IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 25, 2021, accepted November 25, 2021, date of publication December 6, 2021,
date of current version December 20, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3133315

Upward Max-Min Fairness in Multipath High-Speed Networks

PATRICIA LUDEÑA-GONZÁLEZ [1,2], JOSÉ LUIS LÓPEZ-PRESA [2], AND FERNANDO DÍEZ MUÑOZ [2]

[1]Departamento de Ciencias de la Computación y Electrónica, Universidad Técnica Particular de Loja, Loja 1101608, Ecuador
[2]Departamento de Ingeniería Telemática y Electrónica, Universidad Politécnica de Madrid, 28031 Madrid, Spain

Corresponding author: Patricia Ludeña-González (pjludena@utpl.edu.ec)

This work was supported in part by the Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación SENESCYT, Quito, Ecuador, through the Convocatoria Abierta 2012 Scholarship Program (Second stage).

ABSTRACT To take full advantage of the available network capacity, connections need to be able to use multiple paths to route their packets. Max-min fairness (MMF) can be effectively applied to single-path networks, but computing MMF rates in multipath networks requires solving a series of linear programming (LP) problems with high computational cost. Thus, a relaxation of MMF has been proposed, namely, upward max-min fairness (UMMF), which can be solved by simple combinatorial algorithms. Current proposals carry out incremental approximations emulating the waterfilling algorithm, which inherently establishes a dependency between the time required to achieve the optimal solution and the capacity of the links. Thus, the more capacity the network has, the less efficient the algorithms are. We defined the concept of the *saturation level* as the basis for the computation of fair shares. We developed the first centralized algorithm based on this concept, which we call c-SLEN. Unlike its predecessors, its convergence time does not depend on network capacity, and it does not incur link oversaturation. Based on c-SLEN, we derived d-SLEN, a distributed protocol that does not need to maintain per-subflow information in routers and guarantees constant processing time for control packets, making it a good candidate for practical use. Finally, through extensive simulations, we showed that d-SLEN is faster, lighter, and more accurate than its counterparts. Owing to its accuracy and convergence speed, it is able to maintain the size of link queues at minimal values at all times, thus proactively avoiding network congestion.

INDEX TERMS ERC algorithms, multipath networks, upward max-min fairness, high-speed networks, proactive congestion control.

I. INTRODUCTION

Two essential network issues are congestion control and a fair distribution of network capacity among the connections in the network. They have been thoroughly studied in the context of traditional single-path networks (where connections use a single path to route their packets) but much less in the context of multipath networks (where connections might use a variable number of, not necessarily disjoint, paths).

Congestion control algorithms face three main challenges: maximizing capacity utilization of bottleneck links, fair resource sharing, and not stressing link queues [1].

In data center networks, packet losses cause long retransmission delays, which can be catastrophic for many real-time applications. To avoid this issue, lossless networks have been deployed in many production data centers. Nonetheless, when congestion occurs at some link, it may spread to adjacent links, the so-called saturation tree, yielding unfair conditions and seriously degrading global performance [2].

Congestion has traditionally been tackled in a reactive manner. It can be explicitly notified by means of explicit congestion notification (ECN) protocols [3], [4], or it can be detected indirectly [5], [6]. Typical indicators of congestion are packet loss and increment of the round-trip time (RTT). Typically, the sender progressively increases the number of packets sent at each time, the so-called congestion window, until congestion is detected. Then, it drastically reduces the sending rate. This approach was inherited by most congestion-window algorithms. Therefore, these algorithms do not avoid congestion but only mitigate its effects.

Window-based protocols like TCP (Transmission Control Protocol) and its derivatives follow the additive increase

The associate editor coordinating the review of this manuscript and approving it for publication was Fung Po Tso.

multiplicative decrease (AIMD) scheme [7], [8]. Because of their slow convergence, as networks become faster, more subflows will be able to end in less RTTs than the congestion control loop needs to react, making this approach less useful in the future, particularly for short subflows [9]. Furthermore, it has been shown that the slow start model of TCP is a bottleneck in the context of high-speed, large-delay networks, where it may be able to use only approximately three-quarters of the available capacity [10]. Thus, some improvements have been proposed, both in the context of single-path and multipath networks [11]–[14]. Of special interest is the proposal of the bottleneck bandwidth and round-trip propagation time (BBR) [15] algorithm to improve congestion control of TCP connections. BBR estimates the rate at which packets should be sent (which depends on the delivery rate) to achieve high throughput and short RTT, avoiding the buffer overconnection problem of loss-based congestion control approaches, and providing fairness among connections. In [16], wBBR (weighted BBR) was proposed as an extension of BBR to multipath TCP (MPTCP). Each subflow of a connection is assigned a weight that is periodically recomputed to converge to a fair share.

Proactive congestion control has been recently proposed as a viable alternative through explicit rate control (ERC) mechanisms which also provide fairness among competing connections. Calculating explicit rates improves the convergence time [17]. Examples of algorithms that follow this approach are PERC [17] and B-Neck [18], the first quiescent such protocol. These algorithms require per-connection information maintained at the routers. To overcome this problem, s-PERC [19] and SLBN [20] have been proposed as stateless versions of previous algorithms. One of the most interesting properties of this type of algorithm is that the convergence time to the optimal rates depends on the number of bottleneck levels (in the notation of [18]) and the RTT, but not on the capacity of the links, as in window-based algorithms which progressively increase the transmission rate at the sources. This work focuses on the *ERC algorithms*.

Currently, datacenters are built over communication networks with links of very high capacity and multiple available paths between hosts. In this type of network, multipath routing allows for better utilization of the communication infrastructure. High-speed overlay networks have similar requirements with the difference in the propagation time of the links for the long distances involved. Because the approaches used for single-path networks cannot be applied directly to the multipath paradigm, algorithms for multipath systems have been developed, which focus on congestion control and dynamic routing, and apply different fairness criteria [21]. Therefore, this study focuses on *multipath networks*.

Several fairness criteria have been proposed in the literature. Probably, the most popular among them, in single-path networks, is max-min fairness (MMF), because it guarantees the maximum possible rate to the connection with the lowest allocation and achieves high network utilization [22], [23]. Another popular criterion is utility max-min fairness [24], which tries to satisfy the different qualities of service required by the different connections. This criterion can be applied to multipath networks [25], [26]. Because MMF can incur a high computational cost when applied to multipath networks, upward max-min fairness (UMMF) [27], which requires much less computation, has been proposed as a relaxation of MMF in the sense that every MMF allocation is UMMF, but not all UMMF allocations are necessarily MMF. This study focuses on *UMMF allocations*.

In multipath networks, fairness criteria can be used to choose optimal routes among the set of all possible routes for each connection [22], [28]. However, in dense networks, the number of possible routes may grow so much that it is not practical to consider all of them. Furthermore, the set of sessions, on which routing will depend, is highly dynamic. Thus, it may be much more convenient to compute routes statically, considering the topology but not which connections are active in the network. This way, the number of possible subflows per connection is bounded. In this study, we assume that routes are calculated in advance, which includes both the case in which all possible routes are used and the case of single-route routing. The way these routes are chosen is outside the scope of this work.

Traffic control can be faced in a centralized, distributed, or hybrid way. Depending on the size, speed, and type of traffic, one approach may be better than the others. Since centralized algorithms assume global knowledge, they are not well suited for large networks, where distributed or hybrid approaches may be a better option. However, centralized algorithms have been proposed in the context of high-speed datacenter networks [29] and they are always interesting from a theoretical point of view, as they can be used as a reference to validate distributed algorithms and might also lead to efficient distributed versions. Because centralized algorithms are not scalable, we propose a centralized algorithm and its highly scalable distributed version.

A. RELATED WORK

In this section, we review previous work on ERC algorithms for multipath networks, both centralized and distributed, which try to achieve fairness.

There are many congestion control algorithms for single-path networks. Therefore, a traditional approach has been to adapt these algorithms to multipath environments in order to take advantage of the total capacity of the network. Thus, for example, in the field of information centric networks (ICN), MIRCC [30] is proposed as an extension of the rate control protocol (RCP) [31]. A single-path version that outperforms RCP is presented. A multipath extension is then introduced using a hybrid approach, first trying to evenly share the available bandwidth between connections and then to make full use of the remaining capacity. However, the authors recognize that it is a difficult challenge, and their results do not seem to be applicable outside the scope of ICN.

A generalization of the well-known waterfilling algorithm is formulated in [32] as a maximization problem by using a geometric approach to compute max-min fair rates in a multipath network. A throughput polytope is computed, which delimits the range of feasible throughput configurations, so a maximal configuration can be obtained. However, computing the throughput polytope may require exponentially many steps in the general case. Thus, although it may be computed fast for easy scenarios, it does not seem to be of practical use in real networks.

One of the main results in [28] is a centralized (offline) algorithm that computes global max-min fair (GMMF) rates for a set of connections in a multipath network. This algorithm, called OPT_WMMF, requires solving a series of linear programming (LP) problems and can be completed in a polynomial number of steps. However, the size of these problems may grow exponentially, which makes this algorithm impractical for use in real networks. Thus, relaxations of the problem have been proposed, for example, to compute local max-min fair (LMMF) rates (instead of GMMF), and a distributed algorithm that computes an approximation of the MMF rates. However, such distributed algorithm has very strong requirements like global knowledge in routers and synchronization among sources. These requirements make the distributed algorithm impractical.

Since solving MMF in a multipath network by formulating a general LP problem is infeasible for large networks, in [33], the LP problem is reformulated, taking into consideration topological considerations, for the specific case of fat-tree networks. Furthermore, a progressive filling algorithm that does not need to solve any LP problems is also devised. This algorithm is parallelized with a notable improvement in the running times. All these algorithms are centralized, which makes them appropriate for high performance computing (HPC) datacenters, but not for the general case, where a distributed algorithm is necessary.

As the computation of MMF rates in a multipath network seems to be prohibitively expensive in terms of computing power and time, UMMF is an alternative explored in [34], where a distributed algorithm is proposed based on the centralized algorithm of [27]. A price-feedback mechanism is used to design a control protocol that runs on the sources and links in the network. The links compute the prices and report them to sources. When a source (connection) receives the feedback from all its subflows, it recomputes the rates for each individual subflow and notifies the links traversed by each subflow, so the links can recompute prices. This process continues until convergence to the UMMF allocation has been achieved. This approach is very interesting and promising, although it presents some issues:

- Its oscillation nature until convergence, as shown by the experiments performed.
- The number of iterations until convergence (in the order of hundreds for a network with four connections with two subflows each and five links). Note that the time required to perform each iteration is in the order of the RTT.
- A parameter needs to be tuned to achieve better convergence.

As can be seen, there is still work to be done to devise a congestion control mechanism that achieves UMMF and performs well enough to be of practical interest in multipath networks.

B. CONTRIBUTIONS

Because achieving max-min fairness is computationally expensive, a centralized algorithm to compute UMMF rates, called c-SLEN (from Saturation Level Explicit Notification) is proposed. This algorithm resembles the IEWF algorithm [27] using precalculated routes. However, unlike traditional algorithms (including IEWF) that increase rates in discrete steps until a link becomes saturated, our algorithm is based on the concept of *saturation level* of a link. This is the first contribution in this work, which allows one to directly find the next link to saturate and the subflow's rates that saturate it. Thus, an improvement in performance is achieved, since it is not necessary to iterate until a bottleneck is saturated. Therefore, the saturation level of a link may be computed at the link itself and notified to the sources, and the capacity of the links does not affect the performance of the algorithm.

The main result of this work is a scalable distributed protocol that computes UMMF rates. It is a distributed version of c-SLEN called d-SLEN. To the best of our knowledge, this is the first protocol of its kind. Its main features are as follows:

- Computing the saturation level of the links in the path of a subflow and notifying the smallest saturation level to the sources avoids having to successively increase the rates until a link becomes saturated (or oversaturated, depending on the size of the increments) as in [34]. Thus, it is possible to take advantage of all the network capacity from the beginning.
- Its convergence time to the fair rates does not depend on the capacity of the network, which makes it a good candidate for use in high-speed networks.
- Fairness is progressively achieved at a high-speed at the beginning, which allows the connections to get close to their optimum rates in just a few RTTs.
- It does not need to maintain information about the subflows that traverse each link. Only three numbers per link are necessary, and all the protocol packets can be processed in constant time. Therefore, the protocol scales well in terms of the information stored at the routers and the computing time.
- It is able to cope with a dynamic situation where connections enter and leave the network because the saturation level of each link is recomputed each time a protocol packet is processed in a link and only affects the connections that share the affected links.
- The protocol does not depend on any parameters to be tuned to ensure proper convergence.

