SDN Implementation of Multipath Discovery to Improve Network Performance in Distributed Storage Systems

Luis Guillen*, Satoru Izumi*, Toru Abe*†, Takuo Suganuma*† and Hiroaki Muraoka‡
*Graduate School of Information Sciences, Tohoku University, Sendai, Japan
†Cyberscience Center, Tohoku University, Sendai, Japan
‡Research Institute of Electrical Communication, Tohoku University, Sendai, Japan
Email: *{lguillen,izumi}@ci.cc.tohoku.ac.jp, †{beto,suganuma}@tohoku.ac.jp, ‡muraoka@riec.tohoku.ac.jp

***Abstract*—The use of Distributed Storage Systems (DSS) has considerably increased in the past years, alongside the need for effective data transfer from storage to storage. Although current network infrastructure can reliably handle large amounts of traffic, networking techniques have not changed for several years, leading to an under-use of resources, i.e. most routing solutions still use single-path routing. In this paper, we present a pragmatic approach for multipath routing in DSS, which is based on Software Defined Networking (SDN) that uses parallel links at the edge-side. Path discovery is calculated by finding the k-maximum disjoint paths in a multigraph. Preliminary results show that, by using our multipath solution, not only the overall throughput increases but also the efficiency of resources usage.**

***Index Terms*—Software Defined Networking, Distributed Storage Systems, Multipath Routing**

I. Introduction

Distributed Storage Systems (DSS) provide reliable services by networking together copies of data in different nodes, called replicas [1]. Replicas are by nature unreliable since they are unexpensive, prone-to-fail devices; therefore, DSS need to ensure continuity by a fast recovery mechanism. These mechanisms are varied, from costly solutions that keep exact copies in several replicas, to more efficient solutions that fragment data and store them separately [2]. Independent of the mechanism used, at network level, more traffic needs to be handled, added to the inherent complexity of ensuring data-consistency. Currently, network infrastructure is capable of delivering DSS resilience by providing a highly redundant topology, as the one shown in Figure 1, which represents a commonly used topology in data-centers. In this scenario, end-to-end traffic is usually managed by a technique called Equal Cost Multipath (ECMP), which balances packets among a limited number of paths, but ECMP has two major flaw; namely, it is not efficient in terms of resource usage, and does not provide flexible management.

In this context, the following problems were identified:

(P1) *Last-mile bottleneck:* Usually core links use high–speed connections, but at the lower tiers, i.e. aggregation and edge, connection speed is limited and therefore creates a bottleneck to the whole process that we call *last-mile bottleneck*.

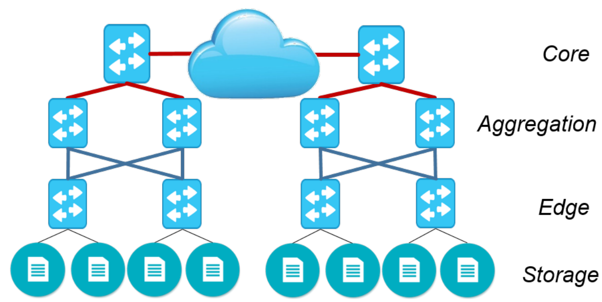


Fig. 1: Fat-tree based topology used in data centers.

(P2) *End-to-end routing:* Traditional networking calculates end-to-end paths in a node-by-node basis, and takes a long time to converge. Moreover, it is limited by legacy protocols such as Spanning Tree Protocol (STP), which prunes redundant branches. Therefore, in spite of the available redundant links, paths along end-to-end nodes are predominantly calculated using single-path approaches.

Software Defined Networking (SDN) changes the way in which networks are managed, since it allows enhanced network programmability by decoupling the control plane from the data (forwarding) plane [3], which makes it ideal for administering traffic in scenarios where dynamic and efficient calculations determine the overall performance, as it is the case with DSS. Moreover, its centralized approach provides an overview of the whole underlying network, allowing more flexible and robust network control.

In this paper, we propose an SDN based control method for path discovery in multipath DSS. At first, we conceptualize the network problem in Section 2 and introduce the solution to *last-mile bottleneck* using a pragmatic approach in Section 3. In Section 4, we implement the approach and find *k-path* candidates which can later be used for more advanced functions, i.e. load balancing. Finally, we evaluate the proposed method in two topologies in Section V.

