Flexible Traffic Splitting in OpenFlow Networks

Daphne Tuncer, Marinos Charalambides, Stuart Clayman, and George Pavlou

Abstract—**Traffic engineering (TE) functionality aims to control and fine-tune the routing configuration and bandwidth allocation in order to optimize the use of network resources and avoid the build-up of congestion. The performance of a given TE scheme is, however, strongly influenced by the degree of flexibility offered in distributing the traffic load. Multipath routing coupled with arbitrary traffic splitting are two essential ingredients for achieving the desired flexibility. Current proposals for multipath routing in OpenFlow have mostly focused on equal splitting solutions, which impose limitations in terms of the level of control that can be achieved. In this paper, we investigate a new approach, which exploits the properties of bit-masking operations to enable flexible TE in OpenFlow networks. The proposed solution relies on the matching entry feature and the multiple table pipeline capability of OpenFlow, and as such, is in line with the current standard. Based on empirical evaluation, we illustrate the influence of the considered masking parameters and how these can be configured to achieve the desired traffic splitting ratios. The results demonstrate that our solution can achieve a similar level of splitting accuracy as the one obtained with a hash-based approach. However, in contrast to current proposals, it does not require complex extensions to the OpenFlow protocol and can be easily implemented in an OpenFlow environment.**

Index Terms—**Software Defined Networking, OpenFlow, Resource Management**

I. Introduction

Over the last few years, significant efforts have been invested in the development of Software-Defined Networking (SDN) solutions, which are seen as enablers for reducing the management complexity of today's networks. One of the key features of such solutions is the ability to re-program the data plane through a well-defined interface and automatically configure it to meet resource management objectives. In the SDN paradigm, OpenFlow [1] has progressively become the *de facto* standard to realize the southbound interface between the control and data planes. The current version of the OpenFlow protocol, however, has limitations when it comes to implementing the functionality of flexible traffic engineering (TE) applications (e.g. for traffic load-balancing [2] [3], energy management [4], etc.), which require multipath routing and unequal traffic splitting capability.

In an early proposal for multipath forwarding support in OpenFlow [5], the authors discussed the use of the *group option* defined in the OpenFlow protocol to enable equal traffic splitting. For unequal splitting, however, the number of rules required to achieve the desired splitting proportions may become excessive, thus incurring scalability limitations. To overcome this issue, the alternative solution discussed in the proposal concerns the use of hashing functions. Hash-based traffic splitting mechanisms have been widely used in the TE literature (e.g. [2] [6]) as they provide the ability to split the load at a fine level of granularity. The decision on which outgoing interface to forward each incoming packet is based on the result of a hashing function applied to the value of some of the fields extracted from the packet header.

Despite these advantages, we believe that the choice of implementing hash-based schemes as features of OpenFlow switches is not in line with the main principle of the protocol. As shown in previous work [6] [7], using a direct mapping between the hashed values and the outgoing interfaces is in general not sufficient to achieve a good level of splitting accuracy. A table-based hashing approach implementing a two-level mapping should be used instead. In this case, the traffic flows are first split into a set of bins based on the hashed value of flow parameters and the bins are then mapped to the outgoing interfaces based on an allocation table. Such an approach however involves a number of challenges in practice. To control the traffic load distribution, a mechanism is required to configure the allocation table (i.e. mapping of the bins to the outgoing interfaces). Not only does this presuppose the availability of a programming language to define and configure the mapping, it also requires a run-time engine to execute the code, which are not trivial issues. In addition, this may encourage different vendors to implement their own proprietary hash-based schemes (i.e. traffic splitting and load allocation strategies[1]), forcing the protocol to be bound to specific vendor implementations and making it difficult for network operators to have control over the functionality implemented in the physical devices.

In this paper, we investigate an alternative approach which exploits the properties of bit mask operations and matching functions to enable flexible traffic splitting in OpenFlow and, as such, facilitate the implementation of resource management applications in an OpenFlow enabled-network. The proposed solution builds upon the basic primitives of networking and matching features of OpenFlow switches to decide how to partition incoming traffic flows into multiple sets of arbitrary size. More specifically, in our approach, partitioning is realized by logically grouping the flows based on their destination IP address, so that one flow set is defined per outgoing interface. At the switch level, this involves implementing a set of matching entries in a multi-entry table, where each entry corresponds to a specific address pattern. Compared to traditional longest prefix match operations employed in IP routing, a more flexible matching function, which relies on a combination of two masks, is used to perform packet matching. To be supported, our approach requires some small extensions to the current protocol (i.e. ability to express

The authors are with the Department of Electronic and Electrical Engineering, University College London, London WC1E 7JE, UK.

[1] It is worth highlighting that the choice of the hashing function itself is not the main issue here. As long as the functions have high-quality hashing properties [8], they will achieve similar splitting performance.

matching field in a more flexible and generic way and to select matching operations, in particular). However, in contrast to previous proposals, these extensions are in line with the current standard and can easily be implemented. In addition, our approach also follows recent research initiatives which call for programmable and flexible protocol-independent packet processing functionality [10] [11].

The main contributions of this work are as follows. We first elaborate on the types of masking operations and structures required to perform unequal traffic splitting. We then demonstrate the capability of the proposed traffic partitioning mechanism based both on theoretical analysis and empirical evaluation of its performance with respect to the level of control it can achieve over the splitting of incoming flows. In addition, we explain how the proposed scheme can be practically implemented in an OpenFlow switch to achieve the objective of realistic load-balancing applications. The results of the evaluation show that, by configuring the masking parameters, it is possible to control the proportion according to which traffic can be split, and, as such, perform flexible load-balancing. Comparison to a table-based hashing scheme [6], in terms of splitting accuracy performance, further demonstrates that the proposed solution constitutes a promising alternative for traffic splitting in an OpenFlow environment.

The remainder of this paper is organized as follows. Section II provides background information. Section III presents the proposed traffic partitioning approach. Section IV describes how this can be implemented in an OpenFlow switch. The results of the empirical evaluation are presented in Section V and further discussed in Section VI. Section VII presents related work. Finally, conclusions are provided in Section VIII.

II. Background

In this section, we provide background information on multipath TE and on the current proposal for multipath forwarding with OpenFlow. We also review the main principles of masking operations in networking.

A. Multipath Traffic Engineering

Over the years, there have been significant research efforts focusing on the development of TE approaches that exploit the features of multipath routing to adapt the distribution of traffic load in the network according to changing conditions, e.g. [12] [13]. Based on the configuration of n paths, the traffic demand between any source-destination (S-D) pair of nodes is logically partitioned into n independent sets at the source node. Each traffic set is then assigned to one of the n paths and routed accordingly. In the proposed approaches, the volume of traffic on each path is driven by splitting ratios that are precomputed according to the traffic demand in order to optimize the utilization of network resources. Solutions for multipath TE have been proposed both in the context of MPLS networks [14] [13] and ordinary IP environments [2] [12], exploiting in particular the features of Multi-Topology Routing (MTR), e.g. [15] [31] [3].

To avoid issues associated with out-of-order packet delivery, most of the TE approaches proposed in the literature have focused on flow-level traffic splitting, which is also the case in this paper. With this approach, packets that belong to the same TCP flow are always assigned to the same path and no further adjustments are permitted along the route. The implications of flow-level traffic splitting are further discussed in Section VI.

B. Multipath Forwarding with OpenFlow

A proposal for multipath routing, which relies on the group option defined in the OpenFlow protocol, has been presented in [5]. This builds upon the existing features of OpenFlow which provide the ability to force one port to point to a set of other ports. In the case of multipath forwarding, this involves sending incoming packets to one port selected out of a group of available ports. The mechanism used to choose the relevant port depends on the type of selection to apply. The case where each of the ports is selected with equal probability (i.e. equal splitting) can easily be achieved based on a round-robin mechanism. In the case of non-equal cost multipath splitting, however, an approach that allows a finer level of control is required. One of the most practical options discussed in the proposal relies on the use of hashing functions. This implies that a hashing scheme such as, for example, the one proposed by Cao et al. in [6], should be implemented in the switch and parametrized for each traffic flow. Although employing a hashing scheme may have the advantage of providing control at a fine level of granularity, we believe that such a design choice is not in line with the general principles of OpenFlow. First, this assumes the availability of a mechanism to program the hashing scheme at run time (i.e. a programming language), which is not a trivial issue. In addition, the protocol may be bound to specific vendor implementations, as vendors may opt for implementing their own proprietary hashing schemes and their own language. In contrast, the OpenFlow paradigm calls for open control functionality, which can provide flexibility, scalability and adaptability.

C. Masking Operations

In network engineering, masking operations are traditionally used to check if a given vector of bits V exhibits a particular pattern. The input vector V is compared to a second vector M (usually referred to as the mask), which represents the considered pattern, based on bitwise operations. The outcome of the comparison is positive if V follows the pattern represented by M, in which case V is said to *match* M.

