Distributed Load Balancing From the Edge in IP Networks

Youcef Magnouche[1], Pham Tran Anh Quang[1], Jérémie Leguay[1], Xu Gong[2], Feng Zeng[2]
Huawei Technologies Ltd., [1]Paris Research Center, France., [2]Dongguan Research Center, China.

***Abstract*—To improve bandwidth utilization in IP networks, flow aggregates are typically split over multiple paths. In this context, we propose a fully distributed load balancing mechanism that operates only from the edge. Each source is able to determine the split ratios based on already available link state information so as to minimize the maximum link utilization in the network. Without extra signaling, our solution provides a feasible load balancing at each iteration and diminishing returns until convergence to a stable state. Through numerical results on a wide variety of instances, we show that it converges to a near-optimal solution in a few iterations. Thanks to packet-level simulations on an SD-WAN scenario, we also compare its performance in a dynamic environment over centralized and legacy load balancing solutions.**

I. Introduction

Traffic engineering (TE) plays a crucial role in optimizing the use of network bandwidth as it helps to maintain a good load balancing in the network over time [1]. Consequently, the Quality of Service (QoS) in terms of end-to-end delay and packet loss can be enhanced while network resources are efficiently utilized. The most popular load balancing mechanism, e.g. equal-cost multipath routing (ECMP) [2], uniformly divides traffic across multiple paths between the origin and the destination. To further improve load balancing, uneven flow splitting mechanisms have been proposed (e.g., [3]). In this case, load balancing weights are used to control the amount of traffic sent over each path.

In the literature, centralized methods such as Niagara [4] or IRSR [5] have been proposed to control load balancing weights so as to minimize a linear cost or minimize the Maximum Link Utilization (MLU). However, these solutions require the presence of a centralized network controller, which may not be desirable for scalability, fault-tolerance, or deployment reasons.

Decentralized load balancing solutions [6], [7], [8], [9], [10] have been proposed to dynamically adjust routing in case of congestion or link failure. For instance, CONGA [6], Contra [8], and HULA [7] can dynamically adjust at ingress nodes load balancing policies based on path-based measurements to reduce network congestion. While these approaches are trying to adapt path selection, they actively involve core nodes to piggy back measurements inside user traffic and do not explicitly optimize a global objective function.

Designing an exact distributed algorithm for the load balancing problem is not obvious by the fact that one source node lacks information (split ratios, paths, traffic demands) about the other source nodes. Without an exchange of information between source nodes, independent decisions may increase the network MLU and lead to link capacity violation. In [11], authors apply Lagrangian relaxation to decompose the load balancing problem for the control of multicast sources and use the sub-gradient algorithm to solve the Lagrangian dual problem. However, the algorithm does not guarantee the satisfaction of the link capacity constraints during all iterations. Indeed, authors use a method given in [12] to construct a feasible solution only at the end of the sub-gradient iterations. In [10], the authors present a distributed algorithm to generate feasible splits of traffic at each iteration so as to minimize a global convex objective. Each intermediate node decides, for all traffic aggregates (i.e., each tunnel), the split ratios over its outgoing links. Therefore, the nodes collaboratively decide both routing and load balancing. As all nodes are involved in the decision process, i.e., both edge and core nodes, this solution may not be practical in some cases due to complexity issues.

In this paper, we present a distributed load balancing solution that operates only from the edge, i.e. from access devices. Each edge device manages a set of Origin-Destination (OD) tunnels and decides, in collaboration with other edge devices, target split ratios in order to minimize the network MLU. This solution is particularly useful in overlay networks such as SD-WAN networks [13], [14] where several paths, related to different access technologies or underlay tunnels, can be used by access routers to load balance traffic. The main advantages of our solution are threefold: 1) it only relies on already available link state information and does not require any additional signaling to coordinate load balancing decisions, 2) it ensures a feasible solution at each iteration so that no traffic is lost due to the violation of capacity constraints, and 3) it improves load balancing with diminishing returns over iterations until it converges to a stable state where load balancing cannot be further improved.

