Distributed Utility Maximization 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 sharing in IP networks, load balancing and rate control are key traffic engineering ingredients. In this context, we propose a fully distributed load balancing and rate allocation mechanism that operates only from the edge. Each access router is able to determine target rates over multiple paths for different traffic aggregates based on already available link state information and, some small and optional, information received from other edge devices. Our distributed utility maximization solution provides a feasible rate allocation at each iteration with diminishing returns. Through numerical results on a variety of instances, we show that it converges to near optimal solutions after a few iterations. Thanks to packet-level simulations on an SD-WAN scenario, we also show that it can well prioritize traffic over centralized and legacy solutions.**

I. Introduction

Traffic engineering [1] gathers a set of mechanisms to optimize network performance and traffic delivery. While routing optimization aims at finding efficient routes, Quality of Service (QoS) mechanisms and load balancing control how bandwidth is shared among the different paths. Traffic engineering plays a crucial role in optimizing the use of network resources and meeting QoS requirements of network users. To improve the utilization of IP networks, load balancing is typically implemented inside switches or routers using Equal Cost Multi-Path (ECMP) [2], where a hash is calculated over packet headers and used to select the outgoing path in a uniform manner, or Weighted Cost Multi Pathing (WCMP) [3], where load balancing weights are used to make decisions for each flows. On top of load balancing, rate control and prioritization can be enforced at the network layer to properly share bandwidth among traffic classes.

The control of load balancing and rate allocation parameters can be realized by a centralized network entity, e.g., a Software-Defined Networking (SDN) controller or a Path Computation Element (PCE) [4]. These centralized control plane units leverage on a global view of the network to decide about the most efficient way to steer traffic. Different objectives can be optimized such as the overall congestion or the QoS experienced by users. Since the seminal work by Kelly et al. [5], the Network Utility Maximization (NUM) framework is commonly used for bandwidth allocation or rate control. The goal is to maximize the overall "utility," where user's level of satisfaction is measured using a concave function of the allocated data rate to that user (i.e., the utility function). For instance, it has been applied for fair bandwidth sharing over single and multiple paths, and efficiently solved in a centralized controller with Lagrangian decomposition [6] or, in the semi-distributed case with multiple domain controllers, using ADMM [7]. However, the presence of a centralized control entity may not be desirable for scalability, fault-tolerance, or deployment reasons.

978-3-903176-32-4 © 2021 IFIP

In this paper, we propose a distributed load balancing and rate control solution that operates only from edge devices with very little support from core nodes. Our goal is to maximize the total network utility and push all the processing at the edge (e.g., on access routers). The challenge is to yield a minimum network overhead when coordination among edge nodes is required. Such a solution is particularly useful for overlay networks in the context of SD-WAN (Software-Defined Wide Area Networks) [8]. It should also quickly *converge* towards the optimal solution, i.e. generate a set of solutions with diminishing returns over iterations, and it must ensure *anytime feasibility*, i.e. that it does not violate capacity constraints at any iteration.

A wide range of distributed solutions for bandwidth sharing have been proposed [9], [10], [11], [12]. However, they actively involve intermediate or core nodes with a request / answer protocol that allocates bandwidth on core links. While they aim at maximizing the network utility, most of the papers use max-min fairness and single path routing. Other solutions rely on an explicit *pricing* of link bandwidth by core nodes [13]. While active participation of core nodes is still required, these pricing solutions do not guarantee that the bandwidth allocation is feasible at all iterations. We direct the reader to Sec. III for more details on related work.

Compared to the state of the art, our distributed solution for load balancing and rate control operates only from the edge. Edge nodes decide outgoing rates over multiple paths for each of the traffic aggregates they manage. Iterative decisions are taken by access routers to continuously optimize network utility. The main advantages of our solution are threefold: 1) it relies on already available link state information (e.g., link loads) and it optionally requires a small additional signaling between edge nodes only to coordinate decisions, 2) it ensures a feasible solution at each iteration so that no traffic is lost due to poor load balancing decisions, and 3) it continuously improves network utility with diminishing returns over iterations.

Our distributed algorithm for utility maximization is based on a sub-gradient method that provides anytime feasibility. It operates at the edge and requires a very low overhead. Without loss of generality, we demonstrate its application with a specific utility function that integrates for each traffic aggregate a preference for outgoing paths and a priority. We analyze through numerical results on various static instances the convergence of the algorithm to optimal solutions. We show that when starting from a bandwidth allocation that minimizes the MLU, an optimality gap lower than 6% can

be achieved after a few iterations. We also show that a near-optimal solution (i.e., gap less than 1.7%) is reached after a sufficient number of iterations. In addition, we perform NS3 [14] simulations on a realistic SD-WAN scenario and show that the algorithm is able to well prioritize traffic and handle paths preferences.

The rest of this paper is structured as follows. The system architecture and problem formulation are introduced in Sec. II. Sec. III reviews the state of the art. Sec. IV presents the Lagrangian relaxation of the problem and the sub-gradient algorithm to solve it. Sec. V details the decomposition of the problem and the distributed procedure to generate feasible solutions at all iterations. Sec. VI shows the performance evaluation and Sec. VII concludes this paper.

II. System description

As traffic evolves over time, load balancing and rate allocation must be continuously adjusted to better use network resources and maximize network utility. This section first presents our distributed architecture where only edge devices are controlling user traffic. Then, it illustrates a typical SD-WAN scenario where it can be applied. Finally, it formulates the optimization problem that the distributed agents need to solve collaboratively.

A. Distributed Architecture

As illustrated in Fig. 1, edge devices are equipped with rate allocation agents in order to continuously compute rate allocations and maximize network utility. 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 rate allocations, on each path, 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. Rate allocations can be strictly enforced in the data plane or loosely used as load balancing weights.