Masking operations are used in traditional IP routing, where the destination addresses extracted from incoming packets are compared against prefix entries maintained in the routing tables. Let's consider the IP address 144.82.111.20 of the URL *www.ucl.ac.uk* as an example. In binary form, this is expressed as 10010000-01010010-01101111-00010100. Let's also consider the netmask of length 16, i.e. 11111111-11111111-00000000-00000000. The result of the masking operation between the considered address and netmask gives the prefix 10010000-01010010-00000000-00000000, which represents the subnet 144.82.0.0.

Packet processing operations implemented in OpenFlow switches also rely on masking. In this case, values extracted from the header of incoming packets are compared against matching fields of the entries maintained in the flow tables. In both cases, a match between the input data and the existing entries is defined as a perfect match, i.e. the input sequence strictly follows the considered pattern. While this has the advantage of enabling strict control on the required bit pattern, the rigidity of a perfect match makes it inadequate as a splitting solution for TE purposes. In this paper, we show how alternative, more flexible, matching functions can be used to realize unequal traffic splitting.

III. Traffic Partitioning

As described in Section II-B, the ability to perform unequal splitting relies on the availability of a mechanism which can logically partition the incoming flows into sets of arbitrary size. In this paper, we investigate a splitting approach based on the IP addresses of the traffic flows, which exploits the properties of masking operations. In this section, we demonstrate the principles of the proposed mechanism based on the destination IP address. It is worth noting that our approach can also work with source IP addresses.

A. Definitions

We first define formally some key principles that were used to develop the proposed splitting approach.

1) IP address: Each IP address[2] is represented as a 32-bit vector $X = < x_i >_{i \in [1,32]}$ where each element i indicates the value of the $i-th$ bit in the address[3], i.e.

$$\forall i \in [1,32],\ x_i \in \{0;1\}$$

We denote $\mathcal{X}$ as the set of all IP addresses.

2) Mask: We define the mask of parameter $r > 0$ as the 32-bit vector $M = < m_i >_{i \in [1,32]}$ where r is the number of bits set to 1, while all other bits are equal to 0. We define parameter r as the masking range and we denote $\mathcal{M}_r$ as the set of all masks of masking range r. Formally, this can be formulated as follows:

$$M \in \mathcal{M}_r \ \text{ iff } \ \sum_{i=1}^{32} m_i = r.$$

An example mask with masking range $r = 3$ is 00001011-00000000-00000000-00000000.

3) Matching function: The main idea of the splitting approach proposed in this paper is to group the incoming traffic flows based on the bit pattern exhibited by their destination IP address. This can be achieved by comparing the input addresses to a set of masks M based on a matching function. To enable unequal traffic splitting, however, it is essential to have flexibility in configuring the operations performed by the matching function (i.e. bitwise operations and comparison).

Given a bitwise operator O and a test condition T, we define the matching function $f_{\mathrm{O,T}}$ as the function of two 32-bit vectors V_1 and V_2 which applies the operator O between V_1 and V_2 and compares the outcome based on test condition T. Bitwise operators can be of any type (e.g. AND, OR, XOR, NOT etc. [9]) and typical examples of test conditions include equality/non-equality. The configuration of the matching function can be used to control how strict the match between an input address and a mask should be. To support flexible partitioning, we consider the function $f_{\&,!=0}$ with O being the "AND" operator (denoted &) and T the test condition defined as "not equal to zero" (denoted $! = 0$).

Definition 1. *For all IP addresses X in $\mathcal{X}$, X is said to match mask M in $\mathcal{M}_r$ with respect to matching function $f_{\&,!=0}$ if and only if X & M != 0.*

Based on the vector representation, this can be formalized by the following equation:

$$\sum_{i=1}^{32} x_i \cdot m_i \neq 0. \qquad (1)$$

According to the function $f_{\&,!=0}$, a match between an address and a mask occurs if only a subset of the bits at the position of the r masking bits are equal to 1 in the address. This relaxes the constraint on the bit pattern which needs to be exhibited by the address.

B. Direct Masking Approach

We first consider a masking approach which build upon the mask range to partition incoming traffic flows. To illustrate the property of such an approach, we consider a hypothetical scenario where the range of destination IP addresses of the incoming flows received at a given OpenFlow switch covers the entire address space. Furthermore, we assume that the flow size, as well as the rate and frequency at which traffic flows are received, follow a uniform distribution. Under these assumptions, we can represent each traffic flow by its destination IP address. It is worth noting that in a real scenario, traffic flows are unlikely to follow a uniform distribution (i.e. elephant vs. mice flows [32]). This assumption is used here to simplify the demonstration; the implications of non-uniform traffic distribution are discussed in detail in Section V.

Let $q_M(r)$ be the proportions of IP addresses X in $\mathcal{X}$ so that X matches mask M in $\mathcal{M}_r$ according to Definition 1. Under the assumption of uniform distribution, we can show that the proportion of IP addresses matching an arbitrary mask of masking range r with respect to the function $f_{\&,!=0}$ is independent of the position of the r masking bits in the mask. For all $r \in [1,32]$, we have the following property:

Property 1. $\forall M \in \mathcal{M}_r, \quad q_M(r) = q(r)$

Proof: The property can easily be shown from Eq. 1 which states that a successful match is obtained when the outcome of the masking operation is strictly positive[4]. ■

[2] We focus on IPv4 addresses only in this paper.

[3] Based on mathematical conventions, we index the bit positions from 1 to 32.

[4] Although, strictly speaking, Eq. 1 states that the sum needs to be non-equal to zero, this can easily be translated into "strictly positive" given that both x_i and m_i are always either equal to 0 or 1.

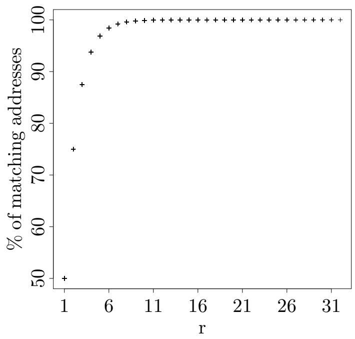


Fig. 1. Proportion of matching addresses according to the masking range r.

Based on Property 1, we can determine the theoretical proportion of matching addresses $q(r)$, and, by extension, matching flows, based on the following equation.

Proposition 1.

$$\forall r \in [1;32], \quad q(r) = 1 - \frac{1}{2^r}. \tag{2}$$

Proof: To get a match, at least one of the bits at the position of the r masking bits in the mask needs to be set to 1 in the input IP addresses. The number of possible combinations of bits satisfying this constraint is $2^r - 1$ (the case with all considered bits set to 0 needs to be disregarded). For each combination, the number of corresponding IP addresses is 2^{32-r}. The total number of matching addresses is therefore $(2^r - 1) \cdot 2^{32-r}$. Since the size of the address space is 2^{32}, the percentage of matching addresses is equal to $\frac{(2^r-1) \cdot 2^{32-r}}{2^{32}}$, which reduces to $q(r)$ after simplification. ■

The evolution of the function $q(r)$ is depicted in Fig. 1 (the values are shown as percentages). The figure shows that only a small set of values ranging from 50% to 100% are available to partition the addresses into different sets. The granularity at which splitting can be realized is therefore very limited. As can be observed, the proportion of matching addresses increases as the value of r increases. The higher the number of bits set to 1 in the mask, the better the chances for a match with an input IP address. In particular, the mask with all bits set to 1 (i.e. $r = 32$) matches all the addresses, while the masks with just one bit set to 1 (i.e. $r = 1$) will match 50% of the addresses (i.e. the considered bit is either set to 0 or 1 in the IP address).

Two main observations can be made from these results. First, they show that by configuring a set of masks against which the destination addresses of the incoming flows are compared, it is possible to partition the flows in different proportions. In addition, the results provide interesting insights regarding the partitioning granularity which can theoretically be achieved. For example, it is not possible to find a combination of masks based on which traffic flows can be partitioned into four strictly independent subsets, each representing 25% of the address space. To overcome these important limitations, in the next subsection we investigate an alternative approach which can enable a finer level of control in terms of partitioning granularity.

C. Prefix Pattern Masking

The approach described in the previous subsection directly compares each mask to all possible IP addresses. Given that the number of matches is independent of the position of the r masking bits, this restricts the number and size of the partitions that can be computed. One way of increasing the range of available partitions is to limit the number of addresses each mask should be compared against. This can be achieved by considering only the addresses that exhibit a pre-determined bit pattern.

More specifically, let's consider the IP address with the bit pattern depicted in Fig. 2.

```
32     28     24     20     16     12      8      4  1
0 000 x xxx x 000 0 000 x xxx x xxxxxxxxxxx
      ↑      ↑          ↑
      V_H    V_L        L
```

Fig. 2. Example bit pattern.

A bit marked as x means that it is either equal to 0 or 1 (free bit), while the bits marked as 0 are strictly equal to 0. In other words, for an address to follow the considered bit pattern, a set of its bits must be equal to 0, while there are no constraints on the others (we disregard the very specific case of address 0.0.0.0). In the example, bits in the intervals $[1,16]$ and $[24,28]$ are free, while bits in the intervals $[17,23]$ and $[29,32]$ are set to 0.

