Performance Routing Using SDN

Shankar Krishnamurthy
School of Computer Science
Arizona State University
Tempe, Arizona-85281
Email: Shakar.Krishnamurthy@asu.edu

Sidharth Singh
School of Computer Science
Arizona State University
Tempe, Arizona-85281
Email: ssing124@asu.edu

***Abstract*— Routing Protocols like OSPF compute the best network path among all available paths based on a metric like hop count or shortest path. Once this routing decision is made and the routing tables are populated, all traffic destined to a particular subnet goes via the same path, unless of course, a router along the way is not available anymore. While web traffic is the most common traffic in the Internet, there are several other applications that have different expectations of the network. To assume that a single path chosen by the routing algorithm works best for all applications is not correct. In this paper, we describe the concept of Performance Routing (Pfr) – a phrase coined originally by Cisco. We implement Performance Routing in a controlled network using Software-defined Networking. To prove our hypothesis, we perform experiments to show that different applications are sometimes better off taking different network paths – despite what the router ordered.**

Keywords—SDN; OpenFlow; Pox; Pfr; Routing

I. INTRODUCTION

Traditional Routing Protocols consider factors like hop count, available bandwidth etc. as metrics for computing the best network path among all available. OSPF, for instance, calculates routes based on Shortest Path First [1]. Modern networks, especially the Internet, have turned into a complex mesh of networking equipment. Often there is more than just one path to reach a destination subnet. Routing protocols typically overlook the second best path. This is done (primarily) to prevent a network loop which can cause considerable anguish to network administrators.

There would be no issue at all if all every application that consumes the network had same characteristics. Unfortunately that is not the case. Consider a File Transfer Protocol (FTP) and Voice over IP Service (VoIP). FTP can be potentially used to transfer large files running into Gigabytes over the network. The primary requirement for FTP, therefore, is access to a higher bandwidth path so that the File Transfer achieves as much throughput as possible. The round trip time of individual packets is irrelevant in this case.

On the other hand, a VoIP service can have quite different expectations of the network. A VoIP service usually comprises exchanges of small messages. The primary requirement is not bandwidth but lower latency. Lower the round trip time of packets in a VoIP call, better the user experience.

Bearing these two examples in mind, one would ask - why does FTP and VoIP – two applications with such contrasting expectations of the network - take the same network path? In other words, can the network path not chosen by a routing algorithm like OSPF be leveraged by another application? In this paper, we introduce the concept of Performance Routing (Pfr). Performance Routing was introduced by Cisco [2] and the specific implementation is not openly available. However, without worrying about the exact implementation, we borrow the idea proposed by Pfr and implement in our way inside a controlled network using Software-defined Networking.

Our hypothesis is that by utilizing all available network paths based on requirements specified by an application, we can extract a higher value from the network. The separation of application across different network paths prevents one application from suffering the consequences of congestion in the other path(s). Moreover, our implementation of Performance not only guarantees performance benefits but also improved network utilization. At this point, it is worth mentioning that our implementation of Performance Routing using SDN must not be confused with a link level load balancer. In this work, we place the onus on applications of stating their requirements/expectations from the network.

To implement this, we add the intelligence to a SDN controller. A network administrator who is aware of the network topology, link bandwidths and network latency can pass this information to a SDN controller by way of a policy file. The SDN controller uses this information to process incoming flows and take apposite action based on the nature of the flow. For example – when a VoIP flow arrives at the controller, it looks at the config file to see if the secondary path (path not picked by the routing algorithm) has an acceptable Round Trip Time. If yes, the controller sends the VoIP flow along the secondary path. The low latency requirements of the VoIP flow are met and at the same time primary flow remains relatively uncongested to take other traffic. In addition, the overall network utilization increases.

Currently, this segregation of different application flows across different network paths is implemented statically based on a configuration file that the network administrator supplies. In the future, we can quite easily extend this to take decisions based on real time conditions across network paths. This could be invaluable to the network when it is plagued by congestion. Usually, when congestion happens along a network path, it leads to timeouts and eventually retransmission of packets (for TCP flows). It is quite ironical that when the network is

already bogged down, end hosts send more packets through retransmissions. The three hallmarks of a congested network are – packet drops, increased round trip time and dip in throughput. Our Performance Routing implementation can easily sense these failings in a network and divert flows to another network path.