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. Additionally, agents can receive useful information from other agents to make decisions. In particular, two scalars called *aggregated utilities* can be periodically received from all the other agents when Polyak step-size is used (see Sec. V for more details). Finally, agents also receive updates about the traffic demand in each of the tunnel they handle from the local monitoring.

The computation of rate allocations is performed by each source device every time new information is received. It takes as input the set of candidate paths for each tunnel, updated traffic information for each tunnel, updated link state information and (optionally) updated aggregated utilities received from other agents. As output, each agent decides for the tunnels it manages a target rate on outgoing paths.

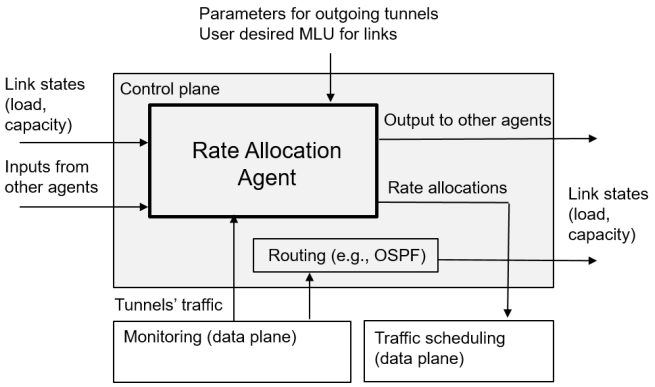


Fig. 1: Device architecture.

To enforce target rate allocations, a traffic scheduling module ensures that traffic evolves from the actual rates towards the targeted ones. As mentioned before, rate allocations can be loosely used as load balancing weights to take routing decisions each time a new micro-flow arrives. The path selection 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 [15] 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. To ensure proper prioritization of traffic in the data plane, packets can be marked with the DSCP code that corresponds to the priority of their tunnel. More advanced traffic scheduling techniques can be used to exploit the computed rate allocations.

B. SD-WAN Use Case

Fig. 2 illustrates a typical SD-WAN network with one headquarter site and three branch sites that are multi-homed with MPLS and broadband Internet connectivity. Traffic can go between the headquarter and remote sites, or it can be between sites themselves. Origin-Destination (OD) tunnels are used to carry traffic aggregates for the different types of application classes (e.g., real-time critical, elastic critical, elastic non-critical).

The network manager can define a desired Maximum Link Utilization (MLU) for each link in the network based on past observations. Typically, he or she may wish that the link utilization stays below 90% to avoid problematic congestion issues. The manager can also configure tunnel attributes that can be used as part of the utility function. In the rest of the paper, we consider for instance that he or she can set for each tunnel a priority or a preference for each outgoing path. Typically, users may prefer to use MPLS against broadband Internet for their mission-critical applications.

C. Problem Formulation

Let's consider a network $G = (V, A)$, where V is the set of nodes and A is the set of links, and a set of Origin-Destination

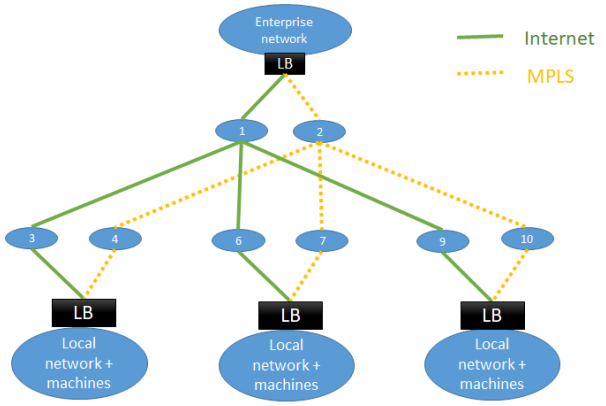


Fig. 2: SD-WAN scenario with one headquarter site and three branch sites. Load Balancing (LB) agents are deployed on access routers with two ports (Internet, MPLS).

(OD) tunnels K (commodities) that must be routed over a set of pre-computed paths P^k with $k \in K$. Given a set of source nodes $S \subset V$, we denote by $K^s \subseteq K$, the set of tunnels having the same origin $s \in S$. Let $C : A \to \mathbb{R}_+$ be the arc capacity function, $d : K \to \mathbb{R}_+$ be the traffic demand function, $U : k \to \mathbb{R}$ be a concave utility function for each tunnel k.

Given a user defined maximum link utilization $MLU \in]0, 1[$, the rate allocation problem consists, for every tunnel $k \in K$, in splitting the traffic demand d_k over multiple paths in P^k such that arc capacities $\tilde{C} = C \times MLU$ are satisfied and the total utility function is maximized.

The load balancing and rate allocation problem can be solved in a centralized manner by solving a Linear Programming (LP) model with the decision variables r_p^k representing the allocated rate on path $p \in P^k$ for tunnel $k \in K$. The problem can be formulated as follows

$$\max \sum_{s \in S} \sum_{k \in K^s} \sum_{p \in P^k} U^k(r_p^k) \tag{1}$$

$$\sum_{p \in P^k} r_p^k = d_k \qquad \forall k \in K^s, \forall s \in S, \tag{2}$$

$$\sum_{s \in S} \sum_{k \in K^s} \sum_{\substack{p \in P^k \\ a \in p}} r_p^k \leq \tilde{C}_a \qquad \forall a \in A, \tag{3}$$

$$0 \leq r_p^k \qquad \forall k \in K^s, \forall s \in S, \forall p \in P^k. \tag{4}$$

Equalities (2) guarantee that all traffic is allocated and Constraints (3) guarantee that $\tilde{C}$ capacities are respected.