The position of the free bits (and conversely the non-free ones) in the example bit pattern is defined over two non-overlapping continuous intervals. More generally, we define three index parameters L, V_L and V_H representing the boundaries of the bit position intervals, as depicted in Fig. 2. For each L, V_L and V_H, we denote as $C(L, V_L, V_H)$ the bit pattern category so that bits in the interval $[L, V_L - 1]$ and $[V_H + 1, 32]$ are equal to 0, while bits in the intervals $[1, L - 1]$ and $[V_L, V_H]$ are free. An IP address X is said to fall within category $C(L, V_L, V_H)$ if it follows the relevant bit pattern. In practice, the indexes L, V_L and V_H can be used to control the bit pattern of input addresses. The larger L and V_H are, the less constrained the pattern is. In contrast, the smaller L and V_H are, the stricter the pattern is (i.e. larger number of 0 bits). The configuration with $V_H = 32$ and $V_L = L = 1$ represents all possible addresses, while the configuration with $V_H = 32$ and $V_L = L \geq 1$ is equivalent to the case of a prefix of length $32 - L$. Although different approaches could be used to decide on the position of the free/non-free bits (by varying the number of intervals), such a four-part bit pattern offers a good trade-off between flexibility and simplicity in controlling how constrained the pattern is (i.e. through a limited number of index parameters).

In a similar fashion to Section III-B, we can determine, under the assumption of uniform traffic flow distribution, the theoretical proportion of addresses following a specific bit pattern that match any mask of masking range r, so that the r bits are in the interval V_L and V_H, with respect to the function $f_{\&,!=0}$. We denote v as the difference $V_H - V_L$ and l as the difference $32 - L$. The proportions of matching addresses, denoted as $q(l, v, r)$, is given by the following equation.

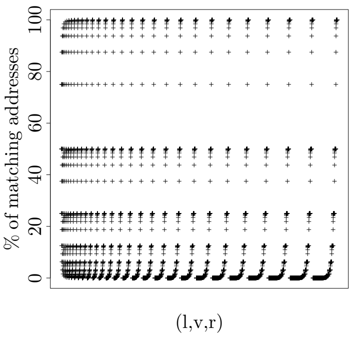


Fig. 3. Distribution of the proportion of matching addresses.

Proposition 2.

$$\forall l \in [1;32],\ \forall v \in [1;l],\ \forall r \in [1;v],$$

$$q(l,v,r) = \frac{(2^r - 1)}{2^{l-v+r}}. \tag{3}$$

Proof: As explained in the proof of Eq. 2, the number of possible combinations which gives a match is $2^r - 1$. In this case, however, the number of addresses associated with each combination is $2^{32-l+v-r}$. As such, the total number of matching addresses in category $C(L, V_L, V_H)$ with $V_H - V_L = v$ is given by $(2^r - 1) \cdot 2^{32-l+v-r}$. The percentage of matching addresses compared to all possibles addresses is therefore $\frac{(2^r-1)\cdot 2^{32-l+v-r}}{2^{32}}$, which reduces to $q(l,v,r)$ after simplification. ■

The evolution of the function $q(l,v,r)$ for all possible values of l, v and r is depicted in Fig. 3. The tuples (l,v,r) on the x-axis are ordered by increasing l, v and r. In contrast to the results depicted in Fig. 1, the number of distinct proportion values for address matching is higher and spans over a larger interval (from 0 to 100%), which shows that partitioning can be achieved at a finer level of granularity. In addition, the plot exhibits a regular pattern, which indicates that identical performance can be achieved with different combinations of parameters.

Fig. 4 shows the influence of each parameter, by varying one parameter at a time. Given the size of the input space, only representative values[5] are depicted. Fig. 4a shows that the proportion of matching addresses is independent of the values of V_L (and by extension of V_H) and as such is not influenced by the position of the r masking bits. In Fig. 4b, where l is varied, the proportion of matching addresses decreases as the value of l increases. As the value of l increases, the number of bits which need to be equal to 0 in the address is larger. The relevant address space is therefore more constrained and less addresses can match. In Fig. 4c, where the varying parameter is v, the proportion of matching addresses increases as v increases. The number of free bits increases as the value of v increases. The address space is less constrained and more addresses can match. Finally, in Fig. 4d, where the varying parameter is r, the proportion of matching addresses increases as r increases. The chances of obtaining a match improve when the number of bits set to 1 are increased (i.e. larger masking range).

From the results, it can be observed that multiple partitions with a wide range of possible sizes can be computed by configuring a set of masks. In contrast to the direct masking approach (Section III-B), however, we show that by considering only those addresses that follow a particular bit pattern, more flexible and finer splitting can be achieved. The results also indicate that the partitioning degree can be controlled through different parameters. In practice, the choice of the parameter(s) to tune can be driven by the characteristics of real traffic flows. We elaborate on the parameter settings in Section V.

IV. Unequal Traffic Splitting

In the previous section on Prefix Pattern Masking, we show that comparing an input mask with masking range r, where the r bits set to 1 are positioned in the interval $[V_L, V_H]$, to only a subset of the addresses that follow a specific bit pattern, can provide more flexibility on how addresses and, by extension, traffic flows, can be partitioned. This section describes how this can be practically implemented.

A. Matching Procedure

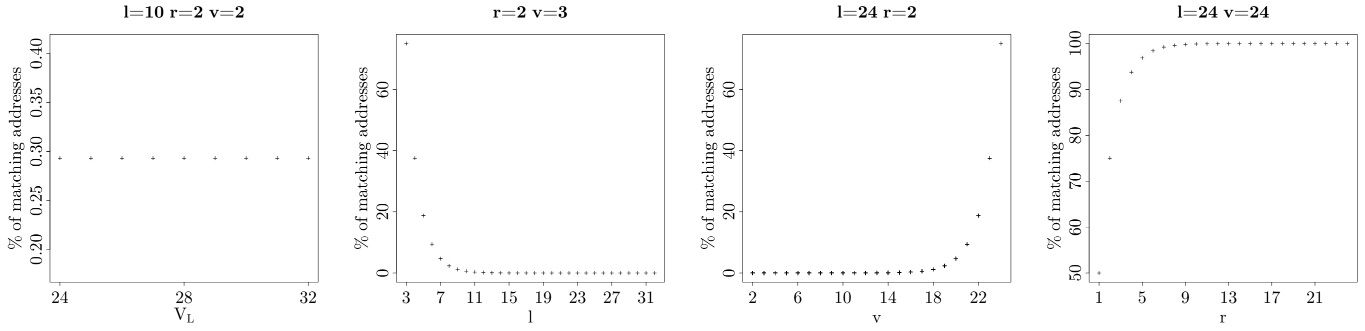
Based on the Prefix Pattern Masking model presented in Section III-C, a successful match occurs if the address follows the required bit pattern *and* if it matches the considered input mask according to Def. 1. From an implementation point of view, the verification of the two conditions can be realized based on two masking operations. Let's consider the bit pattern presented in Fig. 2.

Prefix Pattern Validation The objective of the first operation is to check whether the input address has the required pattern. In this example, it means that bits in the intervals $[1,16]$ and $[24,28]$ are free, while bits in the intervals $[17,23]$ and $[29,32]$ must be equal to 0. This can be achieved by comparing the address to a *prefix pattern* mask M_{P} with respect to the matching function $f_{\&,=0}$ of operator "AND" and test condition "Equal to zero". We refer to this operation as the *prefix pattern validation* operation. The prefix pattern mask for the example bit pattern is defined below:

$$\begin{array}{l} \qquad\quad 28 \qquad 24 \qquad\qquad 17 \\ M_{\mathrm{P}}\textbf{:}\ 1111\ 0\ 000\ 0\ 111111\ 1\ 0000000000000000 \end{array}$$

In this case, the address X is said to match mask M_{P} if and only if X & M_{P} == 0. More generally, we define the *prefix pattern* mask of a given bit pattern as the 32-bit vector so that bits in the intervals $[1, L-1]$ and $[V_L, V_H]$ are set to 0, while bits in the intervals $[L, V_L - 1]$ and $[V_H + 1, 32]$ are set to 1. This is designed to determine whether the address has at least one of its bits set to 1 in the positions forced to 0 in the considered masking pattern, in which case the address does not follow the required pattern. In fact, the proposed prefix pattern masking method can be thought as a generalization of the prefix length matching procedure employed in traditional routing. In contrast to the latter, which has the top N bits set to 1, this provides more flexibility in the choice of the pattern

[5] The same conclusions were drawn with all other values.



(a) Varying V_L.

(b) Varying l.

(c) Varying v.

(d) Varying r.

Fig. 4. Distribution of the proportion of matching addresses based on different parameter combinations.

structure and, as such, it can enable finer granularity traffic splitting.