In the remainder of this paper, we talk about our related work in Section II. In Section III, we talk about our design and architecture we also explain our network topology for the evaluation of this paper. In Section IV, we discuss implementation details before providing experimental results in Section V. Finally we talk about the possible future work in the penultimate section before concluding in Section VII.

II. Related Work

A. Cisco's Performance Routing (Pfr)

Our current work in this paper is inspired from Cisco's proprietary Performance Routing [2] [3] protocol. This protocol runs over routing protocol stack and provides application awareness to the routes. It is inspired from SDN framework. The architecture consists of two blocks – Master Router and Border Router. Master node monitors the border router for certain policies like link utilization which are defined by the network administrator using OER map [3]. OER map is similar to a list of policies which are defined by the administrator. If any violation occurs, master node sends an action according to the rule defined to the border router. We can relate Master node to be acting like a controller in SDN. Similarly Border router is like an OpenFlow switch which doesn't have control flow knowledge.

The problem with Pfr is that it is a proprietary protocol and hence can't be implemented without Cisco boxes. In this paper, the protocol which we have built using Software Defined Networks is able to achieve this up to certain extent.

B. Policy Based Routing

Policy-based Routing (PBR) [4] works on the technique of routing on the basis of a policy. These policies are generally specified using a route-map [4]. PBR can provide flexibility in routing by routing traffic on the basis of source-based instead of traditional destination based routing. It can also be used to change paths of the traffic from primary to some secondary path on the basis of fields in IP header.

The problem with PBR is that it places onus on the network administrator to write a detailed policy. Network administrator should know what all kinds of traffic need to be diverted off the primary path and why. As it is manually decided by humans, possibility of a mistake increases. Certain mistakes in the logic or writing the policy can result in various issues like routing loops or asymmetric routing [5].

The Performance routing protocol which we are introducing aims to solve this by deciding the flow on the basis of differentiated services. The protocol expects that interested traffic (traffic which needs to be re-engineered) will be set with some DSCP [6] bit. Once a DSCP bit is seen, the protocol would know the characteristic requirements of the flow like high bandwidth, low jitter etc. and could route accordingly.

C. Quality of Service

Quality of Service (QoS) [7] is a contract which says that certain kind of traffic will be given special treatment when congestion occurs. In current networks, QoS is often seen when traffic like RTP is flowing through the routers. This is because RTP traffic have a special requirement of low jitter rate and QoS service tries to make sure the application can achieve that.

Problem with QoS is that it has local scope and aims on providing QoS by allocating resources to an application on that link. It fails short in addressing scenarios where better quality could have been achieved just by re-routing that application through some other path if it exists.

III. DESIGN AND ARCHITECTURE

In our implementation of Performance Routing, we have the intelligence to pick different network paths for different applications in the SDN Controller. Figure 1 shows our test topology and an expanded view of the modules inside the controller.

Let us understand the choice of topology before discussing each module inside the controller. Three applications are used in the evaluation of this paper – File Transfer Protocol, Real Time Protocol like VoIP and standard web traffic. Each of these application run on a different server located in the same sub-network. The client is directly plugged into an openflow switch and there are two distinct network paths from client (on the left) to servers (on the right).

We consider the path through Router 1 as the primary path to reach the servers. Note that in reality, a routing algorithm like OSPF will select one of the paths as Primary. The other path that passes through Router 2 is referred to as the backup path. Below we discuss each module in detail.

A. Policy Module

For Performance Routing, knowledge about network characteristics is the first and foremost requirement. In our current implementation, the SDN Controller is supplied with a policy file for each router. The policy file is static and it contains information about provisioned bandwidth for every link emanating from the router and fixed latency along all network paths to reach the edge router or application servers.

