Dynamic Optimization of Load-Balancing and Reconfiguration Overhead in SD-ISP Networks

S. Tomovic, *Student Member, IEEE,* and I. Radusinovic, *Senior Member, IEEE*

.

***Abstract* — In this paper, we propose a new traffic engineering (TE) approach for software-defined Internet Service Provider (SD-ISP) networks that strives to maximize the network throughput and provide adequate QoS (Quality of Service) with a minimal reconfiguration cost. In contrast to the conventional TE approaches, which perform the network optimizations periodically and control the side effects of reconfigurations by carefully choosing the period length between the optimization cycles, we use a bi-objective optimization model that minimizes maximum link utilization and the reconfiguration overhead. A new heuristic algorithm has been proposed in order to generate the approximated Pareto frontier for the bi-objective optimization model, while the Lyapunov drift-plus-penalty algorithm is used to select the most appropriate solution from the approximated Pareto set. Our simulation study shows that the proposed approach adjusts to the ISP's constraint on time-average reconfiguration rate by trading the throughput performance efficiently. Since the reconfiguration overhead is reduced, the network controller could be allowed to optimize resource allocation more frequently, in order to quickly and efficiently respond to the network changes. The paper analyses the impact of the reconfiguration rate constraint, average flow duration and the frequency of TE on the overall network performance.**

***Key words* — QoS, SDN, traffic engineering.**

I. Introduction

With the expansion of Internet of Things (IoT) and the increasing confluence of augmented reality and artificial intelligence, providing quality of service (QoS) guarantees becomes one of the crucial challenges for Internet Service Providers (ISPs). In practice, ISPs avoid QoS issues by over-provisioning backbone links 2 to 3 times relative to the offered load [1], [2]. Since this approach cannot be economically justified in the long run, various Traffic Engineering (TE) mechanisms have been proposed to realize enhanced network performance through efficient usage of network resources [3].

The emergence of Software Defined Networking (SDN) redefined the traditional network architecture by separating the control plane from the data plane. This created opportunity to implement more sophisticated TE techniques, since SDN controller is able to continuously track the network state and reconfigure the data-plane dynamically [1-7]. In order to ensure an optimal network operation and meet QoS requirements, SDN controller must respond to the network events (e.g. arrivals and departures of traffic demands, link and switch failures) in real time. This requires solving complex traffic optimization problems within tight time constraints and reconfiguring flow-tables of SDN switches [7]. However, frequent network reconfigurations adversely impact the network performance [8]. Thus, there is a key problem to balance between the routing optimality and the reconfiguration frequency. The conventional way to cope with the above problem is to perform throughput optimizations periodically (offline) and use greedy QoS-aware routing algorithm for online path computation [6,7]. Increasing the period between TE optimizations means better reliability and stability, but the network keeps a sub-optimal configuration longer. In [8], the authors propose a control policy that decides when to apply a solution of the iterative optimization solver, i.e. when to reconfigure the network. The control policy minimizes a total flow-allocation cost while respecting the network reconfiguration "budget". An important limitation of this approach is the assumption that the network events occur according to a stochastic process of known characteristics. On the other side, [9] proposes an exact model that optimizes traffic distribution while minimizing the number of flow-table updates. However, the optimization objectives are always equally weighted, and the computational complexity of the proposed algorithm is unacceptably large.

This paper is a revised and expanded version of [10], where we have proposed a new approach for scalable TE in SD-ISP networks. In the proposed approach, SDN controller periodically reconsiders a current load-balancing pattern and solves a bi-objective optimization problem to improve it. The optimization model involves two conflicting gaols: i) minimizing maximum link utilizations (MLU), and ii) minimizing the number of flow-table updates. Since this problem is NP-hard [9], we propose a heuristic algorithm to generate an approximated set of Pareto efficient solutions [11]. To choose the most desirable solution from the approximated Pareto set at each optimization instant, we use a drift-plus-penalty algorithm [12]. We have validated the benefits of our proposal over a periodic TE scheme concerned with MLU minimization only. The obtained results show that the proposed solution

Paper received March 31, 2019; revised June 03, 2019; accepted June 11, 2019. Date of publication July 31, 2019. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Grozdan Petrović.

This paper is revised and expanded version of the paper presented at the 26th Telecommunications Forum TELFOR 2018 [10].

