} \\ &< 64\left(\frac{4n}{\epsilon^2}\right)^{(1+2\mathbf{B}/\epsilon)\log_2(2\epsilon/\mathbf{B})} n e^{-\epsilon^2 n/(8\mathbf{B})^2}. \end{aligned}$$

The exponential term on the right hand side decays faster in n than other terms, and hence for sufficiently large n it can

 ISBN: 978-1-61208-450-3

be made smaller than a given probability α. Thus, the expected error $I(\hat{R})$ of the response mean used in the previous section is within ϵ of the optimal error $I(f^*)$ with a probability that increases with the number of observations, and is independent of the underlying distributions. An indirect evidence of this is the increased stability of the response mean as we increase the number of connection degradation events from 10 to 100 in Figures 9(a) and 10, respectively. In summary, this analysis provides a statistical basis for using the response mean $\hat{R}$ as an approximation to the underlying response regression $\bar{R}$ in comparing different methods and configurations.

V. Conclusion

We considered two sites connected over a dedicated, long-haul connection, which must fail-over to a standby connection upon degradations that affect the host-to-host application performance. Current solutions require significant customization due to the vendor-specific software of network devices and applications, which have to be repeated with upgrades and changes. Our objective is to exploit recent SDN and Network Function Virtualization (NFV) technologies to develop faster and more flexible fail-over solutions implemented entirely in software. We first presented a light-weight method that utilizes host scripts to monitor the connection rtt and OpenFlow dpctl API to implement the fail-over. We then presented a second method using two OpenDaylight (ODL) controllers and REST interfaces. We performed experiments using a testbed consisting of HP and Cisco switches connected over long-haul connections emulated in hardware. They showed that both methods restore TCP throughput, but their comparison was complicated by the restoration dynamics of TCP throughput which contained significant statistical variations. To account for them, we developed the impulse-response method to estimate the response regressions, which enabled us to compare these methods under different configurations, and conclude that on the average the dpctl method responds several seconds faster than ODL method.

It would also be of future interest to generalize the proposed methods to trigger fail-overs based on parameters of more complex client-server applications. The performance analysis of such methods will likely be much more complicated since the application dynamics may be modulated by the already complicated TCP recovery dynamics. Currently there seems to be an explosive growth in the variety of SDN controllers [4], including open source and vendor specific ones. Also, there is a wide variety of implementations of OpenFlow standards by switch vendors [1], ranging from building additional software layers on existing products to developing completely native software stacks. It would be of future interest to develop general performance analysis methods that enable us to compare various SDN solutions (that comprise of controllers, switches and application modules) for more complex scenarios such as data centers and cloud services distributed across wide-area networks. In particular, it would be of interest to develop methods that directly estimate the performance differences between different configurations from measurements using methods such as the differential regression [21].

Acknowledgment

This work is funded by the High-Performance Networking Program, Office of Advanced Computing Research, U.S. Department of Energy, and by Extreme Scale Systems Center, sponsored by U. S. Department of Defense, and performed at Oak Ridge National Laboratory managed by UT-Battelle, LLC for U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," Proceedings of the IEEE, vol. 103, no. 1, 2015, pp. 14–76.

[2] Y. D. Lin, D. Pitt, D. Hausheer, E. Johnson, and Y. B. Lin, "Software-defined networking: Standardization for cloud computing's second wave," IEEE Computer: Guest Editorial, November 2014, pp. 19–21.

[3] E. B. Y. M. B. Al-Somaidai, "Survey of software components to emulate openflow protocol as an sdn implementation," American Journal of Software Engineering and Applications, vol. 3, no. 6, 2014, pp. 74–82.

[4] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in Proc. 2nd USENIX Conf. Hot Topics Manage.Internet Cloud Enterprise Netw. Services, 2012, p. 10.

[5] J. Tourrilhes, P. Sharma, S. Banerjee, and J. Pettit, "SDN and OpenFlow evoltion: A standards perspecitve," IEEE Computer, November 2014, pp. 22–29.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. V. aand J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Holzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined WAN," pp. 3–14, 2013.

[7] K. Phemius, M. Bouet, and J. Leguay, "DISCO: distributed multi-domain sdn controllers," 2013, http://arxiv.org/abs/1308.6138.

[8] "Opendaylight," www.opendaylight.org.

[9] V. Tiwari, R. Parekh, and V. Patel, "A survey on vulnerabilities of openflow network and its impact on sdn/openflow controller," World Academics Journal of Engineering Sceinces, vol. 1, 2014, pp. 1005:2–5.

[10] Y. Srikant, The Mathematics of Internet Congestion Control. Birkhauser, Boston, 2004.

[11] W. R. Stevens, TCP/IP Illustrated. Addison Wesley, 1994, volumes 1-3.

[12] N. S. V. Rao, J. Gao, and L. O. Chua, "On dynamics of transport protocols in wide-area internet connections," in Complex Dynamics in Communication Networks, L. Kocarev and G. Vattay, Eds. Springer-Verlag Publishers, 2005.

[13] V. N. Vapnik, Statistical Learning Theory. New York: John-Wiley and Sons, 1998.

[14] C. E. Rothenberg, R. Chua, J. Bailey, M. Winter, C. N. A. Correa, S. C. de Lucena, M. R. Salvador, and T. D. Nadeau, "When open source meets network control planes," IEEE Computer, November 2014, pp. 46–53.

[15] N. S. V. Rao, S. E. Hicks, S. W. Poole, and P. Newman, "Testbed and experiments for high-performance networking," in Tridentcom: International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2010.

[16] "Open networks foundation," www.opennetworking.org.

[17] "Software defined networks," cisco Systems, www.cisco.com.

[18] "Software defined networking," hP, www.hp.com.

[19] I. Rhee and L. Xu, "Cubic: A new tcp-friendly high-speed tcp variant," in Proceedings of the Third International Workshop on Protocols for Fast Long-Distance Networks, 2005.

[20] M. Anthony and P. L. Bartlett, Neural Network Learning: Theoretical Foundations. Cambridge University Press, 1999.

[21] B. W. Settlemyer, N. S. V. Rao, S. W. Poole, S. W. Hodson, S. E. Hicks, and P. M. Newman, "Experimental analysis of 10gbps transfers over physical and emulated dedicated connections," in International Conference on Computing, Networking and Communications, 2012.

 ISBN: 978-1-61208-450-3