978-3-901882-98-2 © 2017 IFIP

II. Related Work and Problem Definition

A. Related Work on SDN multipath routing

Multipath routing based on SDN solutions have already been well studied, especially in wired networks. Izumi et al. [6], for instance, proposed dynamic multipath routing to enhance performance by using parallel data transmission in case of disasters; they calculate link availability based on bandwidth and a risk index, however, the discovery process is still hop-by-hop based. Subedi et al. [7] presented Adaptive Multipath Routing (ARM), which is capable of proactively adapting to network changes based on link capacity and latency, but their approach heavily relies on Link Layer Discovery Protocol (LLDP) as the discovery mechanism. Banfi et al. [8] performed multipath packet forwarding for aggregated bandwidth; the results obtained were promising, however, altought their discovery mechanism claimed to either use LLDP or Layer-3 Ad-hoc deployment, the latter was not fully implemented. Finally, Dinh et al. [9] presented a multipath forwarding architecture, which dynamically selects the best path among *k-paths* for traffic engineering; however, the criteria for the selecting the number of paths is unclear and might not satisfy basic constraints.

B. Network Problem Definition

In this section, we formalize our target network problem. Given a graph $G = (V, E)$, where V is the set of vertices and E the set of edges that connect vertices. The Shortest Path (SP) between two nodes is the one with the least path weight. A path is disjoint (DP) when, depending the requirements, no common edges or vertices are shared [4]. When fully DPs are not possible, finding Maximally Disjoint Paths (MDP) are preferred, wherein it is allowed for paths to share common edges or vertices, as long as the number is minimum. Additional constraints might also apply, e.g. the sum of the weights of all the constituent edges in a path is minimized, also known as the *max-min condition*. Li et al. [5] proved that such condition is strongly *NP-complete*, except in directed acyclic networks, where it is (weakly) NP-complete. Moreover, according to Iqbal and Kuipers [4], for a network to have k-disjoint paths between two nodes, it need to fulfill the following two conditions:

(C1) Both end nodes must have at least k-node degree (i.e. k neighbours); and
(C2) There must also exist enough nodes and edges such that the k-disjoint paths are possible.

III. Proposal of SDN based control method for path discovery

The overall scheme of the proposal is depicted in Figure 2. To solve the MDP problem and fulfill condition (C1), the proposal is to use *k* parallel edges at the lowest tier (edge). In terms of network devices, this represents parallel links between the edge device and the next-hop. The intuition behind is that there is still some control over this segment of the network, hence, *k* links can be added to ensure at least a k-node degree

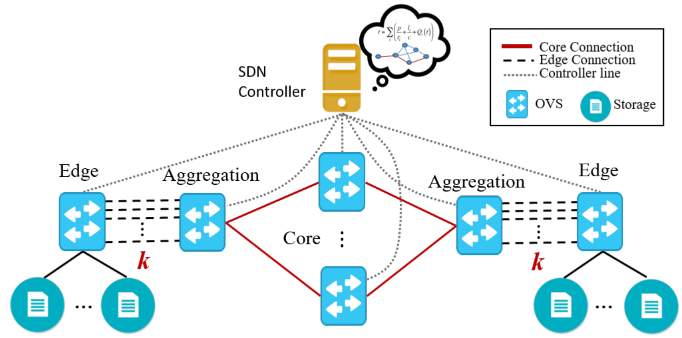


Fig. 2: Proposed scheme for multipath DSS.

(C1) and at the same time increment the number of edges in the, already highly redundant, DSS topology (C2). In case of a two-tier topology (no aggregation) a *commodity network device* can be added in-between so that both conditions are also fulfilled.

This idea, which is central to our approach, ensures that the discovery mechanism successfully finds the k-paths and can also help to solve the last-mile bottleneck problem, as traffic are distributed among several paths and consequently achieve higher aggregated throughput.

Initially k is set as the number of links connected from the edge to the aggregation switch. Then, weights are provided to edges, which can be assigned based on different criteria; e.g. available bandwidth, probability of failure, transmission delay or a combination of them.

Once the weights are set, the MDP problem is solved by running the *selectKPaths* function shown in Algorithm 1, which is based on Suurballe algorithm [10] for a Directed Acyclic Multigraph. Dijkstra shortest path is calculated k-times to find k-disjoint paths, modifying the weights after each path is discovered.

