Noname manuscript No.
(will be inserted by the editor)

CECT: Computationally Efficient Congestion-avoidance and Traffic Engineering in Software-defined Cloud Data Centers

M.M. Tajiki, · B. Akbari, · M. Shojafar, · S.H. Ghasemi, · M.L. Barazandeh, · N. Mokari, · L. Chiaraviglio, · M. Zink

Received: 3 July 2017 / Revised: 13 November 2017/ Accepted: XX XX

Abstract The proliferation of cloud data center applications and network function virtualization (NFV) boosts dynamic and QoS dependent traffic into the data centers' network. Currently, lots of network routing protocols are requirement agnostic, while other QoS-aware protocols are computationally complex and inefficient for small flows. In this paper, a computationally efficient congestion avoidance scheme, called *CECT*, for software-defined cloud data centers is proposed. The proposed algorithm, CECT, not only minimizes network congestion but also reallocates the resources based on the flow requirements. To this end, we use a routing architecture to reconfigure the network resources triggered by two events: 1) the elapsing of a predefined time interval, or, 2) the occurrence of congestion. Moreover, a forwarding table entries compression technique is used to reduce the computational complexity of CECT. In this way, we mathematically formulate an optimization problem and define a genetic algorithm to solve the proposed optimization problem. We test the proposed algorithm on real-world network traffic. Our results show that CECT is computationally fast and the solution is feasible in all cases. In order to evaluate our algorithm in term of throughput, CECT is compared with ECMP (where the shortest path algorithm is used as the cost function). Simulation results confirm that the throughput obtained by running CECT is improved up to 3x compared to ECMP while packet loss is decreased up to 2x.

Keywords QoS-aware Resource Reallocation, Traffic Engineering, Software-defined Cloud Data Centers (SCDC), Network Reprogramming Overhead.

MM. Tajiki, S. Hesamoddin Ghasemi, M. Latifi Barazandeh, N. Mokari, B. Akbari
lectrical and Computer Engineering,
Tarbiat Modares University, Tehran, Iran
E-mail: {mahdi.tajiki, h.qasemi, mahdi.barazandeh, b.akbari, mokari}@modares.ac.ir

M. Shojafar, L. Chiaraviglio
CNIT, Department of Electronic Engineering,
Tor Vergata University of Rome, Rome, Italy.
E-mail: mohammad.shojafar@cnit.it, luca.chiaraviglio@uniroma2.it

Michael Zink
Electrical and Computer Engineering,
University of Massachusetts Amherst, Amherst, USA
E-mail: mzink@cas.umass.edu

1 Introduction

Network Function Virtualization (NFV) has drawn significant attention from industry, government, and academia to improve flexibility and reduce the time to market of new services. Some of these services have a chain of functions (e.g., firewall and load balancer) which need the network to guarantee the required Quality of Service (QoS) constraints. On the other hand, the data centers used to create cloud services represent a significant investment in capital outlay and ongoing costs. Therefore, the cloud data center services are highly adapted at the present time. The dynamic nature of the cloud data center traffic (e.g., VM motion) necessitate support for the diverse class of QoS requirements. Not surprisingly, these QoS requirements have to be guaranteed by the network routing protocol. Additionally, the enormous and dynamic network traffic which is communicating via the network infrastructure imposes congestion in the network links. Clearly, this effect has to be dynamically addressed by the routing protocols in Software-defined Cloud Data Centers (SCDC). In a sequel, the main aim of this paper is to dynamically and

efficiently reallocate resources in a way that i) guarantees QoS requirements of different applications; and; ii) protectively prevent congestion and resource waste.

In this context, several questions arise, like: Is it possible to propose an approach that considers the impact of flows routing among each other? How to model an SCDC and services to evaluate the flow requirements? How to implement a real-time computationally efficient method to preserve SCDC QoSs? The answer to these questions is the goal of the paper.

More in detail, we introduce a dynamic and computationally efficient resource reallocation scheme called Computationally Efficient Congestion avoidance and Traffic Engineering (CECT) in which we guarantee the minimum bandwidth for a specific flow. The main contributions are as follows:

i) The proposed scheme not only maximizes the network throughput but also guarantees the requested QoS level. Since the traffic flow requirements change over time, the mentioned scheme dynamically reallocates the resources in a predefined time period. In order to solve the corresponding optimization problem, two schemes are proposed. The *first one* maximizes the total network throughput, where its computational complexity is high. The *second one* is a low computational complexity meta-heuristic method that finds a near-optimal solution.
ii) To overcome the resource fragmentation in networks with big flows, we consider the impact of each flow on other flows, i.e., in the corresponding optimization problem the rerouting of all flows must be performed simultaneously. Additionally, in order to improve the congestion avoidance as well as increasing the network throughput, multi-path routing is supported in the proposed schemes. Hence, different flows from a similar source to a similar destination can be rerouted via various paths.
iii) To make a tradeoff between computational complexity and performance, the granularity of network rescheduling is adjustable. To this end, we introduce a flow table entry compression technique that makes a tradeoff between the optimality gap and the computational complexity of the solution. In our design, the granularity can be the exchanged information of "a special application in a server with another application on a different server" or "all communications from one data center to another one".
iv) We implement CECT in the MiniNet emulator [31], by considering a realistic network traffic and a realistic fat-tree network topology. Besides, in order to evaluate the impact of flow size on the performance of CECT, we implement a packet generator to generate traffic patterns with micro, small, medium, and big flows.

The full evaluation of other QoSs features (such as the queuing delay, delay variation (jitter), quality of user experiments in SCDC, and the mapping of such parameters in other types of the networks such as WAN) will be some interesting branches of future research.

The rest of the paper is organized as follows. Section 2 presents the most recent existing literature. In Section 3, we present the main functional blocks of the proposed CECT architecture for SCDC. Afterward, in Section 4, we detail the problem formulation, while Section 5 proposed the bio-inspired scalable solution. After detailing the tested application scenarios and performance metrics in Section 7, the performance of CECT algorithm is presented and compared with the corresponding one of the ECMP algorithm [22] in Section 8. Finally, in Section 9, we summarize the main attained results and give some hints for future research.

2 Related work

In the following, we will briefly discuss the main literature engaged in SCDC.

2.1 Congestion Avoidance/Control Methods

In the literature, different works on congestion avoidance/control and traffic engineering have been presented in the past. Authors in [22] presented a multi-path routing technique, ECMP, to perform static load splitting among flows across 8 to 16 multi paths. It is required to deliver high bisection bandwidth for larger data centers. ECMP is applied in current switches in which that are tuned and configured with several possible forwarding paths for a given subnet. In other words, when a packet with multiple candidate paths arrives, it is forwarded on the one that corresponds to a set of selected fields of that packet's headers and modules the number of paths. ECMP does not account for flow bandwidth in making allocation decisions, which can lead to oversubscription that CECT matters this issue.

In [8], a congestion control scheme classifies the network traffic into two classes of *ordinary* and *premium* flows. More in depth, authors consider a non-linear network model based on the fluid flow theory that is able to cope with both the physical network resource constraints and unknown time delays associated with networking systems. However, the proposed scheme does not embed any traffic engineering scheme, therefore, it does not specify routes for the network flows that

the proposed CECT method does. The authors of [28] proposed a traffic engineering approach for networks in which the link capacity and class of service requirements may vary with time. Their scheme does not route flows, however, it produces some control laws which can be used for routing. In [25], a traffic prediction algorithm is exploited to prevent network congestion before it happens. Still, the approach is not applicable in networks with unpredictable traffic pattern. Instead, CECT schedules the resources and flows according to the complete view of the system state and is unbounded to any traffic patterns.

The authors of [21] propose an SDN-based TCP congestion control mechanism at the client side. They focus on long-lived flows and reduce the sending rate by adjusting the TCP receive window of ACK packet after OpenFlow switch triggered a congestion message to the controller. Similarly, authors in [14] proposed a method to control the congestion in SCDCs based on the OpenFlow protocol. Their method monitors the port statistics of the OpenFlow-enabled switches and reroutes some flows in the congested links. Both [21] and [14] do not consider QoS requirements of different flows. In other words, they assume that all flows have similar requirements. Instead CECT considers the flows features and SDN resources properties dynamically.