By simulation, an experimental evaluation of the proposed algorithms was carried out with synthetic networks that serve to compare these algorithms with others proposed in the literature.

Algorithm c-SLEN has been compared with another recent UMMF algorithm, a centralized algorithm devised by Danna *et al.* [27] named IEWF. The experiments show that, although the rates allocated to the connections are similar in all cases, c-SLEN never oversaturates any link, whereas the other tends to both oversaturate links and leave others underutilized. Thus, we show that the concept of saturation level is useful for accurately computing bottleneck rates in links.

Finally, we compare the running times of c-SLEN and IEWF and show that c-SLEN converges much faster than IEWF in all cases owing to its ability to compute the saturation levels in the links.

The distributed protocol d-SLEN was validated by simulation and compared with the recent proposal of Vo *et al.* [34]. The results obtained show that d-SLEN exhibits much faster convergence, which does not depend on the capacity of the network. This convergence does not suffer from oscillations. Furthermore, it can effectively avoid congestion because it keeps link queues at minimal sizes.

C. STRUCTURE OF THE REST OF THE PAPER

In the following section, the concepts of upward max-min fairness and saturation level are presented with a centralized algorithm and a distributed protocol, named c-SLEN and d-SLEN, respectively, which compute UMMF rates. Section III presents the scenarios used during experimentation and shows the high performance of the proposed algorithms compared to other UMMF algorithms. Finally, the results are discussed (see Section IV) and the conclusions are presented in Section V.

II. UPWARD MAX-MIN FAIRNESS

In this section, the system model is presented and the concept of *saturation level* is defined. Then, a centralized algorithm that computes UMMF rates in a multipath system and improves the performance and accuracy of current ones is proposed. Finally, based on this algorithm, a distributed protocol is developed.

A. SYSTEM MODEL

We begin by defining some basic notations and concepts that will be extensively used in the proposed algorithms. The basic notation used is summarized in Table 1. In our system, the network is represented by a directed graph $G = (V, L)$, where V is the set of nodes and L is the set of directed links that connect the nodes. The capacity of the link l is denoted by κ_l. There are two types of nodes: routers (which are connected to other routers and hosts) and hosts (which are connected to a single router).

The set of connections in the network is denoted by C. Connections are established from a source host to a destination host. For simplicity, there is only one connection from a source host to a different destination host. Note that multiple connections between a source and a destination may be represented by multiple virtual sources and destinations. It is assumed that, if there is a link from node a to node b, then there is also a link from node b to node a.

TABLE 1. Basic notation.

Symbol	Description
G	Directed graph which represents the network
V	Nodes in the network (routers and hosts)
L	Network links
c	A connection in the network
C	Set of connections in the network
f	A subflow
$L(f)$	Set of links traversed by subflow f
F	Set of subflows in the network
$F(l)$	Set of subflows traversing link l
$F(c)$	Set of subflows of connection c
κ_l	Capacity of link l
$T_c^*[c]$	UMMF rate of connection c
$T_f^*[f]$	UMMF rate assigned to subflow f
$P^*[f]$	Portion of the throughput of $c(f)$ carried by subflow f, i.e., $T_f^*[f]/T_c^*[c(f)]$
$S^*[l]$	Saturation level of link l
$R^*[l]$	Set of subflows restricted at link l
$N^*[l]$	Set of subflows which traverse link l but are already restricted at some other link

Connections in the system may demand different maximum capacities from the network because they may be unable to generate data at a higher rate. However, for the sake of simplicity, connections are assumed to demand an infinite capacity from the network. Thus, lower demands are represented by the link connecting the source host to its access router having the capacity demanded by the connection.

A connection may use more than one route to send packets to the destination. A route can be represented by a sequence of links (or routers) that connect a source node to its corresponding destination node. The available routes for each connection are assumed to have been precalculated; therefore, the routing is static. Thus, each subflow has a predetermined path, and paths of the same connection may share routers or even links. Each route corresponds to a different *subflow*. The set of subflows of a connection c is denoted by $F(c)$, and $F = \bigcup_{c \in C} F(c)$ denotes the set of all subflows in the system. The connection to which a subflow f belongs is denoted by $c(f)$. Subflows from the same or different connections may share any number of links. The set of subflows traversing link l is denoted by $F(l)$.

The UMMF rate of a connection c is denoted by $T_c^*[c]$. Likewise, the UMMF rate of a subflow f is denoted by $T_f^*[f]$, so that, for each connection $c \in C$, $T_c^*[c] = \sum_{f \in F(c)} T_f^*[f]$. Then, the portion of the throughput of each connection c that corresponds to each of its subflows $f \in F(c)$ is denoted by $P^*[f] = T_f^*[f]/T_c^*[c]$. Thus, for all $c \in C$,

$\sum_{f \in F(c)} P^*[f] = 1$. A link $l \in L$ is said to be *saturated* if $\kappa_l = \sum_{f \in F(l)} T_f^*[f]$, *unsaturated* if $\kappa_l > \sum_{f \in F(l)} T_f^*[f]$ and *oversaturated* otherwise. A rate allocation is *feasible* if no link is oversaturated.

Intuitively, a feasible rate allocation is UMMF if the rate allocated to any connection cannot be increased, even if all the connections that are allocated a larger rate are removed from the system. This is slightly looser than the definition of GMMF, which states that a feasible rate allocation is GMMF if the only way to increase the rate assigned to a connection is at the expense of connections with a lower or equal rate.

Consider Scenario 1 in Section III as an example. As shown in Table 7, the GMMF rates for connections c_1, c_2 and c_3 are $1.\bar{3}$, $1.\bar{6}$ and $1.\bar{3}$ respectively, where $f_{1,2}$ gets a rate of 0 (i.e., a portion of 0), $f_{1,1}$ and $f_{1,3}$ get a rate of $0.\bar{6}$ each (i.e., a portion of 1/2 each), $f_{2,1}$ gets a rate of $1.\bar{6}$ (i.e., a portion of 1), $f_{3,1}$ gets a rate of $0.\bar{3}$ (i.e., a portion of 1/4), and $f_{3,2}$ gets a rate of 1 (i.e., a portion of 3/4). However, a rate assignment of 1.2 to connections c_1 and c_3, and 1.8 to c_2, where $f_{1,1}$, $f_{1,2}$ and $f_{1,3}$ get a rate of 0.4 each (i.e., a portion of 1/3 each), $f_{2,1}$ gets a rate of 1.8 (i.e., a portion of 1), $f_{3,1}$ gets a rate of 0.2 (i.e., a portion of 1/6), and $f_{3,2}$ gets a rate of 1 (i.e., a portion of 5/6), is UMMF because, even removing connection c_2 (which has the highest rate), it is not possible to increase the rate of any flow of connections c_1 and c_3, since all of them traverse at least one saturated link. Those links are l_3, l_6 and l_{10}. This example shows that a UMMF rate allocation needs not be GMMF. Conversely, it is easy to see that every GMMF rate allocation is UMMF.

The concept of *bottleneck* defined in [35] can be extended to a system where connections may have several subflows, each using a different (not necessarily disjoint) path to route its packets.

Definition 1 [34]: A link $l \in L$ is a bottleneck for a subflow $f \in F(l)$ if l is saturated and there is no other subflow $f' \in F(l)$ such that $T_c^*[c(f')] > T_c^*[c(f)]$.

The following theorem provides a very simple alternative definition of UMMF.

Theorem 1 [27]: A feasible rate allocation is UMMF if and only if every subflow has at least one bottleneck.

Let us introduce now the concept of *saturation level* of both subflows and links under a UMMF rate allocation. The saturation level of a subflow is the quotient $T_f^*[f]/P^*[f]$.

Definition 2: Let l be a saturated link. Then, the saturation level of link l, denoted by $S^*[l]$ is defined as the largest saturation level of the subflows that traverse it, that is, $S^*[l] = \max_{f \in F(l)} T_f^*[f]/P^*[f]$.

A subflow $f \in F(l)$ is saturated at link l under a UMMF allocation if $S^*[l] = T_f^*[f]/P^*[f]$. Otherwise (i.e., $S^*[l] > T_f^*[f]/P^*[f]$), f is unsaturated at link l but, since the allocation is UMMF, it must be already saturated at some other link in its path. Then, the following property easily follows from the previous definitions.

Property 1: Let $N^*[l]$ be the set of subflows that are saturated at some other link l', and let $R^*[l]$ be the set of subflows in $F(l)$ saturated at link l under some UMMF allocation. Then, $F(l) = N^*[l] \cup R^*[l]$ and $S^*[l] = (\kappa_l - \sum_{f \in N^*[l]} T_f^*[f])/\sum_{f' \in R^*[l]} P^*[f']$.

Property 1 can be applied to the development of algorithms that compute UMMF rates in the following way. The saturation level of every unsaturated link is computed on the basis of the currently saturated and unsaturated subflows (initially no subflow is considered saturated). Then, the link (or links) with the lowest saturation level is saturated as well as the still unsaturated subflows that traverse that link, which get the saturation level of this link. This process can be repeated until every subflow is saturated. While the algorithm used by IEWF has a running time which depends on the capacity of the links, the saturation level allows to develop an algorithm whose running time only depends on the number of different saturation levels of the links. Besides, while the rates computed using the approach of IEWF grow by discrete increments, what limits the precision of the computed rate, with the saturation level approach, precise rates can be computed, since rates are continuous instead of discrete.

First, a centralized algorithm is presented, which can be executed offline to compute a UMMF allocation for a static set of connections in a multipath network. This algorithm can be used in a static environment and can also be used as a reference to validate the distributed protocols in practice. Then, a distributed version was developed, that can be used in a dynamic environment.

B. CENTRALIZED ALGORITHM

The proposed algorithm, which we call c-SLEN (centralized Saturation Level Explicit Notification), iteratively adjusts the throughput that each connection sends through each of its subflows until it converges to a UMMF rate assignment in a manner analogous to the IEWF algorithm in [27]. Thus, its correctness follows directly from the correctness of IEWF. However, there is a notable difference between the IEWF and our algorithm. IEWF approximates the bottleneck rates by iteratively incrementing the assigned rates by a fixed amount, whereas our algorithm works in the following way: for each unsaturated link, its saturation level is computed and all links with the lowest saturation level are saturated at once. This guarantees that no link is oversaturated, and allows for faster convergence to fair rates.

Algorithm 1 computes the UMMF rate assignment for each connection and subflow in the system. To do so, it uses a few variables with per-connection, per-subflow, and per-link information, which are summarized in Table 2. Thus, P is an array that stores, for each subflow f, the portion of the throughput of connection $c(f)$ routed through subflow f. Recall that, to be consistent, $\sum_{f \in F(c)} P[f] = 1$ for all $c \in C$. Array T_c stores the throughput assigned to each connection in the system. Likewise, T_f stores the throughput assigned to each subflow in the system. These variables will eventually

TABLE 2. Static variables used by algorithm c-SLEN.

Variable	Description
P	For each connection $c \in C$, for each subflow $f \in F(c)$, $P[f]$ is the portion of the throughput of connection c carried by f
T_c	For each connection c, $T_c[c]$ is the rate assigned to connection c
T_f	For each connection $c \in C$, for each subflow $f \in F(c)$, $T_f[f]$ is the rate assigned to f

Algorithm 1 c-SLEN

```
1:  for all c ∈ C do
2:      for all f ∈ F(c) do
3:          P[f] ← 1/|F(c)|
4:      end for
5:      T_c[c] ← 0
6:  end for
7:  repeat
8:      T' ← T_c
9:      T_f ← ConnectionRates(P)
10:     for all c ∈ C do
11:         T_c[c] ← Σ_{f∈F(c)} T_f[f]
12:         for all f ∈ F(c) do
13:             P[f] ← T_f[f]/T_c[c]
14:         end for
15:     end for
16: until T_c = T'
```

converge to P^*, T_c^* and T_f^* respectively. Algorithm 1 resembles IEWF, what ensures convergence to UMMF rates provided that connection rates are correctly computed at Line 9.