S. Tomovic, Faculty of Electrical Engineering, Podgorica, University of Montenegro, Dzordza Vasingtona bb, 81000 Podgorica, Montenegro (e-mail: slavicat@ucg.ac.me).

I. Radusinovic, Faculty of Electrical Engineering, Podgorica, University of Montenegro, Dzordza Vasingtona bb, 81000 Podgorica, Montenegro (e-mail: igorr@ucg.ac.me).

achieves a very satisfying QoS request rejection ratio while being able to meet SD-ISP's requirement with regard to a time-average reconfiguration rate. Moreover, when the reconfiguration "budget" is large enough, it often outperforms the competitive approach.

The rest of the paper is organized as follows. Section II explains the system model as well as assumptions made in our analysis. Sections III presents the simulation methodology and the obtained results. The concluding remarks are given in Section IV.

II. The Proposed SDN Controler Design

The considered SD-ISP network consists of the Core (CR) and Provider Edge (PE) data-plane devices. PE devices are OpenFlow-enabled [13], while SDN capability is not necessarily required at CR routers. A static set of up to K paths (tunnels) is used for traffic delivery between ingress-egress (IE) pairs of PE nodes. We assume that ISP offers two Bandwidth-on-Demand (BoD) services:

1. QoS1 - provides bandwidth guarantees.
2. QoS2 - guarantees very low delay bound and the required bandwidth for time-critical data transfers.

Ingress switches classify incoming traffic according to those two classes. Each class is uniquely labelled.

SDN controller has two modules responsible for flow allocation: *Online Routing* and *Offline-TE*. The users' requests are initially handled by the *Online Routing* module. *Offline-TE* module periodically optimizes the network performance by redistributing aggregated traffic demand of each IE pair over (up to) K pre-configured tunnels. The traffic splitting across the tunnels is performed on a flow level basis, by applying a hash function on packets' header fields. Unequal traffic splitting ratios could be implemented by using the SELECT group-tables [12] on OpenFlow switches. Flow-table of an ingress switch maps incoming packets to one of the total two group-tables, based on the destination address and the label attached. Each group-table corresponds to a different traffic class (QoS1 or QoS2), and is configured with up to K tunnels and K weights, which determine the ratio of the matching traffic to be sent over each of the tunnels.

A. Online Routing

This module performs admission control for BoD requests by running a computationally simple QoS routing algorithm. In the first step, the algorithm checks whether a bandwidth requirement can be met with load balancing weights determined by the *Offline-TE* module in the last running cycle. If yes, the request is accepted. The ingress switch will direct the flow to the appropriate group-table, and tag and balance its sub-flows over multiple pre-configured tunnels. Otherwise, the algorithm seeks for an eligible tunnel to allocate the requested bandwidth. If multiple tunnels are eligible, the preference is given to the one with the largest load-balancing weight.

B. Offline-TE

The *Offline-TE* module periodically (every T seconds) analyses the network state and identifies the most appropriate set of actions to improve it. The optimality of TE decision depends on two opposed criteria: the resulting MLU, and the number of reconfigurations (i.e. group table updates) required. There is no unique solution that simultaneously optimizes both of them. Instead, there exists a set of so-called Pareto optimal solutions (Pareto frontier) [11]. In the rest of the section, we explain the procedure of generating the Pareto optimal solutions and selecting the most suitable one to be applied.

1) Generation of approximated Pareto frontier

For the considered TE problem generating the Pareto optimal solutions is NP-hard [9]. Therefore, in this paper we propose a heuristic method for obtaining the approximated Pareto frontier. The pseudo-code of the heuristic is given in Algorithm 1. We start by solving the linear programming (LP) optimization model that minimizes MLU for the current traffic matrix [6]. The model can be solved in polynomial time if delay-matching tunnels for both traffic classes are known in advance. Then, we determine the number of reconfigurations α that the optimal MLU (MLU^{opt}) requires (line 2). The α value is computed as the number of group-table updates needed to apply the solution of LP solver. Since MLU^{opt} most likely could be achieved with a much smaller number of reconfigurations than α, in the next steps, we try to approach MLU^{opt} by rerouting as few as possible traffic demands. One of the input arguments of the algorithm is N_{max} – which defines the upper bound on the number of iterations where the LP model [6] is run. By dividing α with N_{max} we obtain the "resolution" of the solution space, i.e. the set of possible reconfiguration overheads $O^* = \{\alpha_1^*, \alpha_2^*, ... \alpha_{N_{\max}}^*\}$ in the approximated Pareto frontier is obtained as multiple of α_{step} (lines 7-9). Then, we iterate over the elements of O^* sorted in the ascending order, and determine the corresponding number of traffic demands that will be reconsidered for rerouting in the LP MLU minimization model. This is done by calling *getCriticalDemands()* function (line 14). We assume that SD-ISP also specifies the upper bound on the reconfiguration rate over any time interval T, which we denote with R_{max} in the Algorithm 1. The elements of O^* that impose the reconfiguration rate higher than R_{max} are excluded from the further analysis.