In [30], a QoS-aware resource allocation algorithm which guarantees a minimum overhead on the network during reprogramming phase. The authors mathematically formulate the optimization problem of flow routing in the data center networks and solve it using binary linear programming. The most challenging part of their method is the high computational complexity of solving the mentioned problem which makes the proposed algorithm versy inefficient in medium and large scale data centers. Moreover, the authors of [29], propose a routing algorithm for SCDCs based on traffic prediction. They mathematically formulate the routing problem and propose two schemes to solve it (an exact solution which has a high computational complexity and a suboptimal but fast one). This paper presents a flavor of the significant results and ongoing work, but it is applicable only in predictable networks whilst CECT is not only applicable in the predicting network, but also it is used in unpredicted and any shape of the networks (i.e., based on proof of concept presented in the simulation results).

In addition, authors in [26] present a network-aware resource reallocation technique, in which they use the network topology characteristics of the data center to minimize the maximum latency in communication between VMs. They incorporate the resource heterogeneity by including the computational and communication requirements in the proposed technique. The main focus of the work is on heterogeneity of computational requirements for VMs in CDC, and did not consider heterogeneity of the network bandwidth and the other computational requirements for VMs in CDC. Instead, the proposed method, CECT, covers these limitations and provides load balancing routing to the incoming flows.

In addition, the authors of [34] adopt a two-phase flow embedding approach with an iterative traffic engineering algorithm to address the resource reallocation problem lying in the multimedia communication systems. Some other works such as [23] focus on providing QoS for voice over IP (VoIP) traffics and simultaneously optimizing the power efficiency. In CECT, we not only cover the VoIP traffic class but also consider several types of traffic, e.g., FTP, high definition (HD) video stream that can be applied in 5G network.

2.2 QoS-aware Routing Methods (Single Class of Traffic)

Different works that focus on multimedia and use flow rerouting to guarantee the QoS parameters have been presented in [10, 11, 9] and [12]. In detail, the authors of [10] formulate the dynamic QoS routing problem as a Constrained Shortest path (CSP) problem. In this way, they represent the entire network as a simple graph and define a cost function based on the QoS parameters. The proposed solution improves the QoS of video streaming. However, it cannot support different classes of QoS that is covered in CECT. In [11], the authors form in a group the incoming flows as multimedia and data flow, where the multimedia flows are routed via QoS guaranteed paths. However, the data flow remains on traditional shortest-paths. Instead in CECT, we cover various types of incoming traffic. Besides, authors in [27] address multimedia data processing with computationally intensive tasks and exchange of a big volume of data flow via QoS guaranteed paths per time period and introduce a general framework called *MMGreen* to ensures QoSs of the user flows and achieves maximum energy saving and attains green cloud computing goals in a fully distributed fashion by utilizing the DVFS-based CPU frequencies. Although MMGreen is novel and interesting, but compared to CECT in routing scheme, it does not cover some QoSs such as the utilization of the links between the SDN switches.

Moreover, the authors of [9] propose a distributed QoS routing architecture for video streaming. They use the OpenFlow features to implement their scheme in a multi-domain environment. Finally, [7] devises a simple analytic framework and an experimental platform to

transfer the video streaming. In their framework the video stream has a base layer, which is modeled as a QoS flow, and multiple enhancement layers, which are treated as best-effort flows.

2.3 QoS-aware Routing Methods (Multiple Class of Traffic)

The main challenging deficiency of the mentioned schemes is not to support different classes of QoS, i.e., these solutions only focus onmultimedia flows and ignore other types of data flows. In contrast, there are many works considering different types of data flows such as [17, 20, 18, 15, 33, 24] and [35]. Particularly, authors in [17] propose a new QoS routing algorithm for Multi-Protocol Label Switching (MPLS) networks. The paths are selected based on critical links so as to minimize interference with the future requests. However, the solution [17] is designed for MPLS and it can not be used for other types of network. In addition, the main focus of [18] is to route the flows with the QoS constraint using genetic algorithms. In this way, authors propose a heuristic for unicast routing to find feasible path satisfying the flows requirements. In the same way, [20] guarantees the QoS by defining a new measure called *path weight* which is minimized by the aid of ant colony system. Both [18] and [20] route each flows separately. These methods have three set of limitations. First, they do not guarantee the efficiency of the selected path due to the minimum requirement constraint applied for termination conditions. Second, they do not guarantee the end-to-end performance. In other words, the proposed routing algorithms do not exploit the capability of routing all flows simultaneously, i.e., it is impossible to reroute a flow considering the possible routes of other flows. Third, they examined single flow performance, and unable to handle multiple flows with different QoS requirements. Consequently, these schemes are not applicable for a comprehensive network reconfiguration.

In addition, the scheme reported in [15] explores scalable architectures that jointly optimize rate control and routing. Since the goal of this work is to perform rate control, the proposed approach distributes information and computation across multiple tiers of an optimization machinery. Similarly, in [33], the authors present a resource allocation scheme for inter-data center communication with multiple traffic classes. Although [15] and [33] are practical for inter-data center communications, they are impractical for intra-AS (i.e., inter-autonomous system) network resource allocation, e.g., intra-cloud data centers. The paper [24] deploys the SDN features to manage the differentiating network services with QoS satisfaction. Their problem formulation is in form of integer linear programming (ILP). The weak point of this scheme is its computational complexity which makes it impractical for medium and large scale networks. In [35], a QoS-aware routing mechanism is proposed to balance the network load of industrial Ethernet. In particular, authors exploit the ant colony method to obtain a path for data transmission with different QoS requirements. Since the meta-heuristic approaches may return unfeasible solutions, they need to be discussed from the validity perspective of the solution. However, the time complexity and the validity of their scheme are not discussed.

3 Reference Architecture

In Fig. 1, we describe the considered architecture. In particular, we assume a software-defined cloud network where a logically centralized controller coordinates the network. The switches are all OpenFlow-enabled and the protocol used for communication between the switches and the controller is OpenFlow. Therefore, the controller may query the switches for network topology and current traffic matrix. On the other hand, there are k different classes of traffic flows with different QoS requirement. In addition, the flows are highly dynamic and there are some big flows in the network. In the case of arrival of a new flow, a conventional routing scheme like ECMP [22] is applied. In order to decrease network congestion, the controller reconfigures the network, i.e., some flows are rerouted. To this end, the resources are reallocated based on some predefined measures such as time periods or the event of high packet loss in the network. Table 1 reports the main notations of the paper.

In order to maximize the total network throughput as well as guarantee the QoS requirement of flows, it is necessary to dynamically reallocate the resources with the dynamic pattern of network traffic. In this way, we design a mathematical model that considers the network topology, the flow requirement matrix, and the flow specifications as input, and finds a routing matrix satisfying the QoS constraint and minimizing network congestion. The network topology is given by the matrix $B_{N_L \times N_L}$ where $B_{(i,j)}$ determines the bandwidth of the link from the switch i to the switch j. The number of flows and OpenFlow-enabled switches is N_F and N_L, respectively. The routing matrix $A_{N_L \times N_L \times N_F}$ specifies the path selected for each flow, e.g., if $A^f_{(i,j)} \in \{0, 1\}$ is equal to 1 then the flow f crosses the link or $i \rightarrow j$. The flow requirement matrix $C_{1 \times N_F}$ specifies flows guaranteed requirements based on the corresponding class. The i-th row of the flow requirement matrix defines the guaranteed bandwidth for each flow.