Initially, the throughput assigned to each connection is set to 0 and the portion of that throughput routed through each of its subflows is evenly distributed among them. Then, a loop recomputes the throughput assigned to each connection and subflow until an iteration yields the same former assignment (kept in local variable T'), in which case the algorithm has converged. For practical purposes, an approximation of the equality is enough. At each iteration, a new throughput assignment for each subflow is computed. Then, for each connection, its throughput is computed as the sum of the throughput of its subflows and, for each subflow, its portion is computed as the quotient of its throughput and the throughput of the connection to which it belongs, just like in IEWF.

Function 1 computes and returns the throughput assigned to each subflow in the system. It receives the current portions of each subflow P (which are internally updated) and returns their new rates. To do so, the following local variables (summarized in Table 3) are used. Array R stores, for each link, the set of subflows which are restricted at that link, whereas Array N stores, for each link, the set of connections which traverse that link but are restricted at some other link. The set of links that are already saturated are stored in the variable $SatL$. Likewise, $SatF$ contains the set of subflows

Function 1 ConnectionRates(P): T_f

```
Input: Old subflow portions
Output: New subflow rates
1:  for all l ∈ L do
2:      R[l] ← {f ∈ F(l) : P[f] > 0}
3:      N[l] ← {f ∈ F(l) : P[f] = 0}
4:  end for
5:  SatF ← ⋃_{l∈L} N[l]
6:  SatL ← {l ∈ L : R[l] = ∅}
7:  for all f ∈ F do
8:      T_f[f] ← 0
9:  end for
10: repeat
11:     S ← SaturationLevels(P, R, N, L \ SatL, T_f)
12:     z ← min_{l∈L\SatL} S[l]
13:     for all f ∈ F \ SatF do
14:         T_f[f] ← T_f[f] + z × P[f]
15:     end for
16:     L' ← {l ∈ L \ SatL : S[l] = z}
17:     for all l ∈ L' do
18:         SatL ← SatL ∪ {l}
19:         for all f ∈ R[l] do
20:             SatF ← SatF ∪ {f}
21:             P[f] ← 0
22:             for all l' ∈ L(f) \ L' do
23:                 R[l'] ← R[l'] \ {f}
24:                 N[l'] ← N[l'] ∪ {f}
25:                 if R[l'] = ∅ then
26:                     SatL ← SatL ∪ {l'}
27:                 end if
28:             end for
29:         end for
30:     end for
31:     P ← NewPortions(P, SatF)
32: until SatF = F
33: return T_f
```

TABLE 3. Local variables used in function 1.

Variable	Description
R	For each link $l \in L$, $R[l]$ contains the set of flows which are restricted at link l
N	For each link $l \in L$, $N[l]$ contains the set of flows in $F(l)$ which are not restricted at link l
S	For each link $l \in L$, $S[l]$ contains the saturation level of link l
$SatL$	The set of already saturated links
L'	The set of links to be saturated at the current iteration
$SatF$	The set of already saturated subflows
z	The lowest saturation level among the unsaturated links

that are already saturated. Initially, a subflow f is considered saturated if $P[f] = 0$ and is stored in $N[l]$ for each link l it traverses. Otherwise, it is stored in $R[l]$ for each link l it traverses.

A link is considered saturated if all the subflows that traverse it are already saturated. To start with, all subflows are assigned a throughput of value 0. Then, in a loop, new throughput assignments are incrementally recomputed for each subflow until all subflows become saturated.

Each iteration of the main loop works as follows: First it computes the saturation level of each link (using Function 2). This is the essential difference with IEWF. While IEWF performs small increments in all unsaturated flows (proportionally to their assigned portion) until some unsaturated link becomes saturated, c-SLEN computes the saturation levels of the links directly. Then, it saturates the links with the lowest saturation level, recomputing the throughput of each subflow not previously saturated; finally, it recomputes the portions that correspond to each subflow, in case they are needed for a subsequent iteration (using Function 3).

In order to saturate the links with the lowest saturation level, Property 1 is applied. Thus, all the subflows not previously saturated have their throughput increased by the lowest saturation level times their assigned portion. Thus, the increase in throughput is proportional to their portion and it is guaranteed that all the links with the lowest saturation level will get all their capacity consumed by the subflows that traverse them without exceeding their capacity. It is easy to see that this method is equivalent to the one used in IEWF, but faster, more accurate and not depending on the capacity of the links. Hence, its correctness follows from the correctness of IEWF. Saturated links are included in $SatL$ and all the subflows restricted at those links are moved from $R[l']$ to $N[l']$ at all links l' they traverse, which are not yet saturated. The links l' that have all their traversing subflows in $N[l']$ (so $R[l']$ is empty) are considered saturated (since no connection may get more of their capacity).

Function 2 SaturationLevels($P, R, N, UnsatL, T_f$): S

Input: Current subflow portions, Subflows restricted at each link, Saturated subflows that traverse unsaturated links, Subflow rates
Output: Saturation level of each unsaturated link

1: **for all** $l \in UnsatL$ **do**
2: $\quad p \leftarrow \sum_{f \in R[l]} P[f]$
3: $\quad t \leftarrow \sum_{f \in N[l]} T_f[f]$
4: $\quad S[l] \leftarrow (\kappa_l - t)/p$
5: **end for**
6: **return** S

Function 2 computes the saturation levels of all unsaturated links. The saturation level of a link is the factor by which the throughput of its traversing subflows can be increased (according to their portions) without exceeding its remaining (not already assigned to any subflow) capacity. It is computed as the quotient of the remaining capacity by the sum of the portions of its traversing unsaturated subflows.

Function 3 computes the new portions that correspond to each subflow, according to the current assignment.

Function 3 NewPortions($P, SatF$): P

Input: Current subflow portions, Already saturated subflows
Output: New subflow portions

1: **for all** $c \in C$ **do**
2: $\quad F' \leftarrow F(c) \setminus SatF$
3: $\quad$ **if** $F' \neq \emptyset$ **then**
4: $\quad\quad sum \leftarrow \sum_{f \in F'} P[f]$
5: $\quad\quad$ **for all** $f \in F'$ **do**
6: $\quad\quad\quad P[f] \leftarrow P[f]/sum$
7: $\quad\quad$ **end for**
8: $\quad$ **end if**
9: **end for**
10: **return** P

The portion assigned to each subflow is the quotient of its assigned throughput by the global throughput assigned to its corresponding connection. Only unsaturated subflows are considered, since already saturated ones cannot change their portion, as in IEWF.

In essence, this algorithm is equivalent to IEWF, but improves performance thanks to the application of the concept of saturation level. In fact, the saturation level corresponds to the number of iterations IEWF needs to saturate a link. However, the saturation level needs not be an integer, what results in more accurate rates and avoids oversaturating any link. Thus, c-SLEN serves as a better starting point to develop a distributed protocol.

C. DISTRIBUTED PROTOCOL

In this section we present a distributed protocol called d-SLEN, which resembles the behavior of the centralized algorithm. However, as shown below, it has several advantages over the centralized version and other distributed protocols that compute UMMF rates. For example, it keeps, for each connection, the portions and rates assigned to each of its subflows. In the system, separated data and control planes are considered. Thus, control packets do not consume available throughput in the data plane.

The primitives that provide the interaction between connections and d-SLEN are as follows:

- *API.Join*(): This call is used by a connection to notify d-SLEN that it has joined the system.
- *API.Leave*(): This call is used by a connection to notify d-SLEN that it has terminated.
- *API.Rate*(T): This upcall is used by d-SLEN to notify a connection the rates granted to each of its subflows.

It is assumed that this primitives are invoked coherently, i.e., *API.Join*() is invoked once when a connection starts and *API.Leave*() is invoked once when the connection terminates. Then, *API.Rate*(T) is invoked only while the connection is active.

The behavior of d-SLEN is specified as a set of asynchronous tasks executed at the source and destination of the

TABLE 4. Fields of d-SLEN packets.

Field	Description
f	Subflow identifier
p	p[old] and p[new]: previous and current portion of subflow f
t	t[old] and t[new]: previous and current rate of subflow f
b	Set of potential bottlenecks of subflow f
s	Minimum saturation level for subflow f

connections, and at the network routers, which deploy an event-driven protocol. The basic mechanism used by d-SLEN is the periodic execution of the probe cycles that obtain the minimum saturation level for each subflow. Each cycle starts at the source by sending a *Probe* packet downstream, that is, towards the destination. Then, at the destination, a *ProbeAck* packet is sent upstream, that is, towards the source. While these packets traverse the links in the path, they are updated and the information they carry is also used to update the variables kept at the routers. When the probe cycle ends for all the subflows of a connection, the portions and rates of each subflow are recomputed and notified to the connection, and a new round of probe cycles is started. A *Leave* packet is used to notify that a subflow terminates; therefore, all the information related to this subflow can be deleted. All these packets have the same fields, which are summarized in Table 4.

At the sources, information about the connection parameters is kept. Table 5 summarizes the variables kept at each connection source. More precisely, for each subflow, its current and previous portion and rate are stored in two matrices P and T, respectively, of size $|F(c)| \times 2$. Thus, for each subflow $f \in F(c)$, $P[f]$[old] is the previous portion of subflow f and $P[f]$[new] is the current portion. Likewise, $T[f]$[old] is the previous rate of subflow f and $T[f]$[new] is the current rate. Additionally, the set of apparent bottlenecks for each subflow is stored in (an array of sets of link identifiers) B and their current saturation levels are stored in (an array of numbers) S. Although the list of bottlenecks may be as long as the path length, it is most likely to have a length close to 1. Two auxiliary variables, *cnt* and *leave* are used to count the number of subflows for which the probe cycle has finished, and the fact that the connection has no more data packets to send, so the network resources assigned to that connection can be freed. Hence, the total information needed per connection at the sources is limited to the sum of the length of each subflow's path in the worst case and linear on the number of subflows in the best case, so d-SLEN also scales well in space at the sources.

Task 1 specifies the asynchronous behavior of d-SLEN at the sources. When a connection invokes the *API.Join*() primitive, SourceJoin (Procedure 1) is executed, which starts a round of probe cycles. When a *ProbeAck* packet (which completes a probe cycle) is received at the source, SourceProcessProbeAck (Procedure 3) is executed, which will eventually notify the rates assigned to each subflow and start a new round of probe cycles. This goes on until

TABLE 5. Variables kept at the source of each connection c.

Variable	Description
P	For each subflow $f \in F(c)$, $P[f]$[old] and $P[f]$[new] store the previous and current portion of subflow f
T	For each subflow $f \in F(c)$, $T[f]$[old] and $T[f]$[new] store the previous and current rate granted of subflow f
B	For each subflow $f \in F(c)$, $B[f]$ stores the set of apparent bottlenecks for subflow f
S	For each subflow $f \in F(c)$, $S[f]$ stores the minimum saturation level found for subflow f
cnt	Number of probe cycles completed at the current round
$leave$	TRUE if $API.Leave()$ has been invoked during this round of probe cycles and FALSE otherwise

the connection invokes *API.Leave*() to indicate termination, so SourceLeave() (Procedure 4) is executed to delete every trace of that connection in the system.

