Revisiting Flow-Based Load Balancing: Stateless Path Selection in Data Center Networks

Gregory Detal[a], Christoph Paasch[a], Simon van der Linden[a], Pascal Mérindol[b], Gildas Avoine[a], Olivier Bonaventure[a]

[a] *ICTEAM, Université catholique de Louvain, Louvain-la-Neuve, Belgium*
[b] *LSIIT, Université de Strasbourg, Strasbourg, France*

Abstract

Hash-based load-balancing techniques are widely used to distribute the load over multiple forwarding paths and preserve the packet sequence of transport-level flows. Forcing a long-lived, i.e., elephant, flow to follow a specific path in the network is a desired mechanism in data center networks to avoid crossing hot spots. This limits the formation of bottlenecks and so improves the network use. Unfortunately, current per-flow load-balancing methods do not allow sources to deterministically force a specific path for a flow.

In this paper, we propose a deterministic approach enabling end hosts to steer their flows over any desired load-balanced path without relying on any packet header extension. By using an invertible mechanism instead of solely relying on a hash function in routers, our method allows to easily select the packet's header field values in order to force the selection of a given load-balanced path without storing any state in routers.

We perform various simulations and experiments to evaluate the performance and prove the feasibility of our method using a Linux kernel implementation. Furthermore, we demonstrate with simulations and lab experiments how MultiPath TCP can benefit from the combination of our solution with a flow scheduling system that efficiently distributes elephant flows in large data center networks.

Keywords: Load Balancing; Data Center Networks; Multipath TCP

1. Introduction

Load balancing plays a key role in enterprise, data center and ISP networks. It improves the performance and the scalability of the Internet by distributing the load evenly across network links, servers, or other resources. Load balancing allows to maximize the throughput [24], achieve redundant connectivity [25] or reduce congestion [10]. Different forms of load balancing are deployed at various layers of the protocol stack. At the datalink layer, frames can be distributed over parallel links between two devices [4]. At the application layer, requests can be spread on a pool of servers.

Email addresses: gregory.detal@uclouvain.be (Gregory Detal), christoph.paasch@uclouvain.be (Christoph Paasch), simon.vanderlinden@uclouvain.be (Simon van der Linden), merindol@unistra.fr (Pascal Mérindol), gildas.avoine@uclouvain.be (Gildas Avoine), olivier.bonaventure@uclouvain.be (Olivier Bonaventure)

At the network layer, the most common technique, *Equal-Cost Multi-Path* (ECMP) [24, 13], allows routers to forward packets over multiple equally-good paths. ECMP may both increase the network capacity and improve the reaction of the control plane to failures [25]. Current ECMP-enabled routers proportionally balance flows across a set of equal-cost next hops on the path to their destinations. The most deployed next-hop selection method is solely based upon a hash computed over several fields of the regular packet headers [1, 24]. Using a hash function ensures a somewhat fair distribution of the next-hop selection [10] while preserving the packet sequence of transport-level flows.

Data centers have been designed to run various types of applications [7]. During the last years, many of the applications that are used in large commercial data centers have adopted a data flow computation model. MapReduce [17], Dryad [26] and CIEL [38] among others implement such a computation model. With this kind of applications, large amount of data need to be exchanged between the nodes that implement the different processing stages. This is confirmed by several studies that have analyzed data center traffic. For example, [20] reports that while 99% of the flows carry fewer than 100 MBytes, more than 90% of all the bytes are transported in flows between 100 MBytes and 1 GBytes. Network operators therefore aim to maximize the network utilization in order to improve the performance of these applications [12, 3]. To achieve this goal, data center designs rely heavily on ECMP [2, 20, 21, 37] to spread the load among multiple paths and reduce congestion. This form of load balancing is blind and naive, congestion can still occur inside the data center and lead to reduced performance. Data center traffic contains both mice and elephant flows [7, 28]. Mice flows are short and numerous but generally they do not cause congestion. Most of the data is carried by a low fraction elephant flows. Based on this observation, several authors have proposed traffic engineering techniques that allow to route elephant flows on non-congested paths (see [8, 14, 3] among others). Those techniques often rely on OpenFlow switches [34] to control the server-server paths. Unfortunately, the scalability of such approaches is limited, which may lead to an overload of the flow tables on the OpenFlow switches [14, 15].

In this paper, we show that another design is possible to exploit the path diversity that exists in data center networks without maintaining any state within the network. Current hash-based implementations rely on the IP and TCP headers to select the load-balanced path over which each flow is forwarded. It is therefore very difficult for a host to predict the path that a flow will follow. We show in this paper that hash functions are not the only way to practically enable path diversity. We propose a new deterministic scheme called *Controllable per-Flow Load-Balancing* (CFLB) that allows hosts to explicitly select the load-balanced path they want to use for a specific flow.

CFLB is designed such that it meets the following requirements: all packets of the same flow follow the same path; no overhead; transparent to non-CFLB aware sources; operate at line rate. Thereby, CFLB allows sources to encode inside existing fields of the packet header the load-balanced path that each packet should follow. An invertible function is used to encode and decode the path present in the packets' fields which can be implemented efficiently to enable line rate forwarding. CFLB is transparent for applications that do not want to steer packets and does not require any changes for non-CFLB-aware end hosts. CFLB does not require any extension in packet headers and does not store any state in routers which perform only simple calculations. Instead, the state is stored in end-hosts and conveyed using header fields in each steered packet.

To summarize, our key contributions are:

- A study of path diversity in data center networks;
- The design of a packet steering solution that is stateless for routers and does not require a

shim header;

- The evaluation of the benefits of combining such solution with Multipath TCP.

The remainder of this paper is organized as follows. We first recall in Sec. 2 current hash-based load-balancing basics and discuss related work. We then provide a detailed description of the operation mode of CFLB in Sec. 3. In Sec. 4, we analyze the performance of CFLB, we first use trace-driven simulations to compare CFLB with existing hash-based techniques and, then, we implement CFLB in the Linux kernel to evaluate its packet forwarding performance. We also evaluate the benefits of CFLB for MultiPath TCP hosts. In Sec. 5, we discuss other possible applications of CFLB.

2. Network-based Load Balancing in a Nutshell

In this section, we first provide an in depth overview of the network-layer load balancing using hash based techniques as well as an evaluation of path diversity in data centers. We then discuss other related work on network-based load balancing.

2.1. Path Diversity at the Network Layer

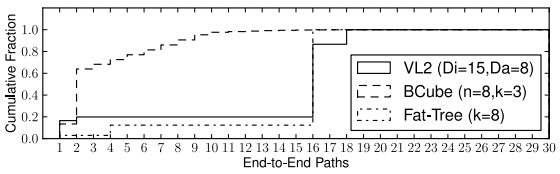
There exist several proposals to enable path diversity at the network layer [35]. In practice *Equal-Cost Multi-Path* (ECMP) [24] is currently the mostly deployed one. ECMP is both a path selection scheme and a load distribution mechanism. To enable path diversity, it uses paths that tie to ensure loop-free forwarding. According to the level of resulting path diversity, routers then proportionally balance packets over their multiple next hops. This proportional aspect is an arbitrary design choice which is not in the scope of this paper. We focus on the practical implementation of the mapping (packet → next-hop). Various next-hop mapping methods exist to practically balance packets over load-balanced paths.

