Ingénierie des Systèmes d'Information
Vol. 30, No. 6, June, 2025, pp. 1523-1534
Journal homepage: http://iieta.org/journals/isi

MaxFlowSDN: SDN-Based Maximum Flow Routing for High-Throughput Data Center Networks

Maazen Alsabaan*, Mohammed J.F. Alenazi, Norah S. Bin Saeed, Abdulrahman Almutari

Department of Computer Engineering, College of Computer and Information Sciences (CCIS), King Saud University, Riyadh 11451, Saudi Arabia

Corresponding Author Email: malsabaan@ksu.edu.sa

https://doi.org/10.18280/isi.300611

Received: 3 March 2025
Revised: 4 June 2025
Accepted: 24 June 2025
Available online: 30 June 2025

Keywords:
MPTCP, maximum flow, SDN, data center, OpenFlow, multipath

ABSTRACT

Large data centers continuously move data from one server to another due to up- or down-scaling virtual server specifications to meet user requirements. Most data center topologies allow multipath routing between any pair of nodes within their network to increase throughput, as well as resilience against link failures. Several approaches have been proposed and developed to utilize these routes to improve network performance. Existing methods often face challenges in achieving maximum throughput across diverse topologies without requiring kernel modifications, leaving room for improvement in practicality and scalability. In this paper, we propose a novel system, MaxFlowSDN, which uses the maximum flow algorithm along with traditional SDN and TCP to deliver higher throughput in data centers. MaxFlowSDN yielded 80% higher throughput in the Fat-Tree topology compared to StandardTCP, ParallelTCP, and MPTCP. In DCell and BCube topologies, it achieved approximately 190% higher throughput than StandardTCP and nearly 50% improvement over ParallelTCP and MPTCP. For evaluation, we deployed our system in different data center topologies and compared our results against existing methods. These results demonstrate that MaxFlowSDN provides maximum flow throughput in the data center environment while addressing the limitations of current approaches.

1. INTRODUCTION

Data centers have a crucial role to play in the computing landscape of any organization. In most recent years, the number of existing data centers has observed a rapid growth. The latest forecast study showed that the total number of data center sites will increase to 3.6 million by 2023 [1]. Furthermore, data centers are also continuously growing in size in terms of the number of servers they host. Such growth has been fueled by an increased demand for diverse facilities provided by these servers and their affiliate resources. The number of hyper-scale data centers was expected to increase to 628 by 2021 [2]. To adequately meet the expected performance, the incrementally growing data centers would need higher network bandwidth as well as more integrated resources. It was expected that traffic within data centers would quintuple by the end of 2021 [2].

Both large and small organizations of big and small sizes need to capture, clean, park, analyze, and use a massive amount of data for data supply chain and production purposes. These processes need essential components of a complex system, including storage and computing resources. Due to these dependencies, a data center remains a vital asset for day-to-day operations. To adequately handle such Big Data hosted in data centers (such as shown in Figure 1), a system requires huge throughput to avoid the degradation of global or local network performance.

Data centers have a set of large-scale data requests that incur a vast number of policies for traffic management transactions per second, which makes it difficult for network administrators to monitor and manage the network [3]. A software-defined network (SDN) [4] is a widely used technology in data centers to customize routing and manage traffic, with a view to achieve more scalability in a network [5]. The structure of an SDN segregates data and control planes. SDNs are applied in diverse ranges of network applications – one of these being Data Centers [6]. Many world-class companies, such as Microsoft [7] and Google [8], have adopted SDNs as a way to provide needed traffic management and throughput for their local- and wide-area networks.

The SDN system itself is widely adopted thanks to its abstraction, management flexibility, and network virtualization capability. A single interface is used by Standard TCP to establish a connection with other networked devices. However, most networked devices have at least two network interfaces; consequently, Parallel TCP utilizes this by creating one TCP connection for each interface. Multipath TCP (MPTCP) [9] was introduced to use other network interfaces and to improve resilience against interface failures. In Data Centers, MPTCP creates multiple subflows between any pair of servers to engage the full bisection bandwidth for techniques such as BCube and dual-homed Fat-Tree [10, 11]. Since its introduction, MPTCP performance has been evaluated in several environments and applications, including

cellular networks, data centers, home networks, and enterprise networks. In fact, MPTCP is supported by some of the largest IT companies in the world, like Apple [12, 13].

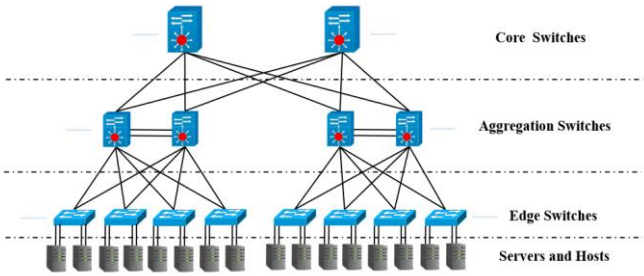

Figure 1. Typical data center architecture

Our contributions in this paper:

1. We introduce MaxFlowSDN, a novel system that integrates SDN with the maximum flow algorithm to enable multipath routing without requiring kernel modifications.
2. Our system dynamically selects multiple paths between source and destination, optimizing throughput based on real-time link capacities and network conditions.
3. We evaluate MaxFlowSDN on multiple data center topologies, including Fat-Tree, DCell, and BCube, demonstrating its superior performance compared to StandardTCP, ParallelTCP, and MPTCP.

The rest of the paper is structured as follows. Background for the proposed methodology is provided in Section 2. This background mainly comprises an introduction to the Software Defined Network, MPTCP, and Maximum Flow algorithms. Related Works are presented in Section 3, whereas the mechanics of the Proposed Methodology (i.e., MaxFlowSDN) are introduced in Section 4. Section 5 presents the Experiment Setup and Performance Metrics used in our paper. Section 6 shows the Evaluation of the results and a Discussion of them. In the final section, i.e., Section 7, the Conclusion and possible remedies and inclusions in Future Work are presented.

2. BACKGROUND

An overview of the architectures of data centers, such as MPTCP, SDN, and maximum Flow problems, is presented in this section.

2.1 Data centers overview and architectures

A data center is an infrastructure element composed of connected computing resources that provide any organization or institution with data storage, applications, and services needed for their essential operations [14]. The data center network is the major part of the design of a data center [15]. Many topologies have been used to meet evolving data center requirements [16]. The following are the most popular and most studied topologies of data center networks.

2.1.1 Fat-Tree topology

The Fat-tree is a multi-rooted tree topology whose roots serve as core switches. In addition, there is an aggregation layer between roots and access switches. Fat-Tree has identical bandwidth at any bisection, and uses a large number of inexpensive switches to allow the deployment of a large number of hosts at low cost. It uses complex routing configurations in switches to prevent creating loops while using the available paths for load balancing [14]. The Fat-Tree topology is depicted in Figure 2.

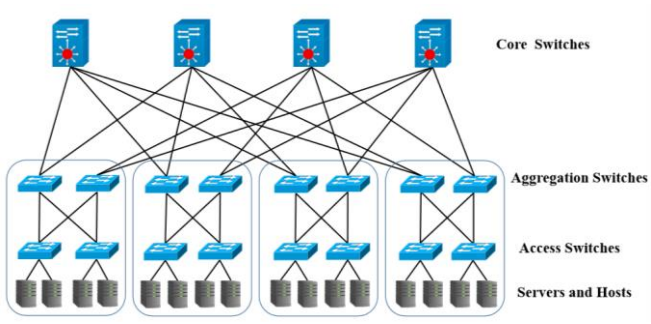

Figure 2. Fat-Tree topology

2.1.2 Leaf-Spine topology

The Leaf-Spine topology consists of two layers of switches: leaf switches attached to servers, and spine switches connected to all leaf switches. The capacity of the cables between leaf switches and servers is different from those between leaf and spine switches. A routing suite with load balancing can be used in the Leaf-Spine topology without causing loops [14]. Figure 3 shows the Leaf-Spine topology.