Task 1 Source d-SLEN()

```
when API.Join() do
    SourceJoin()
end when
when API.Leave() do
    SourceLeave()
end when
when received ProbeAck(f, p, t, b, s) do
    SourceProcessProbeAck(f, p, t, b, s)
end when
```

Procedure 1 SourceJoin()

```
cnt ← 0
leave ← FALSE
restart ← FALSE
for all f ∈ F(c) do
    P[f][old] ← 0
    P[f][new] ← 1/|F(c)|
    T[f][old] ← 0
    T[f][new] ← 0
    S[f] ← ∞
    B[f] ← ∅
    send downstream Probe(f, P[f], T[f], B[f], S[f])
end for
```

Procedure 1 shows the code executed at the source when a connection starts. It sets the number of probe cycles that have already finished to zero and sets *leave* to FALSE to indicate that this connection has not finished yet. Then, for each subflow, the following is performed: The old and current rates are set to zero, as well as the old portion. However, the current portion is initially set to $1/|F(c)|$, as in the centralized algorithm, since, initially, all subflows are considered to carry the same portion. The first time a bottleneck is found for a subflow, its current rate and subflow will be updated

according to the saturation level of the first bottleneck. Thus, at the links with higher saturation levels, a non-zero rate will be considered, which allows for a more accurate saturation level computation. As the rounds of probe cycles are completed, the portions and the rates are recomputed to converge to a UMMF assignment. The initial saturation level is set to infinity, since a minimum is computed during the subsequent probe cycle. Initially, no bottlenecks have been discovered yet, so the list of bottlenecks is left empty. As the probe cycles finish, the list of bottlenecks is updated. For each subflow, a probe cycle is started to discover the saturation level corresponding to each subflow. These probe cycles are performed concurrently.

Procedure 2 SourceLeave()

```
1: leave ← TRUE
```

Procedure 2 shows the actions performed at the source when the connection leaves the system. Since a new round of probe cycles is started when the current round ends (and probe cycles are always completed), variable *leave* is set to TRUE, so when the current round ends, instead of starting a new one, a *Leave* packet is sent to notify the routers and the destination that this connection has left the network.

Procedure 3 SourceProcessProbeAck(f, p, t, b, s)

```
1: cnt ← cnt + 1
2: S[f] ← s
3: B[f] ← b
4: T[f][old] ← t[new]
5: if P[f][new] = 0 then
6:    T[f][new] ← 0
7: else
8:    T[f][new] ← S[f] × P[f][new]
9: end if
10: if cnt = |F(c)| then
11:    if leave then
12:       doLeave()
13:    else
14:       API.Rate(T)
15:       newProbeCycle()
16:    end if
17: end if
```

Procedure 3 is executed when a *ProbeAck* packet is received at the source. The number of completed probe cycles is incremented and the list of bottlenecks for the subflow and its new saturation level are stored. The new rate that comes in the packet is stored as the old one. A new rate is computed by multiplying the new portion by the minimum saturation level just stored for the subflow. However, if the portion is zero, the saturation level might be infinite, which would yield an indeterminate value. Hence, in the case of a zero portion, the new rate is directly set to zero. Thus, if the subflow configurations of the links in this subflow's path did not change after the *ProbeAck* packed was processed at each node, the new rate will not yield oversaturation of any link. Note that as soon as a the subflow gets a zero portion, it will never increase it, what might cause a temporary underutilization of network capacity. Finally, if the current round of probe cycles has finished and the connection has left the network, Procedure 4 is executed. Otherwise, if the round has finished, but the connection has not left the network, then a new round starts executing Procedure 5.

Procedure 4 doLeave()

```
1: for all f ∈ F(c) do
2:    P[f][old] ← P[f][new]
3:    P[f][new] ← 0
4:    T[f][new] ← 0
5:    send downstream Leave(f, P[f], T[f], B[f], S[f])
6: end for
```

In Procedure 4, for each subflow, the old portion is set to the new one, and the current portion and rate are set to 0 so a *Leave* packet, which resembles the *Probe* packet, is sent downstream. Thus, the resources already allocated to each subflow are freed. The main difference in the processing of the *Leave* packet is that it does not generate a *ProbeAck* at the destination.

Procedure 5 newProbeCycle()

```
1: cnt ← 0
2: t ← Σ_{f∈F(c)} T[f][new]
3: for all f ∈ F(c) do
4:    P[f][old] ← P[f][new]
5:    P[f][new] ← T[f][new]/t
6:    send downstream Probe(f, P[f], T[f], B[f], S[f])
7: end for
8: notify rates
```

When a new round of probe cycles must be started, Procedure 5 is executed. First, it resets the counter *cnt*. Then, the portions are updated for each subflow as follows: The new portion of a subflow is computed by dividing its rate by the sum of the rates of all the subflows of the connection (i.e., its portion of the total) and a new round of probe cycles is started. Finally, it notifies the new rates, so each subflow can send data at the specified rate from then on, at least until the new round of probe cycles ends.

At each link l, the protocol only needs to maintain a constant number of values, summarized in Table 6, regardless of the number of subflows traversing the link, what makes it highly scalable in terms of space. These variables are the sum of the portions of the subflows that are restricted at this link p_l, and the sum of the rates t_l and the sum of the portions p'_l assigned to the subflows which are restricted at some other link. The time needed to process each protocol's packet is constant; therefore, it also scales well in terms of

TABLE 6. Variables kept at each link *l*.

Variable	Description
p_l	The sum of the portions of the subflows restricted at Link l, initially 0
p'_l	The sum of the portions of the subflows restricted at some other link, initially 0
t_l	The sum of the rates granted to the subflows restricted at other links, initially 0
s_l	The saturation level of the link, dynamically computed when needed

computation time. The key parameter of a link at each instant of time is its saturation level s_l, which is recomputed every time it is needed.

Task 2 depicts the asynchronous processing carried out by d-SLEN upon the reception of the three different types of protocol packets.

Task 2 Router d-SLEN()

1: **when received** *Probe*(f, p, t, b, s) **do**
2: LinkProcessProbe(f, p, t, b, s)
3: **end when**
4: **when received** *ProbeAck*(f, p, t, b, s) **do**
5: LinkProcessProbeAck(f, p, t, b, s)
6: **end when**
7: **when received** *Leave*(f, p, t, b, s) **do**
8: LinkProcessleave(f, p, t, b, s)
9: **end when**

Function 4 SaturationLevel() : s_l

1: **if** $\kappa_l - t_l \leq 0$ **then**
2: $s_l \leftarrow 0$
3: **else**
4: $s_l \leftarrow (\kappa_l - t_l)/p_l$
5: **end if**
6: $s'_l \leftarrow \kappa_l/(p_l + p'_l)$
7: **return** $\max(s_l, s'_l)$

The saturation level of the link is estimated using Function 4 as follows: Recall first that the capacity of a link is distributed among the subflows that traverse it proportionally to their portions. Thus, the saturation level of a link is the amount by which the portion of a subflow may be multiplied to obtain a rate that maximizes the utilization of the link without exceeding its capacity. Note also that, if a subflow is restricted in some other link, its minimum saturation level must be lower than the saturation level of this link. However, new subflows may arrive at this link, or some subflow may leave, so it is possible that a subflow that was previously restricted in some other link becomes restricted at this link, i.e., its currently assigned rate exceeds the corresponding value according to the current saturation level of this link. To obtain an accurate estimation of the saturation level, the subflow for which the saturation level is being computed is always considered to be restricted in this link. Besides, two estimations are computed: s_l is the estimation considering that no subflow apparently restricted in some other link is indeed restricted in this link (to avoid possible indeterminate and precision errors, if $\kappa_l - t_l \leq 0$, s_l is set to 0), and s'_l is the estimation considering that all subflows that traverse this link are restricted at this link. The final estimation is the maximum of them, since s_l might be smaller than it should because some subflow apparently restricted at some other link might have been assigned an excessive rate and it will be reduced in the next probe cycle for that subflow.

Procedure 6 LinkProcessProbe(f, p, t, b, s)

1: **if** $l \in b$ **then**
2: $p_l \leftarrow p_l - p[\text{old}] + p[\text{new}]$
3: $s_l \leftarrow SaturationLevel()$
4: **if** $s_l > s$ **then**
5: **if** $|b| = 1$ **then**
6: $s \leftarrow s_l$
7: **else**
8: $p_l \leftarrow p_l - p[\text{new}]$
9: $p'_l \leftarrow p'_l + p[\text{new}]$
10: $t_l \leftarrow t_l + t[\text{new}]$
11: $b \leftarrow b \setminus \{l\}$
12: **end if**
13: **else**
14: $s \leftarrow s_l$
15: **end if**
16: **else**
17: $t_l \leftarrow t_l - t[\text{old}]$
18: $p'_l \leftarrow p'_l - p[\text{old}]$
19: $p_l \leftarrow p_l + p[\text{new}]$
20: $s_l \leftarrow SaturationLevel()$
21: **if** $s_l \leq s$ **then**
22: $s \leftarrow s_l$
23: $b \leftarrow b \cup \{l\}$
24: **if** $t[\text{new}] = 0$ **then**
25: $t[\text{new}] \leftarrow p[\text{new}] \times s$
26: **end if**
27: **else**
28: $p_l \leftarrow p_l - p[\text{new}]$
29: $p'_l \leftarrow p'_l + p[\text{new}]$
30: $t_l \leftarrow t_l + t[\text{new}]$
31: **end if**
32: **end if**
33: **send downstream** *Probe*(f, p, t, b, s)

When a *Probe*(f, p, t, b, s) packet is received at a link l, Procedure 6 is executed with the content of that packet. The main task of the probe cycle is to compute the minimum saturation level in the path. Note first that, when the previous probe cycle was performed, either this link l was found to restrict subflow f (and recorded in b and its portion added to p_l) or otherwise its old rate was added to t_l and its old

portion to p'_l. In the case of the first probe cycle, b is empty and $t[\text{new}] = 0$. Thus, two cases arise:

1) $l \in b$: This link was a bottleneck for this subflow, so its old portion must have been accumulated in p_l. Then, since the portion that corresponds to the subflow may have been updated at the source, p_l must be recomputed by subtracting the previous portion and adding the current portion. Thus, the current saturation level of link l may be recomputed according to the new portion of subflow f. Then, two cases may arise now:
   a) $s_l > s$: Although this link was previously a bottleneck for this subflow, it now seems not to be such. However, if this was the only bottleneck for this subflow, it must remain a bottleneck for this subflow, but the saturation level must be updated to the new (less restrictive) one. This may happen if a subflow traversing this link has left the network, or if some subflow has been restricted in some other link, liberating capacity in this link. Otherwise (i.e., there are other links restricting this subflow), l should not be considered a bottleneck of this subflow from now on. In this case, its portion must be subtracted from p_l (remember that it was previously added to compute the saturation level) and added to p'_l, and its current rate is added to t_l.
   b) $s_l < s$: This link remains being a bottleneck for this subflow, and is now more restrictive than before. Hence, the saturation level of the subflow must be set to s_l.
2) $l \notin b$: This link was not a bottleneck for subflow f. Then, to compute the new saturation level of this link, it is necessary to subtract the old rate from t_l and the old portion from p'_l, and to add the new portion to p_l. This way, we can determine if this link has become a bottleneck for this subflow. Recall that, if this is the first probe cycle for this subflow, then the old rate is zero and the saturation level is infinite.
   a) If the saturation level of this link is as restrictive or more than the currently estimated saturation level of the subflow, then this link is added to the list of bottlenecks for this subflow, and the subflow's saturation level is updated. Additionally, if the current rate of this subflow is zero, probably because this is the first probe cycle for this subflow and this is the first link in the path, the new rate is updated to the one corresponding to this saturation level. Thus, in the following links, this estimated rate will be the current rate that will be accounted for in non-bottleneck links. Note that, if the current portion of this subflow is zero, the current rate will remain at zero.
   b) Otherwise (i.e., this link is not a bottleneck of this subflow), the subflow's portion is subtracted from p_l and added to p'_l, and the subflow's rate is added to t_l.

Finally, the updated *Probe* packet is sent downstream, which completes the processing of the *Probe* packet at this link.

Procedure 7 LinkProcessProbeAck(f, p, t, b, s)

```
1:  if l ∈ b then
2:    s_l ← SaturationLevel()
3:    if s_l > s then
4:      if |b| = 1 then
5:        s ← s_l
6:      else
7:        p_l ← p_l − p[new]
8:        p'_l ← p'_l + p[new]
9:        t_l ← t_l + t[new]
10:       b ← b \ {l}
11:     end if
12:   else
13:     s ← s_l
14:   end if
15: else
16:   t_l ← t_l − t[new]
17:   p'_l ← p'_l − p[new]
18:   p_l ← p_l + p[new]
19:   s_l ← SaturationLevel()
20:   if s_l ≤ s then
21:     s ← s_l
22:     b ← b ∪ {l}
23:   else
24:     p_l ← p_l − p[new]
25:     p'_l ← p'_l + p[new]
26:     t_l ← t_l + t[new]
27:   end if
28: end if
29: send upstream ProbeAck(f, p, t, b, s)
```

The processing of the *ProbeAck* packets is specified in Procedure 7. This is similar to Procedure 6. However, since the corresponding *Probe* packet has already been processed in this link, when the *ProbeAck* packet is processed, the portion and rate that are recorded in p_l or p'_l and t_l respectively are already the new ones and not the old ones, as was the case for the processing of the *Probe* packet. Thus, Line 2 in Procedure 6 is not necessary in Procedure 7. Lines 17-18 in Procedure 6 have been changed for Lines 16-17 in Procedure 7, since the rate already accumulated in t_l and the portion in p'_l are the new ones and not the old ones, as stated above. Finally, the updated packet is sent upstream.

