SDLB: A Scalable and Dynamic Software Load Balancer for Fog and Mobile Edge Computing

Ye Yu
University of Kentucky
ye.yu@uky.edu

Xin Li
University of California, Santa Cruz
xinli@ucsc.edu

Chen Qian
University of California, Santa Cruz
cqian12@ucsc.edu

ABSTRACT

Mobile Edge Computing (MEC) provides computing/storage offloading and resource virtualization to mobile devices at the network edge. A load balancer is a necessary network function to determine the destination MEC host of each packet from a mobile device, for such virtualization. Due to the new characteristics of MEC, such as resource limitation and high dynamics, existing solutions of cloud load balancer cannot be directly applied to MEC. This paper presents a new design of a Scalable and Dynamic Load Balancer, called SDLB, that satisfies the requirements of MEC. The core algorithm of SDLB is minimal perfect hashing, which provides two perfect features as a load balancer. Evaluation results show that SDLB is faster by 4x to 10x and uses much less ($< 50\%$) memory, than a widely-used load balancer design for cloud.

CCS CONCEPTS

• **Networks** → *Packet classification*; *Traffic engineering algorithms*; • **Human-centered computing** → *Mobile computing*;

KEYWORDS

Mobile Edge Computing, Software Load balancer, Network functions virtualization

ACM Reference format:
Ye Yu, Xin Li, and Chen Qian. 2017. SDLB: A Scalable and Dynamic Software Load Balancer for Fog and Mobile Edge Computing. In *Proceedings of MECOMM '17, Los Angeles, CA, USA, August 21, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3098208.3098218

MECOMM '17, August 21, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5052-5/17/08...$15.00
DOI: http://dx.doi.org/10.1145/3098208.3098218

1 INTRODUCTION

Although mobile hardware continues to improve, it is still relatively resource-constrained compared to static computing hardware. To provide resource of computation, storage, and bandwidth to massive mobile computing devices such as those of the Internet of Things (IoT), strong back-end servers in a remote cloud is a common solution. However, many modern applications such as augmented reality (AR) and real time monitoring/control are latency-sensitive and may suffer the long round-trip delay to the cloud. Hence, Mobile Edge Computing (MEC) has been proposed to shift computing and storage capacities from the remote cloud to the network edge [9, 17]. The nodes that provide resource to mobile devices are called *MEC hosts*. Fog Computing is a computing paradigm with a similar objective of MEC [3, 24, 26], where the nodes providing resource are called Fog nodes. MEC and Fog may differ in specific characteristics. For example, most MEC hosts are deployed by the mobile service provider while Fog nodes may be network devices or terminals belonging to different users. MEC and Fog are close in their network models and most characteristics [2]. *In this work, we study with a generalized network model which can be applied to both MEC and Fog Computing.*

We consider a MEC network consisting of mobile devices and a number of MEC hosts as shown in Fig. 1. The MEC hosts provide various types of resource to mobile devices, such as CPU, memory, and disk, for difference application requirements including computation, storage, and optimization. The MEC hosts communicates with a remote cloud for further processing or storage if necessary. For example, an IoT sensing device collects the environment data and transmits the data to one of the MEC hosts [9]. The MEC hosts aggregates the data, conducts initial analysis, and transmits aggregated data to the remote cloud. For example, a video analytics application running on MEC hosts may detect special events from the date reported by video cameras, such as a lost child or an intruder, and then reports these events to a control center [17]. As another example, for an AR application running on a smartphone or tablet, the MEC host is able to provide local object tracking and local AR content caching with short latency for the mobile device [17].

MEC provides *resource virtualization* to mobile devices. In fact, a mobile device has no information about which MEC host is actually providing its requested resource. An MEC network deals with the data traffic from mobile devices to MEC hosts, serving various applications and purposes. In the network layer, we classify the packets from mobile devices

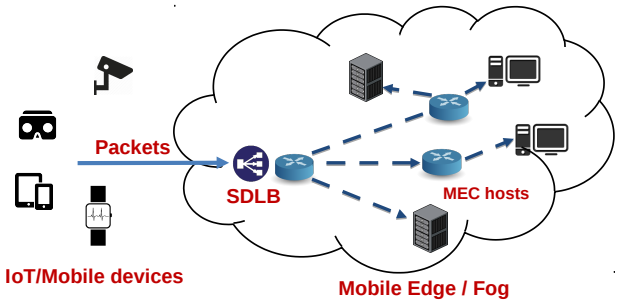

Figure 1: Network model

to MEC hosts into two groups. 1) *Stateful packets*. A packet is stateful means one of the MEC hosts stores its state, i.e., some existing information related to the packet. The packet has to be forwarded to that particular MEC host. The state can be in the device level, application level, or flow level. For example, *device-level state* can be some data reported by the same device of this packet. *Application-level state* can be some data generated by the same application of this packet. *Flow-level state* can be the previous packets of a same flow of this packet. 2) *Stateless packets*. A stateless packet can be forwarded to any MEC host, which can be a query/write to a distributed database, a request of computation offloading, or the first packet of a flow.