The goal of *getCriticalDemands()* function is to find traffic demands that can contribute the most to reducing MLU in the network. By a "traffic demand" we assume a total offered load of IE pair that shares the same QoS policy. At the beginning of each algorithm iteration, the solution set *LPdemands* inherits the list of demands selected in the previous iterations (*prevDemands*). Therefore, in each call, the function *getCriticalDemands()* actually searches for α_{step} "critical demands", while α_n^* - α_{step} demands are already known. In order to find a new "critical demand", we firstly identify the most loaded link based on the network state information (line 26). Then, the demand which loads this link the most is considered "critical" and added to the *LPdemands* set, provided that it has not been already added in the previous iterations (lines 27-28). After the critical demand is determined, the

network state information is updated by adding the bandwidth occupied by that demand. Namely, the existence of this demand is ignored till the end of the function call (line 30). After the α_{step} critical demands are added to *LPdemands* set, the LP model is used to minimize MLU by redistributing the "critical" load (line 16). The resource allocation for the other (non-critical) demands is left unchanged. Based on the solution of the LP model, network state information is updated again so as to reflect the network state in case when the computed load balancing scheme is applied (line 18). The resulting MLU value (MLU_n^*), in pair with α_n^*, makes an approximated Pareto optimal solution. Thus, the resulting approximated Pareto frontier (PF) after *N* iteration becomes:

$$PF = \left\{(MLU_1^*, \alpha_1^*), (MLU_2^*, \alpha_2^*) ... (MLU_N^*, \alpha_N^*)\right\}.$$

***Algorithm 1**: Heuristic*

***Inputs**: networkInfo* = network statistics; *demands* = aggregated demands between IE pairs; N_{max}= maximum number of iterations; R_{max} = upper bound on the reconfiguration rate.
Output: *PF* = aproximated Pareto frontier; *#minimizeMLU* = optimization model [6]; *#LBwieghts* =load balancing weights as a solution of the MLU optimisation model [6]

```
1:  MLU^opt , LBweights = minimizeMLU(demands)
2:  α = countReconfigs(MLU^opt)
3:  PF = { }
4:  PF.add((MLU^opt, α ))
5:  α_step = α / N_max
6:  O*={ }
7:  for i in range (1, N_max ) :
8:      O*.add( i · α_step )
9:  O*.sort_ascending()
10: prevDemands={}
11: for α_n* in O* do:
12:     if α_n* / T > R_max: break
13:     LPdemands = prevDemands
14:     cDemands = getCriticalDemands( α_step , LPdemands)
15:     LPdemands.add(cDemands)
16:     MLU_n*, LBweights = minimizeMLU(LPdemands)
17:     PF.add(( MLU_n*, α_n* ))
18:     updateNetworkState(LBweights)
19:     prevDemands = LPdemands
20: function: getCriticalDemands( α_step , LPdemands):
21:     results={}
22:     dNum=0
23:     stats = netwokInfo.copy()
24:     while dNum< α_step do:
25:         worstLink = getMostLoadedLink()
26:         critical_demand = worstLink.getMaxDemand(stats)
27:         if critical_demand in LPdemands do: continue
28:         results.add(critical_demand)
29:         #neglect load of critical_demand
30:         stats.remove(critical_demand)
31:         dNum+=1
32:     return results
```