In addition to information about the network characteristics, the Policy Module also contains application specific requirements. We refer to this as pre-stated application policies. Consider the VoIP service which is sensitive to latency in the network. A maximum acceptable latency range can be specified in the policy module that states – “pick the non-primary path only if round trip latency on that path is less than 100ms”. This is similar to QoS requirements by applications. The SDN controller takes into account these pre-slated application policies while deciding where to send flows.

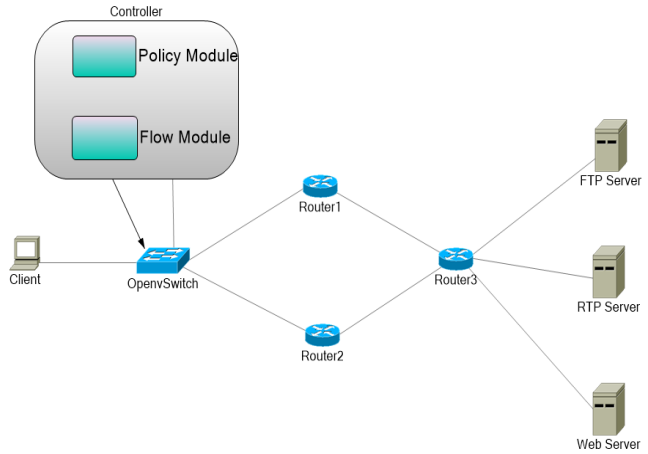

Figure 1: Performance Routing Architecture

B. *Flow Module*

The Flow Module is central to Performance Routing. Controller reads network characteristics and pre-slated application policies from the policy module and a selects a path for the flow. Once the decision is made, it is passed onto the Flow Module which interprets the decision. Based on its interpretation, the Flow Module sends OpenFlow [8] Flow-Mod messages to the OpenFlow Virtual Switch and populates the Flow Table. Necessary and appropriate action is taken by Flow Module once again for the return traffic.

IV. IMPLEMENTATION

We implemented Performance Routing by writing our own SDN Controller. In the following sections, we describe the platform used in our implementation, our controller logic and the all the scenarios that our controller can handle.

A. *Platform*

Our SDN Openflow Controller for Performance Routing is written in POX. Our decision to choose POX over other popular controllers like Floodlight was due to our familiarity and presence of useful API's in POX. Before implementing our controller, we studied the implementation of SimpleL2Learning [9] controller provided by POX.

We used Mininet [10] for network virtualization. We created our own custom topology similar to Figure 1 using a python script. Our openflow switch operates in Layer 2 mode. Router1 and Router2 along the two network paths operate in Layer 3 mode and we have enabled IP Forwarding on them. On router 3, we created separate routing tables for each interface connected to Router 1 and Router 2 and added a couple of IP Rules that specified which routing table to consult.

B. *Controller Logic*

The biggest challenge while implementing the controller was to identify the type of application that a flow represents. The first approach we considered was Differentiated Services where different types of applications can be marked with a particular Differentiated Services Code Point (DSCP) bit in the IP Header [6]. Differentiated services is a networking paradigm that specifies an approach for providing quality of service in modern IP networks. Using this approach, we could mark different applications under test like File Transfer or VoIP with a unique DSCP bit. The controller would then have the task of parsing the IP Header and extracting this bit before making a choice of what network path to use.

However, incorporating DSCP bits to detect application type meant we could not use standard traffic generation tools like Iperf and Curl. One solution was to craft our own packets in entirety but that would have been time consuming. In practice, intermediate Layer 3/4 proxies or other sophisticated Application Delivery Controllers perform the task of DSCP markings. In the end we worked around having to implement Differentiated Services by achieving the same result using Transport Layer port numbers.