Range-based Masking The objective of the second masking operation is to check whether the address matches the mask with masking range *r*, where the *r* bits set to 1 are in the interval $[V_L, V_H]$, based on the matching function presented in Def. 1. We refer to this operation as the *testing* operation and call the relevant mask the *testing* mask (noted M_{T}). An example testing mask for the considered example is depicted below:

$$\begin{array}{l} \phantom{M_{\mathbf{T}}\textbf{:}\ 0000\ }\overset{28}{1}\ 0\,1\,\overset{24}{1}\ 0\ 000000\ \overset{17}{0}\ 0000000000000000 \end{array}$$

$$M_{\mathbf{T}}\textbf{:}\ 0000\ 1\ 011\ 0\ 000000\ 0\ 0000000000000000$$

Summary The main steps of the matching procedure can be summarized as follows:

- Define both the prefix pattern mask and the testing mask.
- Compare the input address to the prefix pattern mask with respect to matching function $f_{\&,=0}$ and to the testing mask with respect to matching function $f_{\&,!=0}$.
- The outcome of the matching procedure is considered successful if the input address satisfies both the conditions of the prefix pattern validation and the testing operations.

To illustrate the procedure, we consider a simple example based on the IP address presented in Section II-C, i.e. 144.82.111.20. The objective is to compare the address against the following prefix pattern of configuration $V_H = 32$, $V_L = 29$, $L = 25$ and testing masks with $r = 2$:

$$M_{\mathbf{P}}\textbf{:}\ 00001111000000000000000000000000$$

$$M_{\mathbf{T}}\textbf{:}\ 10010000000000000000000000000000$$

In this case, the outcome of the masking operation between the address and the prefix pattern mask M_{P} is equal to 0 and between the address and the testing mask M_{T} is not equal to 0. The matching procedure is therefore successful.

B. Traffic Engineering-based Multipath Forwarding

In this subsection, we explain how the proposed masking operations and matching procedure can be used to realize multipath unequal traffic splitting and, as such, support the load-balancing objective of a traffic engineering application. As explained in Section II-A, based on the path diversity provided by an underlying routing/forwarding mechanism, the traffic demand between each S-D pair of nodes is logically partitioned between the available paths according to precomputed splitting ratios to optimize the utilization of network resources.

The proposed traffic splitting solution consists of computing a combination of 2-tuple masks (prefix pattern and testing masks) for each S-D pair. Each 2-tuple is associated with one of the available paths and the configuration of the mask parameters in each tuple is driven by the relevant splitting ratio. In an ideal scenario, masks should be configured in such a way to avoid an input address matched by more than one 2-tuple. However, such a condition may severely constrain the problem and may be difficult to implement in practice. To resolve potential multiple match conflicts, we leverage the priority rule of OpenFlow and associate each 2-tuple with a priority. In this case, the matching tuple is defined as the one with the highest priority. In a similar fashion, the lowest priority tuple is used to ensure that there is always a match and is therefore defined as a wildcard on both the prefix pattern and testing masks.

Based on the OpenFlow principles, the prefix pattern mask and the testing mask can be represented, at the switch level, by a multi-entry table, where each entry is associated with a match field, a priority and a set of instructions components [1]. In the proposed approach, the match fields component concerns the 2-tuple masks against which the destination address extracted from the incoming packet is compared. In particular, the matching operation can be logically expressed as follows:

$$@ \,\&\, M_P \;==\; 0 \quad \&\& \quad @ \,\&\, M_T \;!=\; 0 \tag{4}$$

where @ represents a 32-bit address, M_P is the prefix pattern mask, and M_T the testing mask. In addition, & represents the bitwise operator “AND” and && the logical “AND” condition.

The current OpenFlow specification defines only one table type. The latest version identifies 45 match fields and matching operations are considered as a perfect match [1]. To support the proposed approach, extensions to the protocol are required. First, the ability to express match fields in a more generic way should be supported. This would enable the implementation of different types of tables and, as such, extend packet processing options. In addition, the protocol should allow to configure

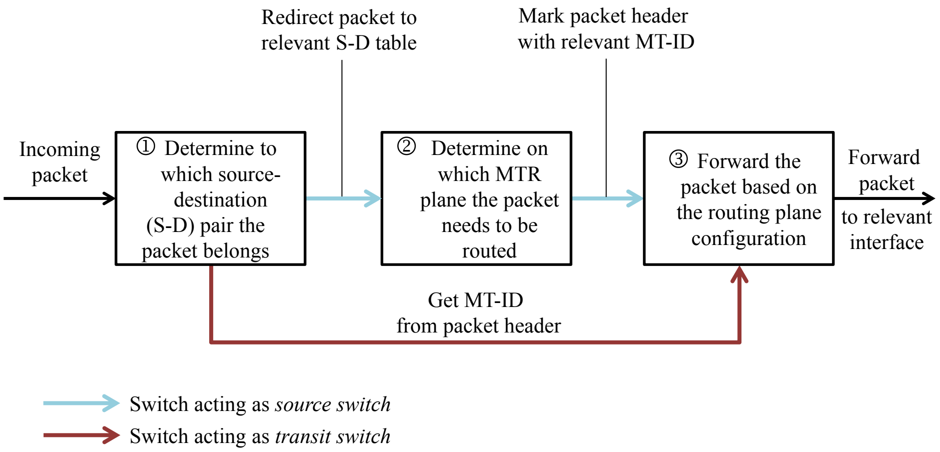

Redirect packet to relevant S-D table
Mark packet header with relevant MT-ID
Incoming packet
① Determine to which source-destination (S-D) pair the packet belongs
② Determine on which MTR plane the packet needs to be routed
③ Forward the packet based on the routing plane configuration
Forward packet to relevant interface
Get MT-ID from packet header
Switch acting as *source switch*
Switch acting as *transit switch*

Fig. 5. Packet processing operations.

matching operations. This would enable the programmability of processing operations and, as such, offer more flexibility in controlling network resources. To be realized, basic bitwise operators and test conditions built into the switch processor can be used.

The packet is then forwarded to the relevant interface based on the selected path.

C. Example Application

To illustrate the sequence of actions which need to be performed at the switch level, we exemplify the packet processing operations based on the MTR-based load-balancing application that we developed in our previous work [3]. Based on path diversity provided by configuring the MTR planes, the objective of the management application is to balance the traffic load in order to minimize the overall network utilization. This is achieved by controlling the volume of traffic between each S-D pair of nodes sent across each plane according to splitting ratios which are periodically adjusted based on network conditions. In [16], we developed a distributed approach to implement the proposed load-balancing application. In this approach, load-balancing decisions are taken by a set of managers which supervise the network edge nodes[6] and control the splitting ratios applied to incoming traffic flows. New configurations (i.e. splitting ratios) are passed to the relevant controllers, which are responsible for planning and applying the required changes in the switches under their control.

The set of operations that need to be executed at the network level in order to achieve multipath routing and unequal traffic splitting depends on the role of the switch. As explained in [16], switches can operate as source or transit based on how they process incoming packets. A switch operates as a *source* for incoming packets if a) it is an edge switch, and b) packets belong to one of the switch's local traffic flows. A switch operates as a *transit* for incoming packets if a) it is an edge switch but incoming packets do not belong to one of the switch's local traffic flows, or b) it is a core switch.

In the case of a source switch, three main actions need to be performed: 1) Determine to which local traffic flow the packet belongs, 2) Determine the MTR plane on which to route the packet, and, 3) Forward the packet according to the configuration of the selected MTR plane. In case of a transit switch, only steps 1 and 3 are executed. In this case, the packet is already marked with the relevant MTR plane identifier (MT-ID). The sequence of actions is depicted in Fig. 5. The main idea behind the proposed packet processing mechanism relies on the multiple table pipeline feature of OpenFlow, which enables multi-processing packet operations. Each incoming packet is sequentially matched against multiple tables in the pipeline so that, at each step, instructions encoded in the matching entry of the current table are applied to the packet. The three main actions involved are detailed below.

Step 1 - Identify the relevant source-destination pair: Given that splitting decisions are taken for each S-D pair, the objective of the first step is to determine to which S-D pair the incoming packet belongs. This can easily be realized based on standard OpenFlow packet processing operations. Source and destination IP addresses extracted from the packet header are matched against the matching fields of the entries of a first flow table, which maintains information for each S-D pair (i.e. source node and destination node IP prefixes). In this case, the actions associated with the matching entry redirect the packet to a second table defined for each S-D pair.

Step 2 - Identify the relevant MTR plane: The objective of this step is to determine on which plane the packet needs to routed. As explained in the previous subsection, this is driven by the splitting ratios whose requirements are translated at the switch level into a set of 2-tuples masks associated with each path.

In this step, the instructions associated with the matching entry consist in marking the packet header with the rele-

[6] We refer to network edge nodes as the set of nodes generating and absorbing traffic, while we call core nodes all the other nodes.

vant MT-ID (each entry represents a MTR plane). Although OpenFlow does not currently support MTR tagging, the latest version of the specifications introduces support for MPLS labeling and VLAN tagging [1]. The VLAN identifier field, in particular, is coded on 12 bits, which represents 4096 options. In practice, a small number of MTR planes (4-5) is sufficient to achieve a traffic engineering objective (e.g. [15] [3]). Two solutions could therefore be followed to support MTR. The protocol could either be extended to incorporate a MTR field, or a small subset of the VLAN identifiers could be reserved to identify the MTR planes and perform MTR tagging.