An ECMP load balancer should fairly share the load over the next hops. Since flows vary widely in terms of number of packets and volume, this is not that easy [24, 10]. Most load-balancing methods that meet these requirements are based on a hash function [10, 1]. They compute a hash over the fields that identify the flow in the packet headers. These fields are usually the source and destination IP addresses, the protocol number and the source and destination ports, i.e. the *5-tuple*. The computed hash can then be used in various ways to select a next hop. The simplest and most deployed method is called *Modulo-N* (see Fig. 3a). If there are N available next hops, the remainder of dividing the hash by N is used as an identifier of the next hop to use. In [10], Cao *et al.* use trace-driven simulations to compare the performance of several hash functions and show that CRC-16 provides the best cost/performance tradeoff.

Due to the non-determinism property of hash functions, forcing a path in a load-balanced network is hard. In general hosts can only vary the transport header fields to try to influence the path selection.

2.2. Path Diversity in Data Center Networks

ECMP is widely used in data centers. Several recent data center proposals have been optimized to support it [39, 37]. To demonstrate the potential presence of ECMP load balancers, we analyze the topology and the resulting routing base of several data center networks. We consider the path diversity between pairs of servers. We use three generic models: Fat-Tree [2], VL2 [20], and BCube [21]. For each model, we instantiate a representative topology with approximately 600



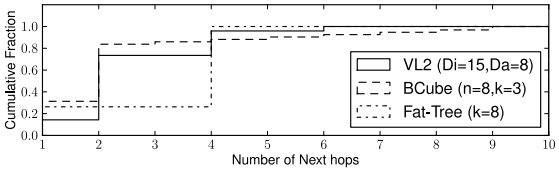


Figure 1: More than 80% of the server pairs in Figure 2: On average, for 70% of destinations, popular models of data center topologies have routers and switches have multiple next hops. two or more paths between them.

servers. Since the simple spanning tree at the link layer also tends to be replaced by multipath-capable routing [42], we also consider switches as load balancers[1].

We observe in Fig. 1 that more than 80% of pairs of servers in these data center networks have at least two paths between them. They have up to 64 paths in BCube. Moreover, Fig. 2 shows that for 70% of destinations the switches and routers have at least a destination for which there are two next hops or more. The maximum number of next hops is 10.

2.3. Related Work

Several researchers have evaluated the performance of dynamic load-balancing techniques. Menth *et al.* consider in [32] dynamic load balancing at the network layer not solely based on static forwarding ratio [10]. Kandula *et al.* propose another dynamic load-balancing technique in [27].

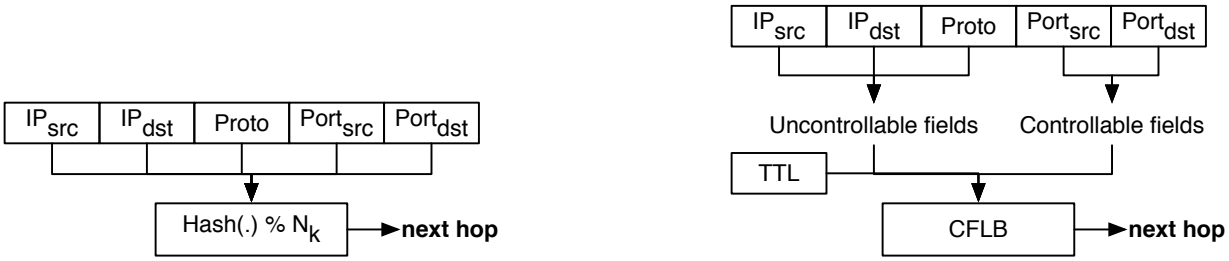
Network-layer load balancing is not limited to the use of ECMP, other forms of multipath routing have also been proposed. BANANAS [29], Routing Deflections [45], Path Splicing [36], and Pathlet Routing [19] are examples of scalable routing primitives that allow end systems to use non-shortest paths as an alternative to explicit source routes. These solutions rely on the utilization of a shim header in addition to the IP header to allow end hosts to exploit the path diversity. Otherwise, the packet is forwarded along the normal route.

Xi *et al.* proposed in [44] *Probe and RerOute Based on ECMP* (PROBE). This technique combines path probing, similar to Paris-traceroute [5], to discover the ECMP paths and uses a NAT technique on the hosts to reroute the flows.

Recent proposals have focused on ways to distribute the flows inside data centers [14, 3, 37]. *SPAIN* [37] is a proposal that configures multiple static VLANs in data centers to expose to end hosts the underlying network paths. In the case of *SPAIN* either the hosts or the switches must be modified to allow selecting a path for each flow.

Other solutions have looked at how dynamically allocate flows to paths, Al-Fares *et al.*, among others, show that it is possible to avoid generating such hot-spots by efficiently utilize aggregate network resources by using *Hereda*, a dynamic flow scheduling system [3]. The idea behind this solution is to assign large flows to lightly loaded paths by querying a scheduler that possesses informations about the current network load. Hereda stores state in the switches, i.e., flow entries in OpenFlow [34], in order to ensure that flows follows the desired path. However, as pointed out by Curtis *et al.* in [14], there currently does not exist any OpenFlow switch that can support the required number of flows at each rack switch.

[1] In BCube models, servers not only act as end hosts, they also act as relay nodes for each other.



(a) Regular ECMP.

(b) CFLB.

Figure 3: Overview of load-balancing mechanisms.

3. Controllable per-Flow Load-Balancing

CFLB enables hosts to select a load-balanced path among the diversity offered by the network layer. Compared to a source routing solution, it allows CFLB-aware sources to steer their packets inside the network without requiring any header extension. Fig. 3 presents the overview of the mechanisms ECMP and CFLB use to select a next hop when several are possible. The mechanisms are quite similar: for IPv4 networks both apply an operation on the 5-tuple to select a next hop. ECMP uses a hash-modulo operation while CFLB uses a more complex operation that will be detailed in the remainder of this section. CFLB uses an additional field of the packet header, the *Time to Live* (TTL), as a way to "identify routers". CFLB is not only limited to layer-3 routing, it can work in any environment where a hop count is used (e.g., TRILL [42], MPLS, IPv6, etc.). CFLB decomposes those header fields in two categories: *controllable* and *uncontrollable*. The controllable fields are the fields that can be selected by the source, e.g., the source and destination TCP ports[2] while the remaining fields are uncontrollable, i.e., the protocol number and the source and destination addresses. In short, the controllable fields are used to convey the path selector while the uncontrollable fields are used to add randomness for non-CFLB aware flows. The CFLB mechanism can be decomposed in four separate operations:

1. The desired path is specified by the source as a sequence of next-hop selections, that we call a *path selector*;
2. This path selector is then encoded by the source inside the *controllable* header fields;
3. Each router recovers from these controllable fields the encoded path selector;
4. The path selector is then used to extract the next-hop selection the routers need to apply to load-balance each packet.

This section is organized to help understand the design choices behind CFLB. We first describe the path selector. We then describe operations (1) and (4) and finally operations (2) and (3).

[2]We see in Sec. 4.2 how the destination port can be controlled.

3.1. Path Selector

In a network offering path diversity, there exist multiple paths having the same cost between a source and a destination. We call these load-balanced path. Fig. 4 shows all load-balanced paths between a source S and a destination D in a simple network. As only load-balanced paths are shown, Fig. 4 does not show links that are not used to route packets to the destination D. In this example there exist five different load-balanced paths between nodes S and D. Table 1 lists the notations used in this paper and their definitions.