It is a simple approach that provides k candidates, whose upper-bound runtime complexity is approximately k-times the runtime complexity of Dijkstra's algorithm using min heaps.

IV. Implementation

The proposed scheme was implemented using OpenDaylight (ODL) and Java to program the $selectkPaths$ function. Once the k-paths are calculated by the application, OpenFlow Switch matching rules are composed and installed in the corresponding switches. Without any loss of generality, for simplicity unit-weights are assigned to each edge, however, as previously mentioned, weights can have more fine-grained values.

V. Evaluation

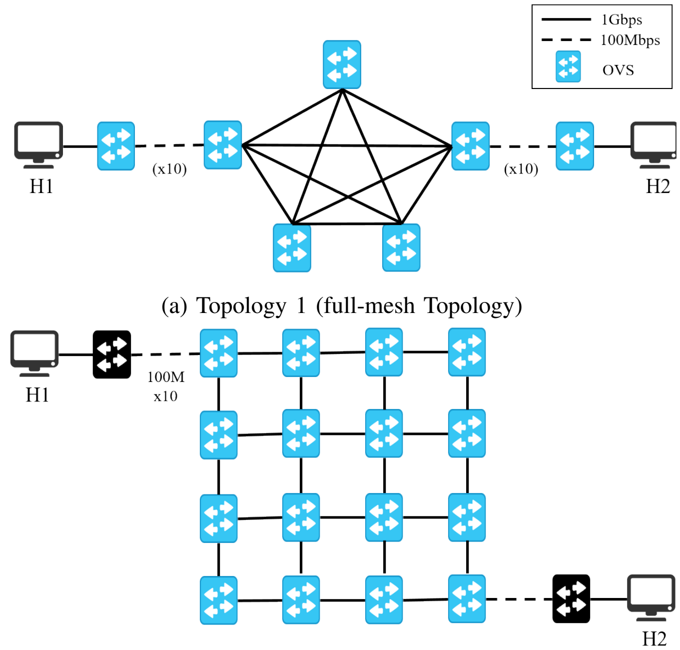
A. Overview

To evaluate the behavior of the proposed approach, we tested the implementation on two contrived topologies, namely: *full-mesh*, as an example of a highly redundant environment; and *grid topology*, as an example of common topologies found in current network infrastructures, as shown in Figure 3a and Figure 3b respectively. All OF-switches are

Algorithm 1: Path selection algorithm to find the k-max disjoint paths from source to destination

function selectKPaths (s, t, k, G);
Input : $s, t, k, G(V, E)$
Output: Set of k–max disjoint shortest paths P

$P \leftarrow \emptyset$;
$currentPath \leftarrow \emptyset$;
$nPath \leftarrow 1$;
do
    if $nPath > 1$ **then**
        $adjustWeights(P[npath - 2])$
    end
    $currentPath \leftarrow getDijkstraShortestPath(s, t)$;
    if $currentPath \notin \emptyset$ **then**
        $nPaths$++;
        $P.add(currentPath)$;
    end
while $currentPath \notin \emptyset$ *and* $nPaths \leq k$;
return P



(a) Topology 1 (full-mesh Topology)

(b) Topology 2 (grid Topology)

Fig. 3: Experimental Network Topologies

connected to a single SDN Controller. Note that in both cases, there are k=10 links connecting the edge with the next-hop switches. Additionally, in Figure 3b the edge switches, painted in dark, are commodity switches added to the main topology to fulfill the conditions in the proposal. The results were compared with the default single path OSPF-based provided by the controller.

B. Experimental Environment

The testbed of both topologies was deployed in a simulated environment using mininet v2.2.2; ODL Lithium SR2 as the SDN controller; and iperf as the throughput measurement tool. The experiments were conducted using a virtual machine running 64-bit Ubuntu 16. 04LTS, with 4 Gb of RAM, hosting ODL and Mininet. In both topologies the procedure was as follows: Initially, L2Switch and STP provided by ODL select the best path from H1 to H2. Then, once the paths were setup, 10 simultaneous iperf request were sent from H1 to 10 different ports in H2 for 100 seconds, using the default values. Finally, the *selectKPaths* function is applied, and re-run the iperf request with the same parameters as in the single path test.