Step 3 - Forward the packet to the relevant interface: The last step involves forwarding the packet based on the configuration of the selected MTR plane identified by MT-ID.

V. Empirical Evaluation

While the results presented in Section III provide interesting insights regarding the level of control in terms of achievable traffic splitting, their impact is limited as they rely on the assumption of traffic uniformity. In a real environment, the characteristics of the traffic flows are unlikely to follow a uniform distribution. In this section, we empirically evaluate the performance of the proposed splitting approach based on realistic traffic flow settings.

A. Traffic Flow Generator

In the absence of real traffic flow data, we base our evaluation on synthetically generated traces. Although some traffic flow generators have been proposed in the literature, we could not directly rely on those as they do not emulate all the necessary characteristics. We have therefore developed our own traffic flow generator.

The flow information required in this work covers two dimensions: 1) the flow dynamics (i.e. inter-arrival time, size and rate), and, 2) the destination IP address distribution. To take into account both dimensions when generating the traffic flows, we divide the flow modeling process into flow dynamics modeling and IP address distribution modeling.

For the flow dynamics modeling part, we follow the methodology presented in [17]. Flow arrivals are modeled as a Poisson process and the input load is controlled by the inter-arrival rate parameter, which we set to 100. Flow sizes are represented by a truncated Pareto distribution of scaling parameter 1.3 and vary between 8 Mbytes and 8 Gbytes. Finally, flow rates are selected from three possible values 0.5Mbps, 1Mbps and 10Mbps with 30%, 60% and 10% probability, respectively. The parameters of each distribution are selected in accordance with the values reported in [17], in which the authors focused on Internet Service Provider backbone networks.

The objective of the IP modeling part is to decide how to assign a destination IP address to each generated flow. To design the address allocation mechanism, we build upon previous research initiatives (i.e. [18] [19] [20]), which have analyzed the characteristics of the IPv4 prefixes maintained in the routing tables of core networks from different perspectives. In [18], Ganegedara et al. investigated the distribution of the prefix length and showed that, in almost all cases, over 50% of the prefixes have a length equal to 24 (i.e. represent class C networks). Based on these observations, they developed a tool to generate IPv4 prefixes. The tool can be configured to select the percentage of class A, B and C networks to consider (via the prefix length) and control the probability of having a 0 or a 1 at each bit position in the prefix. Unlike [18], the research efforts presented in [19] and [20] have focused on the distribution of prefix popularity (i.e. how likely is a given prefix to be hit in the routing table). The results reported in [19] indicate that, in the considered routing tables, 20% of the prefixes account for 80% of the traffic. The findings were further extended in [20], which show that prefix popularity can be represented by a Zipf distribution. With the objective of being as close to reality as possible, based on the above observations, we designed the IP address allocation mechanism as follows.

- **Pre-processing:** We generate a list of 100,000 prefixes based on the tool developed in [18]. Each prefix is then randomly assigned a rank which represents its popularity.
- **Runtime:** For each newly created traffic flow, we select a prefix from the pre-computed list, based on a Zipf distribution with parameter $\alpha = 1$. We then randomly compute an IP address from the selected prefix and assign the address to that traffic flow.

All the results presented in this section were obtained using these mechanisms, based on a total of 1 million flows.

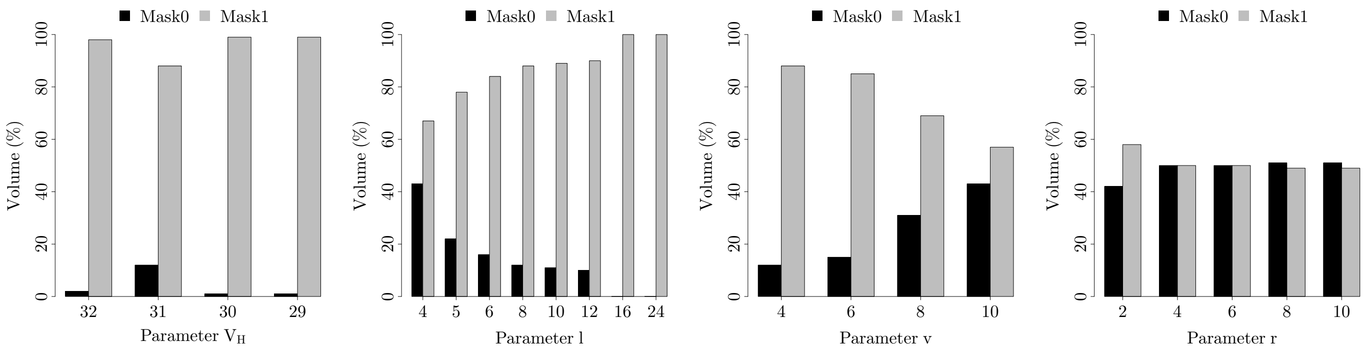
TABLE I
Experiment Settings.

Experiment Index	**l**	**v**	**r**	V_H
Exp1	10	3	2	Varying
Exp2	Varying	3	2	31
Exp3	10	Varying	2	31
Exp4	10	3	Varying	31

B. Influence of Masking Parameters

Based on four experiments, we investigate the influence of the five masking parameters V_H, V_L, l, v, and r on the performance of the traffic splitting approach. In each experiment, we consider a scenario where the traffic flows need to be split into two sets. Each set is represented by a 2-tuple of prefix pattern and testing masks. The first tuple has the highest priority and its masks are configured according to some specific parameter settings. The second tuple has the lowest priority and is defined as a wildcard on both the prefix pattern and testing masks (Section IV-B). In each experiment, we vary the value of one of the parameters of the highest priority 2-tuple. The settings of the parameters for each experiment are reported in Table I.

In all cases, the position of each of the r masking bits is fixed in the interval $[V_H - r + 1; V_H]$. Given the characteristics of the generated traffic flows, we evaluate the performance in terms of the volume of traffic flows assigned to each set. The



(a) Varying V_H (Exp 1).

(b) Varying l (Exp 2).

(c) Varying v (Exp 3).

(d) Varying r (Exp 4).

Fig. 6. Impact of masking parameters on the volume of matching flows.

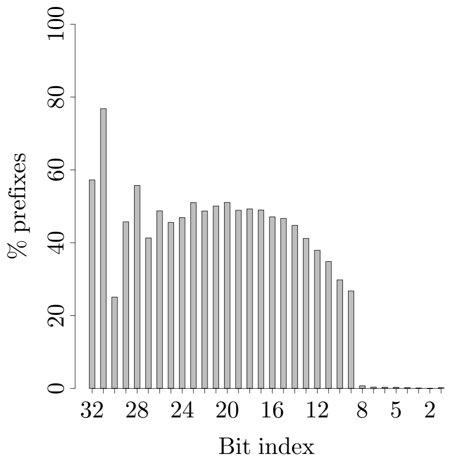


Fig. 7. Percentage of prefixes with a bit set to 1 at each bit position.

results are reported in Fig. 6, where the highest priority tuple is denoted as Mask0 and the lowest priority tuple as Mask1.

Firstly, the results demonstrate that the main conclusion formulated in Section III-C also applies to a realistic scenario: by configuring a set of masks, traffic flows can be split into multiple partitions of arbitrary size. They also highlight some differences. For example, in contrast to the theoretical scenario, it can be observed that the volume of matching flows depends on the value of parameters V_H and V_L. To explain this result, we plot in Fig. 7, for each bit position (index varying from 1 to 32), the percentage of prefixes which have the bit set to 1 at the relevant position. As observed, bits are more likely to be equal to 1 at certain positions. For example, the bit at position 31 is set to 1 in almost 80% of the prefixes, whereas this represents only 25% of the prefixes in the case of the bit at position 30. As explained in Section V-A, the percentage value obtained for each bit position depends on the parameter settings of the IPv4 prefix generator [18]. In our experiments, we set these parameters to represent a realistic distribution of prefixes for core networks. Since parameters V_H and V_L influence the position of the r masking bits, these have an influence on the number of matching flows. Fig. 6b, 6c and 6d corroborate the findings of Section III-C, i.e. the volume of matching flows increase with increasing v and r, and decreasing l.