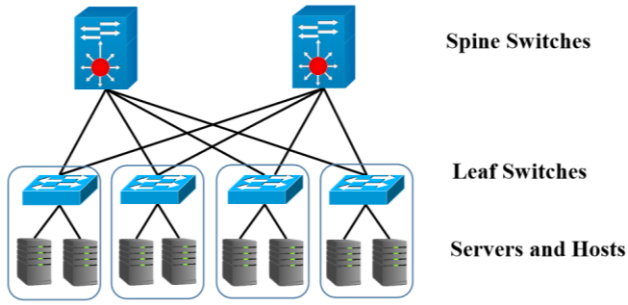

Figure 3. Leaf-Spine topology

2.1.3 VL2 topology

VL2 is similar to Fat-Tree and runs the routing suite on a multi-rooted tree topology. However, it differs from the Fat-Tree topology in terms of link capacities. The links between switch layers have a higher capacity than the links between servers and switches. This results in fewer cables between aggregation and core layers [14]. The VL2 topology is shown in Figure 4.

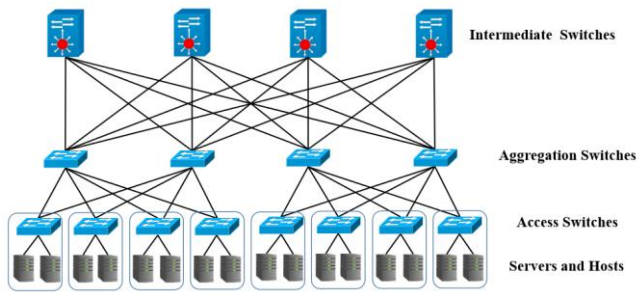

Figure 4. VL2 topology

2.1.4 DCell topology

DCell is a hierarchical data center topology, where the

lower DCell structure level is the building block of the whole system. Higher levels of cells are formed by combining many lower-level DCells together. This topology is highly expandable simply by adding more levels, and has no single failure point. It uses a fault-tolerant, custom routing algorithm, that aims for the shortest routing path [14, 15]. DCell topology is shown in Figure 5.

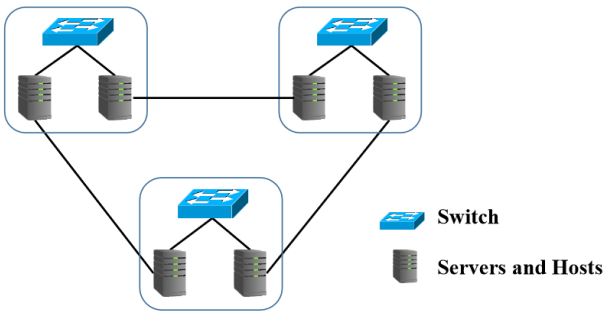

Figure 5. DCell topology

Level 2 DCell is shown in Figure 6.

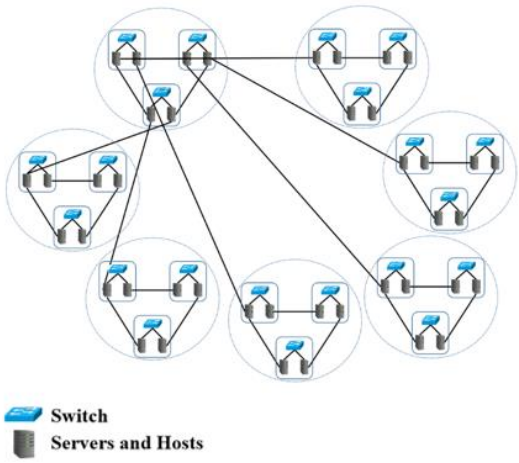

Figure 6. DCell2 topology

2.1.5 BCube topology

BCube is a network structure centered around a server. For this network design, servers which have many network ports can link to multiple switch layers. Additionally, the servers work as forwarding nodes for other servers. The BCube network structure uses source routing and must change the protocol stack of server networking either in hardware or in software [14, 16]. BCube topology is shown in Figure 7.

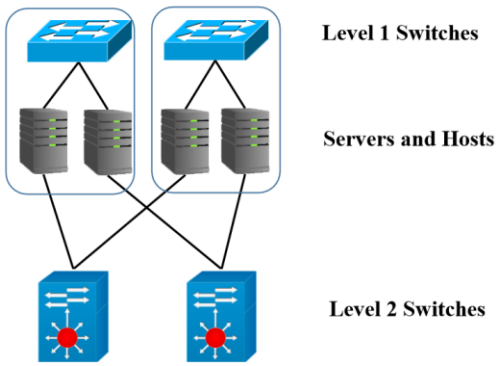

Figure 7. BCube topology

2.2 Data centers overview and architectures

SDN is a network technology which uses the concept of programmable network, simplifying network operations and making it more scalable and adaptable. The main idea is to decouple the control and data plane by using a control console that handles all decision-making tasks (control plane). Switches and routers in the network will only be packet forwarding units (data plane) that may be configured and programmed through an open interface (e.g., OpenFlow17). In contrast, for traditional networks, each individual device needs to be configured, to independently make its traffic forwarding decisions [3, 4].

Figure 8 represents the prominent differences between the software-defined and conventional networks.

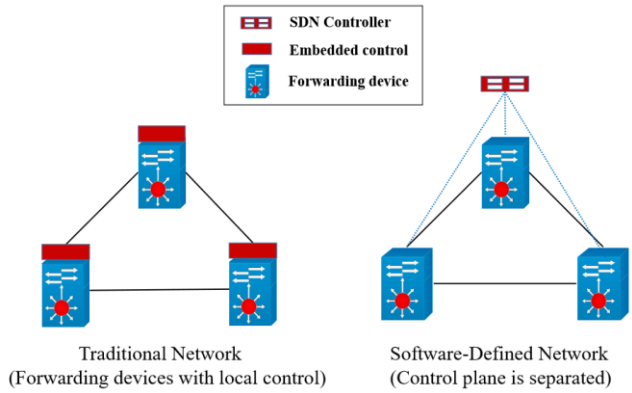

Figure 8. Structure of SDN compared to conventional network

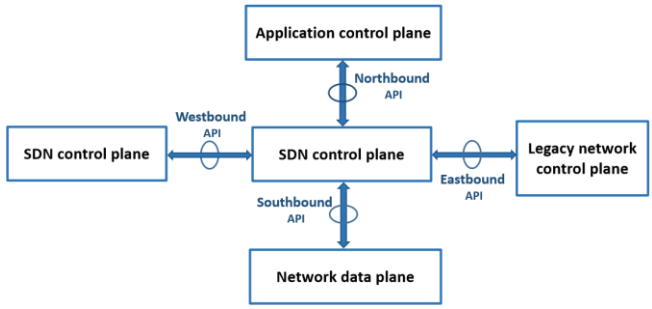

Figure 9. SDN Architecture

Software defined architecture consists of four main interfaces, as shown in Figure 9:

• Southbound-API – The Southbound API acts as interface between controller and forwarding devices (data plane). The OpenFlow protocol [17] is a widely used southbound interface that describes the exchange of information between data plane devices and SDN controller.

• Northbound-API – The Northbound API describes the interfaces between SDN controller and application control plane containing applications running on top of the network. There is no standardized Northbound-API. This interface allows application programmers and developers to administer the network using applications.

• Westbound-API – The Westbound-API represents the interface between different SDN controllers in various network domains. It enables the transfer of network state data used in making routing decisions in every controller by providing the global network view of other domains.

• Eastbound-API – The Eastbound API describes the interface between SDN controller and legacy network control plane in order for the SDN domain to be fully compatible with other (non-SDN) domains, including the routing protocol deployed [18, 19].

2.3 Multipath TCP

Certain limitations exist between communicating pairs when using the standard TCP in the context of multiple interfaces. This is due to the fact that only a single interface is used at each communication end, whereas MPTCP can take advantage of more than one interface by establishing many parallel subflow connections among communicating peers. Parallel connections open a path for increasing resource usage and redundancy, and to enhance overall throughput. At the same time, the socket implementation at the MPTCP transport layer is the same as for standard TCP. A comparison of the MPTCP and standard TCP structures is shown in Figure 10.