Symbol	Definition
B	The radix of the path selector, i.e., the numeral base to encode the path selector.
n_i	The next-hop selection of the i^{th} positioned router.
L	The length of the path selector, i.e., number of next-hop selections that can be encoded in it.
N_k	The number of load-balanced next hops available at a router k (for the sake of clarity, we ignore the destination prefix).
$F(x)$	The invertible function applied on x.
$H(x)$	The Hash function applied on x.
c_f	The controllable fields used.
u_f	The uncontrollable fields used.
$E_i(n_i)$	The function applied on a next-hop selection, it adds "randomness" using u_f.
$D_i(x)$	The function that performs the inverse of E_i, i.e., $D_i(E_i(x)) = x,\ \forall x \in [0, B[$.
$A\|\|B$	The concatenation of A and B.

Table 1: General notations.

CFLB allows sources to control the next-hop selection in routers when they have more than one possible next hop for a destination. For example, router R_4 has 3 potential next hops for destination D. By knowing the number N_k of available next hops towards a destination for each router k in the network, next-hop selections can be mapped to a number $n_i \in [0, N_k[$, where n indicates the index of the next hop which should be selected. Using such mapping, if source S wants to follow the highlighted path in Fig. 4, the next-hop selection for router R_4 is 2. Note that only routers having multiple load-balanced next hops to forward a packet must be controlled by the source if the latter wants to follow a path in the network.

To specify the next-hop selection a router needs to performs, it must be identifiable by the source. As we only want to rely on existing fields of the packet header, to identify the position of a router inside the path selector we use the TTL field present in each packet. The TTL is set by the source and decremented when the packet is forwarded by a router. The field can thus be used by a router to identify its relative position compared to the initial TTL value and thus to represent its position inside the path selector. Therefore, we define the position of each router within the path selector to be $i = ttl \mod L$ where ttl is the TTL value of the packet received at the router and L is the number of next-hop selection that can be encoded in the path selector. The source can, by knowing the initial TTL of the packets[3], encode the next-hop selections of each CFLB router on the path at their corresponding position inside the path selector.

Let us assume that the source S uses an initial TTL of 64, that the length of the path selector is $L = 20$ and that the source wants his packets to follow the bold path in Fig. 4. The positions of the next-hop selections of routers R_1 and R_4 inside the path selector are respectively 4 ($64 \mod 20$)

[3]Most operating systems use a system wide default TTL.

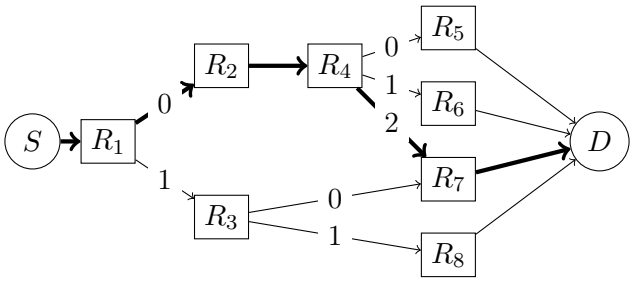


Figure 4: The load-balanced paths between S and D.

and 2 ($62 \mod 20$). The complete load-balanced path can therefore be expressed as the following sequence of next-hop selections: $\{(4 \rightarrow 0), (2 \rightarrow 2)\}$ where the notation $(x \rightarrow y)$ refers to a router at position x which should select its y^{th} next hop.

3.2. Representation and Extraction

CFLB stores a path selector as a positional base-B unsigned integer, where B is known as the radix and shared by all the nodes in the network. This maximizes the number of next-hop selections that can be encoded inside the path selector, while minimizing the number of bits used. As the potential value of n_i is bounded by the radix B, one should choose a B value so that it is the maximum of the number of available next hops N_k towards each destination in every routers of the network. In practice, most routers have a hardcoded upper bound on the number of next hops they can use for a destination, most of them support up to of 16 next hops [5].

Representation. A path selector p can be generalized as:

$$p = \sum_{i=0}^{L-1} n_i \times B^i \tag{1}$$

Where n_i is an unsigned integer in base B that represents the next-hop selection of the router having the i^{th} position within the path selector. If a source wants its packets to follow the same path for different services then it might cause a path selector collision. To overcome this issue, CFLB generates multiple path selectors to describe the same load-balanced path using two solutions that can be combined. First, the unused positions inside the path selector can be filled with random values (as for R_2 and R_7 in Fig. 4). Second, by changing the initial TTL, the position of each router in the path selector changes and thus the path selector value.

Extraction. A router receiving a path selector p can extract the forwarding decision n_i by first extracting the *ttl* of the packet and then computing $i = ttl \mod L$. Finally, n_i can be extracted by applying Eq. 2.

$$n_i = \left\lfloor \frac{p}{B^i} \right\rfloor \mod B \tag{2}$$

The integer division by B^i removes all load-balanced next-hop selections of upstream routers while the modulo operation removes all load-balanced next-hop selections of downstream routers.

Example. Let us now illustrate how CFLB works in the simple network shown in Fig. 4. Assume that the source uses an initial TTL of 64, wants this packet to follow the bold path in Fig. 4. That radix B is 3 and L is 20. The path selector computed by the source based on Eq. 1 can therefore be expressed as:

$$p = \{(4 \rightarrow 0), (2 \rightarrow 2)\} = 0 \times 3^4 + 2 \times 3^2 = 18$$

Note that this implies encoding a next-hop selection of 0 for all other positions inside the path selector. This value is then encoded inside the packet header (we discuss how the path selector is encoded and decoded in the header fields in Sec. 3.3). R_1 retrieves from the packet header the same path selector and the TTL to compute its position inside the path selector, i.e., 4. It then computes the next-hop selection it needs to apply on the packet by using Eq. 2:

$$n_4 = \left\lfloor \frac{18}{3^4} \right\rfloor \mod 3 = 0$$

R_1 decrements the TTL of the packet and forwards it to the next hop labeled 0, i.e., R_2. R_2 does not have load-balanced next hops. It forwards the packet to R_4 and decrements the TTL. R_4 applies the same operation as R_1. R_4 computes:

$$n_2 = \left\lfloor \frac{18}{3^2} \right\rfloor \mod 3 = 2$$

The packet is therefore forwarded to R_7 and then to D as R_7 only has one next hop to forward the packet.

3.3. Encoding and Decoding

CFLB uses the controllable fields of the packet header in order to convey the path selector from routers to routers. These fields must not change during the forwarding process. We use an invertible function F to encode the path selector. The function F must exhibit the avalanche effect [43]. Indeed, a simple bijective function might end up in a poor distribution of the non-controlled traffic [10]. For instance, if only the port numbers are used as controllable fields, we do not want that all web traffic goes through the same next hop. Therefore, we require that the invertible function exhibits the *avalanche effect*, that is for a small variation of the input (different source-ports) a large variation of the output is observed. Based on this requirement, block ciphers such as Skip32[4] or RC5 [40] are good candidates to implement this invertible function when 32 bits are controllable. When more bits are available, we recommend to use a block cipher mode of operation as *Format-Preserving Encryption* (FPE) construction, such as the *eXtended CodeBook* (XCB) mode of operation [33] that accepts arbitrarily-sized blocks, provided they are as large as the blocks of the underlying block cipher. Depending on the number of bits that are controllable in the packet header, different types of block ciphers can be used with XCB, from 32-bit symmetric-key block cipher to most common ones such as DES, 3DES or AES. Furthermore, efficient hardware-based implementations of such block ciphers exist [16, 23]. Using such functions to encode the path selector enables a router to apply the inverse of this function on the controllable fields of the packet header to retrieve the path selector.

[4] `http://www.qualcomm.com.au/PublicationsDocs/skip32.c`.