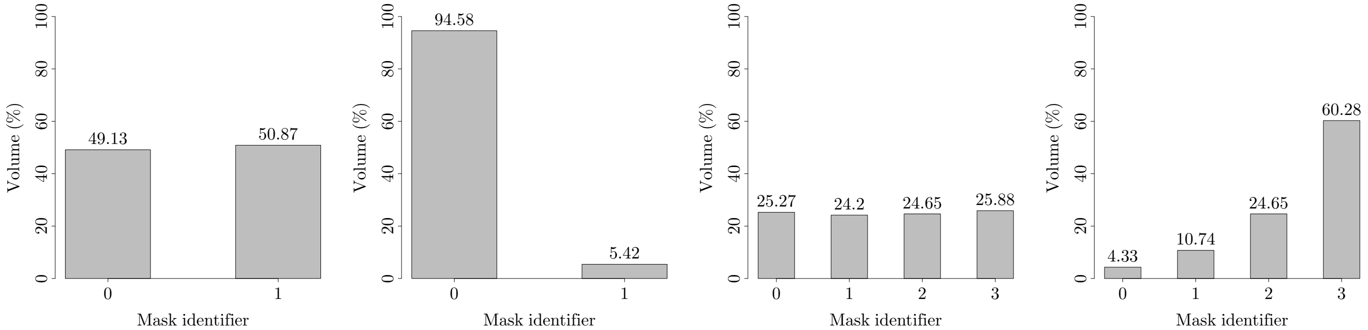
C. Mask Configuration

Based on the results presented in the previous subsection, we now show how it is possible to control the traffic proportion according to which traffic should be split by configuring the prefix pattern and testing masks. To evaluate the splitting accuracy, we consider four scenarios. In the first two scenarios, the traffic is split between two interfaces and, as such, two 2-tuple masks $Mask0$ and $Mask1$ need to be computed. In Scenario 1, the objective is to split the traffic flows into two sets of equal size, i.e. splitting ratios 50%/50%. Scenario 2 represents an unequal split case with splitting ratios 5%/95%. In the two other scenarios, the traffic is split between four interfaces, thus four 2-tuple masks $Mask0$ and $Mask1$, $Mask2$ and $Mask3$ need to be computed. In Scenario 3, traffic is split into four sets of equal size, i.e. splitting ratios 25%/25%/25%/25%, while Scenario 4 illustrates non-equal splitting with ratios 5%/10%/25%/60%. It is worth noting that the scenarios described in this paper are chosen as illustrative examples to present different range of splitting proportions and that similar performance in terms of accuracy was observed with other configurations.

In the following, we first explain how the masks are configured for each scenario and then discuss the results.

1) Scenario 1: 50%/50%: To ensure that a match can always be obtained, $Mask1$ is configured as the wildcard on both the prefix pattern and testing masks (it matches 100% of the flows). To obtain a 50/50 distribution, $Mask0$ should therefore be configured to match 50% of the traffic. Based on Fig. 1, we know that a 50% match can be achieved with a masking range equal to one. $Mask0$ is then set up with $l = 31$, $v = 31$, $r = 1$, $V_H = 32$, and $V_L = 0$. In a non-uniform distribution scenario, the volume of matching flows depends on the position of the r masking bits. To maximize the chances of a match, it is therefore better to place the masking bit at a high bit position - we choose to set it at position 32.

2) Scenario 2: 5%/95%: As in Scenario 1, $Mask1$ is configured as the wildcard to match 100% of the traffic. For $Mask0$, two approaches can be followed: to find either a configuration that can provide a 5% match or one that can cover 95% of the traffic. We focus on the second option, given that, for uniform distribution, a 93.5% match can be obtained with $l = 31$, $v = 31$, $r = 4$ (see Eq. 3). Although this represents a deviation of 1.5% to the desired 95% split, the



(a) Scenario 1.

(b) Scenario 2.

(c) Scenario 3.

(d) Scenario 4.

Fig. 8. Split proportion in terms of volume of matching addresses.

configuration can be regarded as a good candidate given the non-uniform nature of the address distribution. In a similar fashion to Scenario 1, we set the masking bits at the highest positions (i.e. 29 - 30 - 31 - 32) to maximize the chances of a match.

3) Scenario 3: 25%/25%/25%/25%: In this scenario, *Mask*3 is configured as the wildcard and matches 100% of the traffic. To achieve the desired proportions, the combination of *Mask*0, *Mask*1 and *Mask*2 should therefore account for 75% of the traffic. We start by configuring tuple *Mask*0. From Fig. 6b, we can observe that the configuration of Experiment 2 with $l = 5$ is a good candidate to achieve a 25% match. To further improve the performance and obtain a ratio closer to 25%, we can refine the value of parameter v or r (see Section V-B). Fig. 6d shows that the volume of matching flows increases as r increases. We therefore increment the value of r compared to Experiment 2 and set it to 3. For *Mask*1, we start with the configuration of Experiment 2 with $l = 4$. From Fig. 6b, we can observe that this can match around 40% of the traffic. Since the position of the masking bit is maintained constant in Experiment 2, part of the flows matching a tuple configured with $l = 5$ also matches a tuple with $l = 4$. As such, this would result in an actual proportion of around 15% (i.e. $40\% - 25\%$). To increase the range of masking addresses, we therefore increment the values of v and r by 1, which gives a configuration $l = 4$, $v = 4$, $r = 4$ for *Mask*1. Finally, for *Mask*2, we select a configuration which limits the number of possible matches (through the value of r) but does not constrain the prefix pattern (through the value of l and v), i.e. $l = 31$, $v = 31$ and $r = 1$. To match a different set of addresses compared to *Mask*0 and *Mask*1, the masking bit of *Mask*2 is positioned at bit 32 so that it does not overlap with the ones used in the two masks with higher priority.

4) Scenario 4: 5%/10%/25%/60%: In this scenario, we jointly configure the parameters of *Mask*0 and *Mask*1 by targeting a 15% match (i.e. 5% + 10%). Based on Fig. 6b, it can be observed that this can be achieved using a configuration with $l = 8$, $v = 3$ and $r = 2$. To further partition the set of matching addresses in 5%/10% proportions, we adjust the value of the parameters l, v and r, as well as the position of the masking bits. We use the configuration $l = 9$, $v = 5$ and $r = 4$ with the masking bits on the positions 26 - 27 - 28 - 30 for *Mask*0 and the configuration $l = 9$, $v = 5$ and $r = 2$ with the masking bits at positions 29 - 31. For *Mask*2, we use the same setting as in Scenario 3, i.e. $l = 31$, $v = 31$ and $r = 1$ with r positioned at bit 32. Finally, in a similar fashion to all previous cases, *Mask*3 is configured as the wildcard.

5) Splitting Accuracy: The proportions in terms of volume of matching addresses obtained in each scenario are shown in Fig. 8. As can be observed, in all cases, the achieved split is very close to the desired one. All the proportions are within less than 1% to the expected ratios, with a maximum deviation of 0.87% (for Scenario 1). This demonstrates the potential of the proposed approach: it can be used to split a realistic traffic set accurately at a fine level of granularity.

D. Comparison to a Hash-based Solution

We now compare the performance of the proposed approach in terms of splitting accuracy to the one obtained using a table-based hashing scheme [6], based on the four scenarios described in the previous subsection.

Table-based hashing approaches were shown to offer very good performance in terms of traffic splitting accuracy, e.g. [6] [7]. In these schemes, traffic flows are first split into M bins based on the hashed value of flow parameters (e.g. extracted from the IP header). The bins are then mapped to the N outgoing interfaces based on an allocation table. To achieve a fine level of splitting granularity, the number of bins M is usually set one or two orders of magnitude larger than the number of interfaces N [6]. In this work, we set the value of M equal to $500 \cdot N$ (following the settings used in [7]) and apply the 32-bit Cyclic Redundancy Checksum (CRC32) algorithm to get the hashed value of the flow characteristics. CRC32 has been reported as one of the most widely deployed hash functions in hardware equipment [21]. To compare the performance to the proposed mask-based approach, the hashing function is applied to destination IP address only. The proportions obtained for the four scenarios are shown in Table II, which also reports the maximum and the average deviation from the expected split.

It can first be observed that the hash-based approach achieves very good performance in terms of splitting accuracy, with the largest maximum deviation being 0.85% (Scenario 4). More importantly, the results show that the proposed approach can achieve a similar performance to that of a hash-based scheme, which provides additional evidence to support the viability of our solution.

TABLE II
PERFORMANCE COMPARISON.

Scenario 1	**Split (%)**				**Maximum Deviation (%)**	**Average Deviation (%)**
Hash-based splitting	50.40	49.60			0.4	0.4
Proposed approach	50.87	49.13			0.87	0.87
Scenario 2	**Split (%)**				**Maximum Deviation (%)**	**Average Deviation (%)**
Hash-based splitting	4.67	95.33			0.33	0.33
Proposed approach	5.42	94.58			0.42	0.42
Scenario 3	**Split (%)**				**Maximum Deviation (%)**	**Average Deviation (%)**
Hash-based splitting	25.05	24.39	24.76	25.80	0.8	0.425
Proposed approach	25.27	24.20	24.65	25.88	0.88	0.575
Scenario 4	**Split (%)**				**Maximum Deviation (%)**	**Average Deviation (%)**
Hash-based splitting	4.67	10.60	24.15	60.58	0.85	0.59
Proposed approach	4.33	10.74	24.65	60.28	0.74	0.51

In fact, while partitioning is the common objective of both schemes, this is realized in two different ways. In the hash-based approach, the hashed values of destination IP addresses are uniformly distributed into equal size bins defined over the whole hash space, and the desired split is realized by grouping the bins based on the desired ratios. In contrast, partitioning in the proposed approach is achieved by grouping addresses with a similar pattern. To enable fine control over the size of each group, a flexible masking approach is however required, for instance compared to a prefix-based perfect match strategy, which simply divides the address in two parts.