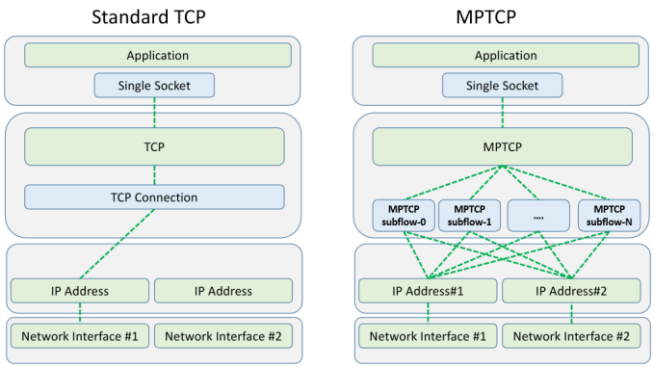

Figure 10. Standard TCP compared to MPTCP

When a single socket is created by the application to use standard TCP, a single TCP connection is created that uses only one IP address and one network interface. However, when MPTCP is enabled and a single socket is created by the application, MPTCP creates multiple subflows to make use of all available network addresses. The packet scheduler in MPTCP divides the segments, and each of these segments is transferred using subflows, which act as a standard TCP's single path [9].

Figure 11 depicts an example of the use of MPTCP in mobile devices. The device benefits from the Wi-Fi connection and cellular network connection at the same time by having one sub-flow in each connection.

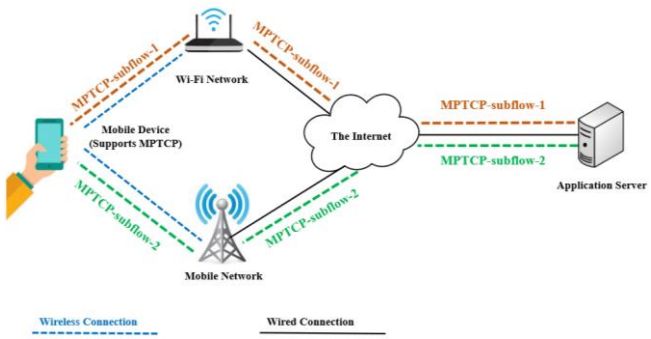

Figure 11. A typical use of MPTCP protocol in mobile devices

2.4 Maximum flow problem

Finding the highest flow value from source (S) to destination (D) node is regarded as a Maximum flow problem. This concept is crucial for almost every network, including communication networks and transportation networks [20]. The earliest method to effectively solve this problem was proposed by Ford and Fulkerson [21], and was seen as the Ford-Fulkerson algorithm. Later on, many other methodologies and algorithms were proposed, like the one proposed by Edmonds and Karp, i.e., the Shortest Augmentation Path [22], but also Dinic's method of Power Estimation [23], Karzanov's algorithm of Preflow Push [24], Goldberg and Tarjan's Push-Relabel Algorithm [25], and Goldberg and Rao's methodology of Binary Blocking Flow [26], to name a few.

3. RELATED WORKS

This section deals with related literature featuring both Software-Defined Network and Multipath TCP to boost data centers' mass data transfer capability. Several studies made use of the MPTCP to improve the throughput of congestion management and control in data center networks. For instance, fine temporal granularity of congestion control and detection for the MPTCP have been proposed to reduce latency for small flows and higher throughput for large flows [27]. Similarly, A machine learning approach has been introduced to improve multipath congestion control by enabling adaptive management of congestion across heterogeneous networks. By leveraging reinforcement learning, this approach addresses challenges like bufferbloat and suboptimal bandwidth usage, achieving improvements in throughput [28]. In addition, Pang et al. designed a queue cache balance factor to estimate the value of a sub-flow congestion window [29]. They avoided throughput collapse, achieved the load balance of MPTCP data transfer, and improved network throughput. However, while these works enhanced MPTCP's congestion control, they still depend on MPTCP's core mechanisms that require kernel modifications, limiting their deployment practicality.

Another line of research has used the MPTCP to enhance routing algorithms of a data center network. For instance, Fu et al. suggested a DQL (Deep Q-Learning) based AI strategy to produce the paths for optimal routing for data centers opting for SDN networks [30]. Moreover, Jung et al. proposed and developed distributed multiple path routing methods to reduce finish time while executing multiple jobs [31]. These studies highlight the benefits of integrating learning or distributed approaches, but do not explore SDN's potential for dynamic rule generation based on global network views.

MPTCP is used to improve the flow completion time of data center networks. For example, Jung et al. [31] and Liu et al. [32] presented a new approach to improve existing data center TCP protocols by decreasing task completion time through receiver-driven coordination. Cheng and Jia [33] suggested an improved network-aware multi-pathing scheme in SDN networks to reduce transmission time by taking heterogeneous network bandwidths into account. Zhang et al. [34] focused on how to provide deadline-sensitive services and achieve high throughput in data center networks. The proposed scheme minimizes average flow completion time by using the full available capacity of the data center network. These studies focused on latency and flow completion time, but did not

target throughput maximization using SDN with standard TCP.

In the literature, many researchers have used multi-pathing TCP to improve load balancing for data center networks. Park et al. [35] presented MaxPass, a novel adaptive load balancing system for data center networks, in which multiple paths are adaptively selected and dynamically changed based on the existing network load. An improvement of TCP-Path scalability was proposed by Alvarez-Horcajo et al. [36] to handle elephant flows for data center networks. While these works improved load balancing, they did not apply maximum flow algorithms within SDN to systematically exploit all available paths.

AlShammari and Alenazi [37, 38] proposed to improve data center performance by using SDN and graph theoretics to make use of all available paths. Their results showed significant improvement against baseline schemes, such as round-robin and least-congested. Furthermore, their methods did not explicitly apply classical network flow algorithms for optimal throughput.

While there has been considerable research on improving data center traffic using SDN and multipath routing, several important gaps remain. Many existing solutions, such as MPTCP, require kernel-level modifications, making them less practical for widespread deployment in real-world environments. Furthermore, there is a lack of solutions that effectively combine SDN programmability with classical flow algorithms, such as maximum flow, to optimize throughput without altering the transport layer protocols. Current approaches often focus on specific data center topologies, such as Fat-Tree, and do not evaluate performance across diverse architectures like DCell and BCube. Additionally, most existing systems do not dynamically adapt to real-time network conditions, such as fluctuating bandwidth, link failures, or congestion hotspots, limiting their ability to optimize traffic flows and ensure resiliency. Addressing these gaps is crucial for developing more flexible, scalable, and easily deployable solutions for data center networks.

In contrast, our work introduces MaxFlowSDN, a novel system that couples SDN with classical maximum flow algorithms to compute and apply optimal multipath routing dynamically. This approach provides kernel-agnostic, high-throughput data delivery across diverse topologies and adapts to current link conditions via SDN rule updates. Table 1 summarizes how MaxFlowSDN addresses limitations found in existing methods.

Table 1. Comparison of MaxFlowSDN with existing methods

Aspect	MPTCP & Variants	SDN Load Balancing	Graph-Theoretic SDN	MaxFlowSDN (This Work)
Transport layer changes	Requires kernel modifications	No	No	No (standard TCP)
Use of classical flow algorithms	No	No	Partial (graph heuristics)	Yes (maximum flow algorithms)
Adaptivity to bandwidth changes	Limited (congestion control only)	Dynamic paths based on load	Static or semi-dynamic	Dynamic via SDN + max flow recomputation
Topology diversity evaluated	Primarily Fat-Tree, dual-homed BCube	Fat-Tree	Limited	Fat-Tree, DCell, BCube
Deployment practicality	Reduced (due to kernel dependency)	Practical	Practical	Highly practical (no kernel changes)

4. MAXFLOWSDN SYSTEM

This section introduces a system that combines a Software-Defined Network and Maximum Flow Algorithm to enhance the flow between two servers in a data center.

4.1 MaxFlowSDN components

The MaxFlowSDN system has several components, as shown in Figure 12. These components include Topology discovery, Maximum Flow Algorithm, Residual Link Capacities, and Rule Generators.