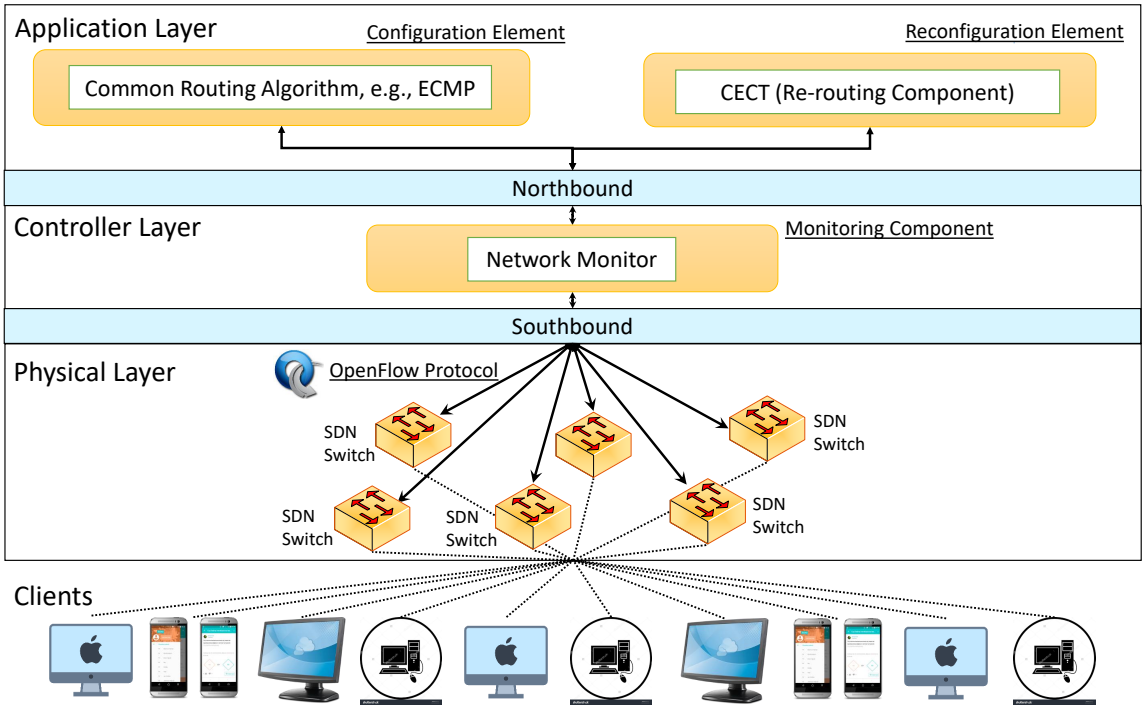


Fig. 1: The proposed architecture.

Table 1: Main Notation.

Symbol	Definition
Mathematical Parameters	
N_L	Number of switches
N_F	Number of flows
B	$N_L \times N_L$ matrix denoting the links bandwidth
R	$1 \times N_F$ vector denoting flows requirement
s	$1 \times N_F$ vector denoting source switch of flows
d	$1 \times N_F$ vector of destination switch of flows
Metaheuristic Parameters	
PO	Population as set of solutions
CH	Chromosome set of x-paths
XP	A gene which is a x-path
V	A set of switches (nodes)
E	A set of edges (links)
R	A set of all x-paths
Decision Variable	
A	$N_L \times N_L \times N_F$ routing matrix
μ	Maximum link utilization

It should be mentioned that CECT is a secondary routing algorithm, which means that there is a primary routing algorithm along with it. In other words, in order to minimize the routing delay of new arrival flows, CECT uses a conventional routing algorithm (as an example, ECMP) to route the flows separately. Thereafter, if the link utilization of some parts of the network exceeds a predefined threshold, CECT algorithm is invoked and some flows are rerouted to prevent network congestion. The average links utilization in different types of networks are different, in [5], several data center traffics are investigated and the flows characteristics are well studied. Based on their study, links with 70 percent and higher utilization are considered as hot-spot links. We followed the same setting. It should be mentioned that using high values as the threshold makes the algorithm more quick while it increases the probability of congestion for burst traffic. On the other hand, considering a low value balances the load across the network while it increases the execution time. As it can be seen in Fig. 1, in our architecture the routing of new arrival flows is done using existing routing algorithms while CECT is used to reroute flows for traffic engineering purposes and minimizing network congestion.

4 Problem Formulation

The main objectives of this paper are to efficiently and dynamically reallocate resources in a way that i) the QoS requirements of different applications are guaranteed, ii) the resource waste and congestion are proactively prevented, and, iii) the computational complexity of rescheduling process is minimized. As a conse-

quence, the routing matrix should be calculated in a way that the mentioned constraints are satisfied. To this end, the routing matrix $A_{N_L \times N_L \times N_F}$ and μ can be obtained such that the network rescheduling overhead is minimized subject to the QoS constraints and the flow conservation constraints. In the following, we present the formulation of the considered problem.

4.1 Capacity Constraint

The link load is guaranteed to be smaller than the maximum target utilization μ by the following constraint:

$$\sum_{f=1}^{N_F} A^f_{(i,j)} R_f \leq \mu B_{(i,j)}, \quad \forall i,j \in \{1,\ldots,N_L\}, \tag{1}$$

Specifically, the l.h.s of (1) calculates the sum of the guaranteed bandwidth of all flows crossing a specific link. The right-hand side specifies the maximum predefined allowable link bandwidth.

4.2 Source and Destination Constraints

The flows are prevented from returning to the source switches via Equation (2). As mentioned earlier, $A^f_{(i,s_f)}$ is one if and only if the flow f crosses the link that connects switch i to the source switch of f called s_f. We then impose the following constraints:

$$\sum_{i=1}^{N_L} A^f_{(i,s_f)} = 0, \quad \forall f \in \{1,\ldots,N_F\}, \tag{2}$$

$$\sum_{i=1}^{N_L} A^f_{(d_f,i)} = 0, \quad \forall f \in \{1,\ldots,N_F\}, \tag{3}$$

For each flow, Equation (2) forces the summation of $A^f_{(i,s_f)}$ (for all i) to be zero. In other words, none of the flows can cross the link between any switch to the source switch of that flow. On the other hand, (3) makes the flows to stay on the destination switches. The aforementioned constraints prevent the flows from entering an invalid switch. Moreover, equations (4) and (5) prevent flows from staying in an invalid switch. These equations force the flows to leave the origin switches and enter to the destination switches, respectively.

$$\sum_{i=1}^{N_L} A^f_{(s_f,i)} = 1, \quad \forall f \in \{1,\ldots,N_F\}, \tag{4}$$

$$\sum_{i=1}^{N_L} A^f_{(i,d_f)} = 1, \quad \forall f \in \{1,\ldots,N_F\}, \tag{5}$$

Equation (4) guarantees the flows to cross from exactly one of the source switch outgoing link. Similarly, equation (5) is considered for the incoming links of destination switch.

4.3 Flow Conservation and Loop Prevention Constraints

If a switch is neither source nor destination of a flow, the flow must leave that switch after it moves in. This restriction is applied by Constraint (6) via balancing the amount of traffic entered to the switch with the amount of traffic left it.

$$\sum_{i=1}^{N_L} A^f_{(i,j)} = \sum_{i=1}^{N_L} A^f_{(j,i)}, \quad \forall f \in \{1,\ldots,N_F\}, \quad \forall i \in \{1,\ldots,N_L\} - \{s_f, d_f\}, \tag{6}$$

$$\sum_{i=1}^{N_L} A^f_{(i,j)} \leq 1, \; \forall f \in \{1,\ldots,N_F\}, \; \forall i \in \{1,\ldots,N_L\}, \tag{7}$$

$$A^f_{(i,j)} \in \{0,1\}, \; \forall f \in \{1,\ldots,N_F\}, \; \forall i,j \in \{1,\ldots,N_L\}, \tag{8}$$

Constraint (7) assures there is no loop in the new routing matrix. It prevents flows from returning to a switch that is met in the past. Finally, we express $A^f_{(i,j)}$ as a binary variable.

4.4 Objective Function

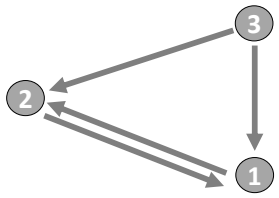
In order to minimize network congestion in case of the existence of burst traffic, the objective function is minimizing the maximum link utilization μ. More formally, we have:

$$\min \quad \mu, \tag{9}$$

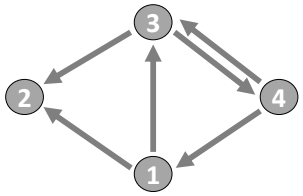
Subject to:

Constraints (1)-(8). (10)

Indeed, we are interested to find the routing matrix A while minimizing μ. The problem belongs to the class of Mixed Integer Linear Programming (MILP) formulations, and it is NP-Hard [16]. Therefore, we rely on a heuristic approach which is detailed in the next section.