Procedure 8 depicts how *Leave* packets are processed at the links. For orthogonality, the fields in the packet are the same as those in the other protocol packets. However, the only information needed is the list of bottlenecks and the old portion in case the link was a bottleneck for the subflow, and the old rate and portion in case the link was not a bottleneck of the subflow. The purpose of this packet is to clear any information concerning this subflow in all the links of its path, so the subflow's portion and rate are subtracted

from p_l or p'_l and t_l respectively, depending on whether l is a bottleneck for f or not, what liberates the capacity assigned to this subflow. Finally, the *Leave* packet is sent downstream.

Procedure 8 LinkProcessLeave(f, p, t, b, s)

```
if l ∈ b then
    p_l ← p_l − p[old]
else
    t_l ← t_l − t[old]
    p'_l ← p'_l − p[old]
end if
send downstream Leave(f, p, t, b, s)
```

When a *Probe* packet reaches the destination, it is transformed into a *ProbeAck* packet with the same content, and sent upstream to complete the probe cycle, as shown in Task 3. When a *Leave* packet is received at the destination, the subflow is closed, and the packet is discarded.

Task 3 Destination d-SLEN()

```
when received Probe(f, p, t, b, s) do
    send upstream ProbeAck(f, p, t, b, s)
end when
```

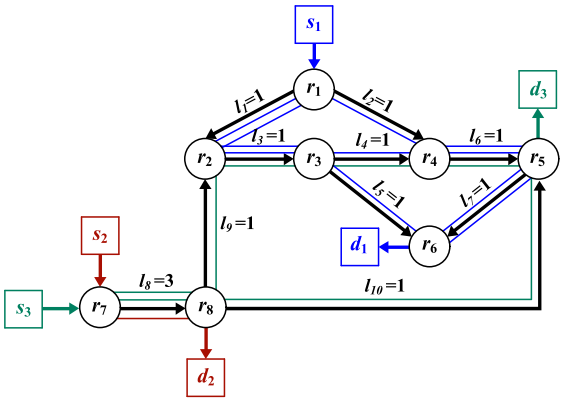
III. EXPERIMENTAL EVALUATION

In this section, we describe the evaluation of the proposed algorithms in practical scenarios as well as the comparison of their performances with those of other algorithms. Various parameters, such as fairness and convergence speed, were evaluated.

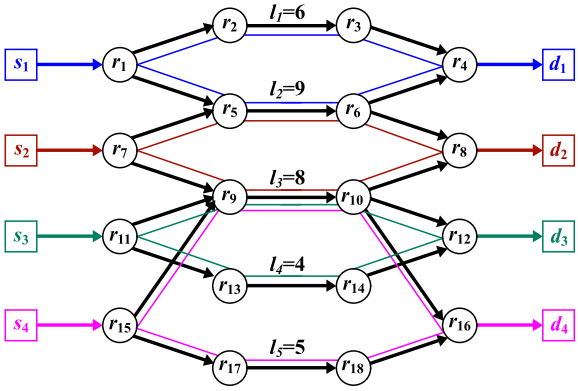
A. SCENARIOS

To test how the proposed algorithms behave in practice, three scenarios with different and typical network problems were deployed. Connections are identified with different colors. The hosts are represented by squares and routers by circles. The source and destination nodes of a connection are given a label that starts with s and d, respectively, and a unique number that identifies that connection. Connections are identified by letter c and a unique number that identifies the source and destination of that connection. For example, connection c_1 goes from host s_1 to host d_1. Subflows are identified by letter f, a connection number, and a subflow number (unique per connection). For example, subflow $f_{1,1}$ is a subflow of connection c_1. Router nodes are identified by a label formed by letter r and a unique number. Inter-router links are represented by black arrows with their labels (letter l plus a unique number) and capacity (in Mbps). The links that connect hosts to routers have infinite capacity, so only inter-router links limit the available throughput of the connections. The paths used by each connection are identified by the color of the connection.

In Scenario 1 [32], there are eight routers and three connections with three, one, and two subflows, respectively.



SCENARIO 1. Connections with different number of subflows [32].



SCENARIO 2. Connections with the same number of subflows [34].

The subflows follow these routes (represented by the sequence of routers traversed) $f_{1,1} = [r_1, r_2, r_3, r_4, r_5, r_6]$, $f_{1,2} = [r_1, r_2, r_3, r_6]$, $f_{1,3} = [r_1, r_4, r_5, r_6]$, $f_{2,1} = [r_7, r_8]$, $f_{3,1} = [r_7, r_8, r_2, r_3, r_4, r_5]$, and $f_{3,2} = [r_7, r_8, r_5]$. Note that subflows share more than one link with subflows of other connections, which complicates the dependencies among them and, consequently, among connections. Subflows that belong to the same connection can also share links. For example, subflow $f_{1,1}$ shares a link with subflow $f_{1,2}$, and another link with $f_{1,3}$. Besides, both links are shared with subflow $f_{3,1}$.

Scenario 2 corresponds to scenario a) in [34]. It has four connections, all of which have two subflows (paths). There is a link shared by three subflows, and one shared by two subflows. The other links are used only by one subflow each. Note that the capacities of the links are much more varied than in the previous case. The subflows follow these routes $f_{1,1} = [r_1, r_2, r_3, r_4]$, $f_{1,2} = [r_1, r_5, r_6, r_4]$, $f_{2,1} = [r_7, r_5, r_6, r_8]$, $f_{2,2} = [r_7, r_9, r_{10}, r_{12}]$, $f_{3,1} = [r_{11}, r_9, r_{10}, r_{12}]$, $f_{3,2} = [r_{11}, r_{13}, r_{14}, r_{12}]$, $f_{4,1} = [r_{15}, r_9, r_{10}, r_{16}]$, and $f_{4,2} = [r_{15}, r_{17}, r_{18}, r_{16}]$.

In addition, a larger scenario was used to test the distributed protocols. This scenario aims to emulate a more realistic environment through the specification of a high-speed, high-density network consisting of 110 routers.

They were randomly arranged according to a transit stub topology. Transit routers are interconnected by 50 Mbps links. Stub routers are connected to other routers by 10 Mbps links. The speed of links connecting hosts to stub routers was set to 10 Gbps to ensure that they were not bottlenecks. However, because these links connect only one host, they can be used to simulate the maximum rate demanded by a connection. In this scenario, 40 connections (pairs of source-destination hosts) were randomly configured to spawn during the first 5 microseconds of the simulation. The connections behave as follows. Each of them is randomly assigned a certain amount of data (between 1 MB and 10 MB) to be sent. Once the connection has transmitted all the data, it leaves the system. Thus, the protocol must be able to react to these changes and recompute the new subflow (and connection) rates when connections join or leave the network.

Due to their small size, in Scenarios 1 and 2, connections use all available routes between a source and a destination. In the case of Scenario 3, a configuration parameter k (which was set to 5 for the experiments) limits the maximum number of routes that can be used by a connection during the simulations. Then, the number of subflows of each connection is selected randomly between 1 and k. The routes are computed using Yen's algorithm, which discovers the k shortest loopless paths between a source and a destination [36]. The version of Yen's K Shortest Path used for the experiments was that in JGraphT [37].

B. SIMULATION

The results obtained for the centralized algorithms were collected by simulating the scenarios using MATLAB [38] and GNU Octave [39].

The results obtained for the distributed protocols were collected using an improved version of Peersim discrete event simulator [40]. To configure a scenario, first the network topology was generated using the GT-ITM (Georgia Tech Internetwork Topology Models) topology generator. Thus, the output obtained can be used as many times as necessary for multiple runs with the same network configuration. The start time and the amount of data to be transmitted for each connection were randomly generated and stored so that they could also be used for multiple runs.

Several characteristics regarding the network links must be highlighted. First, a connection is established through the creation of two links in opposite directions between the two target nodes. Thus, we allow for both downstream and upstream traffic to be transmitted simultaneously and correctly routed, emulating a full-duplex communication system. The link queues are unlimited, to avoid packet losses. Second, separate queues were established at the link level for the transmission of control and data packets. This decision has been made for the sake of simplicity, and taking into consideration that advances in software-defined networks show the effectiveness of separating the control plane from the data plane [41]. Finally, as the simulation runs, data regarding the state of the network were collected periodically and recorded in a logging file. This file is then processed to extract different information, for example, the number of packets in each link queue, which can be used as evidence of congestion, or the rate assigned to each subflow and connection, which can be used to evaluate the speed at which the algorithm is able to converge to the fair rates.

TABLE 7. UMMF Rate allocation per connection and subflow in Scenario 1.

Connection	Subflow	GMMF		Algorithm UMMF			
				c-SLEN		IEWF	
c_1	$f_{1,1}$	$1.\overline{3}$	$0.\overline{6}$	1.20	0.40	1.20	0.40
	$f_{1,2}$		0.0		0.40		0.40
	$f_{1,3}$		$0.\overline{6}$		0.40		0.40
c_2	$f_{2,1}$	$1.\overline{6}$	$1.\overline{6}$	1.80	1.80	1.75	1.75
c_3	$f_{3,1}$	$1.\overline{3}$	$0.\overline{3}$	1.20	0.20	1.25	0.23
	$f_{3,2}$		1.0		1.00		1.02
Total throughput		$4.\overline{3}$		4.20		4.20	

C. CENTRALIZED ALGORITHMS

In this section, the proposed c-SLEN algorithm is compared with IEWF [27], which is the reference algorithm for centrally computing UMMF rates in a multipath network. Scenarios 1 and 2 were used for the comparison.

IEWF assigns rates to connections as multiples of a *unit of subflow*. The smaller the unit of subflow, the more precise the convergence, but more iterations are required to converge to the fair rates. Note also that the oversaturation induced on a link depends on this unit of subflow and the subflows that traverse it; IEWF never undersaturates a bottleneck. For the experiments, the unit of subflow was set to 0.05 Mbps, which represents 5% of the capacity of the link with the smallest capacity.

In the simulations, the rates are considered stable when, for each connection, its current rate differs from its previous rate by less than 0.001 Mbps. Therefore, it is possible that the rate allocations did not reach the UMMF rates even when they were considered stable.

The rate allocations produced by the two algorithms for Scenario 1 are compared with the MMF rates in Table 7. Note that, although the rates calculated by IEWF seem to be closer to the MMF rates, the rate assigned to connection c_3 incurs oversaturation of the three links, as shown in Table 8 in red color. This oversaturation is a consequence of the unit of subflow used and can be reduced at the expense of longer convergence times. However, c-SLEN never incurs oversaturation, which is guaranteed by computing the rates as a function of the minimum saturation level for each subflow. In this scenario, the UMMF assignments do not reach the MMF utility, although they are reasonably close to it. Recall that UMMF is a relaxation of MMF, so they are not always the same.

Table 9 lists the rate assignments for Scenario 2. The overall results are similar and closer to the MMF rates than those for Scenario 1. However, c-SLEN is slightly closer to

TABLE 8. Saturation in bottleneck links in Scenario 1.

Link	Capacity	Throughput	
		c-SLEN	IEWF
l_1	1	0.80	0.80
l_2	1	0.40	0.40
l_3	1	1.00	1.03
l_4	1	0.60	0.63
l_5	1	0.80	0.80
l_6	1	1.00	1.03
l_7	1	0.80	0.80
l_8	3	3.00	3.00
l_9	1	0.20	0.23
l_{10}	1	1.00	1.02

TABLE 9. UMMF Rate allocation per connection and subflow in Scenario 2.