Uncontrollable fields. CFLB is designed based on the properties of hash-based load balancers. It must be transparent to sources that do not need to control the load-balanced path taken by their packets. In this case, the controllable fields are random and do not encode a path selector. CFLB must still distribute such packets efficiently among the available load-balanced next hops. As Cao *et al.* showed in [10], the most efficient packet distribution is achieved when all the headers fields identifying a flow is used as input to the load balancing function. CFLB therefore uses also the uncontrollable fields, source and destination addresses and the protocol number, as input. For that, the way the path selector is encoded slightly changes from Eq. 1:

$$E_i(n_i) = (n_i + H(u_f)) \mod B \tag{3}$$

$$p = \sum_{i=0}^{L-1} E_i(n_i) \times B^i \tag{4}$$

where, H is a hash function and u_f contains the uncontrollable fields of the packet header. This allows to efficiently distribute packets over available next hops of each router, while still allowing routers to recover the next-hop selections encoded by the sources.

Eq. 4 can be inverted to find the next-hop selection n_i to apply on the router whose position is i by applying the following operation on the path selector extracted from the packet header:

$$D_i(x) = (x - H(u_f)) \mod B \tag{5}$$

$$n_i = D_i(\left\lfloor \frac{p}{B^i} \right\rfloor) \tag{6}$$

Next-hop selection. The next-hop selection n_i is a value comprised between 0 and $B-1$. To select a next hop, CFLB applies a mapping between n_i and a value between 0 and $N_k - 1$. However, as $N_k \leq B$, an issue arises when using a simple modulus operation when B is not a multiple of N_k. In this case, the load balancing distribution might be poor. For instance, if $N_k = 2$ and $B = 3$, if the input is uniformly distributed, then the router ends up forwarding 75% of the incoming packets to the first next hop. To resolve this problem, CFLB computes the next-hop selection n_i for the packet as follows :

$$n_i = \begin{cases} D_i(\lfloor \frac{p}{B^i} \rfloor), & \text{if } D_i(\lfloor \frac{p}{B^i} \rfloor) < N_k \\ H(c_f \| u_f) \mod N_k, & \text{otherwise.} \end{cases} \tag{7}$$

The intuition behind Eq. 7 is that CFLB must distinguish whether the packet was controlled by a source or not. If the packet was indeed controlled, the next-hop selected, n_i, must be the one encoded in the path selector. However, the non-controlled packets must be distributed randomly among the N_k available next hops. In Eq. 7, if the packet to forward is a controlled one, then the resulting next hop to select should be lower than N_k (the number of next hops in the routing table for the packet's destination), the decision encoded at the source is correctly taken. Otherwise, it means that the packet is not a controlled one, resulting in a random distribution of the packet on one of the N_k available next hops.

In case of topological changes (transient or permanent), a CFLB-router will renumber indexes, i.e., update the $n_i \rightarrow R_j$ mapping and N_k, of its current available next hops towards each destination. In such a case, while new flows are "aware" of the new state and are so correctly controlled, previous existing ones may be impacted. Indeed, when the desired next hop does not exist anymore or if its

index has changed, the resulting path will change. In CFLB, the impacted controlled flows (i.e., the elephants ones) will fall back to a random load balancing and so will be distributed similarly to classic hash-based load balancing. We consider that such topological changes should be quite marginal (at a time scale greater than flows duration) and new subflows may be created if impacted ones share a common bottleneck.

3.4. Summary

In the previous section, we have explained all the design decisions behind the CFLB algorithm. For clarity, we provide in this section the detailed pseudocode of CFLB.

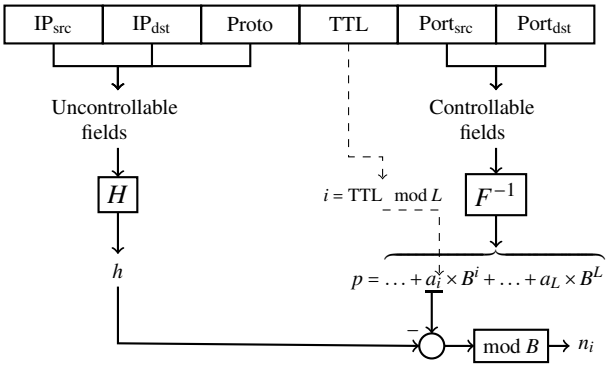


Figure 5: The complete mode of operation of a CFLB router.

Network-wide constant: B = The radix in use in the network.
Network-wide constant: X = The number of bits that are controllable in the packet header.
Input: *pckt* = The packet to forward.
Output: The next-hop selection to apply on *pckt*.
1: $L \leftarrow \lfloor \log_B(2^X) \rfloor$
2: $c_f \leftarrow$ `ExtractControllableFields`($pckt$)
3: $u_f \leftarrow$ `ExtractUncontrollableFields`($pckt$)
4: $ttl \leftarrow$ `ExtractTTL`($pckt$)
5: $p \leftarrow F^{-1}(c_f)$
6: $n_i \leftarrow (\left\lfloor \frac{p}{B^{(ttl \mod L)}} \right\rfloor - H(u_f)) \mod B$
7: **if** $n_i < N_k$ **then**
8: **return** n_i
9: **else**
10: **return** $H(u_f \| c_f \| Router_{ID}) \mod N_k$
11: **end if**

Algorithm 1: Pseudocode showing operations performed by a CFLB router.

Fig. 5 and Alg. 1 show respectively the operations and the pseudocode performed by a CFLB router to forward a packet among load-balanced next hops. The first operation is to extract the controllable and the uncontrollable fields and the TTL from the packet header. Operation F^{-1} is the inverse of the invertible function that extracts the path selector from the controllable fields. In Alg. 1, after having retrieved the next-hop selection n_i, the router performs the operation introduced in Sec. 3.3, i.e., an if-then-else on the value n_i, to determine whether the packet was controlled by the source. The a_i value, in Fig. 5, corresponds to the addition modulo B of the next-hop selection performed at position i and the hash computed on the uncontrollable fields. This a_i value was inserted by the source at the i^{th} position inside the path selector. The router can thus retrieve it

Network-wide constant: B = The radix in use in the network.
Network-wide constant: X = The number of bits that are controllable in the packet header.
Input: $path$ = A sequence (ttl_i, n_i), where ttl_i is the TTL of the packet when received by router i and n_i the next-hop that should be selected by router i.
Input: u_f = The uncontrollable fields the source needs to use.
Output: The controllable fields (c_f) to use to force a packet to follow the load balanced path $path$.

$L \leftarrow \lfloor \log_B(2^X) \rfloor$
$p \leftarrow 0$
for $(ttl_i, n_i) \in path$ **do**
    $p \leftarrow p + ((n_i + H(u_f)) \mod B) \times B^{(ttl_i \mod L)}$
end for
return F(p)

Algorithm 2: Pseudocode showing the path selector construction.

and compute the subtraction modulo B with the same hash value, to finally retrieve the next-hop selection n_i.

Alg. 2 shows the pseudocode used by sources to construct a path selector. The source needs to first compute the length of the path selector. This is needed because the routers position themselves inside the path selector by using the TTL and the length of the path selector. Thus, this length has to be taken into consideration to find the path selector. Then, the source iterates over all routers on the path (with a TTL ttl_i and a next-hop selection n_i) to compute the path selector.

4. Evaluation

In this section, we evaluate the performance of CFLB compared to hash-based load-balancing. We use the source and destination port numbers as controllable fields and the IP addresses are used as the uncontrollable fields for CFLB. Our goal is twofold: first, evaluate its load-balancing and forwarding performances, and second, show through simulations and experiments how MultiPath TCP can benefit from CFLB to exploit the underlying path diversity.