(a) Sample topology with 3 nodes.



(b) Sample topology with 4 nodes.

Fig. 2: Topology samples.

5 CECT Algorithm

In this subsection, the meta-heuristic method called CECT (based on the genetic algorithm [13]) is precisely described and represented as an algorithm in Alg. 1. In brief, CECT pre-computes some feasible paths for each flow and assigns a random path to each flow. In this way, a collection of different solutions are constructed and each solution is ranked based on the constraints mentioned in the previous section. Then, CECT uses the roulette wheel algorithm to select some solutions as the ancestors of the next generation. The new population is generated by applying uniform crossover and multipoint mutation on these ancestors. This process is applied to each generation till a solution violating no constraint is found or a predefined threshold for the number of iterations is met.

In Alg. 1, the first line computes all x-paths (all paths with a length lower than x) in an offline manner (more details will be provided in Alg. 2 and Section 5.1). The second line randomly assigns an x-path to each flow based on the source and destination of flows and x-paths. This means that an x-path which has (a, b) as its (source, destination) cannot be assigned to a flow where the (source, destination) is (c, d) if $(c, d) \neq (a, b)$. Line 4 makes a loop until finding a solution which satisfies all constraints. In other words, the algorithm seeks for a solution with no congestion. It should be mentioned that in two cases the loop breaks after some predefined iterations: i) when the requests are more than the resources (i.e., there is no a solution ensuring all constraints), and, ii) when the proposed algorithm can not find the optimal solution (i.e., in order to prevent an infinite loop). In the next step, we find the fitness function (i.e., using function FF which is precisely described in Section 5.3) for each solution (which is called a chromosome, Section 5.2) for current generation (line 5 of the algorithm). Then, some chromosomes are selected using the roulette wheel algorithm (Section 5.3 to produce the next generation (line 6). Note that, the number of the selected chromosomes is equal to the number of population. Line 8 protects the best chromosome (best fitness value) from further changes (which is known as elitism in the context of genetic algorithms). In line 9, the selected chromosomes are sent to the uniform crossover function (Section 5.4) to produce the next generation. During this step, the parents are replaced with their children.

Algorithm 1 CECT Algorithm

INPUT: $G =< V, E >$, *threshold*
OUTPUT: $A^f_{(i,j)}, \forall\, i, j$

1: *Precompute* all x-paths
2: Randomly select feasible path for each flow (paths with similar source and destination with the flow)
3: $itr = 0$;
4: **while** not(Eqs. (1)-(7)) && ($itr <= threshold$) **do**
5: $FF(CH_i)$ for each $CH_i \in PO$
6: $S =$*Roulette_wheel_selection(PO)*;
7: **for** $i = 1; i <= size(S)$; $i+ = 2$ **do**
8: **if** $CH_i \neq best(FF(S))$ **then**
9: $Uniform\_Crossover(CH_i, CH_{i+1})$;
10: $Multipoint\_mutation(CH_i)$;
11: $Multipoint\_mutation(CH_{i+1})$;
12: **if** no improvement in the solutions for k iterations **then**
13: $mut = mut_{max}$;
14: **else**
15: $mut = mut_{min}$;
16: **end if**
17: **end if**
18: **end for**
19: **end while**
20: **return** A^f_{ij}

Line 10 and 11 are supposed to mutate the newly generated population (Section 5.5). It should be mentioned that two reasons may stop the enhancement of the best chromosome: i) falling into a local optimum, and ii) finding the optimal solution. Therefore, in line 13 of the algorithm the mutation rate is increased to mut_{max} if no improvement is seen in the best chromosome after k iterations. We do this because if the population is in a local optimum then increasing the mutation rate helps the algorithm to escape the local optimum. If the the algorithm is not in a local optimum, in line 15 of the algorithm, the mutation rate is returned to mut_{min} to find an optimum. On the other hand, finding the global optimal solution ends the algorithm (line 20 of the algorithm).

Table 2: 3-paths for sample topology depicted in Fig. 2a.

Label	Path
1	{1→ 2}
2	{2→ 1}
3	{3→ 1}
4	{3→ 2}
5	{3→ 2→ 1}
6	{3→ 1→ 2}

Table 3: 3-paths for sample topology depicted in Fig. 2b.

Label	Path	Label	Path
1	{1→ 2}	7	{1→3→2}
2	{2→ 1}	8	{1→3→4}
3	{3→ 2}	9	{3→4→1}
4	{3→ 4}	10	{4→1→3}
5	{4→ 1}	11	{4→1→2}
6	{4→ 3}	12	{4→3→2}

In the following, we first detail a simple case study for solving CECT problem. Then, we detail the different features of CECT which are the structure of the chromosomes and the subroutines of selection, crossover, and mutation.

5.1 Preliminaries

Consider an x-path is a path with y hops $y \leq x$, e.g., in Fig. 2 the set of 3-path is {{1→2}, {2→1}, {3→1}, {3→2}, {3→2→1}, {3→1→2}}. CECT pre-computes all x-paths (for a predefined x) and marks each path as an unique number starting from 1, e.g., {1→2} is marked as 1, {2→1} as 2, and so on. Tables 2 and 3 contain all entries in the set of 3-path for the sample topologies (i.e., see Fig. 2a for Table 2 and Fig. 2b for Table 3, respectively) along with their labels.

Alg. 2 provides the process of pre-computing the x-paths. Lines 2-7 of the algorithm find all x-paths with length one. To this end, for each switch v links that that directly connect a switch to v are added to the set of results R. In the next step, all x-paths with the length of two are added to the results set R (lines 8-12). To this end, considering each path r in the results set R, switches v that have a direct connection to one of the switches in r are added to the set of results $< r, v >$. Thereafter, all 3-paths are considered and so on. At the end, all x-paths with the length of x will be produced.

Algorithm 2 *Precompute* x-paths

INPUT: $G =< V, E >$, $x >= 1$
OUTPUT: $R =< r >$, r is a set of paths
1: $R = \{\}$;
2: **for** each vertex v in V **do**
3: **for** each vertex v' in V **do**
4: **if** v' is a neighbor of v **then**
5: $R = R + \{< v, v' >\}$;
6: **end if**
7: **end for**
8: **for** each r in R **do**
9: **if** (r is a neighbor of v) & ($Length(r, v) <= x$) **then**
10: $R = R + \{< r, v >\}$;
11: **end if**
12: **end for**
13: $V = V - \{v\}$;
14: **end for**

1	6	3	5	2	5	3	4	2	1

Fig. 3: A sample Chromosome for topology depicted in Fig. 2a and 3-paths labeling of Table 2.

5.2 Chromosomes structure

In the context of the genetic algorithm each solution is called a *chromosome*. In the proposed algorithm each chromosome (or simply each solution) is an array of p labels (where p is the number of flows in the network), e.g., if there are 10 flows in the topology illustrated in Fig. 2a, then a sample chromosome for 3-path is depicted in Fig. 3. The first element of this chromosome is 1 which means that the selected path for the first flow is {1→2} (based on Table 2), therefore, the source of the first flow is switch 1 and the destination is switch 2. Similarly, the sixth element is 5 which means that the selected path for the sixth flow is {3→2→1}. It should be mentioned that all elements of Table 2 are calculated based on the topology depicted in Fig. 2a.

5.3 Selection structure

After the initial population is created, each chromosome is ranked based on the constraints violations (named as the *fitness function (FF)*). In this way, the amount of traffic that violates the QoS constraints is measured and a proper penalty is assigned to each flow. Due to the complex nature of resource reallocation problems, outstanding individuals may introduce a bias in the early stage of the algorithm. As a result, the algorithm may get on a local optimum. To solve this issue, the parents selection exploit a roulette wheel algorithm [3], which is also known as fitness proportionate selection.