III. Related Work

In the state of the art, several solutions have been proposed to solve rate allocation and load balancing problems in a distributed manner.

Distributed bandwidth reservation or rate allocation protocols have been proposed in [16], [17]. In these solutions, a bandwidth allocation request is sent from the source node and processed greedily by core switches. Intermediate nodes or core switches are the ones taking rate allocation decisions using a specific utility function with fairness objective (max-min in general). Routing can be given (in [16]) or not. Similar mechanisms were designed for ATM networks to allocate bandwidth for ABR (Available Bit Rate) traffic [9], [10], [11], [12]. These solutions involve the active participation of core nodes which add complexity in the network.

Other solutions use Lagrangian decomposition and rely on an explicit *pricing* of link bandwidth by adjacent core nodes [13]. Based on the traffic they observe, intermediate nodes update dual variables related to link capacity constraints based on a gradient iterate and they broadcast these link prices in the network. Each source solves a local problem to update its rate allocations. While active participation of core nodes is still required, these solutions do not guarantee that the bandwidth allocation is actually feasible at all iterations. Indeed, primal-dual algorithms are known to fail at providing feasible solutions at any iteration.

Distributed multi-path routing has also been studied. In Halo [18] for instance, each core node decides local split ratios for each flow aggregate. Routing is then decided in a hop-by-hop fashion to minimize MLU (it can actually minimize any convex function). While guaranteeing feasibility at every iteration and convergence to the optimal, this solution actively involves intermediate nodes in the decision-making process and is thus not suitable for our problem.

Decentralized load balancing techniques [19], [20] have been proposed to dynamically adjust at ingress nodes load balancing policies based on path-based measurements about the network congestion. Our work differs from these contributions by not involving core nodes to piggy back measurements inside user traffic and by aiming at maximizing the network utility for all aggregates.

To summarize, our method solves the distributed load balancing and rate allocation problem to maximize network utility and operates only from the edge in a fully distributed fashion. It converges with diminishing returns and provides anytime feasibility.

IV. Lagrangian Relaxation

Lagrangian relaxation method is a well-known method that permits to relax difficult constraints and penalize their violations in the objective function. In general, the resulting problem is easy to solve [21].

A. Problem Relaxation

We apply the Lagrangian relaxation on the compact formulation in order to relax capacity constraints (3) and penalize them in the objective function using Lagrangian multipliers $\lambda \in \mathbb{R}_+^A$ as follows.

$$\min_{\lambda} f(\lambda) \tag{5}$$

$$0 \leq \lambda_a \qquad \forall a \in A. \tag{6}$$

where $f(\lambda)$ is defined as the following linear program

$$\max_{r} \sum_{s\in S}\sum_{k\in K^s}\sum_{p\in P^k} U^k(r_p^k) + \sum_{a\in A}\lambda_a(\tilde{C}_a - \sum_{s\in S}\sum_{k\in K^s}\sum_{\substack{p\in P^k\\ a\in p}} r_p^k) \tag{7}$$

$$\sum_{p\in P^k} r_p^k = d_k \qquad \forall k \in K^s, \forall s \in S, \tag{8}$$

$$0 \leq r_p^k \qquad \forall k \in K^s, \forall s \in S, \forall p \in P^k. \tag{9}$$

Problem (5)-(6) is called the Lagrangian dual problem. One particularity of the Lagrangian relaxation is the fact that the value of any solution of the Lagrangian dual problem is a dual bound of the original problem, i.e., (1)-(4). Moreover, in the case of convex programs, the optimal value of the Lagrangian dual problem coincides with the optimal value of the initial problem [21].

B. Sub-gradient Algorithm

The Lagrangian dual problem can be optimally solved using the sub-gradient algorithm described in Alg. 1.

Algorithm 1: Sub-Gradient algorithm

```
Result: Optimal solution of Lagrangian dual problem
i = 0; step = 1;
λ_a(0) = 0 for all a ∈ A;
while not StopCondition() do
    r* ← Solve optimization problem (7) − (9);
    for a ∈ A do
        g_a = C̃_a − Σ_{s∈S} Σ_{k∈K^s} Σ_{p∈P^k, a∈p} r_p^{*k};
        λ_a(i+1) = max(0, λ_a(i) + step × g_a);
    end
    step = UpdateStepSize(); i = i + 1;
end
```

The StopCondition() function returns "True" if stopping criteria of the algorithm are met and "False" otherwise. Ideally the sub-gradient algorithm can be stopped when all capacity constraints are satisfied. However, in practice this can rarely happen. Then, several criteria are used in the literature [22], [23] as the strong duality conditions, the improvement of the objective value within a given number of iterations or a threshold on the number of iterations.

The UpdateStepSize() function permits to increase or decrease the size of the penalty. In the literature, a common practice is to use $step = \frac{1}{i}$, where i represents the iteration counter. However, in [24], Polyak proposes the following step-size function that does not depend on a counter

$$step = \alpha \frac{UB - LB}{||g||^2}$$

such that, α is a positive scalar, UB is the objective value of (7) − (9) and LB is the utility of the best found solution of (1)-(4). This can be updated each time a solution of (7) − (9) satisfies (3). In our setting, the optimization process never stops. The main advantage of Polyak's method is that the step-size automatically adapts and does not depend on an iteration counter. Later in the paper we refer to as *Iterative* and *Polyak* for the two step-size functions.

V. Distributed Algorithm

We now detail how to decompose the Lagrangian dual problem into $|S|$ subproblems for a distributed algorithm (i.e., one for each agent).