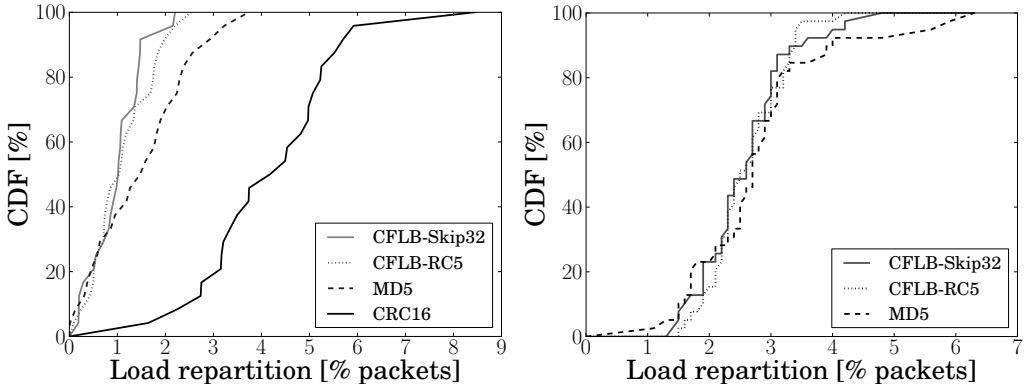
4.1. Performance Evaluation

4.1.1. Load-Balancing for Non-Controlled Flows

Our first requirement evaluates whether a router having multiple next hops for a given destination uniformly distributes the load [10] for non-controlled flows. If there exist N next hops for a given destination prefix, the load balancer should distribute $\frac{1}{N}$ of the total traffic to each next hop.

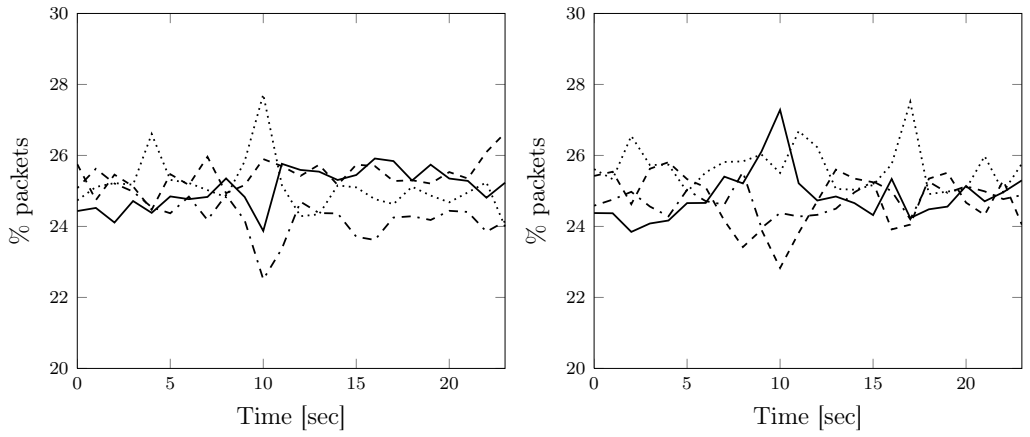
CFLB enables sources to steer controlled packets while also acting as a classic load balancer for non-controlled packets (e.g., mice flows). To compare the hash-based load balancing techniques and CFLB, we simulated each method using realistic traces and evaluated the fraction of packets forwarded to each next hop. We based our simulations on the CAIDA passive traces collected in July 2008 at an Equinix data center in San Jose, CA [41].

To analyze how CFLB balances the non-controlled traffic compared to hash-based techniques, we first simulated 10 million packets (extracted from the CAIDA traces) forwarded through one load balancer performing a distribution among $N = 2$ next hops. Fig. 6a shows the result of this simulation (computed every second). There are three observations resulting from Fig. 6a. First, using CRC16 as a hash-based load balancer gives a rather poor distribution of packets. Second, as the maximum deviation value never goes up to 4% of packets, the load distribution among the two output links is close to an equal 50/50 % repartition of traffic for all evaluated techniques except CRC16. Third, CFLB, whatever the block cipher used, achieves an equivalent load distribution as



(a) After one load balancer. (b) After 4 load balancers.

Figure 6: Deviation from an optimal distribution amongst two possible next-hops.



(a) MD5 Hash-Based (b) RC5 CFLB

Figure 7: Packet distribution computed every second amongst four possible next hops.

a hash-based load balancer using MD5. We did not observe a significant impact on the quality of the load distribution according to the value of B used.

We also evaluated the load balancing performance considering a sequence of several load balancers. Fig. 6b shows the cumulative distribution of the maximum deviation of the load distribution after crossing four subsequent load balancers (computed every half second). The same observation as for Fig. 6a applies, CFLB performs at least as good as a classical hash-based load balancing technique.

Another performance factor is how the load distribution varies over time. Fig. 7a and Fig. 7b show, for respectively a hash-based load balancer using MD5 and CFLB using RC5, the load balancing distribution of packets over time when $N = 4$. We analyze here the case of a router having four outgoing links toward a given destination. A perfect load balancer would send 25% of the packets on each link. We can notice that there are no significant differences between the two techniques, as they behave in the same way over time. They both slightly fluctuate within the same tight interval $[22\%, 28\%]$ and their median is close to 25%. Simulations with other traces and

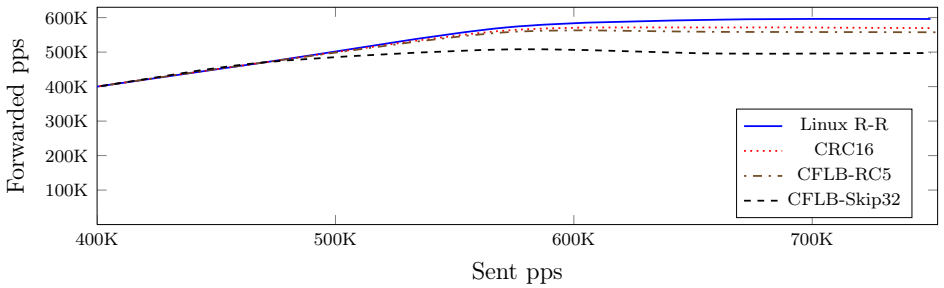


Figure 8: CFLB gives equivalent forwarding performance as hash-based load balancers.

different values of N provide similar results.

4.1.2. Forwarding Performances

The second requirement is the forwarding performance. In order to evaluate it, we implemented the forwarding path of CFLB as a module in the *Linux kernel 2.6.38*[5]. The basic behavior of the Linux kernel when dealing with multiple next hops for a given destination is to apply a round robin distribution of packets based on their IP addresses, therefore performing a pure layer-3 load balancing. As this is not comparable to the hash-based load-balancing behavior introduced in Sec. 2.1, we extended the Linux kernel to take into consideration the 5-tuple of the packets and then apply a hash function to select a next-hop (only CRC like functions are available). To implement CFLB in the kernel, we extend the previously mentioned hash function to enable the deterministic selection of a next hop as described in Sec. 3. We used two different 32-bit block ciphers to implement the invertible function: RC5 and Skip32. The implementation of these two block ciphers has not been optimized, the goal is solely to prove the feasibility of our solution (various techniques could be used to improve its performance [31, 22]). Note that CFLB also computes a CRC function over the uncontrollable fields to add randomness for non-controlled flows.