Connection	Subflow	GMMF	Algorithm UMMF			
			c-SLEN		IEWF	
c_1	$f_{1,1}$	8.00	8.01	6.00	8.15	6.03
	$f_{1,2}$			2.01		2.12
c_2	$f_{2,1}$	8.00	8.00	6.99	8.10	6.90
	$f_{2,2}$			1.01		1.20
c_3	$f_{3,1}$	8.00	7.99	3.99	7.95	3.90
	$f_{3,2}$			4.00		4.05
c_4	$f_{4,1}$	8.00	7.99	2.99	7.95	2.93
	$f_{4,2}$			5.00		5.02
Total throughput		32.00	32.00		32.15	

TABLE 10. Saturation in bottleneck links in Scenario 2.

Link	Capacity	Throughput	
		c-SLEN	IEWF
l_1	6	6.00	6.03
l_2	9	9.00	9.02
l_3	8	8.00	8.03
l_4	4	4.00	4.05
l_5	5	5.00	5.02

the MMF rates, when the rates are considered stable. In this case, the MMF utility is achieved by both c-SLEN and IEWF. Although IEWF seems to improve that utility, it is just a result of the oversaturation produced on all the bottlenecks, as shown in Table 10. As expected, this saturation was due to the subflow unit used.

Let us now focus on the running time of the algorithms. c-SLEN and IEWF were compared in terms of the running times. To do so, at each iteration of the main loop of each algorithm, the running time and current rate assignments were traced. Both algorithms were run on an Intel(R) Core(TM) i9-10900 CPU @ 2.80GHz with 32 GiB RAM, under UBUNTU 20.04 using GNU Octave, version 5.2.0. The results obtained are shown in Figures 1 and 2.

For both scenarios, c-SLEN converges faster than IEWF, although this is more evident in the case of Scenario 2. Although it is not shown, it is easy to see that the convergence speed of IEWF depends on the capacity of the links and

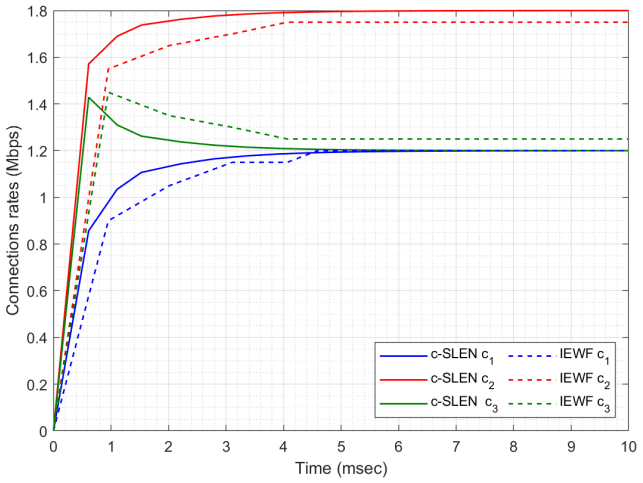


FIGURE 1. Convergence speed for c-SLEN versus IEWF in Scenario 1.

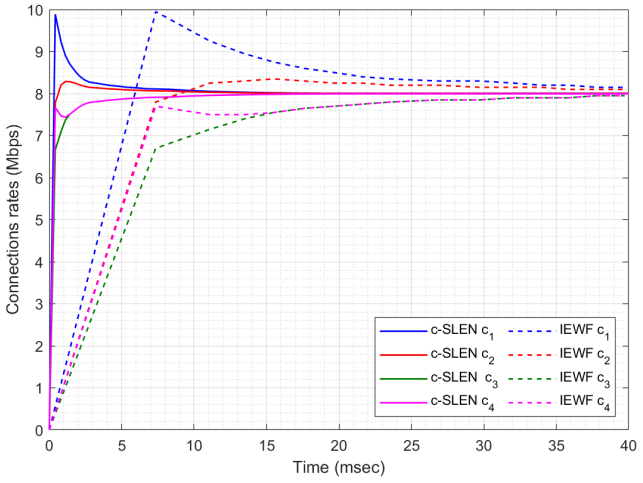


FIGURE 2. Convergence speed for c-SLEN versus IEWF in Scenario 2.

the unit of subflow (which has a clear impact on the rate's accuracy), whereas the convergence speed of c-SLEN does not depend on the capacity of the links and its accuracy is always maximal.

The overall conclusion is that c-SLEN achieves good utilization of network capacity, fairness and fast convergence, while never oversaturating any link.

D. DISTRIBUTED PROTOCOLS

In this section, the d-SLEN protocol is compared with a distributed protocol that computes the UMMF rates in a multipath network, proposed by Vo *et al.* [34]. We refer to this protocol as Vo-Le-Tran (called after their authors). These protocols were evaluated in the three proposed scenarios.

The Vo-Le-Tran protocol relies on a constant γ_l to modulate the amplitude of the possible oscillations until convergence is reached. Its value was set to 0.002, as proposed by the authors [34].

Figure 3 shows, for each connection, how its rate allocation evolves over time, since it joins the system, until it leaves once

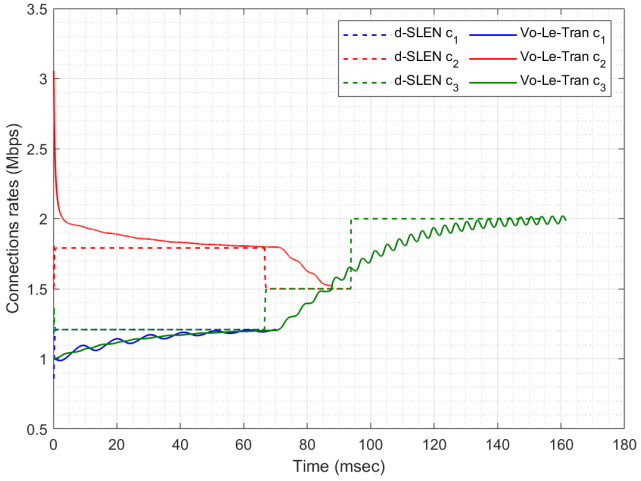


FIGURE 3. Rate allocation per connection in Scenario 1.

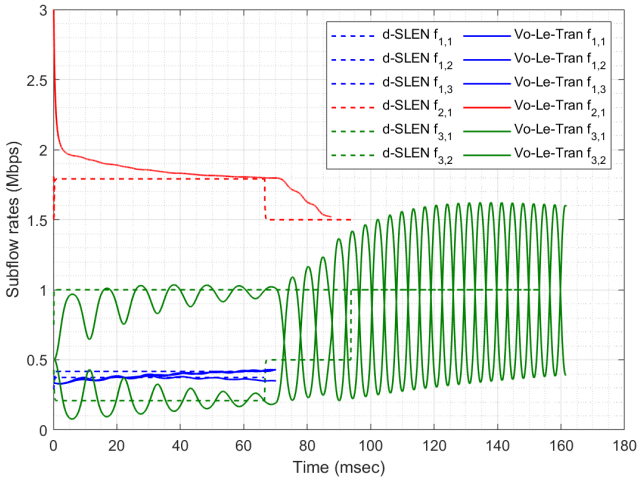


FIGURE 4. Rate allocation per subflow in Scenario 1.

it has transmitted all its data. The most notable difference between both protocols is that, whereas d-SLEN is able to react almost immediately to connections joining or leaving the network, Vo-Le-Tran converges slowly to target rates. Note also the oscillating nature of its design.

Oscillations on subflow allocations are shown in Figure 4. This figure also shows that, while connection c_2 ends first (since it converges slowly to the fair rates from above) in the case of Vo-Le-Tran, connection c_3 (the last to leave) in green, ends first in the case of d-SLEN.

As shown in Figure 5, the link queues reach a maximum length of 3 (the maximum number of subflows per link) in the case of d-SLEN, whereas in the case of Vo-Le-Tran, they are even larger when only one connection remains in the system.

The evolution of the rate allocations in Scenario 2 is shown in Figure 6. As shown in [34], the rate assignments of Vo-Le-Tran oscillate for several seconds in this scenario. However, as shown in Figure 7, d-SLEN converges rapidly to fair rates, with almost no oscillation.

The oscillations of subflow rates in the case of Vo-Le-Tran are even more evident than those of connection rates,

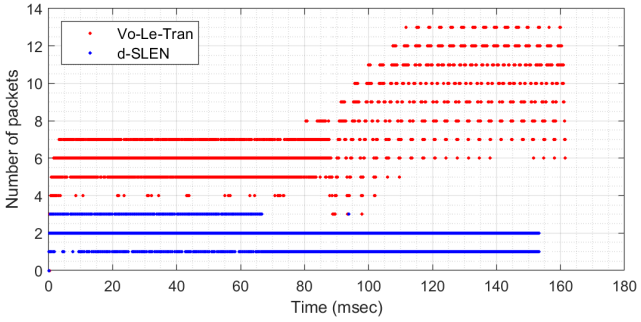


FIGURE 5. Maximum queue size in the links for Scenario 1.

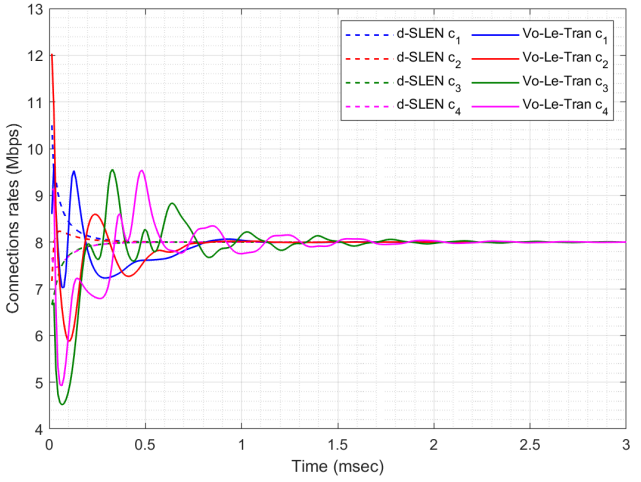


FIGURE 6. Rate allocation per connection in Scenario 2.

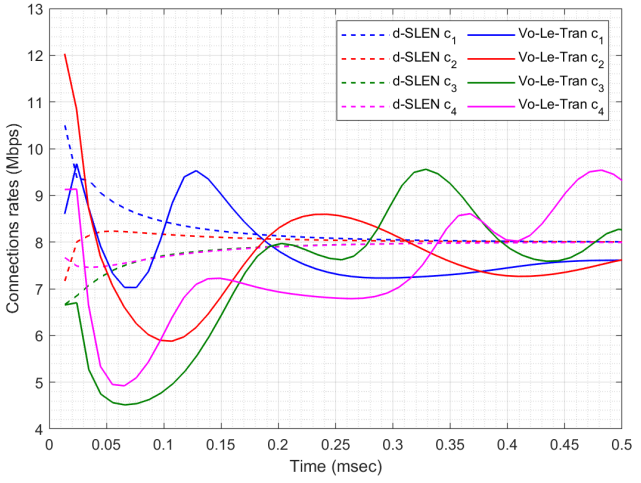


FIGURE 7. Rate allocation per connection in Scenario 2 (first 0.5 seconds).

as shown in Figures 8 and 9, whereas only one subflow oscillates slightly once in the case of d-SLEN.

Regarding the evolution of queue lengths in the links (see Figure 10), as in Scenario 1, the queue lengths in the case of d-SLEN remain extremely small, whereas they are several times larger in the case of Vo-Le-Tran.

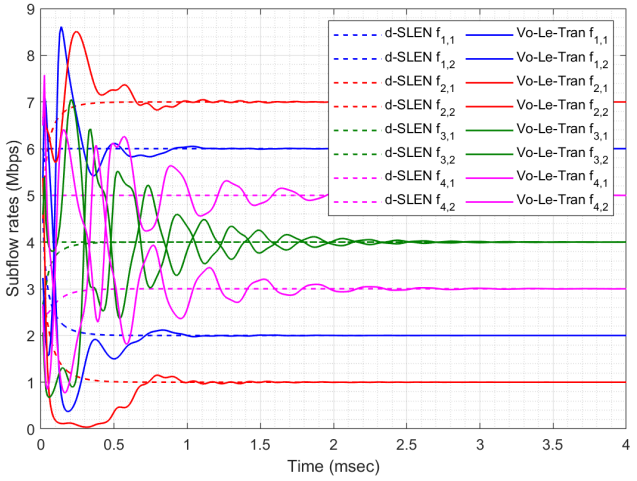


FIGURE 8. Rate allocations per subflow in Scenario 2.