The Topology discovery component is responsible for creating a graph that consists of nodes, i.e., SDN switches, and links that connect these switches. The Maximum Flow Algorithm component is responsible for determining paths that generate the maximum flow between two nodes given currently available link capacities. The Residual Link Capacities component monitors the topology links and identifies currently usable and available bandwidth on every link. The Rule Generators component is responsible for determining the paths.

Multiple paths exist between sender and receiver in a standard computer network. In a standard TCP/IP protocol, a single path is used between receiver and sender. Yet, this method deteriorates the performance due to a bottleneck link that introduces extra constraint on top of end-to-end throughput. To handle this problem, several researchers put forward methodologies revolving around multiple-path solutions that take advantage of alternate paths to increase end-to-end throughput. Multipath solutions depend on the k-shortest path methodology to deliver alternative paths, which yield an inconsistent path without considering the highest flow among senders and receivers.

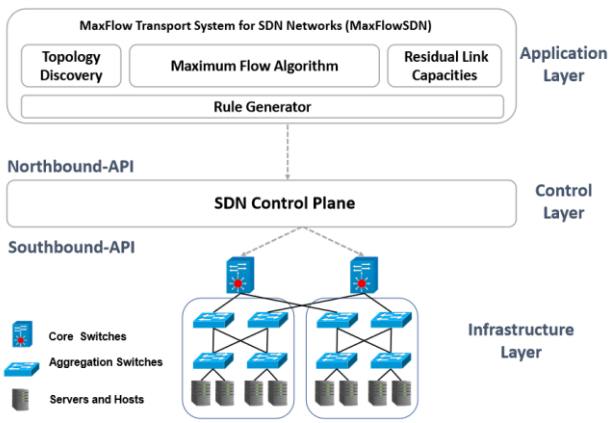

Figure 12. MaxFlowSDN system

The proposed scheme employs the Maximum Flow algorithm among senders and receivers to pick out intermediary switches where flows are split. Once intermediary switches are identified, TCP port number-based rules are pushed using OpenFlow, so that the flow can be split accordingly. Traditionally, forwarding and routing are accomplished using the destination IP addresses. However, for the multipath method, we use both the addresses and port numbers. Finally, senders generate many TCP links with port numbers used in OpenFlow. A typical topology of the SDN is depicted in Figure 12, where the MaxFlowSDN system is implemented. When data is sent to another host, the SDN controller utilizes the Maximum Flow Algorithm to find intermediate switches and forwards information to the sending host about the number of TCP connections it must make to reach the maximum flow to the destination host.

Table 2. Maximum flow solution for the example network [39]

Link	Flow Value
(H1, SW1)	5
(H1, SW2)	2
(SW1, SW2)	2
(SW1, H2)	3
(SW2, H2)	4

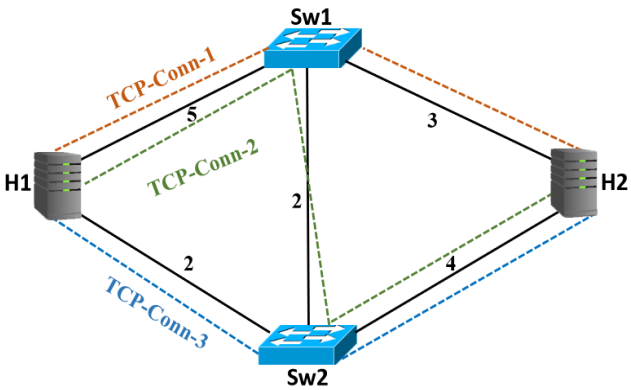

Figure 13. Example topology

Figure 13 shows the graphical representation of the network example. Because the Maximum Flow Algorithm is implemented, source and destination are indicated by H1 and H2, respectively. The solution for the example network is explained in Table 2. The flow-in for SW1 is observed to be 5 when using the (H1, SW1) link, whereas flow-out is split among the (SW1, SW2) links with flow value of 2 and (SW1, H2) link with a flow value of 3. Two TCP connections, i.e., TCP-Conn-1 and TCP-Conn-2, are made from H1 to SW1, since a split at SW1 into two flows occurred as shown in Figure 13. For SW1, two OpenFlow protocols are pushed to forward packets from H1 to SW2, and H2 according to TCP port numbers. Additionally, the value of 4 is noted for flow-in, for SW2, using two links, i.e., (H1, SW2) and (SW1, SW2), whereas the flow-out is noted to be 4 for the (SW2, H2) link. For SW2, OpenFlow rules are added to ensure the forwarding of incoming packets to SW2 and H2 from H1, according to TCP port numbers. As observed, data flow from the (H1, SW2) link is not fragmented; instead, one TCP connection is needed for the specific link, as noted by TCP-Conn-3. Effectively, this example warrants the use of three connections.

4.2 Overview of the MaxFlowSDN

This section explains the application of the MaxFlowSDN algorithm in the suggested system. The algorithm aims to determine routes between source and destination that provide the optimal maximum flow for TCP in a data center topology. The pseudocode is demonstrated in Algorithm 1.

Algorithm 1. MaxFlow Algorithm.
Functions:
MaxFlow(G, src, dst): computes the maximum flow between source src and destination dst for a graph G. SplitFlows2Paths(F): split flows F into several flows.
G: an input graph.
src: source node.
dst: destination node.
$SplittedPaths$: splits paths
flows = MaxFlow(G, src, dst)
SplittedPaths = []
for flow **in** flows **do** SplittedPath = SplitFlows2Paths($flow$)
SplittedPaths.append(SplittedPath)
return SplittedPaths

Two functions make up the algorithm: MaxFlow(G, src, dst) and SplitFlows2Paths(F). The MaxFlow function computes the maximum flow between source src and destination dst for a graph G. The SplitFlows2Paths(F) splits flows F into several paths.

Consider a DCN topology as shown in Figure 13 to illustrate the work of the MaxFlowSDN algorithm. Link weights represent the available bandwidth. In this topology, host H1 sends traffic to H2. The max flow function returns flow values for the given topology as follows: ["H1->SW1":5, "H1->SW2":2,"SW1->H2":3,"SW1->SW2":2, "SW2->H2":4]. The flow values list is passed to SplitFlows2Paths(F), which processes each flow to generate the list of paths. In this example, the list of paths is [(H1,SW1,H2), (H1,SW1,SW2,H2), (H1,SW2,H2)].

Algorithm 2. SDN Rule Generator Algorithm.
Functions:
FindSwitches($path$): Identifies switches along a given path.
DetermineInOut(sw, $path$): Determines optimal input/output ports on switch 'sw'.
GenerateRule(sw, in, out): Creates the SDN rule for the switch sw, for ports in and out
Input:
$paths$: split the paths between the source node (src) and the destination node (dst).
Output:
SDN Rules for multipath routing.
SDNRules = []
for path in paths do
  for sw in findSwitches(path)
    in, out = determineInOut(sw, $path$)
    SDNRule = generateRule(sw,src,dst, in , out) S
SDNRules.append(sw,SDNRule)
return SplittedPaths

After finding the paths, MaxFlowSDN generates the SDN Rules for every switch in all paths. The pseudo-code for generating the SDN rule algorithm is presented in Algorithm 2. It consists of three functions: FindSwitches, DetermineInOut, and GenerateRule. All paths are first passed to the FindSwitches function, which finds the switches for a given path.

Every switch in the path is then passed to the DetermineInOut function to determine input and output ports for switch *sw*, given the path. The last step is the GenerateRule function that creates an SDN code for a switch *sw*, based on source *src* and destination *dst* nodes. In addition to in and out ports, every switch in all paths is given to the GenerateRule function in order to construct all SDN rules.

5. EVALUATION

This section presents our experimental setup and its parameters, and presents the baseline evaluation methods of MaxFlowSDN.