A. Problem Decomposition

In the program (7)−(9), it is easy to see that each constraint of (8)-(9) associated with source $s \in S$, contains only variables associated with tunnels of K^s. Therefore, we can decompose them into $|S|$ subsets, each associated with a source. However, objective function (7) needs to be re-written as follows

$$\sum_{a\in A}\lambda_a\tilde{C}_a + \max_{r}\sum_{s\in S}\sum_{k\in K^s}\sum_{p\in P^k}(U^k(r_p^k) - \sum_{a\in p}\lambda_a r_p^k)$$

Therefore, the optimization problem (7)−(9) decomposes into $|S|$ independent sub-problems, one for each source. The sub-problem for source $s \in S$ is then defined as

$$\max_{r}\sum_{k\in K^s}\sum_{p\in P^k}(U^k(r_p^k) - \sum_{a\in p}\lambda_a r_p^k) \tag{10}$$

$$\sum_{p\in P^k} r_p^k = d_k \qquad \forall k \in K^s, \tag{11}$$

$$0 \leq r_p^k \qquad \forall p \in P^k, \forall k \in K^s. \tag{12}$$

If Polyak step-size function is used, every source node $s \in S$ must be able to compute UB and LB. Hence, each source node $s \in S$ has to share, at each iteration, two scalars with the other source nodes, representing the aggregated utilities:

- $aggUt_1^s : \sum_{k\in K^s}\sum_{p\in P^k}(U^k(r_p^{*k}) - \sum_{a\in p}\lambda_a r_p^{*k})$
- $aggUt_2^s : \sum_{k\in K^s}\sum_{p\in P^k} U^k(\bar{r}_p^k)$

where r^* is the optimal solution of (10)−(12) and $\bar{r}$ is the best feasible solution found. Hence, each source node can compute

- $UB = \sum_{a\in A}\lambda_a\tilde{C}_a + \sum_{s\in S} aggUt_1^s$
- $LB = \sum_{s\in S} aggUt_2^s$

The distributed sub-gradient algorithm to solve the Lagrangian dual problem is presented in Alg. 2.

The getLinkUtilization() function returns the link load for all links in the network. This information can be retrieved from Link State Advertisement (LSA) in the OSPF protocol for instance. The getAggregatedUtilities() function is called only when Polyak step-size is used. It returns two vectors of aggregated utilities associated with all the source nodes in S. The StopCondition() and UpdateStepSize() functions are the same as for Alg. 1.

Algorithm 2: Distributed Sub-Gradient algorithm

Result: Optimal solution of Lagrangian dual problem
$i = 0$; $step = 1$;
$\lambda_a(0) = 0$ for all $a \in A$;
while *not StopCondition()* **do**
  for $s \in S$ **do**
    $\texttt{LU} \leftarrow$ getLinkUtilization();
    Solve optimization problem $(10) - (12)$;
    for $a \in A$ **do**
      $g_a = \tilde{C}_a - \texttt{LU}_a$;
      $\lambda_a(i+1) = max(0, \lambda_a(i) + step \times g_a)$;
    end
    $aggUt_1, aggUt_2 \leftarrow$ getAggregatedUtilities();
    $step$ = UpdateStepSize(); i = i + 1;
  end
end

B. Recovering feasible solutions

Even if optimally solving (5)-(6) gives the optimal value of the original problem, the solution r may not satisfy C capacities. Indeed, relaxing capacity constraints (3) may lead to violations. In order to produce solutions satisfying C capacities at each iteration of Alg. 2, i.e. ensure anytime feasibility, the source of each tunnel solves an additional subproblem given by constraints $(10) - (12)$ together with the following constraints, for all $a \in A$ and for all $s \in S$

$$\sum_{k \in K^s} \sum_{\substack{p \in P^k \\ a \in p}} (r_p^k - r_p^{*k}) \leq \frac{C_a - \texttt{LU}_a^*}{|S|} \tag{13}$$

where r^* represents the rates at the previous iteration and $\texttt{LU}^*$ the associated link utilization. Constraints (13) permit to, artificially, divide the residual capacity of links by the total number of sources, possibly using each link. Let $\overline{r}$ be the rate obtained by solving $(10) - (12)$ and (13) for every $s \in S$.

Proposition 1. *If r^* respects C capacities then $\overline{r}$ as well.*

Proof. By summing constraints (13) we obtain

$$\sum_{s \in S} \sum_{k \in K^s} \sum_{\substack{p \in P^k \\ a \in p}} (\overline{r}_p^k - r_p^{*k}) \leq C_a - \texttt{LU}_a^* \qquad \forall a \in A.$$

Since $\sum_{s \in S} \sum_{k \in K^s} \sum_{\substack{p \in P^k \\ a \in p}} \overline{r}_p^{*k} = \texttt{LU}_a^*$ for all $a \in A$, then

$$\sum_{s \in S} \sum_{k \in K^s} \sum_{\substack{p \in P^k \\ a \in p}} \overline{r}_p^k \leq C_a \qquad \forall a \in A.$$

And the result follows. □

Even if the algorithm guarantees the satisfaction of the capacities C it may violate capacities $\tilde{C}$ (i.e., downscaled capacity with the user MLU). However, the algorithm will increase the penalty on links violating the capacity constraints in order to adapt the rates in the next iterations.

C. Practical Considerations

Fig. 3 shows the execution time line of the algorithm at an edge device. Periodically, every X seconds in the figure (e.g., blue arrows), the edge device retrieves from the local monitoring an update about the traffic demand in each tunnel (i.e., d^k). Based on this, the algorithm starts a new optimization phase and performs one sub-gradient iteration every time new link state update is received (e.g., red arrows). New target rate allocations are calculated every time updates about link states and aggregated utilities are received. In practice, a new iteration can be launched periodically. The computed rates are then used to tune the rate of the actual flows and/or route new flows and FlowLets when they arrive.