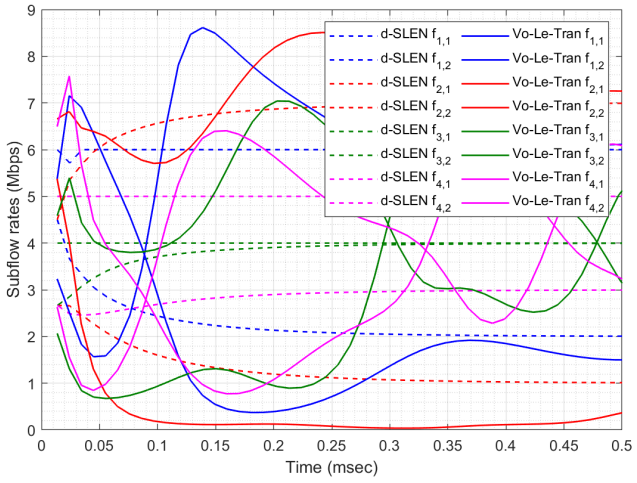


FIGURE 9. Rate allocations per subflow in Scenario 2 (first 0.5 miliseconds).

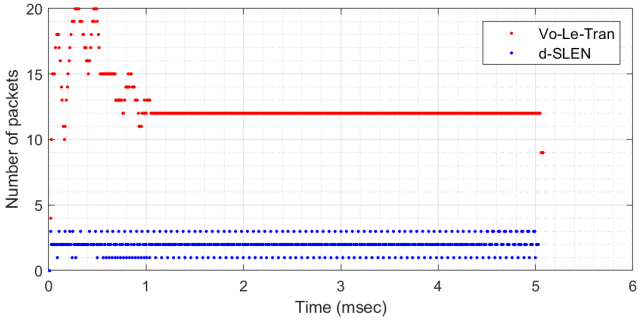


FIGURE 10. Maximum queue size in the links for Scenario 2.

Figures 11 and 12 show the evolution of the rate allocations per connection for the d-SLEN and Vo-Le-Tran protocols, respectively. During stable periods, both protocols allocated the same rates to the connections. However, d-SLEN reacts immediately to connection leaves, while Vo-Le-Tran needs half a second in several cases to reach a stable rate. There are

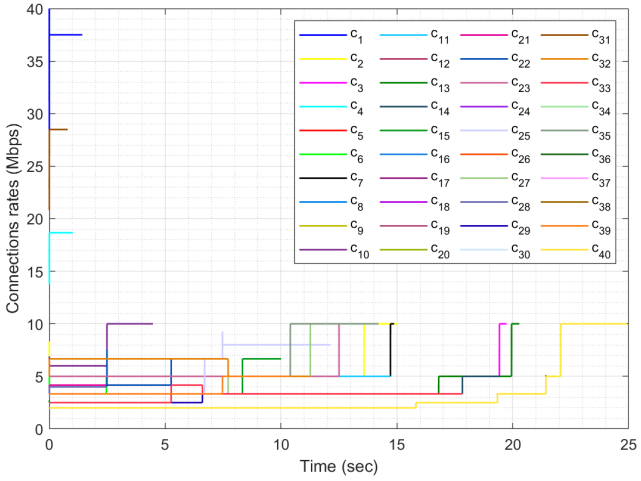


FIGURE 11. Rate allocation per connection in Scenario 3 (d-SLEN).

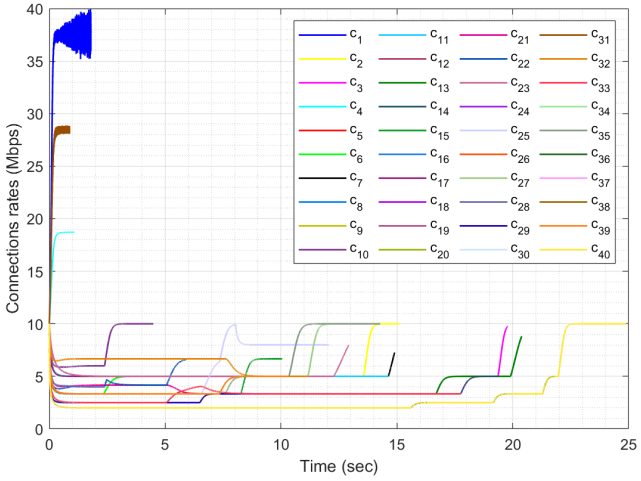


FIGURE 12. Rate allocation per connection in Scenario 3 (Vo-Le-Tran).

even cases in which the connection leaves before reaching a stable rate.

One extreme case is that of connection c_1. Figure 13 shows how its subflow rates keep oscillating during the first 0.5 seconds. In fact, these rates never become stable, as shown in Figure 12. In contrast, d-SLEN exhibits no oscillations and immediately reaches stable rates.

An example where Vo-Le-Tran does not oscillate is shown in Figure 14. It shows the last seconds before the connection leaves the system. In this case, the convergence to the stable rates is very slow in the case of Vo-Le-Tran, compared to that of d-SLEN.

Finally, in Figure 15, the maximum queue length in the links is shown. In this case, the maximum queue lengths in the case of Vo-Le-Tran are one order of magnitude higher than those of d-SLEN. In fact, the maximum number of packets in a link queue is of the order of the number of subflows that traverses it, which is the optimal length. Therefore, d-SLEN effectively avoids congestion while achieving fairness and high utilization of network capacity.

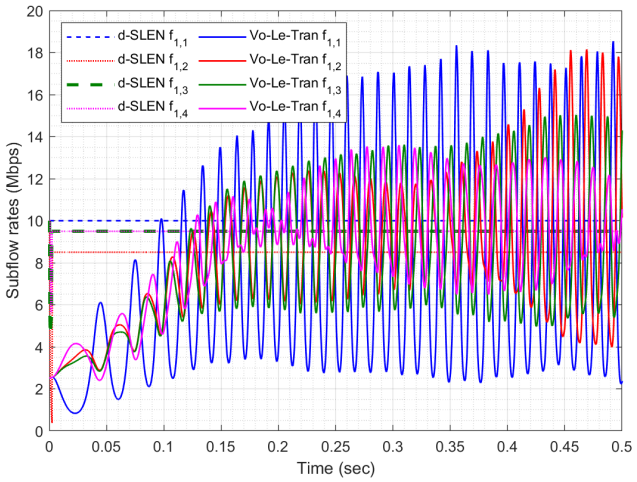


FIGURE 13. Subflow rates of connection 1 in Scenario 3 during the first 0.5 seconds.

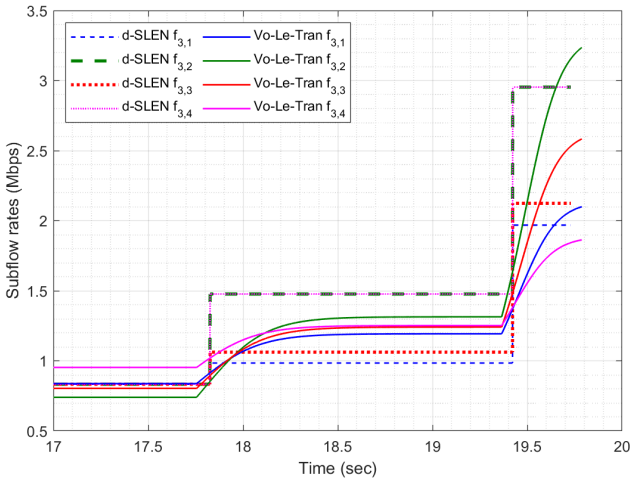


FIGURE 14. Subflow rates of connection 3 in Scenario 3 during the last 3 seconds.

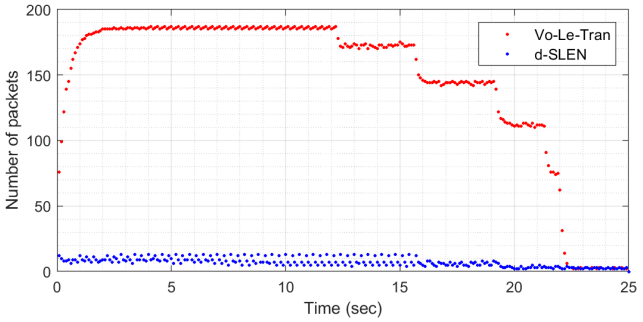


FIGURE 15. Maximum queue size in the links for Scenario 3.

IV. DISCUSSION

Traditionally, connections linearly increase their transmission rates until they detect congestion and then, drastically reduce their rates. This yields oscillations in transmission rates and packet losses. This reactive nature has some major limitations: it does not prevent congestion, generates fluctuations in throughput, is not able to use the full capacity of the network links and requires long packet queues in the routers to try to avoid packet losses. Therefore, proactive congestion control is a good alternative, as long as it does not involve long delays and provides a good balance between fairness and network utilization.

Since the network may offer many possible paths to route the packets of a connection, to get the most out of the network, it is desirable to be able to use multiple paths for each connection and not to be restricted to just one.

Although MMF can be achieved with low-cost algorithms in single-path networks [17]–[20], that is not the case when dealing with multipath, where algorithms require solving a series of linear programming problems with non-negligible computational cost [28], [33], [42]. Thus, UMMF [27] has been proposed as a relaxation of max-min fairness that can be achieved by purely combinatorial algorithms, which makes this fairness criterion a better candidate for use in multipath networks.

In [27], a centralized algorithm (IEWF) to compute the UMMF rates for a set of connections in a multipath environment was proposed. This algorithm is a generalization of the waterfilling algorithm [35] and iteratively increments the rates of the subflows until a link becomes saturated. However, centralized network control is not always a valid option. In addition, the accuracy of IEWF is at odds with its convergence speed. When this algorithm is adapted to a distributed environment, some issues may arise, as in [34]. The protocol proposed in [34] depends on a parameter that needs to be tuned and not always the best convergence time and/or accuracy is achieved with the same value. Since rates are iteratively incremented until the links become saturated, the convergence time increases with the capacity of the network. This protocol also suffers large fluctuations in rates and has large convergence time.

The introduction of the concept of *saturation level* opens a new method to solve this problem. Knowing the minimum saturation level for a subflow, its rate can grow directly to the maximum possible without the need for small increments. Thus, the convergence time does not depend on the capacity of the network. A subflow may compute the minimum saturation level in its path in one probe cycle, that is, in one RTT, it is possible to compute the throughput available to a subflow, so it can start sending data from the first RTT at the computed rate knowing that it will not oversaturate the network, provided that the workload does not change. In fact, as d-SLEN does, when a round of probe cycles for all the subflows of a connection ends, the new portions and rates are computed, and a new round of probe cycles may start. Thus, the minimum saturation level is recomputed after one RTT, allowing a quick reaction to changes in workload.

d-SLEN can be used to provide proactive congestion avoidance and fairness among connections in many different contexts where fairness is required or desirable and multipath is available. This type of protocol is often blamed for requiring multiple RTTs to achieve convergence; therefore, short

connections can be unnecessarily delayed. However, d-SLEN allows the subflows to start sending packets at a secure rate after the first probe cycle, that is, the first RTT. Thus, it can be used even in short connections.

Now, consider the case of different companies with several large data centers located at many different places all around the world, which are connected through a backbone of one or various providers. It is likely that all the traffic between any two data centers could be treated as a single connection (since it could be transmitted through a tunnel) and will last long. In this case, d-SLEN can be used to fairly share the available capacity among the different tunnels using multiple paths for each of them, preventing congestion and, therefore, improving performance by avoiding packet losses and retransmissions.

Since a connection may have a limited capacity to generate traffic, it is interesting for connections to be able to explicitly demand the highest rate at which they can transmit, so the allocated bandwidth will not exceed that value. This can be easily simulated by a virtual link at the source with the requested capacity, so that d-SLEN can be used directly in such an environment.