In the rest of the paper, we first present in Sec. II our distributed load balancing solution and we formulate the overall optimization problem. Then, we introduce in Sec. III our heuristic algorithm to ensure anytime feasibility and diminishing returns to a stable state. Finally, we demonstrate in Sec. IV through extensively simulations its performance on a variety of network scenarios. In particular, we show on static instances with publicly available topologies that, in most cases, our algorithm converges to the optimal solution found with a centralized model. Furthermore, through NS-3 [15] simulations on an SD-WAN network, we show that the MLU is significantly improved compared to ECMP and the centralized solution. Thanks to smooth modifications of the split ratios, we also show that the QoS in terms of end-to-end delay and packet loss is improved over the centralized version. Finally, Sec. V concludes the paper.

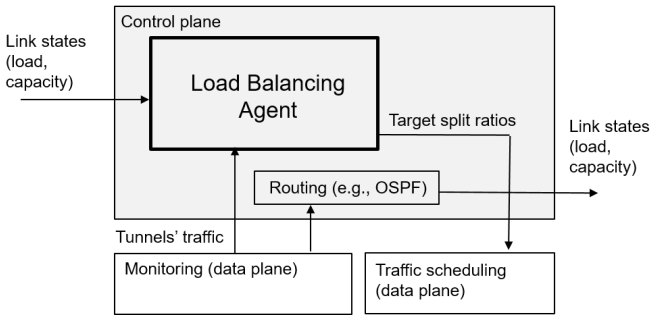


Fig. 1: Device architecture.

II. System Description

We consider a distributed architecture where only edge devices are controlling user traffic.

A. Device Architecture

As illustrated in Fig. 1, edge devices are equipped with load balancing agents that collectively aim at minimizing the MLU of the network. Each agent manages a set of *OD flows* for which they are the origin, also called *tunnels* in the rest of the paper. Tunnels can be split over multiple paths and target split ratios (or load-balancing weights) are continuously updated by agents. The set of candidate paths used by an agent for a given tunnel can be provided by a local or an external path computation module.

Thanks to a link state protocol (i.e., OSPF or any other protocol), each agent periodically receives updates about the network state. In particular, the link states contain link loads as feedback from past load balancing decisions. Link states can be related to physical links or overlay links. They may also include link capacities if they are not given a priori, or if they evolve over time because of some background traffic. Agents also receive updates about the traffic demand in each of the tunnel they handle from the local monitoring.

The computation of target split ratios is performed by every source device each time new information is received. It takes as input the set of candidate paths for each tunnel, updated traffic information for each tunnel, and updated link state information. Once new target split ratios are decided, they can be used to take routing decisions every time a new micro-flow arrives. The decision is made so as to move actual split ratios towards the targeted ones. In some cases, if advanced data plane mechanisms such as FlowLets [16] are used, micro-flows can also be re-routed during their lifetime. As it creates more opportunities to select paths, it accelerates the convergence of split ratios towards their target.

Fig. 2 describes the timeline of the proposed mechanism. Every period of duration τ, agents update their information about the traffic demand for all the tunnels they manage. At a higher frequency, link-state updates are periodically received and agents calculate new target split ratios. The frequency of traffic demand update is intentionally lower compared to link-state updates so that the load balancing algorithm can work on a *stable* optimization problem and performs several iterations before updating the problem inputs. In practice, measurements about the traffic of tunnels and about link loads need to be averaged over a time window to avoid instabilities. The actual split ratios are modified every time the size of individual flows evolves and when new paths are selected at every micro-flow or FlowLet arrival.

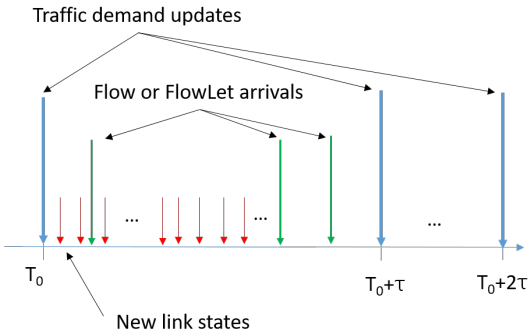


Fig. 2: Timeline from the viewpoint of an edge device.