In roulette wheel, let FF_i be the fitness of individual i in the population. The selection probability of i-th individual is $p_i = \frac{FF_i}{\sum_{j=1}^{N_L} FF_i}$, where N_L is the number of individuals in the population. This selection algorithm is chosen since it discards none of the individuals in the population and gives a chance to all of them. As an example, consider that there are 5 different chromosomes (solutions) which have fitness values of {6.82, 1.11, 8.48, 2.57, 3.08}, respectively. Therefore, the proportional probability of selecting each chromosome is {31%, 5%, 38%, 12%, 14%}, respectively. In a nutshell, the selection process consists of two elements: i) fitness function (reported in Alg. 3); and, ii) roulette wheel selection (detailed in Alg. 4).

In Alg. 3, lines 1-5 compute the imposed traffic to the network by the flow. Line 6 considers the summation of the amount of congestion in all links as the fitness function of the corresponding flow that is returned by the Alg. 3. In Alg. 4, lines 3 and 4 calculate the summation of all chromosome fitness values. The cumulative probability of each chromosome is computed in lines 7-10. The loop in line 11 repeats the following instruction until the number of the selected chromosome reaches a predefined population size. In lines 12-20, a random number between $(0, 1]$ is generated. In particular, a chromosome with the minimum cumulative probability greater than this random number is selected.

Algorithm 3 *FF: Fitness Function*

INPUT: $CH =< XP >$, d, $B =< E, b >$. CH is a set of vertexes (a chromosome), XP is a x-path label, d is the demand size, B is the graph of bandwidth, E is set of edges, and b is the link's capacity
OUTPUT: F. fitness associated with the input chromosome

```
1: for each path XP in CH do
2:     for each edge e in XP do
3:         reduce the bandwidth of edge e in B with d
4:     end for
5: end for
6: F = sum(b);
7: return F
```

5.4 Crossover structure

In genetic algorithms, a crossover is a genetic operator used to vary the features of chromosomes from one generation to the next. The *uniform crossover* uses a fixed mixing ratio between two parents. Unlike one-point and two-point crossover, the uniform crossover enables the parent chromosomes to contribute the gene level rather than the segment level. Therefore, since each chromosome contains several labels, a uniform crossover is exploited in this paper. The corresponding procedure is detailed in Alg. 5. More in detail, a random number

Algorithm 4 *Roulette_Wheel_Selection*

INPUT: PO, which is population
OUTPUT: S, which is selected chromosomes

```
1: sum = 0;
2: S = {};
3: for each chromosome CH in PO do
4:     sum+ = FF(CH);
5: end for
6: sum_pr = 0;
7: for each chromosome CH_i in PO do
8:     pr_i = sum_pr + (FF(CH_i)/sum);
9:     sum_pr += pr_i;
10: end for
11: while Length(S) ≤ Length(PO) do
12:     j = 1;
13:     for j <= 2 do
14:         rn = Random(0,1);
15:         for each chromosome CH_i in PO do
16:             if rn > pr_i && rn < pr_{i+1} then
17:                 S = S + {CH_i};
18:             end if
19:         end for
20:     end for
21: end while
22: return S
```

between $(0, 1]$ is generated for each gene. If the random number is less than a threshold (i.e., 0.5 in [28]), then the first child CH'_1 receives the corresponding gene from the first parent CH_1 and the second child CH'_2 receives the corresponding gene from the second parent CH_2. Otherwise, the first child receives the gene from the second parent while the second child receives the gene from the first parent.

Algorithm 5 *Uniform_Crossover*

INPUT: CH_1, CH_2, which are parents (two chromosomes)
OUTPUT: CH'_1, CH'_2, which are children (two chromosomes)

```
1: for each path in CH'_1 and CH'_2 do
2:     if Random(0,1) < 0.5 then
3:         CH'_1 take the path from CH_1
4:         CH'_2 take the path from CH_2
5:     else
6:         CH'_1 take the path from CH_2
7:         CH'_2 take the path from CH_1
8:     end if
9: end for
10: return CH'_1, CH'_2
```

5.5 Mutation structure

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population to an-

other one. Due to the fact that the probability of falling into a local optimum in resource reallocation problems are high, CECT uses multipoint mutation operator in which several labels of each chromosome are selected and changed randomly as shown in Fig. 4 (i.e., three labels are mutated).

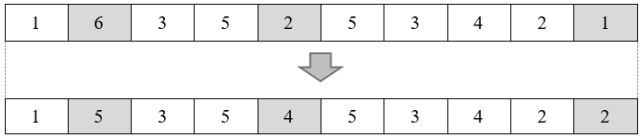


Fig. 4: The multipoint mutation.

Alg. 6 gives an outline of the mutation process. For each gene (which is an x-path label for the corresponding flow), a random number between $(0, 1]$ is generated. If the random number is more than the mutation rate MR (line 3), then the corresponding gene is replaced by a feasible gene (line 6), otherwise it may remain unchanged. Feasible genes are sets of x-paths that have similar source and destination with that gene, e.g., if the source and destination of the i-th gene that is going to be mutated is s_i and d_i, then set of x-paths that have s_i and d_i as their sources and destinations are feasible for this gene.

Algorithm 6 *Multipoint_Mutation*

INPUT: CH, MR, where MR is mutation rate
OUTPUT: CH', which is the mutated chromosome
1: $H' = \{\}$
2: **for** each gene r in CH **do**
3: **if** $Random(0, 1) >= MR$ **then**
4: $CH' = CH' + \{r\}$
5: **else**
6: CH'=CH'+$Random${Feasible Path for this Flow}
7: **end if**
8: **end for**
9: **return** CH'

5.6 Flow table compression structure

In this subsection, in order to reduce the computational complexity of CECT, a technique to reduce the size of the flow table is proposed. For the sake of simplicity, all flows belong to the same pair of source and destination called *SF flows*. There is a large number of small flows (i.e., flows with a size less than 10 Kb/s) [6] in data center networks which can be merged to reduce the computational complexity of the solution. To this end, all SF flows that are smaller than a predefined lower bound are merged. Note that the outcome must be smaller than an upper bound threshold, otherwise it breaks into two or more flows. Interestingly, the increase of the lower bound (fine-grained granularity) reduces the computational complexity while it increases the optimality gap. However, as we will show in the performance evaluation section, the proposed compression method is able to dramatically reduce the number of active flows.

6 Complexity Analysis

In this section the computational and space complexity of the proposed meta-heuristic algorithm is analyzed.

6.1 Computational Complexity

The algorithm is composed of an online part and an offline part (pre-computation of the x-paths). Since the offline part is related to the network topology and is calculated when the network is configured one time for ever, then we ignore this part of the algorithm in our analyze. CECT is composed of four main subroutines: ranking, selection, crossover, and mutation. Consider N_F as the number of flows, itr as the predefined maximum iteration, m as the maximum length of a path, L as the number of links, N_p as the number of chromosomes, and c as the number of pre-computed paths. The computational complexity of ranking and selection parts are $O(N_F \times (m^2 \times \log c + L))$ and $O(N_F)$, respectively [1]. Both the crossover and mutation steps have a computational complexity of the order $N_F \times m$. These subroutines are invoked for each pair of chromosomes, consequently the complexity should multiply by N_p. Since these parts are executed until a valid solution is achieved or a predefined threshold itr is met, in the worst case the computational complexity of CECT is $O(N_p \times N_F \times itr \times (2m + L + m \times L^2 \times \log c))$. In our case, the value of m, c, and itr is selected as 10, 50, and 100, respectively. As a result, computational complexity of CECT is:

$$O(N_p \times N_F \times itr \times m \times L^2 \times \log c).$$

The authors of [2] investigate different approaches of selecting the best population size (number of chromosomes) to be deployed in micro GA algorithms. They propose that the best population size is the square root of the chromosome length. Therefore, considering N as the number of switches, where $N_p = \sqrt{N_F \times \log_2 N}$. So, CECT computation complexity is:

$$O(\sqrt{\log_2 N} \times {N_F}^{3/2} \times itr \times m \times L^2 \times \log c).$$