As the number of subflows that cross a router grows, it is prohibitive for routers to store individual information per subflow. Therefore, for a congestion protocol to be of practical use, it must not need to store information about individual subflows at the routers. However, aggregated information can be stored and maintained, because it requires a constant amount of memory and processing time. Thus, d-SLEN meets the requirements for practical use.

The case of single-path networks is a particular case of multipath, where each connection has just one subflow with a portion of 1. Therefore, this protocol can also be directly applied to single-path networks.

If *QoS* needs to be used to prioritize traffic, a way to apply it could be to introduce a multiplying factor $0 < f \leq 1$ to the portion assigned to a subflow at the source, so it would get a smaller portion of the capacity of the links it traverses, with respect to other subflows with higher priority (and equal portions). A protocol based on the concept of saturation level (such as d-SLEN) is able to directly apply this QoS mechanism since only the sources are affected.

V. CONCLUSION

The concept of saturation level allows for the development of new congestion control protocols, for multipath networks, whose convergence time to the optimal rates does not depend on the capacity of the network.

The first centralized algorithm (c-SLEN) based on this concept is proposed. This algorithm exhibits a much faster convergence to UMMF rates than its predecessor IEWF.

A distributed version of this algorithm (d-SLEN) was developed. This protocol does not need to keep information per subflow at the routers, and the time needed to process each of its packets is constant, so this protocol scales well, which makes it a good candidate for real deployments. By simulation, we have shown that d-SLEN produces a remarkable improvement over state-of-the-art protocols. It converges almost immediately to stable rates and can be used in a dynamic environment where connections arrive and leave the network at any time. It achieves fairness (UMMF). It is able to keep queue sizes as small as possible, which shows its effectiveness in avoiding congestion. Thus, it seems to be a good alternative for many types of high-speed networks.

The proposed design has differentiated data and control planes. However, for future work, it might be interesting to consider the possibility of eliminating this distinction, thus routing control and data packets together.

REFERENCES

[1] N. Mareev, D. Kachan, K. Karpov, D. Syzov, and E. Siemens, "Efficiency of BQL congestion control under high bandwidth-delay product network conditions," in *Proc. 7th Int. Conf. Appl. Innov. (IT)*, 2019, pp. 19–22.
[2] S. Huang, D. Dong, and W. Bai, "Congestion control in high-speed lossless data center networks: A survey," *Future Gener. Comput. Syst.*, vol. 89, pp. 360–374, Dec. 2018.
[3] S. Floyd, "TCP and explicit congestion notification," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 5, pp. 8–23, Oct. 1994.
[4] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ECN marks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 689–702, Oct. 2003.
[5] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, Oct. 2015, doi: 10.1145/2829988.2787510.
[6] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. Renton, WA, USA: USENIX Association, 2018, pp. 329–342. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/arun
[7] Y. R. Yang and S. S. Lam, "General AIMD congestion control," in *Proc. Int. Conf. Netw. Protocols*, Nov. 2000, pp. 187–198.
[8] M. Corless, C. King, R. Shorten, and F. Wirth, *AIMD Dynamics and Distributed Resource Allocation*. Philadelphia, PA, USA: SIAM, 2016.
[9] L. Jose, "Proactive congestion control," Ph.D. dissertation, Stanford Univ., Stanford, CA, USA, Jan. 2019.
[10] G. Huston, "Gigabit TCP," *Internet Protocol J.*, vol. 9, no. 2, pp. 2–26, Jun. 2006.
[11] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, p. 83–91, Apr. 2003, doi: 10.1145/956981.956989.
[12] A. Afanasyev, N. Tilley, P. Reiher, and L. Kleinrock, "Host-to-host congestion control for TCP," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 3, pp. 304–342, 3rd Quart., 2010.
[13] T. Kato, S. Haruyama, R. Yamamoto, and S. Ohzahata, "mpCUBIC: A CUBIC-like congestion control algorithm for multipath TCP," in *Proc. World Conf. Inf. Syst. Technol.* Cham, Switzerland: Springer, 2020, pp. 306–317.
[14] M. Kheirkhah, I. Wakeman, and G. Parisis, "Multipath transport and packet spraying for efficient data delivery in data centres," *Comput. Netw.*, vol. 162, Oct. 2019, Art. no. 106852.
[15] N. Cardwell, Y. Cheng, S. Yeganeh, and V. Jacobson, "BBR congestion control," in *Proc. Working Draft, IETF Secretariat, Internet-Draft Draft-Cardwell-ICCRG-BBR-Congestion-Control-00*, 2017, pp. 1–36.
[16] T. Zhu, X. Qin, L. Chen, X. Chen, and G. Wei, "WBBR: A bottleneck estimation-based congestion control for multipath TCP," in *Proc. IEEE 88th Veh. Technol. Conf. (VTC-Fall)*, Aug. 2018, pp. 1–5.
[17] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, "High speed networks need proactive congestion control," in *Proc. 14th ACM Workshop Hot Topics Netw.*, Nov. 2015, p. 14, doi: 10.1145/2834050.2834096.

[18] A. Mozo, J. L. López-Presa, and A. Fernández Anta, "A distributed and quiescent max-min fair algorithm for network congestion control," *Expert Syst. Appl.*, vol. 91, pp. 492–512, Jan. 2018, doi: 10.1016/j.eswa.2017.09.015.

[19] L. Jose, S. Ibanez, M. Alizadeh, and N. McKeown, "A distributed algorithm to calculate max-min fair rates without per-flow state," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 2, pp. 21:1–21:42, Jun. 2019, doi: 10.1145/3341617.3326135.

[20] A. Mozo, J. L. Lopez-Presa, and A. F. Anta, "SLBN: A scalable max-min fair algorithm for rate-based explicit congestion control," in *Proc. IEEE 11th Int. Symp. Netw. Comput. Appl.*, Aug. 2012, pp. 212–219, doi: 10.1109/NCA.2012.40.

[21] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1492–1525, 2nd Quart., 2017.

[22] D. Nace and M. Pióro, "Max-min fairness and its applications to routing and load-balancing in communication networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 5–17, 4th Quart., 2008.

[23] E. Amaldi, A. Capone, S. Coniglio, and L. G. Gianoli, "Network optimization problems subject to max-min fair flow allocation," *IEEE Commun. Lett.*, vol. 17, no. 7, pp. 1463–1466, Jul. 2013.

[24] Z. Cao and E. W. Zegura, "Utility max-min: An application-oriented bandwidth allocation scheme," in *Proc. 18th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2, Mar. 1999, pp. 793–801.

[25] F. Xu, W. Ye, Y. Liu, and W. Zhang, "UFalloc: Towards utility max-min fairness of bandwidth allocation for applications in datacenter networks," *Mobile Netw. Appl.*, vol. 22, no. 2, pp. 161–173, Apr. 2017.

[26] J. Jin, W.-H. Wang, and M. Palaniswami, "Utility max–min fair resource allocation for communication networks with multipath routing," *Comput. Commun.*, vol. 32, no. 17, pp. 1802–1809, Nov. 2009, doi: 10.1016/j.comcom.2009.06.014.

[27] E. Danna, A. Hassidim, H. Kaplan, A. Kumar, Y. Mansour, D. Raz, and M. Segalov, "Upward max-min fairness," *J. ACM*, vol. 64, no. 1, pp. 1–24, Mar. 2017, doi: 10.1145/3011282.

[28] M. Allalouf and Y. Shavitt, "Centralized and distributed algorithms for routing and weighted max-min fair bandwidth allocation," *IEEE/ACM Trans. Netw.*, vol. 16, no. 5, pp. 1015–1024, Oct. 2008, doi: 10.1109/TNET.2007.905605.

[29] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A centralized 'zero-queue' datacenter network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 307–318, Feb. 2015, doi: 10.1145/2740070.2626309.

[30] M. Mahdian, S. Arianfar, J. Gibson, and D. Oran, "MIRCC: Multipath-aware ICN rate-based congestion control," in *Proc. 3rd ACM Conf. Inf.-Centric Netw.*, Sep. 2016, pp. 1–10.

[31] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Quality of Service—IWQoS 2005*, H. de Meer and N. Bhatti, Eds. Berlin, Germany: Springer, 2005, pp. 271–285.

[32] G. Retvari, J. J. Biro, and T. Cinkler, "Fairness in capacitated networks: A polyhedral approach," in *Proc. IEEE INFOCOM 26th IEEE Int. Conf. Comput. Commun.*, May 2007, pp. 1604–1612.

[33] M. A. Mollah, X. Yuan, S. Pakin, and M. Lang, "Rapid calculation of max-min fair rates for multi-commodity flows in fat-tree networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 1, pp. 156–168, Jan. 2018.

[34] P. L. Vo, T.-A. Le, and N. H. Tran, "An upward max–min fairness multipath flow control," *Mobile Netw. Appl.*, vol. 25, pp. 1174–1177, Dec. 2019, doi: 10.1007/s11036-019-01436-y.

[35] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data Networks*, vol. 6. Upper Saddle River, NJ, USA: Prentice-Hall, 1992.

[36] E. V. Martins and M. B. Pascoal, "A new implementation of Yen's ranking loopless paths algorithm," *Quart. J. Belg., Fr. Italian Operations Res. Societies*, vol. 1, no. 2, pp. 121–133, Jun. 2003.

[37] D. Michail, J. Kinable, B. Naveh, and J. V. Sichi, "JGraphT—A Java library for graph data structures and algorithms," *ACM Trans. Math. Softw.*, vol. 46, no. 2, pp. 1–29, Jun. 2020.

[38] *MATLAB Version 9.6.0.1472908 (R2019a) Update 9*, Mathworks, Inc., Natick, MA, USA, 2019. [Online]. Available: https://www.mathworks.com/products/MATLAB.html

[39] J. W. Eaton, D. Bateman, S. Hauberg, and R. Wehbring. (2019). *GNU Octave Version 5.2.0 Manual: A High-Level Interactive Language for Numerical Computations*. [Online]. Available: https://www.gnu.org/software/octave/doc/v5.2.0/

[40] F. D. Muñoz, "Optimization of peersim: A discrete event simulator for 5G networks," Bachelor's thesis, Departamento de Ingeniería Telemática y Electrónica, Universidad Politécnica de Madrid, Madrid, Spain, Oct. 2020. [Online]. Available: http://oa.upm.es/68357/

[41] T. M. Peixoto, A. B. Vieira, M. Nogueira, and D. F. Macedo, "Does OpenFlow really decouple the data plane from the control plane?" in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 596–601.

[42] H. S. Bin Obaid and T. B. Trafalis, "An approximation to max min fairness in multi commodity networks," *Comput. Manage. Sci.*, vol. 17, no. 1, pp. 65–77, Jan. 2020.

PATRICIA LUDEÑA-GONZÁLEZ received the B.S. degree in electronics and telecommunications engineering from the Universidad Técnica Particular de Loja (UTPL), in 2005, and the master's degree in telecommunications networks from Rey Juan Carlos University (URJC), Madrid, Spain, in 2011. She is currently pursuing the Ph.D. degree with the Technical University of Madrid (UPM).

She worked as an Assistant Professor at UTPL, from 2005 to 2018. Currently, she is an Associate Professor with the Computer Science and Electronics Department (DCCE), UTPL. Her research interests include computer networks, optimization protocols, traffic engineering, wave propagation in wireless telecommunication networks, and wireless network deployments in rural environments.

JOSÉ LUIS LÓPEZ-PRESA received the B.S. degree in computer science from the University of the Basque Country (UPV/EHU), in 1987, and the Ph.D. degree (Hons.) from Rey Juan Carlos University (URJC), in 2009.

From 1987 to 2012, he was an Assistant Professor at the Technical University of Madrid (UPM), where he has been an Associate Professor, since 2012. His research interests include computer networks, traffic engineering, distributed systems, fault-tolerant systems, parallel and distributed processing, and graph theory with special focus on graph isomorphism.

FERNANDO DÍEZ MUÑOZ received the B.S. degree in telematics engineering and the master's degree in the Internet of Things from the Technical University of Madrid (UPM), in 2020 and 2021, respectively. He worked as a Research Engineer at IMDEA Networks, Leganés, Spain. His research interests include computer networks, traffic engineering, and process mining analysis applied to healthcare systems.