2) The decision making

A solution (MLU_n^*, α_n^*) from the PF set requires α_n^* updates of group-tables and introduces the MLU optimality gap:

$$\Gamma = MLU_n^* - MLU^{opt} \quad (1)$$

We are interested in minimizing an average MLU optimality gap while keeping the time-average reconfiguration rate below R_{avg} - which is the parameter defined by SD-ISP [9]. To achieve the mentioned goal, we model the reconfiguration rate constraint as a virtual queue, with dynamics over time slots $t \in \{0, T, 2T, ...\}$ as:

$$V(t+T) = \left\lceil V(t) - R_{avg} \right\rceil^+ + \alpha^*(t)/T \quad (2)$$

In the above equation, V(t) is a queue size at a given time *t,* while $V(t+T)$ is expected queue size at time *t*+*T*, when *Offline-TE* module will run again. The service rate of the queue is equal to the desired time-average reconfiguration rate R_{avg}. The number of queue arrivals during the time interval [*t*, *t*+*T*] corresponds to the number of reconfigurations that *Offline-TE* module decides to perform at time *t,* i.e. $\alpha(t)^* \in O^*$. The initial queue size V(0) is set to 0. It has been proven that such a virtual queue could be stabilized by minimizing the Lyapunov drift function [12], which captures the changes in the queue backlog from one slot to the next:

$$L(t) = V(t+T)^2 - V(t)^2 \quad (3)$$

However, while minimizing $L(t)$ helps to meet the time-average constraint for $\alpha(t)$, the average MLU optimality gap might become unacceptably large over time. Thus, instead of choosing the *PF* solution that minimizes $L(t)$, we choose the one that minimizes drift-plus-penalty function $A \cdot \Gamma(t) + L(t)$. The nonnegative constant *A* is used to emphasize the importance of the objective function $\Gamma(t)$, which captures changes of MLU optimality gap over time slots $t \in \{0, T, 2T, ...\}$. By increasing *A* we are able to push MLU optimality gap towards 0, at the expense of increasing an average queue size linearly proportional to *A*.

The complexity of the proposed algorithm can be controlled with $N_{max.}$ While conventional MLU minimization TE approach requires solving LP optimization model only once, the proposed approach solves the MLU minimization model up to N_{max} times. However, it should be noted that in each iteration MLU minimization is performed over a reduced demand set, i.e. load balancing is optimized only for "critical" demands.

III. THE SIMULATION RESULTS

In order to evaluate the performance of the proposed approach, we built a simulator in Python that uses CPLEX [14] to solve the optimization problems. We used POP (Points of Presence)-level Sprint ISP topology [6], where link capacities were set to 1000 units, while delay on each link was derived from a great circle distance between the connected POPs and the speed of light in the fiber. Equal

link capacities were assumed based on [15]. Since this information is not up-to-date, we express link capacities in abstract units. Traffic traces for Sprint ISP network are not available to research community. Thus, we used the gravity model [16] to generate an average value of traffic matrix for the hour of the maximum load during a day. By scaling the gravity model matrix, we derived an "admissible" traffic matrix, which brings MLU to at most 70%, provided that flows are not delay-sensitive. Based on the "admissible" traffic matrix, we simulated dynamic arrivals of BoD requests. In particular, BoD requests have been generated according to the Poisson distribution, whereas the bandwidth demand of each request is randomly chosen from the set of [3, 5, 7] units. The mean rate of the BoD request arrivals during the hour of the greatest load is computed such that the resulting traffic model in average converges to the "admissible" traffic matrix. The duration of generated traffic demands was modeled with an exponential distribution, with the mean of 20 minutes by default. The delay bound for QoS2 requests was set to 35ms. The number of tunnels between each IE pair (K) was bounded by 10. The analysis has shown that using a lager number of tunnels does not lead to noticeable improvement of service capacity. The paths for the tunnels have been computed with Yen's K-shortest path algorithm [17], whereby the computation is performed subject to the constraint that each link can be traversed by two tunnels of the same IE pair, at most. We consider this link-overlapping constraint beneficial for ensuring a better performance in case of link failures, which are frequent in ISP networks [18]. The duration of simulations was set to one day, and BoD request arrival rate during the day was scaled as illustrated in Fig 1.

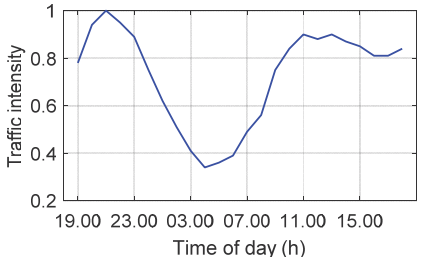

Fig. 1. Traffic load pattern.