B. Load Balancing Problem

The load balancing problem consists in minimizing the maximum link utilization defined for a set A of links as

$$\texttt{MLU}(A) = \max_{a \in A} \{\frac{V_a}{C_a}\}$$

where V_a and C_a are the amount of traffic and the capacity of link a, respectively. Let us denote by $\texttt{LU}_a = \frac{V_a}{C_a}$ the link utilization of $a \in A$.

We formally present the compact formulation of the problem that can be solved in a centralized manner. Let's consider a network $G = (V, A)$, where V is the set of nodes, A the set of arcs and $C : A \to \mathbb{R}_+$ a function giving the capacity of each arc. Let K be the set of tunnels, i.e. user tunnels or flow groups, and $d : K \to \mathbb{R}_+$ be the function that indicates the traffic demand of each tunnel. Let $P^k = \{p_0^k, \dots, p_{|P^k|}^k\}$ be a set of available paths for tunnel $k \in K$. We assume that for all $k \in K$, $|P^k| \geq 2$. Otherwise, the split of traffic is trivial and the associated tunnel can be removed from the optimization problem.

The problem can be formulated using the variables

- $x_p^k \in [0, 1]$: split ratio on path $p \in P^k$ for tunnel $k \in K$.
- $\theta \in \mathbb{R}_+$: maximum link utilization of A.

The load-balancing problem is equivalent to the following linear program:

$$\min \quad \theta$$

$$\sum_{p \in P^k} x_p^k = 1 \qquad \forall k \in K, \quad (1)$$

$$\sum_{k \in K} d_k \sum_{p \in P^k, a \in p} x_p^k \leq \theta C_a \qquad a \in A, \quad (2)$$

$$0 \leq x_p^k \qquad \forall k \in K, \forall p \in P^k, \quad (3)$$

Constraints (1) guarantee that all traffic is split on the paths and Constraints (2) permit to compute the MLU.

III. Distributed Load Balancing Algorithm

In this section, we now propose a heuristic for the distributed resolution of the load balancing problem. The algorithm is fully-distributed, i.e., only the source node of each tunnel decides on the split ratios over its outgoing paths independently from the other source nodes. It also ensures *anytime feasibility*, i.e., that a feasible solution is produced at each iteration.

From the compact formulation introduced in Section II-B, we show how to adapt this model to obtain small independent sub-problems that can be solved in a distributed manner. We prove the convergence of the algorithm and we illustrate its operation on toy examples.

A. Preliminary Definitions

The heuristic, described in the next subsection, starts from an existing solution and modifies it to improve (decrease) the $\texttt{MLU}(A)$ of the network until reaching a *stable state*, i.e, a state that cannot be further improved by the edge device. To formally define the *stable state*, let $A^k \subseteq A$, for every $k \in K$, be the set of links composing all paths of P^k, i.e. $A^k = \{\, a \in A \mid \exists\, p \in P^k \text{ such that } a \in p\}$. A solution is called a *stable state* if for each $k \in K$:

- every path $p \in P^k$ contains a link reaching $\texttt{MLU}(A^k)$, or
- there exists a path $p \in P^k$ containing a link reaching $\texttt{MLU}(A^k)$, with a split ratio equal to 0.

Claim 1. *There must exist an optimal solution of* (1)-(3) *that is a stable state.*

Consider an existing load balancing solution such that $\overline{x}^k_p \in [0,1]$ represents the split ratio vector and $\overline{\texttt{LU}} \in \mathbb{R}^A_+$ the associated utilization of links. For all $k \in K$, let $\overline{\theta}^k = \max_{a \in A^k}\{\overline{\texttt{LU}}_a\}$ and $\overline{\theta} = \max_{k \in K}\{\theta^k\}$. In the following fully-distributed algorithm for the load balancing problem, the source node of every tunnel $k \in K$ attempts to minimize $\texttt{MLU}(A^k)$ while guaranteeing that $\texttt{MLU}(A)$ does not increase whatever the decision of the other source nodes.