The throughput obtained was recorded in both TCP and UDP for each path, and finally, the used routes were traced in both cases and in both topologies.

C. Simulation Results

Figure 4 shows the aggregated throughput obtained in the full-mesh and grid topology in the TCP and UDP tests. As can be observed, in both cases, throughput using our multipath approach outperforms the default single-path approach provided by the controller. In the case of the full-mesh topology, the aggregated throughput during the TCP test reached around 960 Mbps compared to the single-path that only reached around 96 Mbps, since only 1 of the 10 available links were used. Similar results were obtained in case of the grid topology. Conversely, in case of the UDP test, aggregated throughput in the iperf log, a higher raw throughput is seen for the case of single path. A possible explanation for this phenomenon is that iperf only reported the intended traffic sent from the origin, ignoring the packet-loss rate.

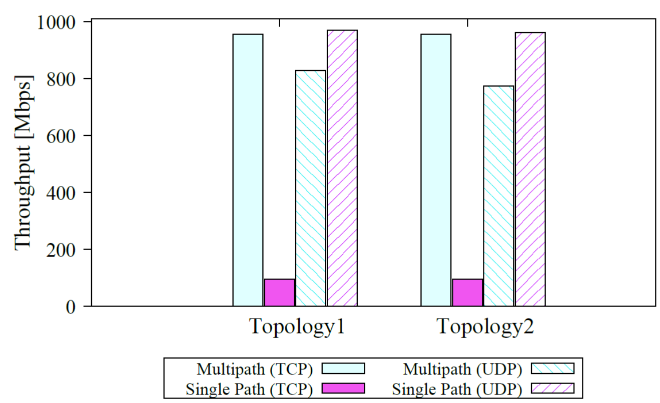


Fig. 4: Aggregated Throughput.

Table I presents the effective amount of data transferred after 100 seconds of the iperf test. Our multipath approach was able to transfer on average 987 and 925 Mbytes in Topology 1 and 2 respectively, whereas in the case of single-path it could only transferred around 110 Mbytes on average in both topologies. In terms of the overall jitter (the delay packets experience due to congestion), in our multipath approach, it was only around 0.25ms, while in the case of single-path, on average 7.5ms–

TABLE I: Average amount of Data Sent and jitter

Topology #	Data Sent[Megabytes]		Jitter [ms]	
	Multipath	***Single-path***	***Multipath***	***Single-path***
1	986.7	113	0.226	7.426
2	924.6	114	0.278	6.330

TABLE II: Average Number (#) of Datagrams Sent

Topology #	#of Datagrams Sent		Loss Percentage [%]	
	Multipath	***Single-path***	***Multipath***	***Single-path***
1	704,039	825,752	0.058	90.0
2	659,560	810,507	0.054	90.0

which is considerably higher. This is, of course, a result of packet collisions in the shared links.

Table II, presents datagram information of both topologies. It is worth noting that the number of lost datagrams is significantly higher in the case of the single-path, even though the number datagrams is greater. This leads to an average of 90% of packets lost, compared to 0.06% in our approach.

Finally, in terms of effective resources usage, the number of paths used in both topologies was computed. In the case of Topology 1 (full-mesh) the percentage of links used in single-path reached a maximum of 16%, whereas with our approach we reached around 90% utilization. In Topology 2 (grid) the single-path solution only used around 20% of the edges whereas with the proposed approach 100% of the links were used. Of course, not all the links were used to the same extend; for example, Figure 5 shows a graphic interface, wherein links in Topology 2 are colored based on the degree of utilization; as observed, independent of the the links at the source and destination nodes, the majority of the traffic was equally distributed.

D. Discussion

The main advantage of using an SDN approach is that controllers usually include control-plane functionalities such as: topology discovery, route selection and path failover mechanisms [11] that can be used as a starting point. However, these functionalities are not designed to handle multipath solutions or highly redundant topologies, hence, redundant and parallel links are pruned by STP.

It is worth noting that using the available infrastructure efficiently would greatly contribute data transfer in DSS, however, the proposed approach might not be very useful for short transmissions, where a single path suffices to cover the requirements, as would be the case if the topology does not satisfy constraints C1 and C2.