Well known applications usually use well-known port numbers which we exploited in our controller logic. For example – File Transfer Protocol's data plane uses TCP Port number 20 and Real Time Protocol (RTP) normally uses UDP Port 5004. In any case, for purely academic purposes, a service can be made to run on any free port. For every new packet that reached our controller (usually the first packet of a flow), the following checks were performed –

- Whether the IP payload contained a TCP or UDP header? This was easily checked using API's in POX.
- A TCP packet was forwarded to a TCP Handler.
- The TCP Handler matched the destination port in the packet to a list of application port numbers in the policy module.
- If a match is found, a network path is selected based on pre-stated application policies and/or network characteristics and controller logic specific to various applications.
- If no match found, POX's default/inbuilt Layer 2 forwarding module is called (L2Learning). [9]

The handling of an incoming UDP packet at the controller is same as TCP, except that a UDP Handler is called instead. In both cases, based on the pre-stated application policies, the backup path could be a better suited path. To handle this, two functions have been implemented in our controller that select backup routes, whenever applicable, in case of TCP and UDP.

C. *Scenarios handled in Controller*

Now that our controller is able to identify flows belonging to different kinds of applications based on Layer 4 ports, we describe the different scenarios that our controller currently handles. Recall that according to our network topology described in Figure 1, we have two network paths from client to the destination subnet where application servers like File

Transfer (FTP), Real Time Protocol (RTP) and Web Server are hosted. Also recall that we call the network path via Router 1 as 'Primary' and the network path via Router 2 as 'Secondary'.

Figure 2 describes five scenarios handled by the controller where it performs Performance Routing for two applications – File Transfer Protocol and Real Time Protocol (VoIP).

#	Bandwidth	Latency	Behavior
1	**Primary = Backup**	**Primary = Backup**	Equal Cost Multipath Routing
2	**Primary = Backup**	**Primary > Backup**	FTP on Primary RTP on Backup
3	**Primary = Backup**	**Primary < Backup**	FTP on Backup RTP on Primary
4	**Primary > Backup**	**Primary < Backup** Latency on Backup within acceptable range for RTP as per pre-stated application policy	FTP on Primary RTP on Backup
5	**Primary > Backup**	**Primary < Backup** Latency on Backup ***NOT** within acceptable range for RTP as per pre-stated application policy	Both FTP and RTP on Primary

Figure 2: Scenarios handled by Controller

In scenario (1), primary and backup network paths are identical to each other in terms of both total provisioned bandwidth and total latency from client across to the destination subnet where servers reside. Although the two paths are exactly identical, a traditional routing algorithm will (typically) select only path. In our performance routing implementation however, the policy module has enough information while the controller has enough intelligence to see that the two links are indistinguishable for both FTP and RTP. Therefore, we implement Equal Cost Multipath Routing [11] in this scenario.

In scenario (2) and (3), primary and backup network paths are identical in terms of total provisioned bandwidth. However, in (2), the total latency along the primary path is higher than the backup path. Our controller immediately senses that lower latency is more important for RTP than FTP. Therefore, the controller's flow module redirects the RTP flows to the backup path. FTP flows take the primary path with slightly higher latency. As long as the difference in Round Trip Time between primary and secondary is not too big, FTP should be reasonably unaffected. Scenario (3) is the exact opposite of Scenario (2).

Scenario (4) and Scenario (5) present a more intriguing case when primary path is definitely better than the backup – both in terms of bandwidth (higher) and latency (lower). The question that our controller now asks is – "how worse off is the backup path in terms of latency?" At this point, the policy module is consulted. If the RTP application has specified an upper bound on tolerable latency, the controller can make a choice and send RTP flows via backup path.

To understand this clearly, let us take an example. Assume that the primary path has an overall provisioned bandwidth of 100Mbits/sec in comparison to a lower bandwidth of 50Mbits/sec on the backup path. Moreover, the primary path has an end to end latency of 50ms in comparison to secondary's 75ms. It would seem at first glance that the primary path is certainly a better path and the routing algorithm was spot on. However, an application's requirement for effective functioning is best known by the application itself, not by software running on a router. If an RTP application's worst case acceptable end to end latency is in the range of 100ms, it would still make sense to send the RTP flow on backup. This is scenario (4) from Figure 2.

The list of scenarios that our controller handles currently cover all aspects of variability in bandwidth and latency across two available paths. With some trivial modifications, our controller can easily pick among more than two available paths.

V. EVALUATION

In this section, we evaluate the performance of our Performance Routing implementation on an emulated network using the Network Virtualization software Mininet. The five scenarios explained in the previous section were put to test and the effects of Performance Routing were analyzed on File Transfer and Real Time Protocols.