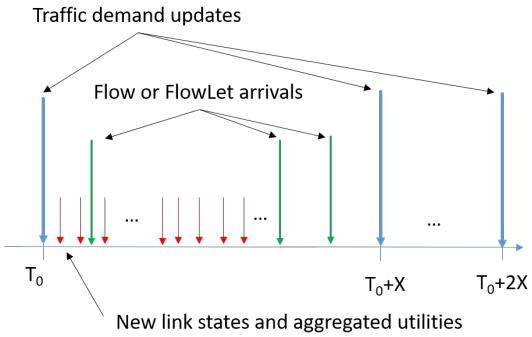


Fig. 3: Execution time line of the algorithm.

To avoid instabilities, measurements about traffic and link loads can be averaged over a moving time window. Furthermore, the frequency of traffic demand updates (i.e., of new optimization phases) must be sufficiently lower than the one for algorithm iterations so that the algorithm has enough time to work on a stable instance of the problem. As the different agents execute this process in an asynchronous manner, loose synchronization mechanisms can help nodes to start their optimization phases all together. For instance, one may use a modulo function over the sequence number of the LSA for the link with smallest identifier.

D. Network Overhead

At the beginning of an optimization phase, every source node optionally receives as link states an update of link capacities (if not known already). And, at each iteration of the subgradient algorithm, every source node receives an update of the utilization of every link in the network (i.e., link loads). Moreover, if Polyak step-size function is used, every source node $s \in S$ has to share two scalars, representing the aggregated utilities, with the other source nodes at each iteration. In the best case, the overhead is extremely low as only already available link states (e.g., link loads) are periodically exchanged.

VI. Numerical Results

The distributed algorithm and the compact model have been implemented in C++, using CPLEX as a LP-solver. They were

				Distributed (Polyak)						Distributed (Iterative)						Centralized	
Instance	|A|	|V|	|K|	GAP20	GAP80	GAP200	CPU20	CPU80	CPU200	GAP20	GAP80	GAP200	CPU20	CPU80	CPU200	GAP	CPU
Abilene	28	11	11	16.6	16.6	16.6	0.00	0.00	0.01	17.0	17.6	17.6	0.00	0.00	0.00	17.6	0.00
BtEurope	74	24	24	3.7	3.7	3.7	0.00	0.00	0.00	3.7	3.7	3.7	0.00	0.00	0.01	3.7	0.01
ColtTelecom	354	153	15	43.1	43.1	43.1	0.01	0.01	0.01	56.3	56.4	56.4	0.01	0.01	0.01	56.4	0.02
Geant	122	40	40	10.5	10.5	15.7	0.00	0.00	0.01	14.8	15.8	15.8	0.01	0.01	0.01	15.8	0.03
GTSCE	386	149	14	42.2	42.2	42.2	0.01	0.00	0.01	53.8	56.2	56.2	0.01	0.01	0.01	56.8	0.01
ITCDeltacom	322	113	11	49.3	49.3	49.3	0.00	0.00	0.01	49.3	49.3	49.3	0.01	0.01	0.01	49.3	0.01
Kentucky	1790	754	75	12.7	12.7	12.7	0.00	0.01	0.00	13.2	13.2	13.2	0.00	0.00	0.00	13.2	0.09
SD-WAN	46	15	15	40.2	40.2	40.2	0.00	0.00	0.00	36.5	40.2	40.2	0.00	0.00	0.00	41.9	0.00
US_Carrier	378	158	15	31.6	31.6	31.6	0.01	0.01	0.01	31.8	32.2	32.2	0.01	0.01	0.01	32.2	0.01
IPRAN_1	1114	485	500	55.6	55.6	55.6	0.01	0.01	0.75	57.7	57.9	57.9	0.01	0.01	0.76	58.8	0.00
IPRAN_2	1086	477	500	63.9	63.9	63.9	0.01	0.01	0.76	67.1	67.1	67.1	0.01	0.01	0.76	67.1	0.00

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

tested on an Intel(R) Xeon(R) CPU E5-4627 v2 of 3.30GHz with 504GB RAM, running under Linux 64 bits. A maximum of 1 thread has been used. In the following, 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 [25]. Without loss of generality, we use for the evaluation a utility function that combines $Prio_k$, the priority of tunnel k, $Pref(k,p)$ the preference of tunnel k for path p, and r_p^k the target rate allocated to tunnel k on path p as follows

$$U^k(r_p^k) = Prio^k \times Pref(k,p) \times r_p^k$$

We compare the performance of the distributed algorithm against a centralized solution solving the compact model to optimality. The sub-gradient algorithm stops when the number of iterations reaches a specified threshold (see below). Two step-size functions are used: Iterative and Polyak with $alpha = 2$ (see sub-section IV-B).

A. Evaluation on Static Instances

In this first evaluation, the goal is to evaluate the convergence of the distributed algorithm. We use three types of synthetic instances:

1) Public instances: Abilene, BtEurope, Geant from SNDLIB [26], ColtTelecom, GTSCE, ITCDeltacom, Kentucky, US_Carrier from Internet topology zoo [27].
2) A SD-WAN instance with one headquarter site and three sites (see Fig. 2).
3) Two large IPRAN, typical from radio access networks.

Link capacities and the traffic demand of tunnels are generated randomly. The source and destination of tunnels are picked at random in all the instances, except for SD-WAN where tunnels are between sites and the headquarter. Tunnels are randomly ranked and the tunnel position represents its priority (priorities start from 1). The maximum number of paths generated per tunnel is 5. However, in practice, fewer paths are used. For each tunnel, the paths are randomly ranked and the path position represents its preference (preferences start from 1).