The idea of using several parallel links or even adding intermediate switches is not conventional, but given the obtained results is a viable alternative to traditional single-path approaches or other networking techniques.

VI. Conclusion

In this paper, we presented a high-level description of an SDN-based network management solution for data transmission in DSS, which uses k parallel links at the edge side, to allow multipath routing. Initially we calculate the k-max disjoint paths, and based on that initial assignment the traffic is distributed among those paths. Preliminary results applied to common network topologies shown that the proposal outperforms the single-path approach implemented by default in SDN controllers, improving the overall throughput as well as resource utilization.

As future work, we plan to improve the proposal by implementing adaptive load balancing among the initially discovered k-paths.

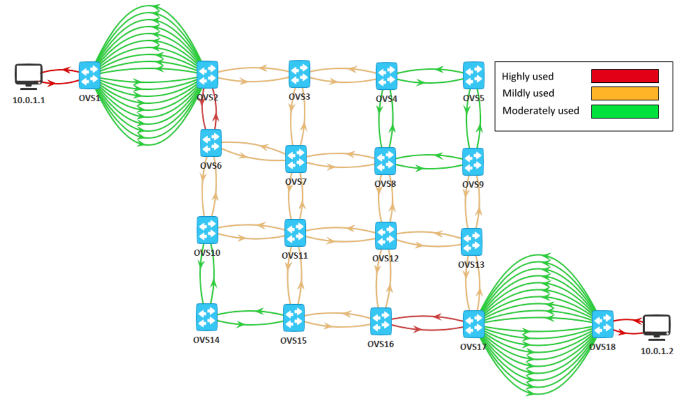


Fig. 5: Link usage in topology 2.

References

[1] Y. Liu, N. Rameshan,E. Monte,V. Vlassov, and L. Navarro, “ProRenaTa: Proactive and Reactive Tuning to Scale a Distributed Storage System,” In *15th IEEE/ACM ISCCGC*, pp. 453–464, 2015.

[2] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran, “Network Coding for Distributed Storage Systems,” in IEEE Transactions on Information Theory, 56, pp. 4539–4551, 2010.

[3] W. Xia, Y. Wen, C. H. Foh, D. Niyato and H. Xie, “A Survey on Software-Defined Networking,” in *IEEE Communications Surveys Tutorials*, 17(1), pp. 27–51, 2015.

[4] F. Iqbal, and F.A. Kuipers, “Disjoint Paths in Networks.” in *Wiley Encyclopedia of Electrical and Electronics Engineering*. Wiley & Sons, Inc., 1999.

[5] C.L. Li, S. T. McCormick, and D. Simchi-Levi, “The complexity of finding two disjoint paths with min-max objective function,” in *Discrete Applied Mathematics*, 26(1), pp. 105–115, 1990.

[6] S. Izumi, M. Hata , H. Takahira, M. Soylu, A. Edo,T. Abe and T. Suganuma. “A Proposal of SDN Based Disaster-Aware Smart Routing for Highly-available Information Storage Systems and Its Evaluation,” International Journal of Software Science and Computational Intelligence, 9(1), pp.68–82, 2017.

[7] T. N. Subedi, K. K. Nguyen and M. Cheriet. “OpenFlow-based in-network layer-2 adaptive multipath aggregation in data centers,” Computer Communications, 61(1), pp.58–69, 2015.

[8] D. Banfi, O. Mehani, G. Jourjon, L. Schwaighofer, and R. Holz, “Endpoint-transparent Multipath Transport with Software-defined Networks,” in Proc. of *LCN2016, pp.307–315, 2016.*

[9] K. T. Dinh, S. Kukliński, W. Kujawa, M. Ulaski. “MSDN-TE: Multipath Based Traffic Engineering for SDN,” in Proc. 8th Asian Conference, ACIIDS 2016*, pp. 630–639, 2016.*

[10] J. W. Suurballe, and R. E. Tarjan. “A quick method for finding shortest pairs of disjoint paths,” in Networks*, 14(2),pp. 325–336, 1984.*

[11] RFC7426 S. Denazis, E. Haleplidis, J.S. Salim, O. Koufopavlou, D. Meyer, and K. Pentikousis. “Software-Defined Networking (SDN): Layers and Architecture Terminology.” in: IETF RFC 7426 (Informational Document)*, 2015.*