Source node of tunnel k is able to compute the new $\texttt{MLU}(A^k)$ using $\overline{\texttt{LU}}$, $\overline{x}^k$ together with the new split ratios x^k. This information is available locally.

$$\texttt{MLU}(A^k) = \max_{a \in A^k}\{\overline{\texttt{LU}}_a - \sum_{p \in P^k, a \in p} \frac{d_k \overline{x}^k_p}{C_a} + \sum_{p \in P^k, a \in p} \frac{d_k x^k_p}{C_a}\}$$

To ensure a fair sharing of the residual capacity under the constraint that the $\texttt{MLU}(A)$ does not increase, we artificially divide the residual capacity of links by the total number of tunnels possibly using each link. Then, the source node of every tunnel must satisfy the following link capacities

$$C_a \overline{\texttt{LU}}_a + \frac{C_a(\overline{\theta} - \overline{\texttt{LU}}_a)}{|K|+1}, \quad \forall a \in A$$

In this case, $\texttt{MLU}(A)$ of the new solution must be lower than or equal to $\overline{\theta}$.

For each tunnel $k \in K$, let $\theta^k \in \mathbb{R}_+$ be the maximum link utilization of A^k and let $\mathcal{P}_k$ be the following linear program

$$\min \quad \theta^k$$

$$\sum_{p \in P^k} x^k_p = 1 \tag{4}$$

$$\overline{\texttt{LU}}_a + \sum_{p \in P^k, a \in p} \frac{d_k}{C_a}(x^k_p - \overline{x}^k_p) \le \theta^k \qquad \forall a \in A^k, \tag{5}$$

$$\sum_{p \in P^k, a \in p} (x^k_p - \overline{x}^k_p) \le \frac{C_a(\overline{\theta} - \overline{\texttt{LU}}_a)}{d_k(|K|+1)} \qquad \forall a \in A^k, \tag{6}$$

$$0 \le x^k_p \qquad \forall p \in P^k. \tag{7}$$

Constraints (4) guarantee that all traffic of tunnel k is split over paths of P^k. Constraints (5) permit to compute $\texttt{MLU}(A^k)$ and Constraints (6) guarantee that the new $MLU(A) \le \overline{\theta}$.

Let x^{*k} and θ^{*k} be the optimal solution of $\mathcal{P}_k$.

Proposition 1. *For all $k \in K$, $\theta^{*k} \le \overline{\theta}$.*

Proof. It is easy to see that, solution $\overline{x}^k$ is a feasible solution of $\mathcal{P}_k$. This implies that θ^k is at least equal to $\overline{\theta}$. □

Proposition 2. *Solution x^* represents a stable state unless there exists $k \in K$ such that $\theta^{*k} < \overline{\theta}^k$.*

Proof. Let us consider the contrary, x is not a stable state and for all $k \in K$, $\theta^{*k} = \overline{\theta}^k$. By definition of a stable state, there must exist a tunnel $k \in K$ such that

- there exists a path $p \in P^k$ with $\overline{\texttt{LU}}_a < \overline{\theta}^k \le \overline{\theta}$ for all $a \in p$, and
- every path containing a link reaching $MLU(A^k)$ has a non-zero split ratio. Let $P^k_{mlu} \neq \emptyset$ be the set of these paths.

It follows that

$$\delta = \min_{a \in p}\{\frac{C_a(\overline{\theta} - \overline{\texttt{LU}}_a)}{d_k(|K|+1)}\} > 0$$

.

By adding δ to x^{*k}_p and substracting $\frac{\delta}{|P^k_{mlu}|}$ to every path in P^k_{mlu}, we obtain a feasible load balancing solution with $MLU(A^k) < \theta^{*k}$. This contradicts the optimality of x^*. □

B. Distributed Algorithm

From Propositions 1 and 2, we propose the distributed algorithm detailed in Alg. 1. It uses the two following functions:

1) *ReceiveLinkUtilization():* this function returns the link utilization for all links in the network. In practice, it can be received from the OSPF protocol that exchanges link state information between routers.
2) *SolveLinearProgram($\mathcal{P}_k$):* this function solves the linear program $\mathcal{P}_k$ using any linear solver.