The initial solution satisfying C capacities is obtained by solving the load balancing problem that minimizes the maximum link utilization. Let θ^* be the maximum link utilization of this solution. The user target MLU to determine soft link capacity constraints $\tilde{C}$ is calculated with $MLU = Tanh(\theta^* \times 2.0)$ (see sub-section II-C).

Convergence. In Table I, three algorithms are compared over all instances: the distributed algorithm with Polyak step-size, the distributed algorithm with Iterative step-size and the centralized method. Column heads are defined as follows:

- GAP20, GAP80 and GAP200: improvement percentage over the initial solution and obtained by the distributed algorithm after 20, 80 and 200 iterations, i.e.,

$$\text{GAP} = \frac{\text{Currrent\_Objective} - \text{Initial\_Objective}}{\text{Initial\_Objective}}$$

- CPU20, CPU80 and CPU200: CPU time of the distributed algorithm after 20, 80 and 200 iterations. It is obtained by summing the maximum CPU times of all source nodes at each iteration.
- CPU, GAP: time and improvement percentage over the initial solution obtained by the centralized method.

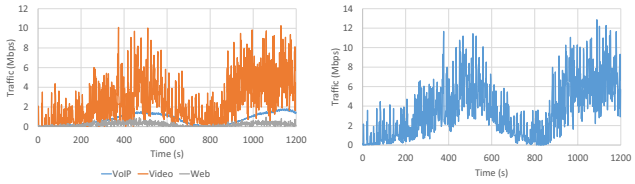
Table I displays the numerical results associated with all instances. We notice that the distributed algorithm with Polyak step-size gives a similar improvement after 20 and 80 iterations. Moreover, except for Geant the improvements over the initial solution are the same after 20, 80 and 200 iterations. Compared to the centralized method (optimal), the algorithm gives the optimal solution in 18% of the cases and near-optimal solution (difference lower than 4% between GAP200 and GAP) in 81.8% of the instances.

The distributed algorithm with Iterative step-size gives similar improvements after 20, 80, and 200 iterations in 63.6% of the instances. Moreover, the algorithm gives the optimal solution in 72.7% of the cases. These results show a fast convergence of the distributed algorithm with both step-size functions. Indeed, significant improvements of the initial solutions have been obtained after a few iterations.

While Polyak step-size is more adaptive for continuous optimization, the distributed algorithm performs better with Iterative step-size. Indeed, Polyak step-size is sensitive to the alpha parameter that needs to be tuned for each instance.

The CPU times are relatively small for all algorithms, less than 0.76 second on all the tests.

Overhead. In the following, we analyze the amount of traffic received by each agent at each iteration of the distributed algorithm. Consider, for example, a network of 500 source nodes and 1000 links. Suppose that each scalar consumes 32



(a) VoIP, Video, and Web traffic. (b) Total Traffic.

Fig. 4: Traffic of each service and total traffic.

bits and a duration of each iteration of 200ms. Therefore, if Polyak step-size is used, every source $s \in S$ receives, at each iteration, the following information

- aggregated utilities: $2 \times \frac{32 \text{ bits}}{0.2 \text{ s} \times 1e6 \text{ bits}} \times 499 \text{ nodes} = 0.159$ Mb/second.
- link loads: $\frac{32 \text{ bits}}{0.2 \text{ s} \times 1e6 \text{ bits}} \times 1000 \text{ links} = 0.16$ Mb/second.

However, with Iterative step-size, only link loads are received by each source at each iteration. This show the low overhead required by the algorithm with the two step-size functions.

B. NS3 Simulations

Simulation settings. We present results using NS3 [25] with OpenFlow 1.3 [28] to evaluate algorithms in a dynamic environment. As the number of iterations may fluctuate between two traffic demand updates, we only use Polyak step-size which does not rely on an iteration counter.

The simulation scenario is depicted in Fig. 2. It comprises a single Headquarter (H) connected to three remote sites, i.e. S1, S2, S3. There are 6 tunnels where 3 are from H to S1, S3, and S3 and the other 3 are opposite tunnels from sites to headquarter. Access routers have dual homing with Broadband Internet and MPLS, having propagation delays of 60ms and 1ms respectively. The packet loss rate at transmission level on Internet is 0.01% and null for MPLS.

We consider that each tunnel prefers MPLS over Internet (i.e., $Pref(k,p)$ equals 1 or 2, respectively for Internet and MPLS), and that sites and their associated tunnels have different priorities (i.e., $Prio_k$): low (1), medium (3) and high (5) as shown in Table IIa. The traffic pattern in each tunnel is diurnal and formed by 3 types of services: Voice over IP (VoIP), Video, and Web. Each service has a specific SLA (Service Level Agreement) requirement that we do not directly optimize. We use DSCP to differentiate services in the data plane with a strict priority queuing discipline.

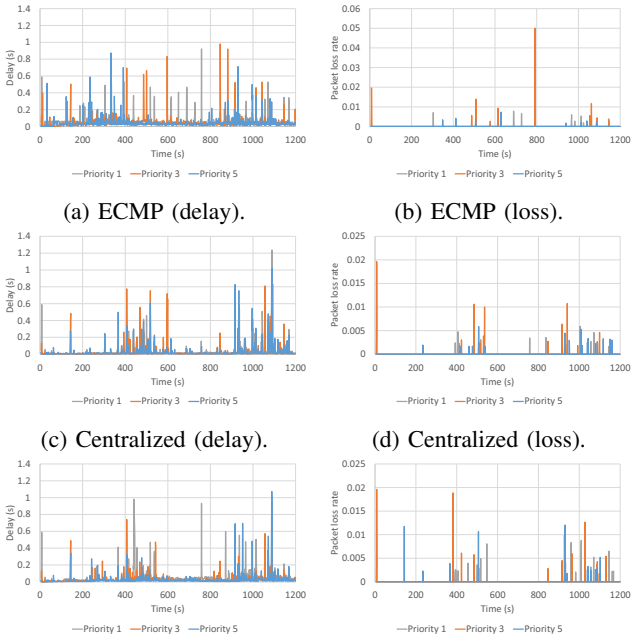
Tunnel Identifier	1	2	3	4	5	6
Source - Dest.	H-S1	S1-H	H-S2	S2-H	H-S3	S3-H
Priority ($Prio^k$)	5 (high)		3 (medium)		1 (low)	

(a) Priority for each tunnel.