(1) (1)

A. Traffic Patterns

In each experiment, we ran a mix of the following four kinds of traffic –

- File Transfer using FTP protocol – A 1 Gigabyte was uploaded from Client to FTP Server.
- Real Time Protocol – We wanted to simulate a real world VoIP service like Skype for our evaluation but we could not get one to work on the Mininet Setup in time. We studied the typical characteristics of RTP traffic and research showed that average size of RTP packets is close to 250 Bytes. We simulated UDP Traffic using Iperf and restricted packet size to 250 Bytes. The RTP traffic was run for the same duration as FTP traffic.
- Web Traffic – We wrote custom python scripts to spawn 20 threads and send HTTP requests using Curl. A total of 1000 curl requests were sent during every iteration. The homepage of Google News'

(approximately 600KB) was uploaded in each Curl Request. On the web server, we ran Python's inbuilt SimpleHTTPServer. Please note that in the current implementation, we do not perform any specific QoS for Web Traffic. The primary aim of Web Traffic is to prevent the network paths from being completely idle. For Web Traffic, our controller performs a simple Equal Cost Multi Path load balancing.

- Transient Traffic – In addition to the above three, we also run short bursts of intransient TCP along both networks traffic using Iperf. Transient traffic is of little value to us and it simply represents the other traffic in the network.

B. *Evaluation Metrics*

Our current implementation of Performance Routing works for applications with either high bandwidth requirements or low latency requirements. The two primary evaluation metrics in our evaluation are throughput and delay jitter. Our assumption is that with Performance Routing, File Transfer application must get a higher throughput while RTP application should see reduced delay jitter.

C. *Experimental Results*

In this section, we present some of the results from our evaluation of Performance Routing using SDN. As explained above, we ran four different kinds of flows – FTP, RTP, Web Traffic and Transient TCP. The five scenarios in Figure 2 are tested with and without Performance Routing. To assess the consistency of results and account for outliers in the data set, we ran each iteration ten times. Note that we used Linux utility 'tc' and 'tbf' to set delay and limit bandwidth across links.

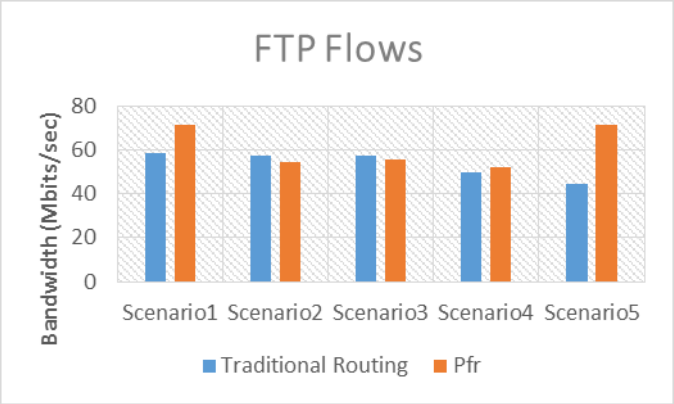

Figure 3: Bandwidth used by FTP flows with and without Pfr

The testing methodology for scenario 1 was a little different from other scenarios. In scenario 1, primary and the backup path are indistinguishable in terms of both bandwidth and latency. Both network paths have a provisioned bandwidth of 100Mbits/sec and an end to end latency of 100ms. Our controller implements an Equal Cost Multipath Load Balancing in this case. To test this, we created ten parallel FTP connections. Figure 3 shows that the average throughput achieved with Performance Routing is approximately 20% higher in scenario I. Similarly for RTP connections we create ten parallel UDP flows and Figure 3 shows a slight decrease in jitter with Performance Routing in scenario 1.

In Figure 3, scenario 2 and scenario 3 show an unexpected, although small, dip in throughput for the FTP flow with Performance Routing. In scenario 2, both network paths still have the same bandwidth of 100Mbits/sec, but the end to end delay on primary link is greater by 25ms. Our controller sends the FTP flow along the primary path and RTP flow along the backup. Figure 4 shows that RTP's delay jitter with Performance Routing falls to almost a third of the value without Performance Routing in scenario 2.