Algorithm 1: Distributed heuristic for load balancing problem

Data: $G = (V, A)$, K, P, d, C and $\overline{x}$.
Result: Split ratio vector x.

```
while x̄ is not a stable state do
    for k ∈ K do
        LU_a ← ReceiveLinkUtilization();
        θ̄^k ← max_{a∈A^k} {LU_a};
        θ̄ ← max_{a∈A} {LU_a};
        x̄^k ← SolveLinearProgram(P_k);
    end
end
x ← x̄;
```

Theorem 1. *Algorithm 1 converges to a stable state.*

Proof. From Proposition 1, the $MLU(A)$ does not increase during all iterations of Alg.1. Moreover, from Proposition 2, at each iteration of the algorithm, there exists one tunnel $k \in K$ such that $MLU(A^k)$ decreases. This is enough to show the result. □

C. Illustration with toy examples

Figure 3 displays four network configurations that we now use to illustrate the operations of our algorithm. In cases 1-3, there are two tunnels, the blue from node 1001 to node 1002 and the purple from node 1003 to node 1002. In case 4, there is an additional brown tunnel from node 1004 to node 1003. All traffic demands are 100Mb/s. Dashed lines represent paths. All links adjacent to nodes 1001, 1002, 1003 and 1004 have infinite capacities, and all other links have a capacity of 100Mb/s. The values on paths represent the amount of traffic in Mb/s.

- case 1 : link $(1, 3)$ reaches the $MLU(A) = 90\%$. At the next iteration of Algorithm 1, source node 1001 will move $16.66Mb/s$ from path p to path p' while source node 1003 moves $6.66Mb/s$ from path p to p'. Then, $MLU(A)$ decreases to $76.66Mb/s$ reached by link $(2, 3)$.
- case 2: represents a stable state, since the blue tunnel has a path with a link reaching the $MLU(A)$ and the split ratio is equal to 0. Moreover, all paths of the purple tunnel have at least one link reaching $MLU(A)$.
- case 3 : illustrates a traffic loss since $MLU(A) > 1$. At the next iteration of Algorithm 1, source node 1001 moves $60Mb/s$ from path p to p'. Then $MLU(A)$ decreases from 190% to 130%.
- case 4 : Since the algorithm is distributed, the source of every tunnel will move $15Mb/s$ from path p to p'. Even if they add traffic together on link $(1, 3)$, the $MLU(A)$ decreases from 90% to 75%.

IV. Numerical Results

The distributed algorithm and the compact model described in the previous sections have been implemented in C++, using CPLEX as a LP-solver. They were tested on an Intel(R) Xeon(R) CPU E5-4627 v2 of 3.30GHz with 504GB RAM and 32 cores, running under Linux 64 bits. A maximum of 32 threads have been used.

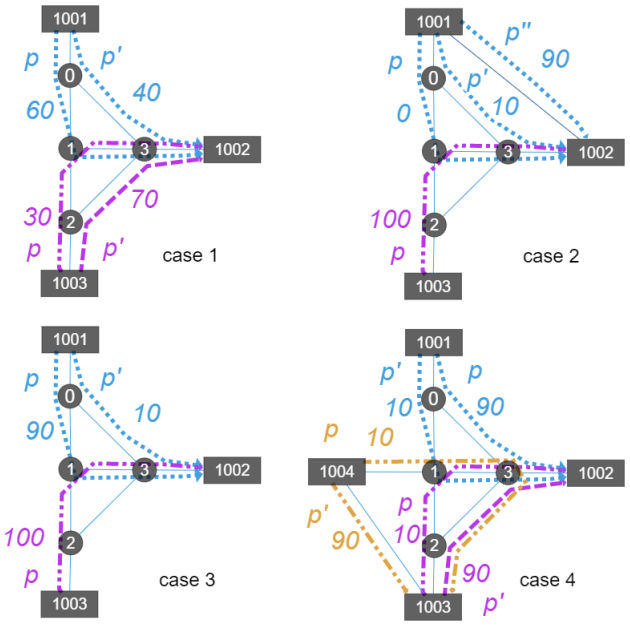


Fig. 3: Example of load balancing configurations.

In this section, we present evaluation results on 1) static instances in a simplified network environment and on 2) a dynamic SD-WAN instance in the NS3 network simulator [15].