Service type	VoIP	Video	Web
SLA requirement (delay)	30ms	75ms	100ms
Priority queue (DSCP code)	0	1	2

(b) Traffic types and corresponding priority queues.

TABLE II: Characteristics of tunnels and traffic in NS3.



(a) ECMP (delay). (b) ECMP (loss).

(c) Centralized (delay). (d) Centralized (loss).

(e) Distributed (delay). (f) Distributed (loss).

Fig. 5: End-to-end delay and packet loss rate for all priority levels with distributed and centralized algorithms.

Table IIb summarizes the desired SLA requirements and the corresponding priority queues for each traffic type.

The traffic pattern for each traffic type and the total traffic are shown in Fig. 4. We generated micro-flows with typical patterns of VoIP, Video and Web applications for a duration of 300s. The inter-arrival time of flows follows an exponential distribution. We randomly generated diurnal patterns by adjusting the mean of inter-arrivals of flows. The idea is to have congestion episodes representing peak hours.

Each node in the network measures the link states and broadcasts them periodically. The source of each tunnel, therefore, is able to collect them. Link states are measured and sent every 200ms, and averaged over a 1s window to mitigate fluctuations. The source of each tunnel measures the current traffic and averages it over a 5s moving window. The traffic demand is an input of algorithms at the beginning of each optimization phase (i.e., every 30) and the link states are the inputs of Alg. 2 at each iteration (i.e., every 200ms). Every time algorithms are called, every 30s or 200ms, new target split ratios are computed and available to the data plane.

When a new micro-flow arrives, it is assigned to a path that minimizes the difference between target and actual split ratios. In our implementation, the selected path is converted into forwarding rules deployed at every OpenFlow switch.

Simulation results. Fig. 5 shows the evolution of the end-to-end delay and the packet loss rate at network layer for ECMP, centralized, distributed solutions. The difference in end-to-end delay across priorities for ECMP is not remarkable while it is noticeable for centralized and distributed algorithms (i.e., higher priority has lower delay). Both centralized and

End-to-end delay (ms)			
	ECMP	Centralized	Distributed
	Tunnel 1 (priority = 5, high)		
Average	46.27	24.96	26.35
95-percentile	97.07	90.52	67.9
	Tunnel 3 (priority = 3, medium)		
Average	45.98	27.13	27.48
95-percentile	97.1	110.68	76.76
	Tunnel 5 (priority = 1, low)		
Average	50.39	28.16	45.82
95-percentile	109.85	117.81	127.8
End-to-end packet loss (%)			
	ECMP	Centralized	Distributed
	Tunnel 1 (priority = 5, high)		
Average	0.009	0.004	0.006
	Tunnel 3 (priority = 3, medium)		
Average	0.012	0.008	0.009
	Tunnel 5 (priority = 1, low)		
Average	0.01	0.007	0.009

TABLE III: End-to-end delay and packet loss rate.

distributed algorithms prefer MPLS to Internet; therefore, the higher priority tunnels have more chance to be transferred over MPLS which has a significantly lower propagation delay. The evolution of the packet loss rates follow a similar pattern. They increase when traffic is high and congestion happens (between 400s and 600s, and after 900s). ECMP has the highest peak loss rate (5%) and there is no noticeable difference between priorities. Meanwhile, the high priority (i.e., 5) in centralized and distributed algorithms has a significantly low packet loss rate.

Table III summarizes results for ECMP, centralized, and distributed solutions with average and 95^{th} percentile for end-to-end delay and packet loss rate. Note that the 95^{th} percentile is 0 for the packet loss and that performance are similar for opposite tunnels (i.e., 2, 4, 6). The end-to-end delay of centralized and distributed algorithms are about 50% lower than the ones of ECMP. The average end-to-end delay of the distributed algorithm, except the low priority tunnel, is similar to the one of the centralized algorithm. The packet loss rates of distributed algorithm are slightly higher than one of centralized algorithm. These results are explained by the fact that the distributed algorithm places more traffic on Internet where the propagation delay is higher.

We also measure the SLA violation rate which is the ratio of flows violating their SLA requirements to the total number of flows. Fig 6 shows the SLA violation rate over time for ECMP, centralized, and distributed solutions. As expected, ECMP has the highest violation rate while the centralized

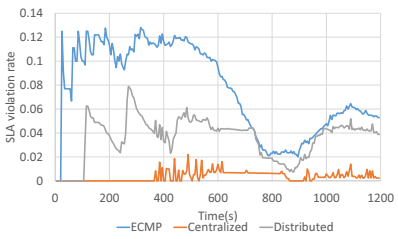
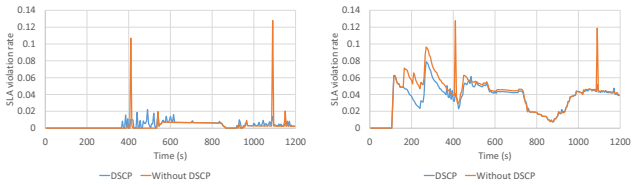


Fig. 6: Ratio of SLA violations (when DSCP is used).