Scenario 4 shows marginal improvement for both FTP flow and RTP flow. Figure 4 shows that the jitter with Performance Routing reduces almost by 40% for RTP Traffic. This is a particularly good result because our controller sent the RTP flow on backup because of pre-stated application policies described above.

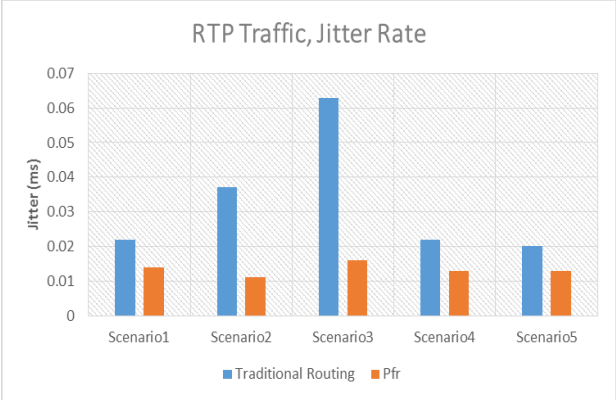

Figure 4: Delay Jitter in RTP flows with and without Pfr

VI. FUTURE WORK

In this paper, we present a very basic implementation of Performance Routing using SDN. In the future, our first challenge is to implement a routing protocol like OSPF on the Openflow switch so that our primary routes are computed by OSPF instead of being arbitrarily chosen. Furthermore, in the current model, the network characteristics are statically picked up from the policy module. In the future, we plan to select network paths based on current available bandwidth and not the overall provisioned bandwidth specified in the policy module. We have already implemented this as part of Project II and it can be integrated easily. We also plan to improve our traffic profile and incorporate Performance Routing for Web Traffic. Lastly, in this paper we simulate RTP traffic using Iperf but in the future works, we would like to incorporate a real RTP client in order to showcase the benefits of Performance Routing more effectively.

VII. CONCLUSION

Traditional routing algorithms lack an element to flexibility and they are not amenable to different requirements of different applications. In this paper, we propose that Performance Routing (Pfr) using SDN can make a network more flexible. To evaluate our hypothesis, we have created a SDN controller that identifies the type of incoming flow by looking at its destination TCP Port and performs SDN Routing based on policies inside the policy module. Our evaluation in a topology with two network paths show improvements in bandwidth for File Transfer Protocol and a reduction of delay jitter for Real Time Protocol (VoIP). In addition to improved performance, Pfr also guarantees a better network utilization. There is an unmistakable link between Performance Routing and Quality of Service and in our current implementation, we have created a basic QoS system for File Transfer and Real Time applications.

REFERENCES

[1] "Dijkstra's Algorithm" [Online] Available: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[2] "Pfr Technology Overview" [Online] Available: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.

[3] Cisco Validated Design I "Transport Diversity: Performance Routing (Pfr) Design Guide"., February 2008.

[4] "Policy-Based Routing, White Paper" [Online] Available: http://www.cisco.com/en/US/products/ps6599/products_white_paper09186a00800a4409.shtml

[5] "Understanding Asmmetric Routing – Riverbed" [Online] Available: https://www.google.com/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#

[6] "Differentiated Services Code Point" [Online] Available: http://en.wikipedia.org/wiki/Differentiated_services

[7] "Quality of Service" [Online] Available: http://en.wikipedia.org/wiki/Quality_of_service

[8] "Openflow Specification" [Online] Available https://openflow.stanford.edu/display/ONL/POX+Wiki

[9] "SimpleL2Learning : Controller available as part of GENI Tutorial" [Online] Available http://groups.geni.net/geni/wiki/GENIExperimenter/Tutorials/OpenFlowOVS/Execute

[10] "Mininet : An instant virtual network on your laptop" [Online] Available http://mininet.org/

[11] "Equal Cost Multipath Routing (ECMP)" [Online] Available http://en.wikipedia.org/wiki/Equal-cost_multi-path_routing