5.1 Experimental setup and parameters

This section presents our experiment and its parameters. We used Mininet 2.3.0d4 emulation to evaluate the MaxFlowSDN system. The host device has Ubuntu 16.04 as operating system, with a six-core 2.2 GHz processor and 16 GB of RAM. In the given topologies, Mininet connects virtual Linux hosts together using OpenvSwitch switches. The switches were configured to use the OpenFlow 1.3 protocol, and the control logic was implemented using the Ryu 4.32 controller framework. iperf was employed to generate data traffic between destination and source hosts. Five runs were performed in each evaluation test, with a duration of 30 seconds of data-flow for each run. The throughput sampling-rate was noted as 1 sample per second. Table 3 shows all evaluation parameters.

Table 3. Default emulation parameters

Parameter	Values
Emulator	Mininet 2.3.0d4
Memory	16GB
Operating System	Ubuntu–16.04
Sampling-Rate	1 sample per sec
CPU	six-core 2.2 GHz
Number of Runs	Five for each method
TCP Congestion	CUBIC
Virtual Switches	OpenvSwitch 2.5.5
Experiment Duration	30 seconds

In a previous work [39], we evaluated multiple TCP congestion control variants to define the most suitable one for the experiments. Based on evaluation results, we chose Cubic. Experimental data center topologies are: Fat-Tree, DCell, and BCube. The Fat-Tree topology used in the evaluation is shown in Figure 14.

The red highlighted path shows the shortest path from source H1 to destination H2. The link capacities were manually selected due to two reasons. First, the Mininet virtualized environment imposed computational limitations, making gigabit-level capacities infeasible. Second, the capacities were configured to highlight disjoint paths between the source and destination nodes. Table 4 shows the link capacities used for the Fat-Tree topology, which were similarly applied to other topologies for consistency in evaluation. The DCell topology used in the evaluation is shown in Figure 15.

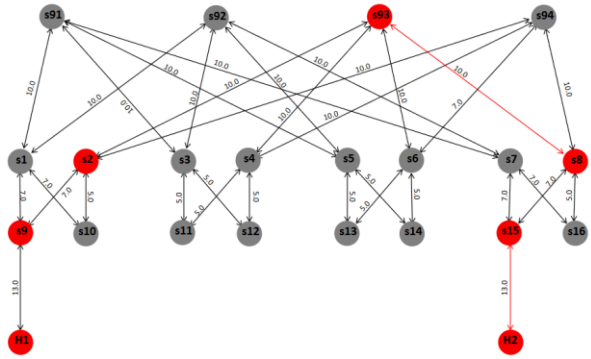

Figure 14. Fat-Tree test topology

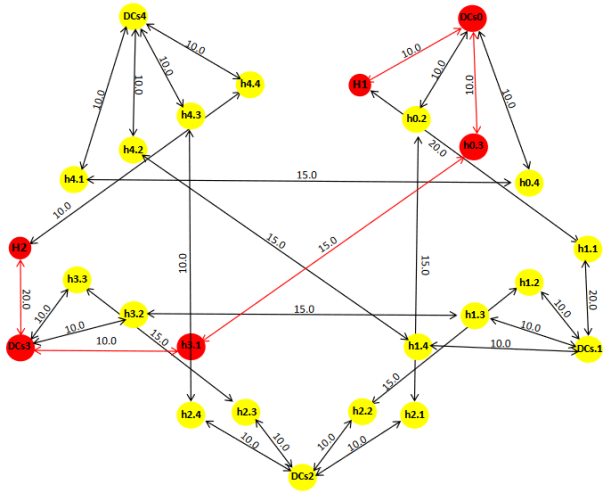

Figure 15. DCell test topology

Table 4. Fat-Tree test topology links bandwidth

Link Type	Link: Bandwidth
Core to Aggregation links	s91 - s1 link: 10 Mbpss91 - s3 link: 10 Mbpss91 - s5 link: 10 Mbpss91 - s7 link: 10 Mbpss92 - s1 link: 10 Mbpss92 - s3 link: 10 Mbpss92 - s5 link: 10 Mbpss92 - s7 link: 10 Mbpss93 - s3 link: 10 Mbpss93 - s4 link: 10 Mbpss93 - s6 link: 10 Mbpss93 - s8 link: 10 Mbpss94 - s2 link: 10 Mbpss94 - s4 link: 10 Mbpss94 - s6 link: 10 Mbpss94 - s8 link: 10 Mbps
Aggregation to Access links	s1 - s9 link: 7 Mbpss1 - s10 link: 5 Mbpss2 - s9 link: 7 Mbpss2 - s10 link: 5 Mbpss3 - s11 link: 5 Mbpss3 - s12 link: 5 Mbpss4 - s11 link: 5 Mbpss4 - s12 link: 5 Mbpss5 - s13 link: 5 Mbpss5 - s14 link: 5 Mbpss6 - s13 link: 5 Mbpss6 - s14 link: 5 Mbpss7 - s15 link: 7 Mbpss7 - s16 link: 5 Mbpss8 - s15 link: 7 Mbpss8 - s16 link: 5 Mbps
Access to Hosts links	s9 - h1 link: 13 Mbpss15 - h2 link: 13 Mbps

The red highlighted path shows the shortest-path from H1 to H2. Bandwidth details of the links are shown in Table 5. The BCube topology used in the evaluation is shown in Figure 16.

The red highlighted path shows the shortest-path from H1 to H2. Bandwidth details of the links are shown in Table 6.

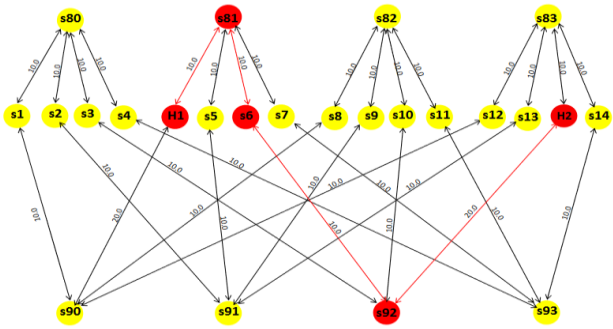

Figure 16. BCube test topology

Table 5. DCell test topology links bandwidth

Link Type	Link: Bandwith
Switch-to-host links	DCs0 - h0.1(h1) link: 10 MbpsDCs0 - h0.2 link: 10 MbpsDCs0 - h0.3 link: 10 MbpsDCs0 - h0.4 link: 10 MbpsDCs1 - h1.1 link: 20 MbpsDCs1 - h1.2 link: 10 MbpsDCs1 - h1.3 link: 10 MbpsDCs1 - h1.4 link: 10 MbpsDCs2 - h2.1 link: 10 MbpsDCs2 - h2.2 link: 10 MbpsDCs2 - h2.3 link: 10 MbpsDCs2 - h2.4 link: 10 MbpsDCs3 - h3.1 link: 10 MbpsDCs3 - h3.2 link: 10 MbpsDCs3 - h3.3 link: 10 MbpsDCs3 - h3.4(h2) link: 20 Mbps
Between-cells links	h0.1(h1) - h1.1 link: 20 Mbpsh0.2 - h1.4 link: 15 Mbpsh0.3 - h3.1 link: 15 Mbpsh0.4 - h4.1 link: 15 Mbpsh1.2 - h2.2 link: 15 Mbpsh1.3 - h3.2 link: 15 Mbpsh1.4 - h4.2 link: 15 Mbpsh2.3 - h3.3 link: 15 Mbpsh2.4 - h4.3 link: 15 Mbpsh3.4 (h2) - h4.4 link: 15 Mbps

Table 6. BCube test topology links bandwidth

Link Type	Link: Bandwidth
Level 1 links to servers	s80 - s1 link: 10 Mbpss80 - s2 link: 10 Mbpss80 - s3 link: 10 Mbpss80 - s4 link: 10 Mbpss81 - h1 link: 10 Mbpss81 - s5 link: 10 Mbpss81 - s6 link: 10 Mbpss81 - s7 link: 10 Mbpss82 - s8 link: 10 Mbpss82 - s9 link: 10 Mbpss82 - s10 link: 10 Mbpss82 - s11 link: 10 Mbpss83 - s12 link: 10 Mbpss83 - s13 link: 10 Mbpss83 - h2 link: 10 Mbpss83 - s14 link: 10 Mbps
Servers to level 2 links	s90 - s1 link: 10 Mbpss90 - h1 link: 20 Mbpss90 - s8 link: 10 Mbpss90 - s12 link: 10 Mbpss91 - s2 link: 10 Mbpss91 - s5 link: 10 Mbpss91 - s9 link: 10 Mbpss91 - s13 link: 10 Mbpss92 - s3 link: 10 Mbpss92 - s6 link: 10 Mbpss92 - s10 link: 10 Mbpss92 - h2 link: 20 Mbpss93 - s4 link: 10 Mbpss93 - s7 link: 10 Mbpss93 - s11 link: 10 Mbpss93 - s14 link: 10 Mbps

