# Draft

jpduan*

***Abstract*—Datacenter network is known for its high burstiness and irregularity of its traffic pattern. Apart from this, two conflicting goals, providing high throughput for large flows and low latency for small flows, have always harassed the performance of the datacenter network, causing low network utilization. The SDN paradigm alleviates the performance bottleneck of the datacenter network by handing the control of the traffic admission and routing to a centralized controller, which knows the global information and coordinates the competing traffic. In this paper, we aim to address the inherit network utilization problem of the datacenter network using the powerful SDN scheme combined with end-host MPTCP transmission. Our proposed system comprises of two parts. First, we designed a intermediate layer at the end host that inspects the MPTCP subflows. This layer supports dynamically increasing of the number of subflows in use. Second, we designed a centralized SDN controller that distribute the flow traffic evenly on links between each layer in the multi-rooted tree. We conduct comprehensive experiment to evaluate the performance of our proposed system.**

## I. Introduction

Improving the link utilization rate in the datacenter network has been a critical problem bothering researchers and network administrators. A typical datacenter generates tens of thousands of flows every second, with highly random traffic patterns. Flows with varying transmission goals, either low latency or high throughput, coexists in the datacenter. Even though today's fat-tree based datacenter topology provides redundant paths between any pairs of end-hosts, without proper routing and scheduling, flows may collide with each other on their paths, resulting in low throughput for large flows and increased latency for small flows. Traditional routing solution, such as ECMP(Equal Cost Multiple Path), selects one of the shortest paths for a flow according to the hash of the flow's five tuple. However, ECMP causes many flow collision and unbalanced links because it does not take the global network information into consideration.

*Software Defined Network* provides a promising solution to the existing datacenter network problem. By separating the data plane from the control plane and giving the centralized controller a full control over the entire network, SDN could make routing and scheduling decisions based on the global information to achieve better throughput and link utilization. Hedera [1]is such a centralized SDN controller that schedules large flows inside a datacenter. Hedera detects large flows at switch, estimates the flow rates and dynamically schedules large flows to less loaded path. However, the 500ms scheduling interval used by Hedera is too optimistic. Considering the incredibly large flow arrival rate in production datacenter network, Hedera will suffer from great scalability problem.

To overcome the scalability issue caused by the centralized controller, Sen *et al.* [2] proposes to use a switch local solution to route flows. In [2], each switch has its own controller and spread flows to the same destination evenly among all the out-going links. Flows are split when needed. This approach achieves optimal performance by solving max-commodity flow problem in a distributed fashion, but the overhead of assigning each switch a controller is too high. Dard [3] solves the scalability issue by handing the controller's work to the end-host. It assigns multiple IP addresses to each end host. End host estimates the flow rate and dynamically changes the flow path by changing the sending IP address. However, end host in Dard needs to send query messages to switches for path information. This querying process needs careful coordination among the switches. It is sensitive to fault and the latency of such a process may be unpredictable.

MPTCP[4] provides a good alternative method to improve the datacenter performance by using MPTCP to transmit large flows. Each MPTCP connection consists of multiple subflows. Each subflow takes an independent shortest path based on ECMP hashing. MPTCP can shift the traffic from more congested subpath to less congested one. By randomly choosing path for each subflow, MPTCP achieves good throughput without the coordination of the centralized controller. The author in [4] reports that it takes about 8 subflows per MPTCP connection to reach desirable throughput.

Apparently, if all the shortest paths between two end host could be used by MPTCP, then MPTCP resembles the packet spraying method described in [5] and theoretically achieves the optimal throughput. However, adding too many subflows increases the management overhead of the MPTCP protocol stack and it takes more time for the receiving end to recover the data. Cao *et al.* [6] shows that by using a modified MPTCP protocol, they managed to achieve a high throughput with only 4 subflows. So MPTCP can survive with fewer number of subflows. However, decreasing the number of the subflows means increasing the probability that individual subflow will encounter congestion. Just like the case for single path TCP, the downstream collision caused by ECMP hashing for MPTCP with fewer subflows needs to be considered again. This tradeoff could be solved in SDN environment, where we could decide the path for each subflow and schedule the subflow.