We compared the proposed TE approach with the TE approach that minimizes MLU periodically (PTE). The simulations were run for two different values of the period length between MLU optimizations: T=5min and T=1h. The first setting is used as a representative of dynamic SDN-based TE [1], while the latter setting corresponds to the timescale of TE in conventional ISP networks, according to [19]. For our approach, which we here denote as TEBRO (TE with Balanced Reconfiguration Overhead), we limit the maximum number of iterations (N_{max}) to 100, which was sufficient to achieve a near optimal performance. For this setting, the running time of the algorithm was around 15s in the worst case. In the further analysis, T is always assumed to be 5 minutes for the proposed solution, unless stated otherwise.

We provide the results in terms of the rejected bandwidth demand, time-average reconfiguration rate per minute, and averaged MLU in Fig. 2, Table 1 and Fig. 3, respectively. We measured the reconfiguration rate as the rate of group-table updates initiated by the *Offline-TE* module, as those updates affect aggregated traffic demands. The results are shown as a function of the percentage contribution of QoS2 requests to the total number of BoD requests. The R_{max} parameter for our approach was set to 200/T, while R_{avg} was set to 1.6 reconfigurations per minute. The R_{avg} value was chosen based on the results obtained with PTE (T=1h), which is considered the current practice solution, as mentioned before. By running multiple simulations for different values of A and R_{avg} parameters, we found out that for $A = 5 \cdot 10^5$ the proposed TE approach offers the most efficient tradeoff between the average MLU optimality gap and the reconfiguration rate. By decreasing the A parameter, we can further reduce the reconfiguration overhead, but the MLU optimality gap increases. From Fig. 2, it is interesting to note that for a sufficiently large A the proposed approach mostly outperforms the PTE (T=5min) in terms of all performance metrics analyzed. This can be attributed to the fact that TEBRO strives to optimize load balancing for the most demanding IE pairs, to which the largest number of BoD requests refers. This results in a lower averaged MLU when compared to PTE approach, even though TEBRO TE algorithm does not produce optimal MLU in each run (Fig. 3). The performance of PTE degrades as the percentage of QoS2 traffic increases, because load balancing is mostly limited to a small number of low-latency paths, whereas transmission bottlenecks are created very often.

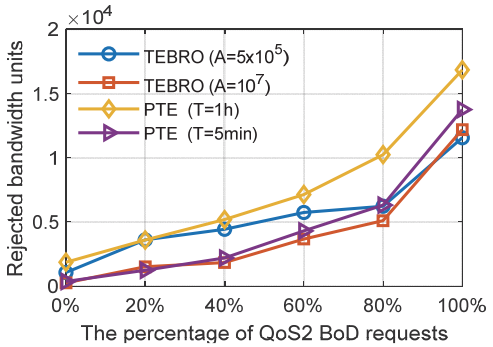

Fig. 2. The comparission of PTE and TEBRO in terms of the amount of rejected bandwidth demand.

TABLE I
AVERAGE RECONFIGURATION RATE PER MINUTE.

QoS2 ratio	PTE (T=5min)	PTE (T=1h)	TEBRO ($A = 5 \cdot 10^5$)	TEBRO ($A = 10^7$)
0%	15.44	1.43	1.57	1.72
20%	21.51	1.98	1.63	1.82
40%	21.69	2.15	1.58	1.62
60%	22.13	2.25	1.63	1.78
80%	22.37	2.47	1.68	1.98
100%	7.14	0.72	1.29	2.43

From Table 1, it can be seen that TEBRO ($A = 5 \cdot 10^5$) reduces reconfiguration overhead up to 13 times compared to PTE (T=5min). Moreover, it achieves a lower reconfiguration rate than PTE (T=1h), which performs TE 6 times less often. Further reducing the frequency of TE in PTE approach would result in the inability to respond to

sudden traffic distortions. On the other side, TEBRO has means to control the reconfiguration overhead, thus, the SDN controller can be allowed to optimize resource allocation more frequently. In the rest of the paper, we assume that A always equals $5 \cdot 10^5$ for the proposed approach, as this setting meets the requirement in terms of time-average reconfiguration rate.