We deploy a testbed of three computers to evaluate the performance of the forwarding path of a Linux router. The computer acting as a load balancer is an Intel Xeon X3440 @2.53GHz, and both sender and receiver are AMD Opteron 6128 @2GHz. The sender is connected through a 1Gbps link to the load balancer which balances traffic amongst two 1Gbps links to the receiver. The traffic was generated using 8 parallel *iperf*[6] generators, creating UDP-packets with a payload of 64 Bytes, in order to overload the load balancer. The result of this experiment is given in Fig. 8.

The classic Linux Round-Robin on the IP-addresses obviously performs the best (it only requires a lookup of the destination IP address in the routing cache to forward the packet). It forwards approximatively 600,000 packets per second. Not far below, both the classical hash-based technique using CRC16 and CFLB using RC5 are able to forward respectively 570,000 and 560,000 packets per seconds. This performance decrease compared to the standard Linux Round-Robin is mainly due to the more complex hash-algorithm that selects the next hop. Finally, CFLB using Skip32 forwards up to 500,000 packets per seconds. We can conclude that CFLB, even using non-optimized block ciphers, comes with a marginal additional cost as it gives equivalent forwarding performances as classical hash-based techniques.

[5]More information can be found at: `http://inl.info.ucl.ac.be/cflb`.
[6]`http://iperf.sourceforge.net/`

4.2. MPTCP improvements with CFLB

Multipath TCP (MPTCP) is an extension to TCP that splits a data stream over multiple TCP subflows while still presenting a standard TCP socket API to applications [6]. The subflows can be established using the same IP address pair and different ports [39] or just using distinct IP addresses of the same end hosts [18]. Raiciu *et al.* evaluated MultiPath TCP inside data centers in [39]. Simulations and measurements show that performance improves when MultiPath TCP is allowed to use multiple subflows in such data centers. However, the non-deterministic nature of the hash function does not guarantee that subflows will follow distinct paths inside the data center. Therefore a large number of subflows are needed to increase the probability to use disjoint paths and thus utilize all the available network resources. Using a large number of subflows comes with a cost in CPU cycles and network resources. With CFLB, Multipath TCP is guaranteed to efficiently use the network resources as subflows can be deterministically mapped to paths.

In the following section, we evaluate the advantages of using MPTCP hosts conjointly with a CFLB-enabled network. We first simulate a data center environment to show that when using CFLB, MPTCP establishes fewer subflows for elephant connections than with a probabilistic approach. Finally, we also show in a small testbed that MPTCP can benefit from the usage of CFLB to avoid crossing hot spots.

4.2.1. Data Center Simulations

We performed simulations of MPTCP-enabled data centers and evaluate the performances achieved when a simple central flow scheduling algorithm allocates elephant flows. The scheduler that we use for simulations simply consists in minimizing the number of flows going through each link of the data center. In practice, hosts can use a similar technique as in [14] to detect whether one connection corresponds to an elephant flow, and if so query the scheduler to establish additional subflows. The scheduler then specifies to the host the TCP ports to be used to setup a new subflow. The required TCP ports are computed using the CFLB mode of operation allowing to map a subflow to a specific path in the network. We refer to this combination of MultiPath TCP and CFLB in the remaining of this section as MPTCP-CFLB.

To evaluate the benefits of CFLB with MPTCP in data centers, we first enhanced the *htsim* packet-level simulator used in [39] to support path selection with CFLB. We consider exactly the same Fat-Tree datacenter topology as discussed in Fig. 2 of [39]. This simulated datacenter has 128 MPTCP servers, 80 eight-port switches and uses 100 Mbps links. The traffic pattern is a permutation matrix, meaning that senders and destinations are chosen at random with the constraint that destinations do not sink more than one connection. The regular MPTCP bars of Fig. 9 are the same as Fig. 2 of [39]. It shows the throughput achieved by MPTCP when MPTCP subflows are load-balanced using ECMP. The MPTCP-CFLB bars show the throughput that MPTCP is able to obtain when CFLB balances the MPTCP subflows over the less loaded paths. The simulations show that with only 2 subflows, MPTCP-CFLB is much closer to the optimum than MPTCP with hash-based load balancing. Even with only one subflow (smartly allocated by the scheduler), improvements are considerable and MPTCP-CFLB achieves a good utilization of the network. This can be explained by the fact that relying on a random distribution of subflows ends in a poor use of available resources.

Similar results have been observed on other data center topologies such as VL2 and BCube. We also performed simulations for an overloaded data center and observed that using MPTCP-CFLB conjointly with a flow scheduler focusing on less congested paths offers more fairness amongst the different connections.

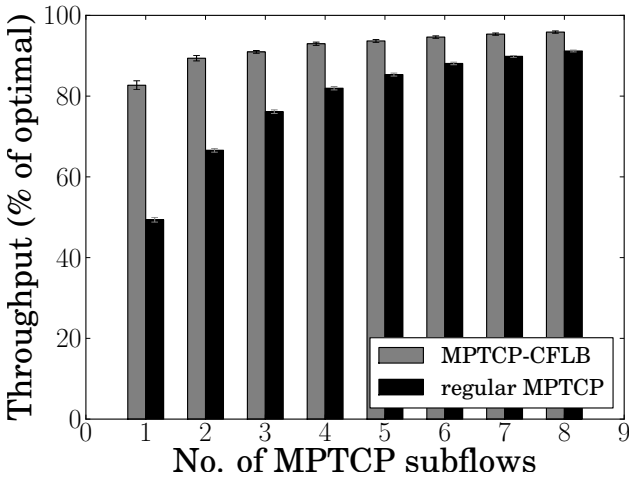


Figure 9: MPTCP needs few subflows to get a good Fat Tree utilization when using CFLB.

4.2.2. Testbed Experiments

We modified the MPTCP *Linux kernel 2.6.36* implementation [6] to add the deterministic selection feature offered by CFLB. We created a *netlink* interface to the kernel so that a user-space module can interact with MPTCP and announce to the kernel the subflows to create. The CFLB functionality has been implemented in user-space. Our prototype allows a source to control the source and destination TCP ports to follow a specific path inside an IPv4 network. The two block ciphers (RC5 and Skip32) were also implemented inside the kernel crypto library.

A python library *pycflb* was developed to provide a simple API for interacting with the user-space CFLB. We also developed an RPC server to show the feasibility to centralize the computation of CFLB in a server. This server has information about the network topology and is the only one that interacts with the pycflb library. pycflb can be configured with the cipher and key parameters in use in the network. Sources only query it to retrieve the ports to use or to recover the path taken by a specific flow. These three implementations (Linux MPTCP netlink-interface, user-space CFLB and pycflb library) allow a source to deterministically map subflows to paths and represent approximatively 4,000 lines of code.

When MultiPath TCP runs on a single homed server, additional subflows are created by modifying the port numbers in a random manner. Since MultiPath TCP relies on tokens to identify to which MPTCP connection a new subflow belongs, both the source and destination TCP ports can be used to add entropy. Combining CFLB and MultiPath TCP in the Linux MPTCP implementation provides a significant benefit because the subflow 5-tuple can be selected in such a way that the underlying path diversity offered by the network can be easily exploited. We evaluate the benefit of this technique in a small testbed with a client and a server (AMD Opteron 6128 @2GHz) and two CFLB-capable routers (Xeon X3440 @2.53GHz).