A *MEC load balancer* (MEC-LB) is a network function deployed between the mobile devices and MEC hosts, which assigns a MEC host as the destination for every packet from a mobile device. A MEC-LB must serve the following two properties. **P1: For every stateful packet, the MEC-LB performs a lookup to get the designated MEC host based on the packet ID. P2: For every stateless packet, the MEC-LB selects one of the available MEC hosts according to capacity of the hosts.** Different MEC hosts are assigned with different probabilities based on their capacities.

Due to the new characteristics of MEC, existing solutions of cloud load balancers [6, 10, 11] cannot be directly applied to MEC-LB. We summarize the desired requirements of a MEC-LB with comparison to a cloud load balancer as follows.

1. *Scalable*. A MEC-LB should be fast and scalable to process massive amount of packets from a large number of mobile devices. A cloud load balancer has a similar requirement [6, 10].

2. *Software based*. Compared to the hardware load balancers in a cloud [10, 11] which is a fixed amount of extra financial cost, a MEC-LB implemented in software is more flexible and cost-efficient for MEC. It can be deployed as a virtual network function running on a MEC host or a component of a router/switch.

3. *Memory-efficient*. With small memory cost, a MEC-LB is easier to fit with fast (and hence expensive) memory such as cache. Unlike a resource-rich cloud, which can use a server cluster for its software load balancer [6], a MEC network has limited resource to host a MEC-LB.

4. *Adaptive to MEC changes*. MEC hosts are heterogeneous [24]: each carrying different amount of physical resource available to mobile devices. Since many MEC hosts are built on existing devices or terminals deployed for other purposes [26, 27], they may join/leave the network depending on their own needs. A MEC host may run out of its capacity for its own applications and hence becomes unavailable to mobile devices. In contract, servers in a cloud are mostly homogeneous and more stable in the network.

5. *Portable*. The MEC-LD should be a portable solution which does not rely on any special hardware platform. It is because a MEC-LD may be deployed at any network device or terminal. The packet ID can be arbitrary, such as 5-tuple [29], MAC address, or any network names in new Internet proposals [14, 21].

This paper presents a new design of a Scalable and Dynamic Load Balancer for MEC, called SDLB, that satisfies the above requirements. SDLB is built on a new data structure named POG [28]. POG applies the theoretical studies on *minimal perfect hashing* [1, 16]. Different from the theory studies, POG supports network dynamics. In addition, it takes the advantage of the Software Defined Networking (SDN) paradigm [18, 19] which has been widely applied in enterprise and organization networks. By using POG, SDLB uses very small memory and has an important feature which is a perfect fit of a MEC load balancer: to process a packet with an ID, if the ID-host relationship is specified in SDLB, it quickly returns the host ID to which the packet should be forwarded; if no ID-host relationship is specified, it quickly returns a random host ID, and the *probability distribution of host IDs can be specified and adjustable* according to the host capacity.

Evaluation results show that SDLB is faster and uses less memory, than a widely-used load balancer design (hash table + consistent hashing) which is the core algorithm in Google's cloud load balancer Maglev [6]. SDLB is also adaptive to network dynamics.

The rest of this paper is organized as follows. We present the system design in Section 2. We show the evaluation results in Section 3. We discuss possible future work in Section 4 and present related work in Section 5. Finally we conclude this work in Section 6.

2 SYSTEM DESIGN

2.1 System overview

The key idea of using SDN for SDLB is as follows. Since the data plane of a load balancer is resource limited, we let the data plane of SDLB only include the packet processing component and is optimized for memory efficiency and processing speed. The construction and update components of SDLB are moved to an *LB controller*, which uses more memory to keep the data plane consistent to network changes. The data plane of SDLB (called *SDLB-DP*) is able to fit to fast memory such as cache to achieve better processing speed. Since network changes are much less frequent than the incoming packet rate, the LB controller can be implemented with relatively