End-to-end delay (ms)		
	Centralized	Distributed
	Tunnel 1 (priority = 5, high)	
Average	23.54	26.99
95-percentile	89.44	67.41
	Tunnel 3 (priority = 3, medium)	
Average	23.81	28.28
95-percentile	109.72	72.11
	Tunnel 5 (priority = 1, low)	
Average	29.5	47.34
95-percentile	113.66	130.6

TABLE IV: End-to-end delay when DSCP is not used.



(a) Centralized algorithm (b) Distributed algorithm

Fig. 7: Ratio of SLA violations (with DSCP vs. no DSCP).

algorithm has the lowest. The SLA violation rate increases when traffic increases and we can observe that the distributed algorithm needs some time to converge at the beginning. Indeed, violations are steadily decreasing for the distributed algorithm until the network becomes congested at 400s.

Fig. 7 demonstrates the effect of prioritization using Diff-Serv in the data plane. When different DSCP codes are used for all traffic types, SLA violations are remarkably decreased even though the average delay is not improved as shown in Table IV. DSCP puts different services into different priority queues while every packet are put into the same queue when DSCP is not used. This result shows that the performance of algorithms are mainly due to the control plan algorithms themselves, even though a consistent prioritization of traffic in the data plane boosts the overall performance.

Discussion. While the distributed and centralized algorithms are closed, the centralized solution performs best. Many parameters can be tuned to further improve the performance. On the algorithm itself, the $alpha$ parameter in Polyak step-size can have an influence on the convergence rate. While from a system perspective, the average window for link states could either give a noisy feedback if too small, or give a too slow feedback if too large.

VII. Conclusion

We have proposed a fully-distributed load balancing and rate allocation algorithm that maximizes network utility and that only operates from the edge using already available link state information and some lightweight exchange of aggregated utilities when Polyak step-size is used. The algorithm converges 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 a few iterations. The distributed solution significantly helps to improve the QoS compared to legacy load balancing solutions like ECMP.

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] "Multipath Issues in Unicast and Multicast Next-Hop Selection," RFC 2991, Nov. 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 *Proceedings of the Ninth European Conference on Computer Systems*, 2014, pp. 1–14.

[4] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[5] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the OR society*, vol. 49, no. 3, 1998.

[6] B. McCormick, F. Kelly, P. Plante, P. Gunning, and P. Ashwood-Smith, "Real time alpha-fairness based traffic engineering," in *Proc. of HotSDN*, 2014.

[7] Z. Allybokus, K. Avrachenkov, J. Leguay, and L. Maggi, "Multi-path alpha-fair resource allocation at scale in distributed software-defined networks," *IEEE Journal on JSAC*, vol. 36, no. 12, 2018.

[8] 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.

[9] Y. Afek, Y. Mansour, and Z. Ostfeld, "Phantom: a simple and effective flow control scheme," *Computer Networks*, vol. 32, no. 3, 2000.

[10] B. Awerbuch and Y. Shavitt, "Converging to approximated max-min flow fairness in logarithmic time," in *Proc. IEEE INFOCOM*, 1998.

[11] Y. Bartal, M. Farach-Colton, S. Yooseph, and L. Zhang, "Fast, fair and frugal bandwidth allocation in ATM networks," *Algorithmica*, vol. 33, no. 3, pp. 272–286, 2002.

[12] Y. T. Hou, H.-Y. Tzeng, and S. S. Panwar, "A generalized max-min rate allocation policy and its distributed implementation using the abr flow control mechanism," in *Proc. IEEE INFOCOM*, 1998.

[13] D. P. Palomar and M. Chiang, "A tutorial on decomposition methods for network utility maximization," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1439–1451, 2006.

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

[14] G. F. Riley and T. R. Henderson, "The ns3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010.

[15] E. Vanini, R. Pan, M. Alizadeh, P. Taheri, and T. Edsall, "let it flow: Resilient asymmetric load balancing with flowlet switching."

[16] L. Jose, S. Ibanez, M. Alizadeh, and N. McKeown, "A distributed algorithm to calculate max-min fair rates without per-flow state," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 2, pp. 1–42, 2019.

[17] F. Skivée and G. Leduc, "A distributed algorithm for weighted max-min fairness in mpls networks," in *International Conference on Telecommunications*. Springer, 2004, pp. 644–653.

[18] N. Michael and A. Tang, "Halo: Hop-by-hop adaptive link-state optimal routing. networking," *IEEE/ACM Transactions on, PP (99)*, pp. 1–1, 2014.

[19] 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.*, Aug. 2014.

[20] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *Proceedings of the ACM Symposium on SDN Research*, 2016.

[21] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, "Network flows: theory, algorithms, and applications. 1993," *Google Scholar Google Scholar Digital Library Digital Library*, 1993.

[22] P. Putz, *Subgradient optimization based Lagrangian relaxation and relax-and-cut approaches for the bounded-diameter minimum spanning tree problem*. na, 2007.

[23] M. B. Hasan and M. Toha, "An improved subgradiend optimization technique for solving ips with lagrangean relaxation," *Dhaka University Journal of Science*, vol. 61, no. 2, pp. 135–140, 2013.

[24] B. T. Polyak, "Introduction to optimization. optimization software," *Inc., Publications Division, New York*, vol. 1, 1987.

[25] G. F. Riley and T. R. Henderson, *The NS3 Network Simulator*, 2010.

[27] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on JSAC*, oct 2011.

[28] L. J. Chaves, I. C. Garcia, and E. R. M. Madeira, "OFSwitch13: Enhancing NS3 with OpenFlow 1.3 Support," in *NS3 Workshop*, 2016.