Finally, it is worth highlighting that since the splitting accuracy of any traffic partitioning scheme depends on the characteristics of a specific traffic set, those characteristics can influence the results.

VI. DISCUSSION

By nature, the proposed traffic splitting approach depends on the distribution of the IP addresses. In particular, the results presented in Section V demonstrate that, although the insights gained from the theoretical analysis can be useful to decide how the masks should be configured to achieve some expected performance, these are not sufficient. They should be augmented with information regarding the characteristics of the incoming traffic flows, which could be acquired from different sources. For example, the distribution of the observed IP addresses may depend on the type of network. The range of addresses monitored at an edge network is likely to significantly differ from the one observed in a core network. As such, useful information can be provided regarding the type of prefix patterns to expect. In addition, prediction strategies, coupled with lightweight learning techniques, could be used to infer the dynamics and characteristics of the observed IP addresses. Previous research efforts (e.g. [19] [20] [22]) have shown that relevant properties, such as the variability of heavy hitters over time or the burstiness in the occurrence of prefixes, could be extracted from the traffic flows and have proposed mechanisms that take advantage of these properties.

The performance of our approach is also influenced by the dynamics of the traffic flows themselves (i.e. size and rate), which is a general issue with any flow-level splitting approach and is independent of the partitioning scheme [23]. Since the objective is to balance the traffic load over multiple paths, the traffic dynamics can have an impact on the volume of traffic sent over each path and, as such, on the level of load-balancing that can be achieved. As reported by Rost et al. in [24], the skewness in the distribution of flow rates, the grouping flexibility of the mechanisms used to partition the flows and the burstiness of traffic are key aspects that can affect the accuracy of splitting. To address these issues, previous work (e.g. in [7]) has proposed the use of dynamic splitting algorithms that can react to load imbalance by modifying the algorithm parameters at runtime. Such an approach could apply to our proposal as well. In this case, a two-level traffic management process could be implemented in which a pro-active mechanism would provide the initial mask configurations based on historical flow information, while a reactive scheme would adapt to the traffic dynamics by fine tuning the parameters l, v and/or r.

Most of the TE approaches proposed in the literature have focused on the use of hashing schemes to enable traffic splitting. However, as shown in [23], these schemes themselves can have their performance significantly affected in terms of splitting accuracy (e.g. 20% splitting error reported in [23]). In general we believe that, in the context of SDN, the use of such schemes - despite their flexibility - poses a number of difficulties given that different implementations of the hash-based splitting method are possible, which can also be closed to vendors. Mechanisms (and associated language) to program the partitioning scheme are currently not supported by OpenFlow and their complexity is unknown. In contrast, the approach proposed in this paper builds upon the basic features and primitives of networking and, as such, requires simpler extensions to the current protocol. Also, our solution is in line with recent research initiatives which call for programmable and flexible packet processing functionality [10] [11]. As a result, although our approach is more dependent on the characteristics of the input traffic compared to the hashing method, we believe that, due to the inherent dynamic nature of the traffic flows, the trade-off in terms of flexibility and simplicity of implementation offered by the proposed solution makes it a promising alternative.

Finally, while the proposed solution focuses on applying the masks to the IP address, it could theoretically be extended to any other match field that supports masking (OpenFlow currently supports arbitrary bitmasking on IP and Ethernet addresses [1]). A combination of fields could also be considered as long as this does not increase the complexity of the approach.

VII. Related work

Functionality to support flexible allocation of traffic among multiple alternative paths is currently missing from the OpenFlow specification. Proposals have mainly focused on equal-cost splitting solutions based on OpenFlow multipath forwarding groups [5]. Although these are fairly easy to implement, they have a limited level of control over the distribution of traffic and may not allow TE objectives to be effectively achieved. The same limitation is shared by direct hashing approaches, e.g. the Internet Checksum algorithm [25], which can only split traffic into equal amounts. Table-based hashing schemes on the other hand can split the load at a fine granularity, which is defined by the ratio of bins to outgoing interfaces [6], and have been widely used. However, as discussed in Section VI, we believe that adding hash-based schemes as a feature of OpenFlow may not be the best design choice.

In [26], Wang et al. have proposed an approach to enable unequal traffic splitting in an OpenFlow environment by taking advantage of the wildcard rule capability offered by the protocol. In that work, the authors focus on uniform traffic distribution and show that, for non-uniform cases, their approach can lead to severe load imbalance (25% deviation is reported). This is in contrast to our solution which can achieve accurate split even in the case of non-uniform distribution. To improve their solution, Wang et al. suggest to refine the wildcard rule at a finer level. However, as mentioned by the authors themselves, this will result in an increase in the number of rules required in each switch, which can lead to limitations in terms of scalability. To overcome these limitations, Kang et al. have recently presented a new version of the approach that leverages the metadata tags available from the switch hardware to group traffic types associated with similar splitting ratios [33]. Compared to our solution, however, it may provide less flexibility since the control parameters used can result to insufficient partitioning granularity, especially in the presence of heterogeneous addresses. In our approach, the accuracy is not a factor of the number of mask tuples, which is only driven by the number of paths between each S-D pair of nodes. As shown in previous work (e.g. [12] [15] [3]), a small number of paths is enough in practice to achieve near-optimal load-balancing performance (typically between 3 and 5).

Although TE at the packet level [27] can achieve fine granularity splitting, packet re-ordering issues can falsely signal network congestion. For this reason, most approaches focused on flow-based splitting techniques (including the one proposed in this paper) [2] [3] [13] [4]. However, due to varying flow sizes and rates the load may not be balanced well over the available paths, given that a flow persists on the originally assigned path during its lifetime. FLARE [23] overcomes this problem by operating on bursts of packets within a flow, which are carefully chosen to avoid reordering. We plan to consider flowlet level splitting in future extensions of this research.

Traffic engineering in the context of software-defined networks was investigated by [28] [29] and [30]. The authors of [28] focused on ordinary IP networks with partial deployment of SDN capability and have not considered how unequal traffic splitting can be realized (ECMP is assumed). The same shortcoming is shared by [29] in which the authors proposed an SDN-based implementation of MPLS functionality where the load can be shared between multiple LSPs. The TE approach in [30] employs a hash-based approach to achieve unequal traffic splitting and has the implementation shortcomings mentioned above. A survey of TE initiatives in SDN has been recently presented in [34].

In traditional routing, the table matching entry is the one with longest destination prefix match. In [24] the authors use prefix matching for the purpose of balancing traffic, which has two main drawbacks: (1) a perfect match is required, which makes the approach rather rigid with respect to the granularity level that can be achieved, and (2) it may not scale due to the very long tables involved in the process. In contrast, our solution is more flexible and can achieve finer traffic splitting granularity. In addition, its complexity is dominated by the number of S-D pairs and path options to consider, which, in practice, represent a lower order of magnitude compared to individual prefixes.

VIII. Conclusions

The SDN paradigm has emerged as a promising solution for reducing the management complexity of network infrastructures through the creation of a unified control plane independent of specific vendor equipment. The current version of OpenFlow, the prevalent southbound interface, can only support equal splitting of traffic flows between alternative paths, which may not allow TE objectives to be successfully met. In this paper we propose a new approach based on bit-masking operations that can enable a fine level of control over the splitting of incoming flows. We describe how unequal distribution of traffic can be achieved and demonstrate through experimentation the capabilities of the approach as well as the parameters that influence its performance. To further demonstrate the potential of our solution, we compare its performance in terms of splitting accuracy to the one obtained using a hash-based scheme and show that similar level of accuracy can be achieved. More generally, we argue that our approach is a better match for OpenFlow oriented traffic management compared to hash-based schemes, and also has a lower implementation complexity.

Future extensions of this research will focus on the implementation of the proposed solution in a test-bed environment. In particular, this will involve the development of a management application to compute the required masks, integrating both pro-active and reactive processes. Our objective is to design an algorithm in such a way that the mask-compute time will be negligible compared to the frequency of reconfigurations. In addition, we will perform an extensive evaluation of the overall approach based on a wide range of traffic conditions and network configurations.

Acknowledgment

This research was funded by the EPSRC KCN project (EP/L026120/1) and by the Flamingo Network of Excellence project (318488) of the EU Seventh Framework Programme.

References

[1] “OpenFlow Specifications v.1.5.0," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf. [Online; accessed 26-October-2015].

[2] S. Fischer, N. Kammenhuber, and A. Feldmann, “REPLEX: dynamic traffic engineering based on wardrop routing policies," in *Proc. of the ACM CoNEXT’06*, Lisboa, Portugal, 2006, pp. 1-12.