A. Evaluation on Static Instances

In this first evaluation, the goal is to evaluate the convergence of the algorithm. We compare its performance against a centralized solution solving the compact model to optimal. We use three types of synthetic instances:

1) Publicly available instances: Abilene, BtEurope and Geant from SNDLIB [17], and Colt_Teleco, GTS_CE, ITC_Deltaco and Kentucky, US_Carrier from Internet topology zoo [18].
2) A SD-WAN instance with one headquarter site and three sites (see Fig. 5)
3) Two large IPRAN instances that are typical from radio access networks.

Link capacities and the traffic demand of tunnels are generated randomly. Except for SD-WAN where tunnels are between sites and the headquarter, sources and destinations of tunnels are picked at random in all the instances. The maximum number of paths generated per tunnel is 10. In practice, fewer paths are used.

For this static evaluation, we start from an initial solution calculated with a *greedy algorithm*. All the traffic of each tunnel is routed over only one path chosen at random. In such a case link capacity may be initially violated. In the presentation of results, a link utilization exceeding 1 means that there is a

Instances	|A|	|V|	|K|	MLUd	MLUc	MLUg	CPUd	CPUc	#Iter
Abilene	28	11	11	**0.64**	**0.64**	0.78	0.05	0.00	5
BtEurope	74	24	24	**0.79**	**0.79**	0.81	0.02	0.00	2
Colt_Teleco	354	153	15	0.48	**0.44**	1.11	0.08	0.00	20
Geant2012	122	40	40	**0.70**	**0.70**	1.49	0.08	0.00	12
GTS_CE	386	149	14	**0.22**	**0.22**	0.57	0.06	0.00	14
ITC_Deltaco	322	113	11	**0.35**	**0.35**	0.51	0.04	0.00	8
Kentucky	1790	754	75	**0.37**	**0.37**	0.37	0.06	0.01	1
SDWAN	46	15	15	0.46	**0.44**	1.77	0.06	0.00	13
US_Carrier	378	158	15	**0.84**	**0.84**	0.85	0.02	0.00	3
IPRAN_1	1114	485	500	**0.30**	**0.30**	0.42	1.69	0.09	77
IPRAN_2	1086	477	500	**0.27**	**0.27**	0.37	1.21	0.10	51

TABLE I: Numerical results comparing the distributed heuristic and the centralized method.

traffic loss. Algorithm 1 stops when a stable state is obtained (see Sec. III-A).

In Table I, the characteristics for all instances are given and results are provided with the following column heads:

- MLUd, CPUd and #Iter for the distributed algorithm. It shows the maximum link utilization of the best solution, the CPU execution time in seconds when sub-problems are solved by batches of 32, and the number of iterations obtained until the algorithms stops.
- MLUc and CPUc for the centralized model. It shows the maximum link utilization of the optimal solution and the CPU execution time in seconds, obtained by solving the compact model.
- MLUg : the maximum link utilization given by the greedy solution at the initialization.

Bold values indicate the approaches giving the best MLU.

Table I and Fig. 4 display the numerical results associated with SND-LIB, SD-WAN and IPRAN instances. We notice that in 81.8% of the instances Alg. 1 finds the optimal solution after few iterations while the maximum optimality gap for the remaining instances is 9%. Even if the centralized method is faster than Alg. 1 on a single machine, the latter necessitates a few iterations to converge to a stable state. Indeed, the number of iterations is lower than 8 in 81.8% of the instances while the maximum is 77 for the remaining instances. In practice, Alg. 1 is distributed on all edge devices and the duration of each iteration depends on the frequency of link states reception.

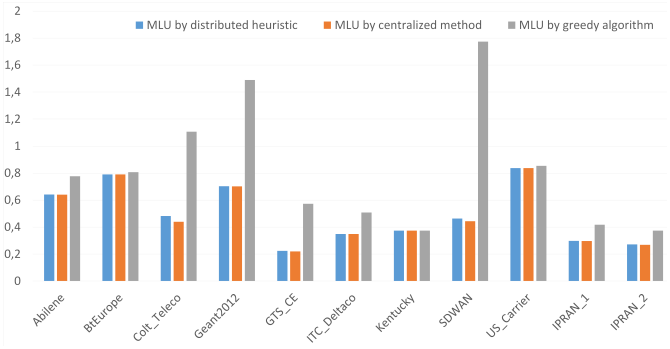


Fig. 4: Comparison of the MLU given by distributed heuristic, the centralized method and the greedy algorithm.