In the first experiment, each host is connected to one router via a 1Gbps link. The routers are directly connected via seven 100 Mbps links. These 7 links offer 7 different distinct paths between the client and the server. If seven MPTCP-subflows are created, an optimal usage of the network should result in about 700 Mbps of throughput. To evaluate this, we ran iperf between the hosts, creating traffic during one minute. The experiment has been repeated 400 times to collect representative results. Fig. 10 provides the probability distribution function of the number

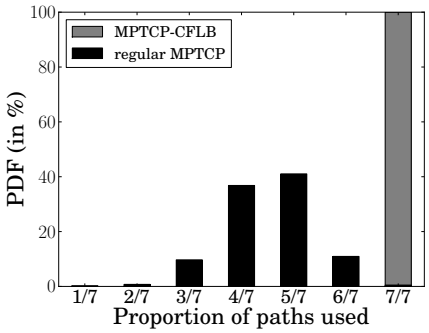


Figure 10: Regular MPTCP is unlikely to use all paths. MPTCP-CFLB on the other hand always manages to use all the paths.

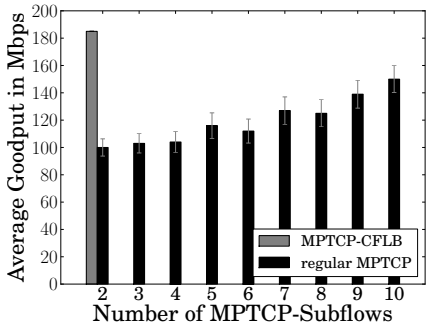


Figure 11: Regular MPTCP has a very small probability of using link A of Fig. 12 and is thus suboptimal compared to MPTCP-CFLB.

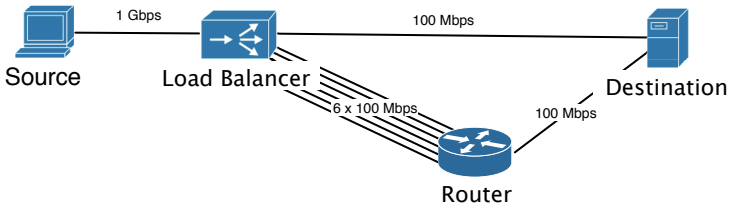


Figure 12: Testbed – The maximum throughput available between S and D is at 200 Mbps due to the bottleneck link between the router and the destination.

of distinct paths used by the classical MPTCP and our enhanced MPTCP-CFLB implementation.

Fig. 10 raises the following observation: as expected, using seven subflows, MPTCP-CFLB is able to take the full benefit of the seven paths while the classical MultiPath TCP cannot efficiently utilize them. Indeed, the performance of MPTCP-CFLB is completely deterministic as the MPTCP connection balances exactly its seven subflows over the seven paths. Among the 400 experiments, when paths are randomly selected, only two experiments were able to use the seven paths. This can be confirmed by the probability that k subflows go through k different paths amongst the l paths available, which can be written as:

$$P^m_{(k,l)} = \frac{l!}{(l-k)! \times m^k} \qquad (\forall\ k,l,m \in \mathbb{N} \mid k \leq l \leq m)$$

Therefore, covering the whole 7 load-balanced paths with 7 subflows probability is $P^7_{(7,7)} = 0.6\% \approx \frac{2}{400}$ which explains the poor result of MPTCP. Most of the experiments result in four or five paths being used. This implies that two or three paths carry two competing TCP subflows from the same MPTCP connection.

Our second evaluation (Fig. 12) still offers 7 distinct paths from the source to the destination, but this time the destination has two 100 Mbps links. One is a direct link from the load balancer to the destination and the second is attached to the router. With only two subflows, MPTCP-CFLB is able to saturate the two 100 Mbps interfaces of the destination. Fig. 11 compares the performance of MPTCP and MPTCP-CFLB when the number of subflows varies. Each measurement with MPTCP was repeated 100 times and Fig. 11 provides the average measured goodput. These measurements clearly show that when using random TCP port numbers, MPTCP is unable to efficiently use the

two different 100 Mbps links. Increasing the number of subflows slowly increases the performance, but adding a subflow to an MPTCP connection comes with a significant cost. Thus, the less TCP sublows are established, the best it is. MPTCP-CFLB is able to cover all the available paths with the minimal cost.

5. Discussion

In this paper, we have mainly focused on the utilization of CFLB in data centers networks carrying TCP/IPv4 packets. CFLB could be applied to other problems in different networking technologies. Extending CFLB to support another networking technology can be done by selecting the controllable and the uncontrollable fields of the packet header that are used as input to the load-balancing algorithm.

A first natural extension of CFLB would be to deploy it in IPv6 networks. In an IPv6 network, CFLB could also rely on the source and destination ports, but IPv6 packets contain a 20 bits flow label field. The semantics of this field is still being debated within the IETF [11]. IPv6 sources could leverage CFLB to encode a path selector in the flow label field of their packets.

Although we illustrated the benefits of CFLB with MultiPath TCP, the same could be applied for single TCP/UDP flows, in this case only the source port could be controlled as applications running on top of those transport protocols require to specify the destination port.

A CFLB network has other benefits. One of the side effect benefits is the monitorability of the load-balanced paths. Commercial networks often deploy monitoring tools that probe network paths to verify whether their network meets the stringent SLAs that are requested by their customers. Unfortunately, when there are load-balanced paths, it is very difficult for the monitoring station to steer packets on a specific path, which complicates network monitoring. This operational problem is one of the reasons why the MPLS-TP architecture prohibits the utilization of ECMP [9]. With CFLB, this problem disappears since a monitoring station can easily steer packets along specific paths through CFLB routers.

MPLS networks often use ECMP to load balance the traffic. To enable MPLS routers to support ECMP even when carrying non-IP packets, router vendors have proposed the utilization of special MPLS entropy labels [30] to identify flows that can be load-balanced. CFLB could easily exploit these entropy labels and ensure that flows are both well balanced and that paths can be efficiently monitored.

6. Conclusion

Most data centers networks rely on hash-based load-balancing to distribute the load over multiple paths. Hash-based techniques are able to efficiently spread the load but it is difficult to predict and force the next hop selection that such load balancers will take. In this paper, we have shown that it is possible to achieve both efficient load-balancing while enabling hosts to explicitly select the paths of their flows without storing any state in the network. This open new ways to enable hosts and load balancers to interact.

Controllable per-Flow Load-Balancing relies on invertible functions, such as block ciphers, instead of solely using classical hash-based selection. Thanks to an invertible operation, it is possible to preserve the best properties of currently used hash functions with the added benefit of enabling CFLB-aware sources to steer elephant flows over selected load-balanced paths. We envision a data center where mice flows are distributed using the classical load balancing model while elephant

flows are deterministically allocated to less congested paths. Our simulations indicate that CFLB is as efficient as classical hash-based techniques to achieve an optimal load distribution. Performance measurements in our lab have shown that our prototype implementation in the Linux kernel achieves almost the same performance as the default per-destination load balancing. Furthermore, we have shown that by coupling CFLB with Multipath TCP it is possible to greatly improve the utilization of data center networks offering path diversity between pairs of servers.

We hope that CFLB will encourage other researchers and network manufacturers to reconsider the utilization of blind hash functions in various types of load balancers.

Acknowledgment

Simon van der linden is supported by a FRIA grant. This work was partially supported by the FP7 EU project CHANGE. We particularly thank Sébastien Barré for his work on MPTCP. We would like to thank Costin Raiciu for his MPTCP simulator. We also thank Jean-Jacques Pansiot and Damien Saucez for their useful comments.

References

[1] Load balancing with Cisco Express Forwarding. Tech. rep., Cisco Systems, Inc., 1998.

[2] Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. *ACM SIGCOMM CCR 38* (Aug. 2008), 63–74.

[3] Al-fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., and Vahdat, A. Hedera: Dynamic flow scheduling for data center networks. In *Proc. USENIX NSDI* (2010).

[4] Association, I. S. IEEE Std 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks - Link Aggregation. 2008.