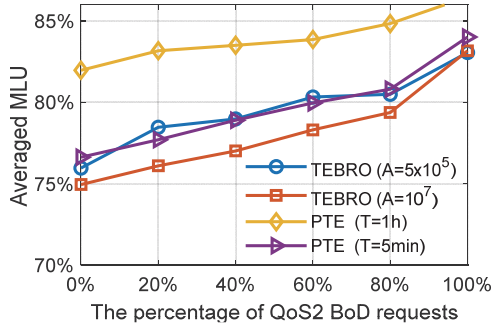

Fig. 3. The comparission of PTE and TEBRO in terms of averaged MLU.

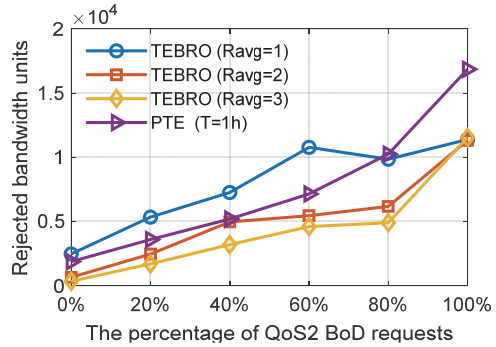

Fig. 4. The impact of R_{avg} on rejected bandwidth demand.

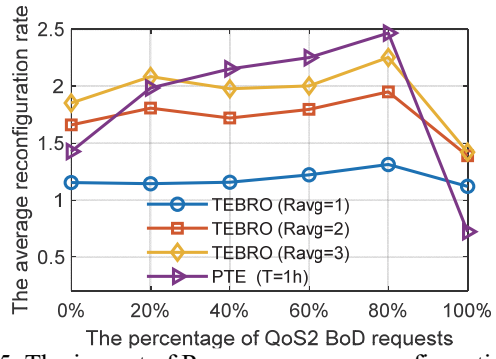

Fig. 5. The impact of R_{avg} on average reconfiguration rate.

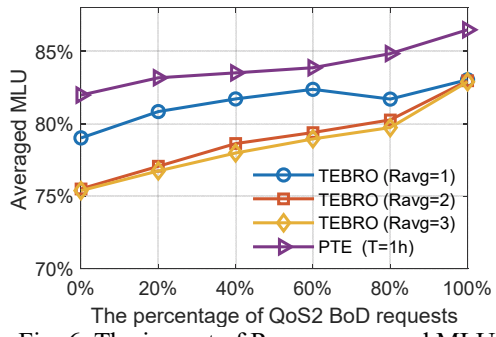

Fig. 6. The impact of R_{avg} on averaged MLU.

The impact of R_{avg} parameter on TEBRO performance in terms of rejected bandwidth demand, average reconfiguration rate and averaged MLU is illustrated in Figs. 4-6. The simulations have been performed for three different R_{avg} constraints: 1, 2 and 3. In general, increasing R_{avg} results in a smaller amount of rejected bandwidth and lower averaged MLU. The results from Fig. 5 confirm that TEBRO is a promising solution to meet even very strict reconfiguration rate constraints imposed by ISP. In all of the analyzed scenarios, the reconfiguration rate averaged over the period of 24 hours was just slightly above the given R_{avg} limitation, in the worst case.

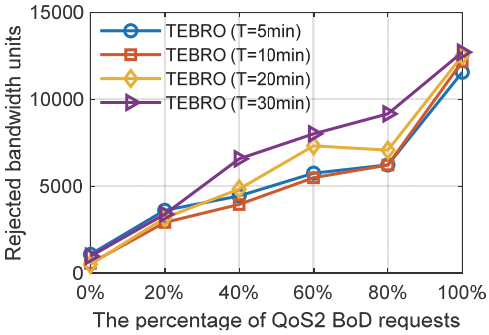

Fig. 7. The impact of T on rejected bandwidth demand.

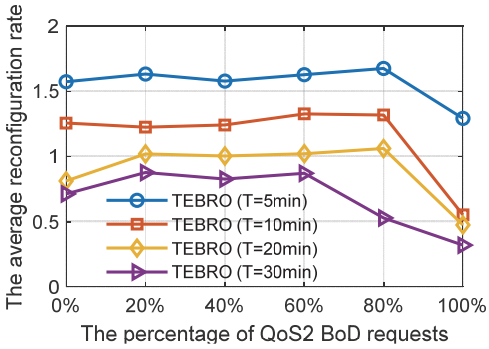

Fig. 8. The impact of T on average reconfiguration rate.