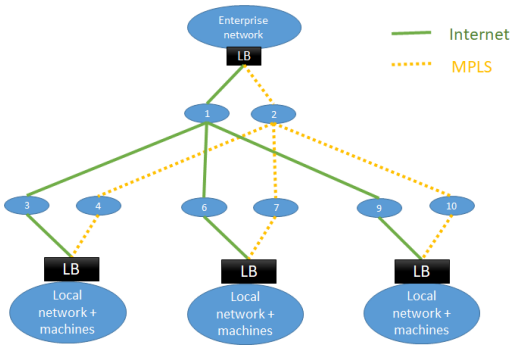
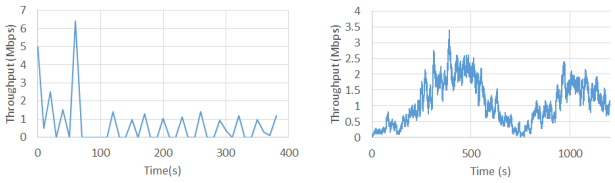


Fig. 5: SD-WAN scenario with 1 headquarter and 3 sites connected through broadband Internet and MPLS. Load Balancing (LB) agents are deployed on access routers with two ports (Internet, MPLS).

B. *Network Simulations with NS3*

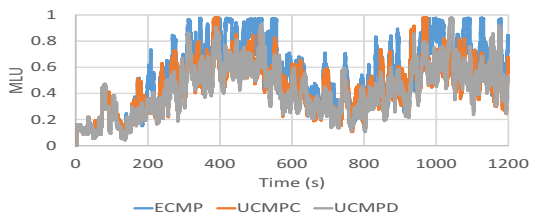
Beside numerical results for the convergence, we conducted a number of simulations using NS3 [15] with OpenFlow 1.3 module [19] to evaluate the performance of the algorithms under dynamic traffic. Fig. 5 describes the simulation scenario. We consider the SD-WAN scenario, where we have a single headquarter (HQ) connected to several remote sites, e.g. 3 sites in our simulation. In this scenario, we created 6 tunnels where 3 are from HQ to sites and 3 are from sites to HQ. The aggregate traffic pattern of each tunnel is diurnal and is formed by video flows. The pattern of each video flow and the total traffic pattern are shown in Fig. 6. Note that we generate diurnal traffic from video flows by adjusting the inter-arrival time of video flows over time.



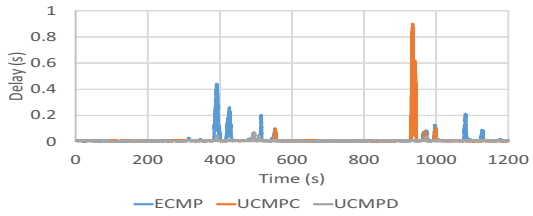
(a) Video flow traffic (b) Total traffic

Fig. 6: One video flow and aggregated traffic for one tunnel.

Link states are measured and broadcasted periodically. The source of each tunnel, therefore, is able to collect them. Link states are measured every 100ms and averaged over a 1s window to avoid fluctuations. The source of each tunnel measures the current traffic and averages it over a 5s window to also avoid fluctuations. The traffic demands and link states are the inputs of Alg. 1 to compute the new target split ratios. In this simulation, we consider that each load balancing agent updates the traffic demand every 30 s and receives link state updates every 100ms. When a new micro-flow arrives, it is assigned to a path to minimize the difference between target and actual split ratios. The path of this micro-flow is converted into forwarding rules that are deployed at every switches in our implementation.



(a) MLU



(b) End-to-end delay

Fig. 7: Evolution of the MLU and the end-to-end delay.