5.2 Performance metric and baseline methods

The performance of MaxFlowSDN is matched with three other approaches, considering all tested data center topologies. For each method, we had five runs with data produced via iperf sent between H1 and H2 hosts. The number of runs is sufficient because data variance is low, as will be observed in results later. The methods which are compared here are given below:

1. StandardTCP: Only one TCP connection is used to transfer application data, without considering the number of available interfaces.
2. ParallelTCP: To transfer application data, one TCP connection is created for each available interface. These connections exist and work in parallel.
3. MPTCP: MPTCP connection is used to transfer application data. For all available interfaces, MPTCP internally generates multiple sub-TCP connections for everyone.
4. MaxFlowSDN: To transfer application data, multiple TCP connections are created for each available interface. These connections exist and work in parallel. The MaxFlow algorithm's flow value is used to determine the number of connections.

For each method, the throughput from H1 to H2 was observed.

6. RESULTS AND DISCUSSION

This section shows the findings of three types of data center topologies, as shown in subsection 5.1.

6.1 Fat-Tree topology

In the Fat-Tree test topology as shown in Figure 14, we compared the performance of MaxFlowTCP against three methods: StandardTCP, ParallelTCP, and MPTCP. Figure 17 shows the average throughput from H1 to H2 for all methods in the Fat-Tree topology. Table 7 shows the full data and throughput results obtained for the Fat-Tree topology. In the Fat-Tree test topology, we compared the MaxFlowTCP performance against three methods: StandardTCP, ParallelTCP, and MPTCP. Figure 17 shows the average throughput from H1 to H2 for all methods in the Fat-Tree topology. Table 7 shows the full data and throughput results obtained for the Fat-Tree topology.

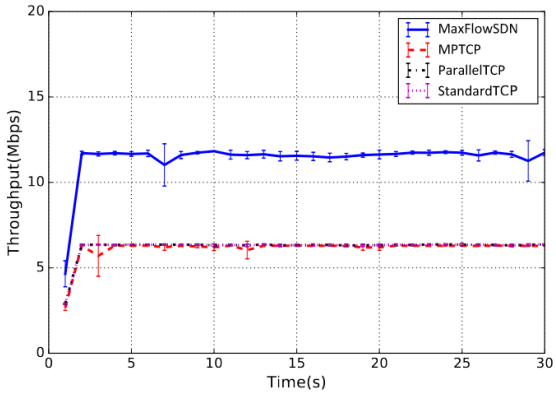

Figure 17. Fat-Tree average throughput

In the Fat-Tree topology experiment, only one network interface was available in H1 source node and H2 destination node. Consequently, the Parallel TCP and StandardTCP performance were identical, since only one TCP connection was created for both methods. Both obtained an average throughput of 6.5 Mbps after the first two seconds of the experiment. MPTCP performance was similar to StandardTCP and ParallelTCP because it also could not benefit from its main advantage of having multiple internal paths created for

different interfaces. MaxFlowSDN outperformed the three methods with an average throughput around 12 Mbps, because it created two connections to reach the maximum flow possible from source to destination. Figure 18 shows the two connection paths created by the MaxFlowSDN algorithm. Results show that MaxFlowSDN had approximately 80% more throughput compared to ParallelTCP, StandardTCP and MPTCP.

Table 7. Throughput in the Fat-Tree topology

Methods	Average		Standard Deviation	
	Total Data	Throughput	Total Data	Throughput
MaxFlowSDN	410.86 Mb	11.76 Mbps	5.55	0.17
MPTCP	216.60 Mb	6.38 Mbps	1.17	0.04
ParallelTCP	214.21 Mb	6.49 Mbps	5.60	0.02
StandardTCP	214.21 Mb	6.49 Mbps	5.60	0.02

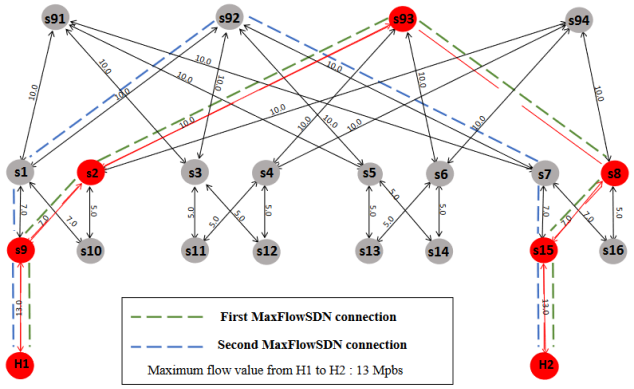

Figure 18. MaxFlowSDN connections in Fat-Tree test

6.2 DCell topology

In the DCell test topology, we compared MaxFlowSDN performance against the three methods: StandardTCP, ParallelTCP and MPTCP. Figure 19 shows the average throughput from H1 to H2 for all methods in the DCell topology. Table 8 explains the full data and throughput results obtained for DCell topology.

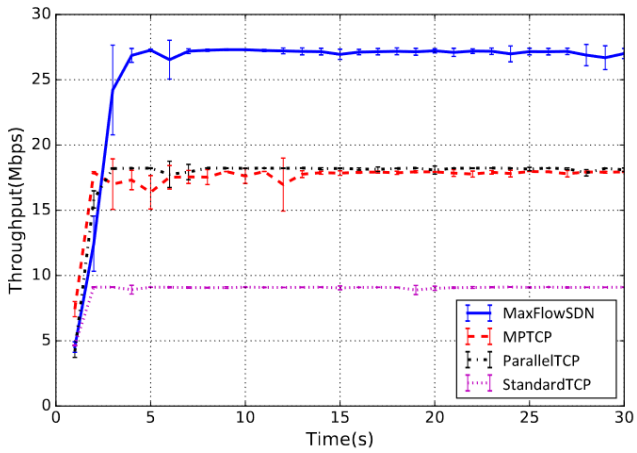

Figure 19. DCell average throughputTopology

In the DCell topology experiment, two network interfaces were available in the H1 source node and H2 destination node. In StandardTCP method, one TCP connection was created between source and destination. The average throughput for StandardTCP after two seconds of the experiment was 9.33 Mbps. The average throughput for ParallelTCP and MPTCP was around 18 Mbps. They obtained similar results because they both have two connections going from the two interfaces available at the source. MaxFlowSDN outperformed all the three methods as the MaxFlow algorithm used in MaxFlowSDN creates three connections to use the maximum available bandwidth. Figure 20 shows the three connection paths created by the MaxFlowSDN algorithm. The average throughput reached 27 Mbps after two seconds of the experiment. Results showed that MaxFlowSDN had around 185% more throughput when compared to StandardTCP, and 47% more compared to ParallelTCP and MPTCP.

Table 8. Throughput in DCell topology

Methods	Average		Standard Deviation	
	Total Data	Throughput	Total Data	Throughput
MaxFlowSDN	937.58 Mb	26.58 Mbps	6.12	0.16
MPTCP	582.09 Mb	17.94 Mbps	11.62	0.35
ParallelTCP	647.43 Mb	18.36 Mbps	1.38	0.05
StandardTCP	329.03 Mb	9.33 Mbps	0.81	0.03

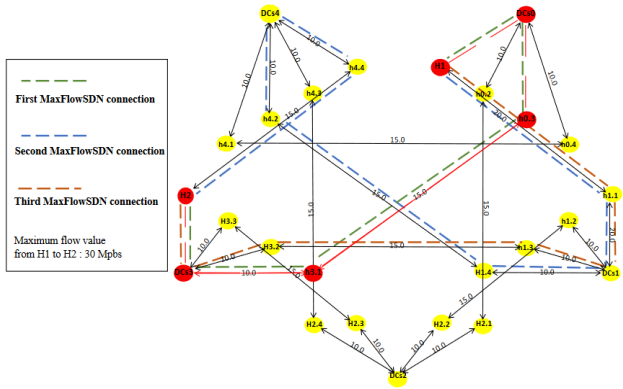

Figure 20. MaxFlowSDN connections in the DCell

6.3 BCube topology

In the BCube test topology, we compared MaxFlowSDN performance against three methods: StandardTCP, ParallelTCP, and MPTCP. Figure 21 shows the average throughput from H1 to H2 for all methods in the BCube topology. Table 9 shows the results of total data and throughput in BCube topology.