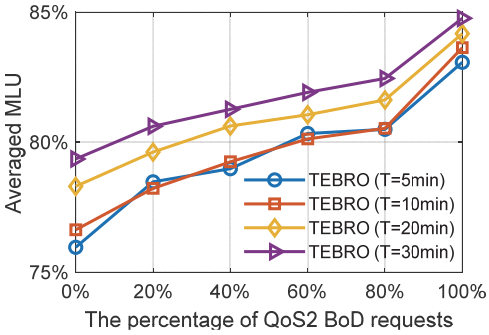

Fig. 9. The impact of T on averaged MLU.

We have also analyzed the impact of the period length (T) between successive runs of *Offline TE* module on TEBRO performance. Simulations were performed for T=5, 10, 20 and 30 minutes, and the obtained results are shown in Figs. 7-9. As can be seen from Fig. 7, large T values generally increase the BoD request rejection ratio, but reduce the average reconfiguration rate. This is mainly because *Offline TE* module optimizes load balancing weights only based on the current traffic matrix. Thus, the optimality of the routing scheme decreases rapidly over time, as a consequence of emerging traffic fluctuations. The results from Fig. 8 are in line with the conclusions that optimal MLU could be achieved by reconfiguring load balancing weights of a relatively small number of IE pairs. It is interesting to note that increasing T from 5 to 10 minutes does not lead to the increase of rejected bandwidth

demand (and averaged MLU), and helps to further reduce the reconfiguration overhead. However, in order to determine an optimal value for *T*, many other factors should be considered, such as average BoD request duration and traffic variability.

Finally, Figs. 10-12 show how the -average traffic flow duration (FD) affects performance of PTE (T=1h) and TEBRO (T=10 minutes). As expected, the performance of both aproaches degrades when flow duration decreases in relation to *T*. However, the performance gap between TEBRO and PTE increases for smaller values of FD. This suggests the importance of dynamic TE optimizations under variable traffic load.

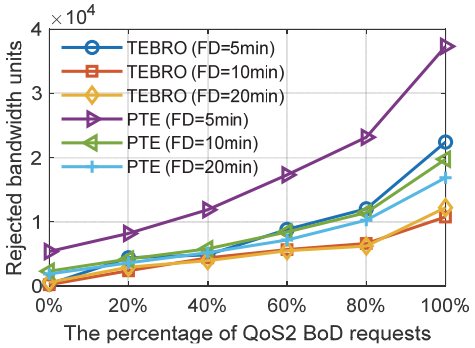

Fig. 10. The impact of flow duration (FD) on rejected bandwidth demand.

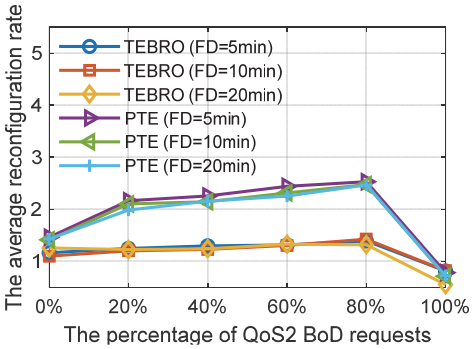

Fig. 10. The impact of flow duration (FD) on average reconfiguration rate.

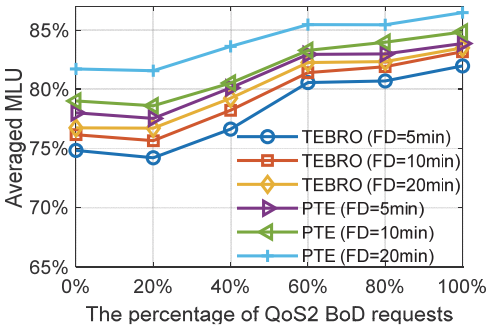

Fig. 11. The impact of flow duration (FD) on averaged MLU.