Fig. 7 shows the evolution over time of MLU and end-to-end delay for all OD flows. Table II provides a summary of results for ECMP, UCMPC (centralized model), and UCMPD (our distributed solution) in terms of MLU, end-to-end delay, and end-to-end packet loss rate with average and 95-percentile figures. We can observe that ECMP plots the worst performance for both performance metrics. Indeed, ECMP splits the traffic equally on all available paths without considering the traffic demand and capacity of the paths. Consequently, it has the highest MLU, which leads to congestion and high end-to-end delay. For UCMPC, the centralized algorithm, the MLU is 6% lower than ECMP on average. Interestingly, our distributed solution UCMPD leads to slightly better performance than UCMPC, with an MLU 10% lower than ECMP, because it frequently updates split ratios (every 100 ms instead of 30s). UCMPD is more agile in reacting to the congestion in the network, but it also generates smooth updates of split ratios that improve QoS metrics, thanks to the capacity constraints in program $\mathcal{P}_k$. The average end-to-end delay of UCMPD is similar to UCMPC and 50% lower than of ECMP. The end-to-end packet loss of UCMPC and UCMPD are equal (0.002%) and much lower than ECMP (0.006%).

V. Conclusion

We have proposed a fully-distributed load balancing solution that only operates from the edge using already available link state information to minimize the MLU. The algorithm converges to a stable state with diminishing returns and generates a feasible solution at each iteration. We demonstrated through numerical results and network simulations that in most cases, it converges to an optimal solution after few iterations. This distributed solution significantly helps to improve the QoS in terms of delay and packet losses compared to legacy load balancing solutions like ECMP.

Further work may include improving the convergence using the exact number of tunnels using each link and the resolution of a single linear at each source node.

MLU			
	ECMP	UCMPC	UCMPD
Average	0.53	0.454	0.426
95-percentile	0.96	0.76	0.74
End-to-end delay (ms)			
	ECMP	UCMPC	UCMPD
Average	18.99	4.62	4.84
95-percentile	104.63	5.37	5.44
End-to-end packet loss			
	ECMP	UCMPC	UCMPD
Average	0.006%	0.002%	0.002%

TABLE II: Average and 95-percentile for MLU, end-to-end delay and end-to-end packet loss in the SD-WAN scenario.

References

[1] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[2] D. Thaler and C. Hopps, "RFC 2991: Multipath issues in unicast and multicast next-hop selection," 2000.

[3] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers," in *Proc. ACM EuroSys*, 2014.

[4] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, "Efficient traffic splitting on commodity switches," in *Proc. ACM CoNEXT*.

[5] P. Medagliani, J. Leguay, M. Abdullah, M. Leconte, and S. Paris, "Global optimization for hash-based splitting," in *Proc. IEEE GLOBECOM*, 2016.

[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "Conga: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, p. 503–514, Aug. 2014.

[7] C. H. Benet, A. J. Kassler, T. Benson, and G. Pongracz, "Mp-hula: Multipath transport aware load balancing using programmable data planes," in *Proc. NetCompute*, 2018.

[8] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *Proc. USENIX NSDI*, 2020.

[9] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, p. 253–264, Aug. 2005.

[10] N. Michael and A. Tang, "Halo: Hop-by-hop adaptive link-state optimal routing," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1862–1875, 2015.

[11] J. Zhang, X. Zhang, M. Sun, and C. Yang, "Minimizing the maximum link utilization in multicast multi-commodity flow networks," *IEEE Communications Letters*, vol. 22, no. 7, pp. 1478–1481, 2018.

[12] H. D. Sherali and G. Choi, "Recovery of primal solutions when using subgradient optimization methods to solve lagrangian duals of linear programs," *Operations Research Letters*, vol. 19, pp. 105–113, 1996.

[13] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (SD-WAN): Architecture, advances and opportunities," in *Proc. IEEE ICCCN*, 2019.

[14] O. Michel and E. Keller, "Sdn in wide-area networks: A survey," in *Proc. IEEE SDS*, 2017.

[15] G. F. Riley and T. R. Henderson, *The ns-3 Network Simulator*, 2010.

[16] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *Proc. USENIX NSDI*, 2017.

[17] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proc. INOC*, 2007.

[18] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.

[19] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "Ofswitch13: Enhancing ns-3 with openflow 1.3 support," in *Proceedings of the Workshop on Ns-3*, 2016.