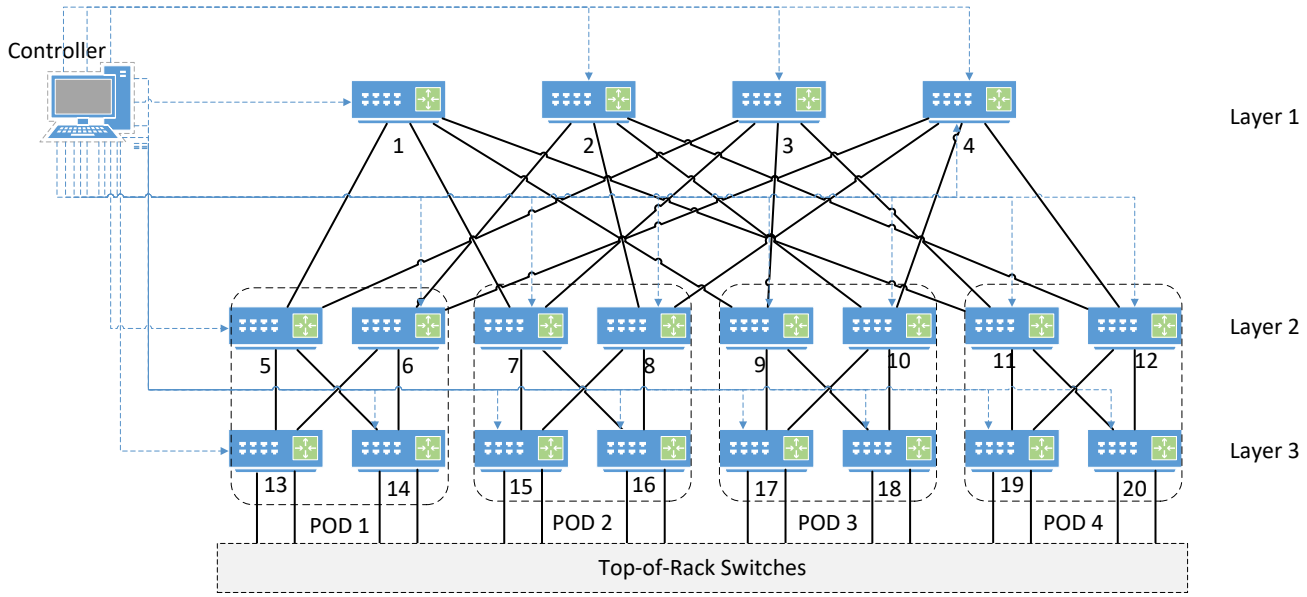


Fig. 5: The Fat-tree topology with $k = 4$.

The values of $\sqrt{\log_2 N}$ and $\log c$ are less than 10 even for huge networks, therefore, the computational complexity of CECT is as follows:

$$O(CECT) \triangleq O({N_F}^{3/2} \times itr \times L^2 \times m). \quad (11)$$

It should be mentioned that m and itr are small values (usually less than 10 and 100, respectively).

6.2 Space Complexity

Consider N_p as the number of chromosomes, IL and CL as the length of an integer variable and a character variable, N_F as the number of flows, m as the maximum length of a path, and c as the number of pre-computed paths. We analyze the space complexity of the offline and the online parts separately. In the offline part, CECT should save the table of x-paths and corresponding labels. Each x-path in the worst case consists of m characters where each character contains a switch name. On the other hand, for each x-path there is a corresponding label which is an integer variable. There are c x-paths in the network, therefore, the space complexity of the offline part of the algorithm is $O(c \times (IL + m \times CL))$.

On the other hand, in the online part of the algorithm, there are N_p chromosomes, each chromosome consists of N_F integer variables (labels). Therefore, the space complexity of the online part of the algorithm is $O(N_F \times N_p \times IL)$.

7 Simulated scenarios and considered performance metrics

In this section, we describe the considered test scenarios and the adopted performance metrics.

7.1 Setup Description

The proposed analytical model is evaluated on the network topology shown in Fig. 5 in order to simulate the SCDC. The illustrated topology called fat-tree topology is a scalable data center network architecture that is universally adopted [19]. All simulations are carried out on a desktop equipped with Intel Core 2 dual 2.6 GHZ CPU and 4.0 GB RAM. In our simulations, we consider 32 port switches (e.g., NEC PF5340-32QP) on the 3^{rd} layer of the topology (switches 13-20) and 48 port top-of-rack switches (e.g., Cisco Catalyst 4948 Switch), this topology can support up to 11.28 K servers. We exploit a network traffic pattern which can be found in [4]. Due to lack of information about the IP layout, we assign hosts to the switches randomly. In addition, the access

and aggregation switches are classified as POD I. The probability of leaving the originating POD for each flow is considered as PLR parameter, e.g., $PLR = 0$ means that all flows stay in their originating POD. We use Mininet [31] along with POX controller [32] to emulate the network. In the sequel, network throughput of ECMP [22] and CECT are discussed to show the impact of the proposed scheme on the network performance. ECMP is selected as a comparisons with our solution (i.e., CECT) for two reasons: i) ECMP and CECT as multi-path routing method distribute packets across multiple links in the network in such away to preserve the load balancing, and, ii) ECMP is considered as an interesting and prevalent real method which is applied in large data centers and it is implemented as a common routing protocol in Mininet [31].

7.2 Performance Metrics

In the carried out simulations, the following three performance metrics have been numerically evaluated:

(i) *Throughput*: it is the rate of successful message delivery over a communication channel in the SCDC;
(ii) *Data Transfer*: it is the average amount of the data transferred through a link in the SCDC;
(iii) *Packet Loss*: it is the network congestion metric that is the percentage of packets lost with respect to packets sent in the SCDC.

It should be mentioned that Wireshark is used to capture the traffic in all of the hosts and switches. Thereafter, all captured traffics are merged to calculate the mentioned parameters.

8 Performance Evaluation and Comparisons

In this section, we test and compare the performance of the proposed CECT algorithm against the corresponding one, namely, the ECMP [22] algorithm.

8.1 Throughput

The first group of carried out tests aims to evaluate and compare the throughput of CECT and ECMP (Equal Cost Multi-Path) [22] routing algorithms. Hence, in our emulation, ECMP considers the path length as the cost of that path which means that it uses the shortest-path algorithm to find paths and distributes the traffic between these paths. The obtained numerical results for the total number of flows are reported in Fig. 6.

In Fig. 6, we increase the total number of flows in the network from 200 flows up to 2000 flows by adding 200 new flows in each iteration and calculate the network throughput. To this end, we run *TCP dump* on all of the hosts in the network and merge these TCP-dumps to calculate the total network throughput. As can be seen, since growing the number of flows increases the network traffic load, the probability of congestion in the network is higher. Therefore, the superiority of CECT is more evident in a large number of network flows. Based on the emulation results, CECT improves the network throughput up to 3x compared to the ECMP.

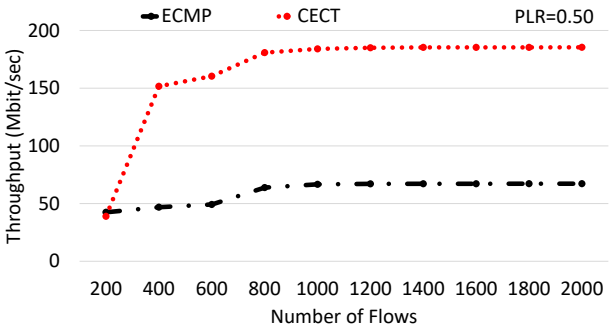
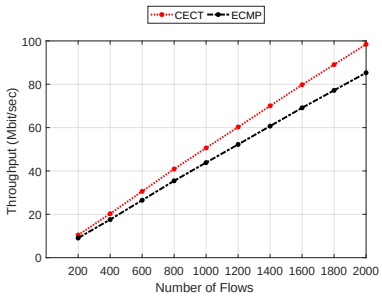


Fig. 6: Total network throughput.