IV. Conclusion

In this paper, we propose a new TE approach for BoD service provisioning in SD-ISP networks. The proposed approach strives to balance the network resource consumption in order to maximize the BoD request acceptance ratio, considering as a constraint the ISP's preference in terms of the time-average reconfiguration rate. Since the analysed TE problem is NP-hard, we have proposed a heuristic method to dynamically find an efficient tradeoff between the MLU optimality gap and the reconfiguration overhead. The obtained results show that the proposed approach maintains an acceptable MLU optimality gap with a signifficantly reduced reconfiguration overhead compared to a conventional MLU minimization TE scheme.

In our future work, we plan to generalise the proposed QoS provisioning model to include more diverse requirements in terms of delay, bandwidth and reliability. Also, a distributed version of TEBRO algorithm will be developed, that can be deployed on a hierarchiacal SDN control plane, such as [20].

References

[1] C.Y. Hong et al., "Achieving high utilization with software-driven wan," Proc. of SIGCOMM '13, pp.15–26, 2013.

[2] S. Jain et al., "B4: Experience with a globally-deployed software defined WAN", in Proc. of ACM SIGCOMM, vol. 43, no. 4, pp. 3–14, 2013.

[3] A. Mendiola et al., "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Comm. Surveys Tutorials*, vol.19, no.2, pp. 918–953, 2017.

[4] J. L. Martins and N. Campos, "Short-sighted routing, or when less is more," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 82-88, October 2016.

[5] S. Tomovic and I. Radusinovic, "Fast and efficient bandwidth-delay constrained routing algorithm for SDN networks," in Proc. IEEE NetSoft Conference (NetSoft), pp. 303-311, 2016.

[6] S. Tomovic and I. Radusinovic, "Traffic engineering approach to virtual-link provisioning in software-defined isp networks," 2017 25th Telecommunication Forum (TELFOR), pp.1–4, Nov 2017.

[7] S. Tomovic, I. Radusinovic, "An effective use of SDN for virtual-link provisioning in ISP networks", *IEICE Transactions on Communications*, Vol.E102-B, No.4, pp.855-864, April 2019.

[8] S. Paris, A. Destounis, L. Maggi, G. S. Paschos and J. Leguay, "Controlling flow reconfigurations in SDN," IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications, San Francisco, CA, 2016, pp. 1-9.

[9] Usman Ashraf, "Rule Minimization for Traffic Evolution in Software-Defined Networks", *Communications Letters IEEE*, vol. 21, no. 4, pp. 793-796, 2017.

[10] S. Tomovic and I. Radusinovic, "Dynamic Optimization of Load-Balancing and Reconfiguration Overhead in SD-ISP Networks," 2018 26th Telecommunications Forum (TELFOR), Belgrade, 2018, pp. 1-4.

[11] P. Ngatchou, A. Zarei and A. El-Sharkawi, "Pareto Multi Objective Optimization," Proc. of the 13th Int. Conf. on Intelligent Systems Application to Power Systems, 2005, pp. 84-91.

[12] Michael J. Neely. Stochastic Network Optimization with Application to Communication and Queueing Systems. Morgan & Claypool, 2010.

[13] Open Networking Foundation - OpenFlow v1.4 specification. [Online]. Available: https://www.opennetworking.org/, 2017.

[14] CPLEX Optimization Studio. [Online]. Available: http://www-3.ibm.com/software/products/en/ibmilogcpleoptistud

[15] The Internet Topology Zoo database. [Online]. Available: http://www.topology-zoo.org/dataset.html

[16] A. Gunnar, M. Johansson, and T. Telkamp, "Traffic matrix estimation on a large IP backbone: a comparison on real data", In *Proc. of the 4th SIGCOMM*, pp.149-160, Taormina, Italy, 2004.

[17] Yen, Jin Y. (1970). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks". Quarterly of Applied Mathematics. 27 (4): 526–530.

[18] H. H. Liu et al., "Traffic engineering with forward fault correction," SIGCOMM Comput. Comm. Rev., vol. 44, no. 4, pp. 527–538, Aug. 2014.

[19] J. He and J. Rexford, "Date: Distributed adaptive traffic engineering," in In Proc. IEEE INFOCOMM, 2006.

[20] M. Moradi, Y. Zhang, Z. M. Mao, and R. Manghirmalani, "Dragon: Scalable, flexible, and efficient traffic engineering in software defined isp networks," IEEE Journal on Selected Areas in Communications, vol. 36, no. 12, pp. 2744–2756, Dec 2018.