[3] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, “DACoRM: A coordinated, decentralized and adaptive network resource management scheme," in *Proc. of NOMS’12*, Maui, Hawaii, Apr. 2012, pp. 417-425.

[4] M. Charalambides, D. Tuncer, L. Mamatas, and G. Pavlou, “Energy-aware adaptive network resource management," in *Proc. of IM’13*, Ghent, Belgium, May 2013, pp. 369-377.

[5] “OpenFlow Multipath Proposal," http://archive.openflow.org/wk/index.php/Multipath_Proposal. [Online; accessed 26-October-2015].

[6] Z. Cao, Z. Wang, and E. Zegura, “Performance of hashing-based schemes for Internet load balancing," in *Proc. of INFOCOM’00*, vol. 1, Tel Aviv, Israel, 2000, pp. 332-341 vol.1.

[7] R. Martin, M. Menth, M. Hemmkeppler, “Accuracy and Dynamics of Hash-Based Load Balancing Algorithms for Multipath Internet Routing," *Proc. of BROADNETS’06*, San Jose, California, USA, 2006, pp. 1-5.

[8] https://github.com/Cyan4973/xxHash. [Online; accessed 02-February-2016].

[9] D.Kernighan, and D. M. Ritchie, “The C Programming Language (2nd ed.)," Englewood Cliffs, NJ: Prentice Hall, March 1988, *ISBN 0-13-110362-8*.

[10] P. Bosshart et al, “P4: programming protocol-independent packet processors," in *SIGCOMM Comput. Commun.*, rev. 44, no.3, pp. 87-95, July 2014.

[11] P. Bosshart et al, “Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN," in *Proc. of ACM SIGCOMM’13*, New York, NY, USA, pp. 99-110.

[12] J. Wang, Y. Yang, L. Xiao, and K. Nahrstedt, “Edge-based traffic engineering for OSPF networks," *Computer Networks*, vol. 48, issue 4, pp. 605-625, July 2005.

[13] S. Kandula, D. Katabi, B. Davie, and A. Charny, “Walking the tightrope: responsive yet stable traffic engineering," in *Proc. of SIGCOMM’05*, vol. 35, Philadelphia, PA, USA, August 2005, pp. 253-264.

[14] A. Elwalid, C. Jin, S. Low and I. Widjaja, “MATE: MPLS adaptive traffic engineering," in *Proc. of IEEE INFOCOM’01*, Anchorage, Alaska, USA, 2001, vol. 3, pp. 1300-1309.

[15] N. Wang, K. Ho, and G. Pavlou, “Adaptive multi-topology IGP based traffic engineering with near-optimal network performance," in *Proc. of NETWORKING’08*, vol. 4982. Springer Berlin/Heidelberg, 2008, pp. 654-666.

[16] D. Tuncer, M. Charalambides, S. Clayman, G. Pavlou, “Adaptive Resource Management and Control in Software Defined Networks," *IEEE Transactions on Network and Service Management (TNSM), special issue on Efficient Management of SDN and NFV-based Systems*, vol.12, no.1, pp.18-33, March 2015.

[17] A. Kvalbein, C. Dovrolis, and C. Muthu, “Multipath load-adaptive routing: putting the emphasis on robustness and simplicity," in *Proc. of ICNP’09*, Princeton, New Jersey, USA, October 2006, pp. 203-212.

[18] T. Ganegedara, J. Weirong, and V. Prasanna, “FRuG: A benchmark for packet forwarding in future networks," *Proc. IEEE IPCCC’10*, Albuquerque, NM, USA, Dec. 2010, pp. 231-238.

[19] K. Gadkari, D. Massey, and C. Papadopoulos, “Dynamics of Prefix Usage at an Edge Router," *Proc. of PAM’11*, Atlanta, GA, USA, 2011, pp. 11-20.

[20] N. Sarrar, S. Uhlig, A. Feldmann, R. Sherwood, and X. Huang, “Leveraging Zipf’s Law for Traffic Offloading," *SIGCOMM Comput. Commun. Rev.*, January 2012, vol. 42, pp. 16-22.

[21] Nan Hua, Eric Norige, Sailesh Kumar, and Bill Lynch, “Non-crypto Hardware Hash Functions for High Performance Networking ASICs," *Proc. of ANCS’11*, Brooklyn, NY, USA, 2011, pp. 156-166.

[22] K. Papagiannaki, N. Taft, C. Diot, “Impact of flow dynamics on traffic engineering design principles," *Proc. of INFOCOM’04*, Hong-Kong, 2004, vol.4, pp.2295-2306.

[23] S. Kandula, D. Katabi, S. Sinha, and A. Berger, “Dynamic Load Balancing Without Packet Reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, pp. 51-62, April 2007.

[24] S. Rost and H. Balakrishnan, “Rate-aware splitting of aggregate traffic," *Technical report, MIT*, 2003.

[25] USC/ISI, “Internet Protocol - Specification," RFC 791, Internet Engineering Task Force, Sept. 1981. [Online; accessed 26-October-2015]. Available: https://tools.ietf.org/html/rfc791

[26] R. Wang, D. Butnariu, and J. Rexford, “OpenFlow-based server load balancing gone wild", *Proc. of Hot-ICE’11*, Boston, MA, USA, 2011.

[27] M. Shreedhar and G. Varghese, “Efficient fair queuing using deficit round robin", *Proc. of SIGCOMM’95*, 1995.

[28] S. Agarwal, M. Kodialam, and T. Lakshman, “Traffic engineering in software defined networks," in *Proc. of INFOCOM’13*, Turin, Italy, April 2013, pp. 2211-2219.

[29] S. Das, “PAC.C: A Unified Control Architecture for packet and circuit network convergence," PhD Thesis, Stanford University, June 2012.

[30] S. Jain et al, “B4: Experience with a Globally Deployed Software Defined Wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3-14, Aug. 2013.

[31] D. Tuncer, M. Charalambides, G. Pavlou, and N. Wang, “Towards Decentralized and Adaptive Network Resource Management," in *Proc. of the 7th IEEE/IFIP International mini-Conference on Network and Service Management (mini-CNSM’11), Paris, France, October 2011.*

[32] R. S. Prasad, and C. Dovrolis “Beyond the Model of Persistent TCP Flows: Open-Loop vs Closed-Loop Arrivals of Non-persistent Flows," in *Proc. of the 1st Annual Simulation Symposium (ANSS-41 2008)*, April 2008, pp. 121-130.

[33] N. Kang, et al, “Efficient Traffic Splitting on Commodity Switches," in *Proc. of CoNEXT’15*, Heidelberg, Germany, December, 2015.

[34] I. F. Akyildiz, et al, “A roadmap for traffic engineering in SDN-OpenFlow networks," *Comput. Netw.*, 71 (October 2014), pp. 1-30.

Daphne Tuncer is a postdoctoral researcher in the Department of Electronic and Electrical Engineering at University College London, UK. She received her Ph.D. from the same department in November 2013. Before joining UCL, she studied in France, where she obtained a "Diplôme d’ingénieur de Télécom SudParis" in 2009. Her research interests are in the areas of software-defined networking, network self-management, adaptive network resource management, energy efficiency and cache/content management.

Marinos Charalambides is a senior researcher at University College London. He received a BEng (First Class Hons.) in Electronic and Electrical Engineering, a MSc (Distinction) in Communications Networks and Software, and a Ph.D. in Policy-based Management, all from the University of Surrey, UK, in 2001, 2002 and 2009, respectively. He has been working in a number of European and UK national projects since 2005 and his current research interests include software-defined networking, in-network caching, energy-aware networking and on-line traffic engineering. He is on the technical program committees of the main network and service management conferences.

Stuart Clayman received his PhD in Computer Science from University College London in 1994. He has worked as a Research Lecturer at Kingston University and at UCL. He is currently a Senior Research Fellow at UCL EEE department. He co-authored over 30 conference and journal papers. His research interests and expertise lie in the areas of software engineering and programming paradigms; distributed systems; virtualised compute and network systems, network and systems management; networked media; and knowledge-based systems. He has been involved in several European research projects since 1994. He also has extensive experience in the commercial arena undertaking architecture and development for software engineering, distributed systems and networking systems. He has run his own technology start-up in the area of NoSQL databases, sensors and digital media.

George Pavlou (g.pavlou@ucl.ac.uk) is Professor of Communication Networks in the Department of Electronic and Electrical Engineering, University College London, UK, where he coordinates research activities in networking and network management. He received a Diploma in Engineering from the National Technical University of Athens, Greece and MSc and PhD degrees in Computer Science from University College London, UK. His research interests focus on networking and network management, including aspects such as traffic engineering, quality of service management, autonomic networking, information-centric networking, grid networking and software-defined networks. He has been instrumental in a number of European and UK research projects that produced significant results with real-world uptake and has contributed to standardization activities in ISO, ITU-T and IETF. He has been on the editorial board of a number of key journals in these areas, he is the chief editor of the bi-annual IEEE Communications network and service management series and in 2011 he received the Daniel Stokesbury award for "distinguished technical contribution to the growth of the network management field".