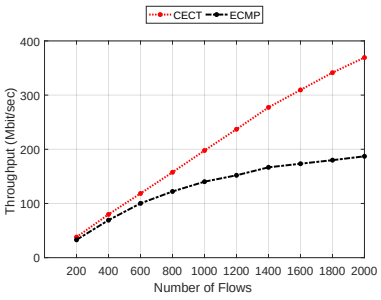
In the following, in order to investigate the impact of the network size and traffic pattern over the proposed scheme, we evaluate the network throughput over a network topology with 45 switches (i.e., fat-tree k=6). Additionally, we implement a network packet generator to generate different traffic patterns. To this end, we use MATLAB programming language to generate the traffic pattern with different flow sizes and simulate the network environment. Furthermore, we use a real network traffic with 4 different traffic patterns where the average rate of flows are micro (Fig. 7a), small (Fig. 7b), medium (Fig. 7c), and big (Fig. 7d). Consider b as the ratio of the flow rate to link bandwidth. We consider flow f a micro flow, if $b_f = 0.005$. Similarly, small, medium, and big flows are flows with b_f equal to 0.02, 0.2, and 0.5, respectively. As can be seen, the superiority of CECT over ECMP increases whether the size or the number of traffic flows increases.

8.2 Data transfer

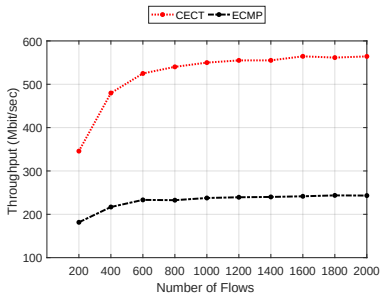
The second group of numerical tests focuses on the communicated traffic in the network in each time interval and the transferred data versus the number of flows between CECT, ECMP routing strategies that are presented in Fig. 8. Hence, Fig. 8a shows the number of bytes that are communicating in each time interval. Based on this figure, the superiority of CECT



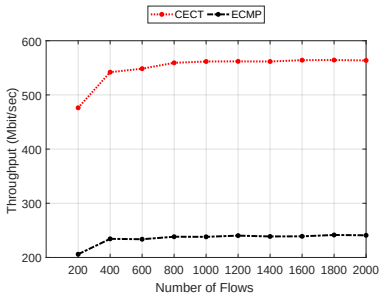
(a) Micro Flows.



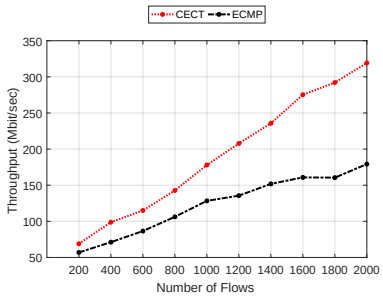
(b) Small Flows.



(c) Medium Flows.



(d) Big Flows.

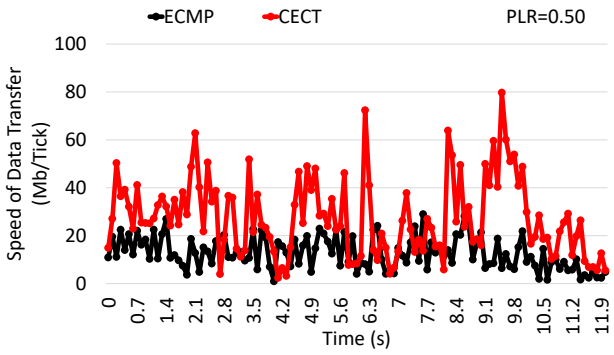


(e) Real Network Traffic.

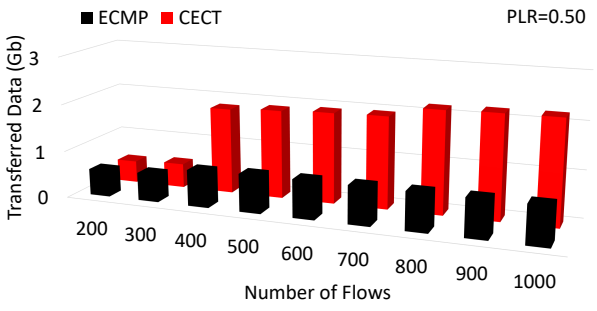
Fig. 7: Network Throughput, fat-tree k=6, 45 switches.

over ECMP (the shortest path algorithm is used as the cost function) is clear. Due to greedy nature of TCP connections, each flow tries to obtain as much bandwidth as possible, therefore, the TCP connections try to reach the maximum speed of data transfer. However, since increasing the traffic rate increases the probability of congestion, the traffic rate rises and then drops periodically. On the other hand, Fig. 8a presents the transferred data versus the number of flows in the network. As can be seen, increasing the number of flows increases the gap between the results of ECMP and CECT. This happens because increasing the number of flows, increases the probability of congestion in the network and makes the impact of the rerouting algorithm clearer.

It should be mentioned that in our emulation, each flow tries to communicate a special amount of traffic (e.g., flow 1: 10 Mb, flow 2: 400 Kb, flow 3: 100 Mb, etc.).



(a) Traffic transferred.vs. time interval.



(b) Traffic transferred.vs. Number of flows.

Fig. 8: Network Traffic.

8.3 Packet Loss

The third group of simulations aims at evaluating the packet loss of the network. The obtained numerical results (expressed in terms of the multiple number of flows) are reported in Fig. 9. The percentage of packet loss versus the number of flows in presented in the mentioned figure. It is clear from the figure that increasing

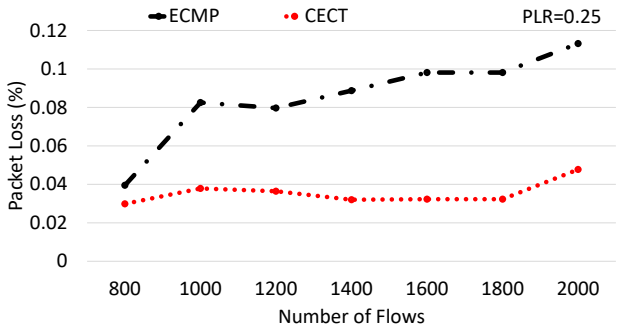


Fig. 9: Packet loss.

the number of flows increases the packet loss in both approaches, however, the percentage of packet loss in the proposed algorithm is sufficiently lower in compared with the packet loss of ECMP. As a result, CECT decreases the packet loss up to 2x compared to traditional approach (i.e., ECMP).

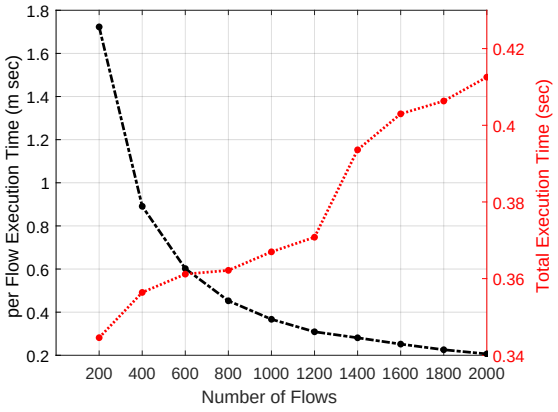
8.4 Execution Time

In order to investigate the impact of network size and flows number on the execution time of CECT, we exploit a network topology with 45 switches and 2000 flows to test CECT and the results are depicted in Fig. 10. In the right side of the figure, *total execution time* of CECT versus the number of flows is presented. Correspondingly, on the left side, *per flow execution time* versus the number of flows is illustrated. As can be seen, the total execution time for 20 and 45 switches are less than 0.42 and 2 second and the execution time per each flow is less than 1.8 and 4.5 milli-second.

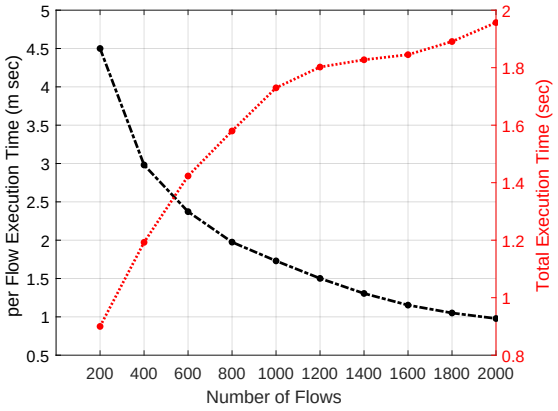
Since the execution times is completely dependant on the programming language, the configuration of the PC which hosts the network controller, and the optimality of the implementation, therefore, we mathematically analyzed the computation and space complexity of CECT in Section 6.