[5] Augustin, B., Friedman, T., and Teixeira, R. Measuring Load-balanced Paths in the Internet. In *Proc. ACM IMC* (2007), vol. 6.

[6] Barré, S., Paasch, C., and Bonaventure, O. MultiPath TCP: From Theory to Practice. In *IFIP Networking, Valencia* (May 2011).

[7] Benson, T., Akella, A., and Maltz, D. A. Network traffic characteristics of data centers in the wild. In *Proc. ACM IMC* (Melbourne, 2010), pp. 267–280.

[8] Benson, T., Anand, A., Akella, A., and Zhang, M. The case for fine-grained traffic engineering in data centers. In *Proc. of INM/WREN* (2010), pp. 2–2.

[9] Bocci, M., Bryant, S., Frost, D., Levrau, L., and Berger, L. A Framework for MPLS in Transport Networks. RFC 5921, IETF, July 2010.

[10] Cao, Z., Wang, Z., and Zegura, E. Performance of Hashing-Based Schemes for Internet Load Balancing. In *Proc. IEEE INFOCOM* (2000).

[11] Carpenter, B., and Amante, S. Using the IPv6 flow label for equal cost multipath routing and link aggregation in tunnels. Internet draft, draft-carpenter-flow-ecmp-05, IETF, July 2011.

[12] Chowdhury, M., Zaharia, M., Ma, J., Jordan, M. I., and Stoica, I. Managing data transfers in computer clusters with orchestra. In *Proc. ACM SIGCOMM* (2011), pp. 98–109.

[13] Cisco. Server Cluster Designs with Ethernet. `http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_Infra2_5/DCInfra_3.html`.

[14] Curtis, A., Kim, W., and Yalagandula, P. Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. In *INFOCOM* (2011), IEEE, pp. 1629–1637.

[15] Curtis, A. R., Mogul, J. C., Tourrilhes, J., Yalagandula, P., Sharma, P., and Banerjee, S. Devoflow: scaling flow management for high-performance networks. In *Proc. ACM SIGCOMM* (2011), pp. 254–265.

[16] De Cannière, C., Dunkelman, O., and Knežević, M. KATAN and KTANTAN – A family of small and efficient hardware-oriented block ciphers. *In Proc. CHES 2009* (2009), 272–288.

[17] Dean, J., and Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM 51* (Jan. 2008 pages = 107–113,).

[18] Ford, A., Raiciu, C., Handley, M., and Bonaventure, O. TCP Extensions for Multipath Operation with Multiple Addresses. Internet draft, draft-ietf-mptcp-multiaddressed-04, IETF, July 2011.

[19] Godfrey, P. B., Ganichev, I., Shenker, S., and Stoica, I. Pathlet Routing. In *Proc. ACM SIGCOMM* (Aug. 2009).

[20] Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: a scalable and flexible data center network. In *Proc. ACM SIGCOMM* (2009).

[21] Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. BCube: a high performance, server-centric network architecture for modular data centers. In *Proc ACM SIGCOMM* (2009), pp. 63–74.

[22] Han, S., Jang, K., Park, K., and Moon, S. Packetshader: a gpu-accelerated software router. In *Proc.ACM SIGCOMM* (2010), pp. 195–206.

[23] Hodjat, A., and Verbauwhede, I. A 21.54 Gbits/s Fully Pipelined AES Processor on FPGA. In *Proc IEEE Symp. Field-Programmable Custom Computing Machines* (2004), pp. 308–309.

[24] Hopps, C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, IETF, Nov. 2000.

[25] Iannaccone, G., Chuah, C., Bhattacharyya, S., and Diot, C. Feasibility of IP Restoration in a Tier-1 Backbone. *IEEE Network 18*, 2 (Mar. 2004).

[26] Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. Dryad: distributed data-parallel programs from sequential building blocks. *ACM SIGOPS Operating Systems Review 41*, 3 (2007), 59–72.

[27] Kandula, S., Katabi, D., Sinha, S., Berger, A., and Roughan, M. Dynamic Load Balancing Without Packet Reordering. *ACM SIGCOMM CCR 37*, 2 (Mar. 2007), 51–62.

[28] Kandula, S., Sengupta, S., Greenberg, A., Patel, P., and Chaiken, R. The nature of data center traffic: measurements & analysis. In *Proc. ACM SIGCOMM IMC* (2009), pp. 202–208.

[29] Kaur, H., Kalyanaraman, S., Weiss, A., and Kanwar, S. BANANAS: An Evolutionary Framework for Explicit and Multipath Routing in the Internet. In *Proc. ACM SIGCOMM FDNA Workshop* (2003).

[30] Kompella, K., Drake, J., Amante, S., Henreickx, W., and Yong, L. The Use of Entropy Labels in MPLS Forwarding. Internet draft, draft-ietf-mpls-entropy-label-00, IETF, May 2011.

[31] Kounavis, M. E., Kang, X., Grewal, K., Eszenyi, M., Gueron, S., and Durham, D. Encrypting the internet. In *Proc. ACM SIGCOMM* (2010), pp. 135–146.

[32] Martin, R., Menth, M., and Hemmkeppler, M. Accuracy and Dynamics of Multi-Stage Load Balancing for Multipath Internet Routing. In *Proc. IEEE ICC* (June 2007).

[33] McGrew, D. A., and Fluhrer, S. R. The Extended Codebook (XCB) Mode of Operation. Cryptology ePrint Archive, Report 2004/278, 2004.

[34] McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. Openflow: enabling innovation in campus networks. *SIGCOMM CCR 38* (March 2008), 69–74.

[35] Mérindol, P., Francois, P., Bonaventure, O., Cateloin, S., and Pansiot, J.-J. An efficient algorithm to enable path diversity in link state routing networks. *Computer Networks 55*, 1 (April 2011), 1132–1149.

[36] Motiwala, M., Elmore, M., Feamster, N., and Vempala, S. Path Splicing. In *Proc. ACM SIGCOMM* (Oct. 2008).

[37] Mudigonda, J., Yalagandula, P., Al-Fares, M., and Mogul, J. C. SPAIN: COTS data-center Ethernet for multipathing over arbitrary topologies. In *Proc. USENIX NSDI* (2010), pp. 18–18.

[38] Murray, D., Schwarzkopf, M., Smowton, C., Smith, S., Madhavapeddy, A., and Hand, S. Ciel: a universal execution engine for distributed data-flow computing. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation* (2011), USENIX Association, p. 9.

[39] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. Improving datacenter performance and robustness with multipath tcp. In *Proc. SIGCOMM* (August 2011).

[40] Rivest, R. L. The RC5 Encryption Algorithm. In *Proc. FSE* (1994), vol. 1008, pp. 86–96.

[41] Shannon, C., Aben, E., Claffy, K., and Andersen, D. The CAIDA Anonymized 2008 Internet Traces – 2008-07-17 12:59:07 - 2008-07-17 14:01:00. `http://www.caida.org/data/passive/passive_2008_dataset.xml`.

[42] Touch, J., and Perlman, R. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556, IETF, May 2009.

[43] Webster, A. F., and Tavares, S. E. On The Design Of S-Boxes. In *Proc. CRYPTO* (1986), pp. 523–534.

[44] Xi, K., Liu, Y., and Chao, J. Enabling flow-based routing control in data center networks using probe and ecmp. In *INFOCOM Workshop on cloud computing* (2011), pp. 614–619.

[45] Yang, X., and Wetherall, D. Source Selectable Path Diversity via Routing Deflections. In *Proc. ACM SIGCOMM* (2006).