In the BCube topology experiment, two network interfaces were available in H1 source node and H2 destination node. In the StandardTCP method, one TCP connection was created between source and destination. The average throughput for StandardTCP after two seconds of the experiment was 9.35 Mbps. The average throughput for parallel and MPTCP was around 18 Mbps. Their results were similar as they have two connections from both interfaces available. MaxFlowSDN outperformed the three methods because the algorithm used in MaxFlowSDN creates three connections to make use of the

maximum available bandwidth. Figure 22 shows the three connection paths created by MaxFlowSDN. The average throughput yielded 27.32 Mbps after two seconds of the experiment. Results showed that MaxFlowSDN had around 191% more throughput compared to StandardTCP, and 48% more compared to ParallelTCP and MPTCP.

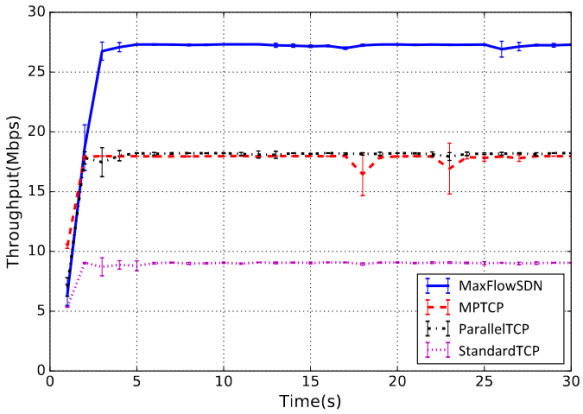

Figure 21. BCube average throughput

Table 9. Throughput in BCube topology

Methods	Average		Standard Deviation	
	Total Data	Throughput	Total Data	Throughput
MaxFlowSDN	926.39 Mb	27.32 Mbps	14.01	0.25
MPTCP	588.72 Mb	18.21 Mbps	8.01	0.30
ParallelTCP	644.70 Mb	18.54 Mbps	6.40	0.03
StandardTCP	326.96 Mb	9.35 Mbps	1.13	0.03

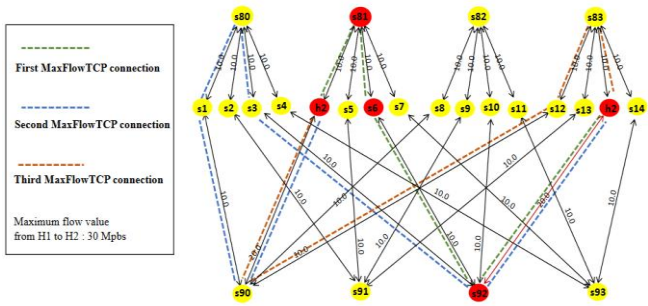

Figure 22. MaxFlowSDN connections in Bcube

The evaluation results demonstrate that MaxFlowSDN outperforms existing methods across Fat-Tree, DCell, and BCube topologies. The performance advantage of MaxFlowSDN stems from its ability to determine optimal paths using the Maximum Flow algorithm coupled with SDN-based network control. In the Fat-Tree topology, MaxFlowSDN achieved 80% higher throughput than StandardTCP, ParallelTCP, and MPTCP by creating two optimal connections between source and destination nodes. In DCell and BCube topologies, MaxFlowSDN showed approximately 190% higher throughput compared to StandardTCP and almost 50% improvement over ParallelTCP and MPTCP by effectively utilizing three connection paths. Unlike MPTCP, which requires kernel modifications, MaxFlowSDN achieves these improvements using conventional TCP implementation while leveraging SDN capabilities for dynamic path configuration and flow management through OpenFlow rules. These results establish MaxFlowSDN as an effective solution for optimizing intra-datacenter traffic routing.

7. CONCLUSIONS AND FUTURE WORK

A data center is an essential asset for any organization with vital systems that perform critical daily operations. Data center networks need to process and compute bulk data using the most practical and efficient method possible. The use of SDNs in data centers has allowed them to improve their performance in several respects, since SDN makes network configurations programmable and dynamic. This has yielded improved flexibility, better management, and more scalable schema.

In this paper, we proposed a new methodology named MaxFlowSDN, which uses SDN coupled with conventional TCP to deliver the highest possible data flow throughput in data centers. Our methodology achieved this maximum throughput by generating multiple paths between destination-source pairs. The proposed methodology (MaxFlowSDN) yielded significant improvement when compared with three other methodologies, i.e., StandardTCP, ParallelTCP, and MPTCP. The comparison was made using test topologies commonly employed in data centers. The proposed methodology outperformd others thanks to its ability to use the maximum available bandwidth via the Maximum Flow Algorithm, which permits the maximum possible throughput. In the Fat-Tree data center topology, MaxFlowSDN had 80% higher throughput than MPTCP, ParallelTCP, and StandardTCP. In DCell and BCube topologies, evaluation results demonstrated that MaxFlowSDN provides around 190% higher throughput than StandardTCP, and almost 50% improvement compared to ParallelTCP and MPTCP.

However, MaxFlowSDN has limitations. It has been tested on a limited number of topologies, and its scalability in larger, real-world data centers needs further evaluation. While the system is optimized for throughput, it does not address latency-sensitive traffic, and it currently relies on a single SDN controller, which may become a bottleneck. Additionally, it lacks mechanisms for real-time fault detection and rerouting.

Future work will focus on addressing these limitations by developing a dynamic, adaptive version of MaxFlowSDN that handles real-time network changes, uses a distributed SDN controller architecture for better scalability and fault tolerance, and incorporates latency-awareness, so that optimizing path selection not just for throughput but also for minimizing end-to-end delays. This would be beneficial for latency-sensitive applications such as online gaming, data exchanges between autonomous vehicles, or telemedicine, where even slight delays can significantly impact user experience. We also plan to evaluate the system in larger, real-world environments, such as GENI (Global Environment for Network Innovations), to further refine its capabilities.

ACKNOWLEDGMENT

The authors extend their appreciation to Deanship of scientific research in King Saud University, Saudi Arabia.

REFERENCES

[1] O'Connell, A. (2019). Forecast: Data centers, worldwide, 2016-2023, 2019 update. https://www.gartner.com/en/documents/3956376.

[2] IEA. (2019). Data centres and energy – From global headlines to local headaches? IEA, Paris. https://www.iea.org/commentaries/data-centres-and-energy-from-global-headlines-to-local-headaches.

[3] Nunes, B.A.A., Mendonca, M., Nguyen, X.N., Obraczka, K., Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. IEEE Communications Surveys & Tutorials, 16(3): 1617-1634. https://doi.org/10.1109/SURV.2014.012214.00180

[4] ONF. (2012). Software-defined networking: The new norm for networks. https://opennetworking.org/wp-content/uploads/2011/09/wp-sdn-newnorm.pdf.