9 Conclusions and Future Work

In this work, an efficient resource reallocating algorithm for software-defined data centers was introduced. In this way, the problem was mathematically formulated and an optimal scheme was proposed to solve the corresponding optimization problem. Since the computational complexity of the proposed solution is high, we proposed a meta-heuristic approach based on genetic



(a) Fat-tree K=4, 20 switches.



(b) Fat-tree K=6, 45 switches.

Fig. 10: Execution time of CECT.

algorithm, called CECT, to propose a sub-optimal solution which has a low computational complexity. The computational complexity of CECT was discussed and showed that it is applicable for real-world networks. Additionally, CECT was compared with ECMP from throughput, data transfer, and packet loss perspective. Emulation results show that CECT improves the total network throughput up to 3x while the packet loss is decreased up to 2x. Future work would be dedicated to minimizing the side effect of network reconfiguration. Additionally, one can minimize the network energy consumption by chaining the objective function. In this way, the mathematical formulation should be extended in a way that some new constraint implement the network energy consumption model.

References

1. Fast proportional selection. URL `http://jbn.github.io/fast_proportional_selection/`. (Date last accessed Jun. 2017)

2. Abu-Lebdeh, G., Benekohal, R.F.: Convergence variability and population sizing in micro-genetic algorithms. Computer-Aided Civil and Infrastructure Engineering **14**(5), 321–334 (1999)
3. Back, T.: Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press (1996)
4. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: 10th ACM SIGCOMM conference on Internet measurement, Melbourne, Australia, pp. 267–280 (2010)
5. Benson, T.A.: New paradigms for managing the complexity and improving the performance of enterprise networks. Ph.D. thesis, The University of Wisconsin-Madison (2012)
6. Benson, T.A.: New paradigms for managing the complexity and improving the performance of enterprise networks. Ph.D. thesis, University of Wisconsin-Madison (2012)
7. Civanlar, S., Parlakisik, M., Tekalp, A.M., Gorkemli, B., Kaytaz, B., Onem, E.: A qos-enabled openflow environment for scalable video streaming. In: IEEE GLOBECOM Workshops (GC Wkshps), Miami, FL, USA, pp. 351–356 (2010)
8. Do, M.T., Jin, J., Wang, H., Man, Z.: Sliding mode learning based congestion control for diffserv networks. IET Control Theory & Applications **10**(11), 1281–1287 (2016)
9. Egilmez, H.E., Civanlar, S., Tekalp, A.M.: A distributed qos routing architecture for scalable video streaming over multi-domain openflow networks. In: 19th IEEE International Conference on Image Processing (ICIP), Orlando, FL, USA, pp. 2237–2240 (2012)
10. Egilmez, H.E., Civanlar, S., Tekalp, A.M.: An optimization framework for qos-enabled adaptive video streaming over openflow networks. IEEE Transactions on Multimedia **15**(3), 710–715 (2013)
11. Egilmez, H.E., Dane, S.T., Bagci, K.T., Tekalp, A.M.: Openqos: An openflow controller design for multimedia delivery with end-to-end quality of service over software-defined networks. In: Asia-Pacific, Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), Hollywood, CA, USA, pp. 1–8 (2012)
12. Egilmez, H.E., Gorkemli, B., Tekalp, A.M., Civanlar, S.: Scalable video streaming over openflow networks: An optimization framework for qos routing. In: 18th IEEE International Conference on Image Processing (ICIP), Brussels, Belgium, pp. 2241–2244 (2011)
13. Eiben, A.E., Raue, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: International Conference on Parallel Problem Solving from Nature, pp. 78–87. Springer (1994)
14. Gholami, M., Akbari, B.: Congestion control in software defined data center networks through flow rerouting. In: 23rd Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, pp. 654–657 (2015)
15. Ghosh, A., Ha, S., Crabbe, E., Rexford, J.: Scalable multi-class traffic management in data center backbone networks. IEEE Journal on Selected Areas in Communications **31**(12), 2673–2684 (2013)
16. Guck, J.W., Reisslein, M., Kellerer, W.: Function split between delay-constrained routing and resource allocation for centrally managed qos in industrial networks. IEEE Transactions on Industrial Informatics **12**(6), 2050–2061 (2016)
17. Kulkarni, S., Sharma, R., Mishra, I.: New qos routing algorithm for mpls networks using delay and bandwidth constraints. International Journal of Information **2**(3) (2012)
18. Leela, R., Thanulekshmi, N., Selvakumar, S.: Multi-constraint qos unicast routing using genetic algorithm (muruga). Applied Soft Computing **11**(2), 1753–1761 (2011)
19. Leiserson, C.E.: Fat-trees: universal networks for hardware-efficient supercomputing. IEEE transactions on Computers **100**(10), 892–901 (1985)
20. Liang, B., Yu, J.: One multi-constraint qos routing algorithm cgea based on ant colony system. In: 2nd International Conference on Information Science and Control Engineering (ICISCE), Shanghai, China, pp. 848–851 (2015)
21. Lu, Y., Zhu, S.: Sdn-based tcp congestion control in data center networks. In: IEEE 34th International Performance, Computing and Communications Conference (IPCCC), Nanjing, China, pp. 1–7 (2015)
22. Martini, L., Rosen, E., El-Aawar, N., Heron, G.: Ieee standard for local and metropolitan area networks– virtual bridged local area networks amendment 13: Congestion notification. IEEE Std 802.1Qau-2010 (Amendment to IEEE Std 802.1Q-2005) pp. c1–119 (2010)
23. Mushtaq, M.S., Fowler, S., Mellouk, A., Augustin, B.: Qoe/qos-aware lte downlink scheduler for voip with power saving. Journal of Network and Computer Applications **51**, 29–46 (2015)
24. Ongaro, F.: Enhancing quality of service in software-defined networks. Ph.D. thesis, University of Bologna (2014). URL `http://amslaurea.unibo.it/id/eprint/7356`

25. Otoshi, T., Ohsita, Y., Murata, M., Takahashi, Y., Ishibashi, K., Shiomoto, K., Hashimoto, T.: Traffic engineering based on stochastic model predictive control for uncertain traffic change. In: IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, Canada, pp. 1165–1170 (2015)
26. Shetty, S., Yuchi, X., Song, M.: Optimizing network-aware resource allocation in cloud data centers. In: Moving Target Defense for Distributed Systems, pp. 43–55. Springer (2016)
27. Shojafar, M., Canali, C., Lancellotti, R., Abawajy, J.: Adaptive computing-plus-communication optimization framework for multimedia processing in cloud systems. IEEE Transactions on Cloud Computing (2016)
28. Su, W., Lagoa, C.M., Che, H.: Optimization-based, qos-aware distributed traffic control laws for networks with time-varying link capacities. Automatica **72**, 158–165 (2016)
29. Tajiki, M.M., Akbari, B., Mokari, N.: Qrtp: Qos-aware resource reallocation based on traffic prediction in software defined cloud networks. In: 8th IEEE International Symposium on Telecommunications (IST), Tehran, Iran, pp. 527–532 (2016)
30. Tajiki, M.M., Akbari, B., Mokari, N.: Optimal qos-aware network reconfiguration in software defined cloud data centers. Computer Networks **120**, 71–86 (2017)
31. Mininet. `https://github.com/mininet/mininet`. [Online; accessed 30-June-2017]
32. PoX. `https://github.com/noxrepo/pox`. [Online; accessed 30-June-2017]
33. Wang, J.M., Wang, Y., Dai, X., Bensaou, B.: Sdn-based multi-class qos-guaranteed inter-data center traffic management. In: IEEE 3rd International Conference on Cloud Networking (CloudNet), Luxembourg, Luxembourg, pp. 401–406 (2014)
34. Zhang, L., Tizghadam, A., Bannazadeh, H., Leon-Garcia, A.: Iterative traffic engineering in the data plane of multimedia ip communications. In: 2nd IEEE NetSoft Conference and Workshops (NetSoft), Seoul, South Korea, pp. 107–111 (2016)
35. Zhao, J., Ge, X.: Qos multi-path routing scheme based on acr algorithm in industrial ethernet. In: Third International Conference on Communications, Signal Processing, and Systems, pp. 593–601 (2015)