Another inherit problem of MPTCP is that it can not shift its traffic from more congested path to less congested one [6]. When using MPTCP, we expect that the traffic shifting acts as a substitute for flow scheduling, so the performance of traffic shifting is critical to the overall performance of the datacenter network. However, MPTCP achieves traffic shifting through the coupling of the congestion control of different subflows. Traffic shifting happens when end hosts detects congestion signals or acknowledgement packets from the network. There is a substantial latency of this distributed process. If end host can acquire exact path information as a feedback, then it can react immediately to traffic shifting. In a SDN network, this special feed back could be obtained from the controller.

In this paper, we further explore MPTCP use case in *software defined datacenter network* to improve the datacenter performance. Our design is based on three criteria:

*First*, a centralized controller is in charge of controlling the flow setup process and informing the end host about the network condition. This controller should have good scalability, so it should not be involved in the heavy task of re-computing and re-routing of established flows.

*Second*, flows are categorized into mice flows, which require low latency, and elephant flows, which require high throughput. Mice flows are transmitted through single path TCP while elephant flows are transmitted through MPTCP. And applications should use unified interface for flow transmission without specifying which category the flow belongs to.

*Third*, the number of the subflows used by a MPTCP connection could dynamically change and should not be too large in order to reduce the protocol overhead.

Based on the above criteria, we come up with a datacenter network system that digs out the potential of MPTCP and SDN. Our contribution are two fold:

*First*, on the controller side, we developed a datacenter network controller. It uses simple algorithm to select a suitable path for each subflow. It queries the switch for the network condition and periodically multicasts the network condition to the end host.

*Second*, on the receiver side, we add a intermediate layer that monitors the transmission condition of each MPTCP connection. It will notify the MPTCP to add more subflows when needed. Upon the reception of the multicast information from the controller, the shim layer triggers the MPTCP to shift traffic according to the acquired network condition.

We present the design details in following sections.

## II. System Model

There are several disadvantages to use too many subflow per MPTCP connection. *First*, too many subflows increase the management cost of the MPTCP protocol stack. The MPTCP sender uses a packet scheduler to inject the packet to different subflows for transmitting, which means that the packets will arrive at the receiver out of order. The receiver needs to maintain a large buffer in order to recover the received data sequence and the buffer increases as more subflows are established. *Second*, considering using MPTCP in a software defined datacenter. If we want to do fine-grained control over the path of the subflow, then we need to use exact match to route each subflow. The more subflows we use, the more rule space we consume in the switch. Considering the limited rule space in the OpenFlow switch, achieving good performance with less subflows established is favorable.

In order to achieving a better performance without incurring the above mentioned problems, we design a combined system to dig out the full potential of MPTCP with assistance from the SDN controller. The system is capable of dynamically changing the number of subflows that a MPTCP connection use, achieving good over all throughput with less subflows.

Our controller is a specially designed reactive controller. It handles the MPTCP connection setup and subflow setup by selecting one of the least loaded path from all the available shortest paths. It also maintains a database recording the MPTCP subflow status. An active thread running in the controller is responsible for gathering the subflow statistics of a MPTCP connection and the statistics of the link from the aggregation switch. When the statistic gathering is finished for one MPTCP connection. The controller thread will forward all the information to the server that initiate this MPTCP connection.

On the server side, an active user space process listens to the controller. Whenever it receives the path condition information, it will alert the kernel using user space to kernel communication. The underlying MPTCP protocol stack will react to the path condition information and make decisions on whether to create new subflows and how to adjust the congestion control schemes to meet the path condition.

## References

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, 2010, pp. 19–19.

[2] S. Sen, D. Shue, S. Ihm, and M. J. Freedman, "Scalable, optimal flow routing in datacenters via local link balancing," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 151–162.

[3] X. Wu and X. Yang, "Dard: Distributed adaptive routing for datacenter networks," in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on*. IEEE, 2012, pp. 32–41.

[4] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 266–277.

[5] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *INFOCOM, 2013 Proceedings IEEE*. IEEE, 2013, pp. 2130–2138.

[6] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies (CoNEXT 2013)*, 2013, pp. 73–84.