[5] Hwang, R.H., Tseng, H.P., Tang, Y.C. (2015). Design of SDN-Enabled cloud data center. In 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), Chengdu, China, pp. 950-957. https://doi.org/10.1109/SmartCity.2015.193

[6] Alenazi, M.J.F., Almutairi, A., Almowuena, S., Wadood, A., Çetinkaya, E.K. (2020). NFV provisioning in large-scale distributed networks with minimum delay. IEEE Access, 8: 151753-151763. https://doi.org/10.1109/ACCESS.2020.3017667

[7] Hong, C.Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., Wattenhofer, R. (2013). Achieving high utilization with software-driven WAN. In Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM, Hong Kong China, pp. 15-26. https://doi.org/10.1145/2486001.2486012

[8] Jin, X., Li, Y.R., Wei, D., Li, S.M., Gao, J., Xu, L., Li, G.Z., Xu, W., Rexford, J. (2016). Optimizing bulk transfers with software-defined optical WAN. In Proceedings of the 2016 ACM SIGCOMM Conference, Florianopolis Brazil, pp. 87-100. https://doi.org/10.1145/2934872.2934904

[9] Ford, A., Raiciu, C., Handley, M., Bonaventure, O. (2013). TCP extensions for multipath operation with multiple addresses (No. rfc6824).

[10] Peng, Q.Y., Walid, A., Hwang, J., Low, S.H. (2014). Multipath TCP: Analysis, design, and implementation. IEEE/ACM Transactions on networking, 24(1): 596-609. https://doi.org/10.1109/TNET.2014.2379698

[11] Tang, W.S., Fu, Y.S., Dong, P.P., Yang, W.J., Yang, B., Xiong, N.X. (2019). A MPTCP scheduler combined with congestion control for short flow delivery in signal transmission. IEEE Access, 7: 116195-116206. https://doi.org/10.1109/ACCESS.2019.2933880

[12] Bonaventure, O., Seo, S. (2016). Multipath TCP deployments. IETF Journal, 12(2): 24-27.

[13] Mehani, O., Holz, R., Ferlin, S., Boreli, R. (2015). An early look at multipath TCP deployment in the wild. In Proceedings of the 6th International Workshop on Hot Topics in Planet-Scale Measurement, pp. 7-12. https://doi.org/10.1145/2798087.2798088

[14] Noormohammadpour, M., Raghavendra, C.S. (2017). Datacenter traffic control: Understanding techniques and tradeoffs. IEEE Communications Surveys & Tutorials, 20(2): 1492-1525. https://doi.org/10.1109/COMST.2017.2782753.

[15] Sharma, V., Mishra, R. (2020, June). A comprehensive survey on data center network architectures. In 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, pp. 222-228. https://doi.org/10.1109/ICRITO48877.2020.9197934

[16] Guo, C.X., Lu, G.H., Li, D., Wu, H.T., Zhang, X., Shi, Y.F., Tian, C., Zhang, Y.G., Lu, S.W. (2009). BCube: A high performance, server-centric network architecture for modular data centers. In Proceedings of the ACM SIGCOMM 2009 conference on Data communication, Barcelona, Spain, pp. 63-74. https://doi.org/10.1145/1592568.1592577

[17] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., Turner, J. (2008). OpenFlow: Enabling innovation in campus networks. ACM SIGCOMM Computer Communication Review, 38(2): 69-74. https://doi.org/10.1145/1355734.1355746

[18] Wu, X., Lu, K., Zhu, G. (2018). A survey on software-defined wide area networks. Journal of Communications, 13(5): 253-258.

[19] Sahoo, K.S., Mohanty, S., Tiwary, M., Mishra, B.K., Sahoo, B. (2016). A comprehensive tutorial on software defined network: The driving force for the future internet technology. In Proceedings of the International Conference on Advances in Information Communication Technology & Computing, Bikaner, India, pp. 1-6. https://doi.org/10.1145/2979779.2983928

[20] Kinariwala, B., Rao, A.G. (1977). Flow switching approach to the maximum flow problem: I. Journal of the ACM (JACM), 24(4): 630-645.

[21] Ford, L.R., Fulkerson, D.R. (2009). Maximal flow through a network. In Classic Papers in Combinatorics, pp. 243-248.

[22] Edmonds, J., Karp, R.M. (1972). Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the ACM (JACM), 19(2): 248-264.

[23] Dinic, E.A. (1970). Algorithm for solution of a problem of maximum flow in networks with power estimation. Soviet Mathematics Doklady, 11: 1277-1280.

[24] Karzanov, A.V. (1974). Determining the maximal flow in a network by the method of preflows. Soviet Mathematics Doklady, 15: 434-437.

[25] Goldberg, A.V., Tarjan, R.E. (1988). A new approach to the maximum-flow problem. Journal of the ACM (JACM), 35(4): 921-940. https://doi.org/10.1145/48014.61051

[26] Goldberg, A.V., Rao, S. (1998). Beyond the flow decomposition barrier. Journal of the ACM (JACM), 45(5): 783-797. https://doi.org/10.1145/290179.290181

[27] Ye, J., Feng, L.T., Xie, Z.Q., Huang, J.W., Li, X.H. (2019). Fine-grained congestion control for multipath TCP in data center networks. IEEE Access, 7: 31782-31790. https://doi.org/10.1109/ACCESS.2019.2902860

[28] Li, W., Zhang, H., Gao, S., Xue, C., Wang, X., Lu, S. (2019). SmartCC: A reinforcement learning approach for multipath TCP congestion control in heterogeneous networks. IEEE Journal on Selected Areas in Communications, 37(11): 2621-2633. https://doi.org/10.1109/JSAC.2019.2933761

[29] Pang, S.C., Yao, J.M., Wang, X., Ding, T., Zhang, L. (2019). Transmission control of MPTCP Incast based on buffer balance factor allocation in data center networks. IEEE Access, 7: 183428-183434. https://doi.org/10.1109/ACCESS.2019.2960180

[30] Fu, Q., Sun, E., Meng, K., Li, M., Zhang, Y. (2020). Deep Q-learning for routing schemes in SDN-based data center networks. IEEE Access, 8: 103491-103499. https://doi.org/10.1109/access.2020.2995511

[31] Jung, E.S., Vishwanath, V., Kettimuthu, R. (2014). Distributed multipath routing algorithm for data center networks. In 2014 International Workshop on Data Intensive Scalable Computing Systems, New Orleans, LA, USA, pp. 49-56. https://doi.org/10.1109/DISCS.2014.14.

[32] Liu, S., Huang, J.W., Zhou, Y.T., Wang, J.X., He, T. (2019). Task-aware TCP in data center networks. IEEE/ACM Transactions on Networking, 27(1): 389-404. https://doi.org/10.1109/TNET.2018.2890010

[33] Cheng, Y.Y., Jia, X.H. (2020). NAMP: Network-aware multipathing in software-defined data center networks. IEEE/ACM Transactions on Networking, 28(2): 846-859. https://doi.org/10.1109/TNET.2020.2971587

[34] Zhang, X.M., Liu, S.K., Xu, J. (2018). An efficient scheduling scheme for XMP and DCTCP mixed flows in commodity data centers. IEEE Communications Letters, 22(9): 1770-1773. https://doi.org/10.1109/LCOMM.2018.2853616

[35] Park, M., Sohn, S., Kwon, K., Kwon, T.T. (2019). MaxPass: Credit-based multipath transmission for load balancing in data centers. Journal of Communications and Networks, 21(6): 558-568. https://doi.org/10.1109/JCN.2019.000047

[36] Alvarez-Horcajo, J., Lopez-Pajares, D., Martinez-Yelmo, I., Carral, J.A., Arco, J.M. (2019). Improving multipath routing of TCP flows by network exploration. IEEE Access, 7: 13608-13621. https://doi.org/10.1109/ACCESS.2019.2893412

[37] AlShammari, W.M., Alenazi, M.J. (2020). Performance analysis of a graph-theoretic load balancing method for data centers. International Journal of Advanced Computer Science and Applications, 11(8): 666-674.

[38] AlShammari, W.M., Alenazi, M.J.F. (2021). BL-Hybrid: A graph-theoretic approach to improving software-defined networking-based data center network performance. Transactions on Emerging Telecommunications Technologies, 32(1): e4163. https://doi.org/10.1002/ett.4163

[39] Saeed, N.S.B., Alenazi, M.J. (2020). Utilizing SDN to deliver maximum TCP flow for data centers. In Proceedings of the 3rd International Conference on Information Science and Systems, Cambridge United Kingdom, pp. 181-187. https://doi.org/10.1145/3388176.3388216