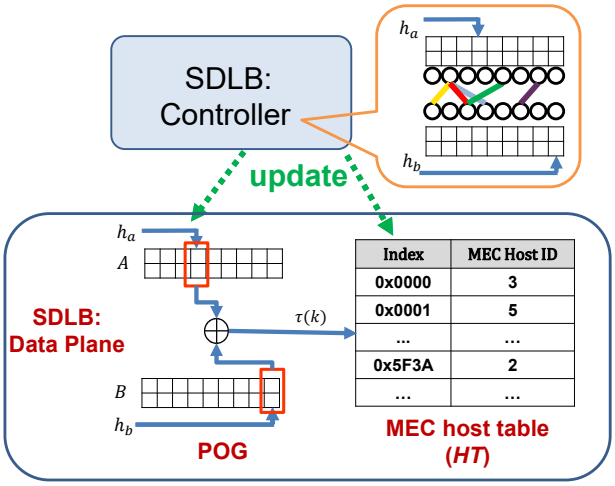

Figure 2: SDLB structure

slow memory such as RAM. Upon network dynamics, the controller computes the updated SDLB-DP and sends the modification to SDLB-DP similar to existing SDN APIs [19].

We consider the SDLB-DP as *a key-value query structure as well as a deterministic randomizer*. Every packet carries an ID and is processed by the SDLB-DP. 1) For a stateful packet, SDLB-DP takes the packet ID as the key and returns a MEC host ID as the value. The MEC host specified by the value should be the host that includes the state of this packet. Here the SDLB-DP servers as a key-value query structure. 2) For a stateless packet, SDLB-DP returns a random MEC host ID depending on the packet ID and the probability distribution of this random selection should be adjustable. Here it serves as a deterministic randomizer. In the following. we introduce the data structure that can achieve these two functions simultaneously.

2.2 Data structure and algorithms

As an overview, SDLB-DP is in a two-level structure as shown in Fig. 2. The first level provides a many-to-one mapping from a packet ID k to an l-bit value $\tau(k)$, the data structure is called *POG*, and the second level is a many-to-one mapping from $\tau(k)$ to a MEC host h, called the MEC host table (HT).

2.2.1 First level: POG. We present the data structure POG , a data structure that applies the theoretical results of MWHC perfect hashing [16] in the SDN scenario.

The l-POG data structure maintains a mapping function τ. Let S be a set of keys (in our application, IDs of all stateful packets). Let U be the universal set (all possible valid packet IDs). $S \subset U$. For any key $k \in U$, a *query* on the POG returns an l-bit integer $\tau(k)$. POG serves as a key-value store for keys $k \in S$. It supports the following operations.

- `add(`k`,`$\mathtt{r}$`)`: Insert k to S and specify $\tau(k) = \mathtt{r}$. This operation does not change the τ value of any other $k \in S$.
- `delete(`k`)`: Remove k from S; after this operation, the $\tau(k)$ value may change in the future.

The underlying mechanism of POG is that it maintains two arrays of l-bit integers, A and B. The length of these two arrays m_a and m_b satisfies $2.67n \le m_a + m_b < 4n$. POG also maintains two hash functions h_a, h_b to map keys to the corresponding indexes in A and B. The query result τ is computed by

$$\tau(k) = A[h_a(k)] \oplus B[h_b(k)].$$

Here $\oplus$ is the exclusive or operation. In order to determine the values of A and B, the POG control structure, running on the controller, maintains an *acyclic* bipartite graph $G = (U, V, E)$. Each edge in E represents a key. There is a edge $(u_i, v_j) \in E$ connecting nodes $u_i \in U$ and $v_j \in V$ if and only if there is a key $k \in S$ such that $h_a(k) = i$ and $h_b(k) = j$. We prove that the *expected* time to construct an POG control structure with $n = |S|$ names is $O(n)$ and the expected time to add/delete/alter a name is $O(1)$. The detailed data structure and algorithm design of POG can be found in [28].

2.2.2 Second level: MEC host table (HT). SDLB also maintains an array in the data plane as the MEC host table (HT). HT contains 2^l elements. $l = 16$ is sufficiently big for SDLB. After computing the value $\tau(k)$ from the POG, SDLB-DP returns the value stored in $HT[\tau(k)]$ as the MEC host ID. Since $0 \le \tau(k) < 2^{16}$, the table HT is very small and is able to fit into fast memory.

When SDLB-DP gets a packet with key k, it returns a particular value $HT[\tau(k)]$. This value is deterministic and hence SDLB guarantees to forward all packets with a particular packet ID to the same host.

2.2.3 Property as a deterministic randomizer. An important property of POG is that when $k \notin S$, $\tau(k)$ returns an arbitrary value. The property was considered a weakness of a key-value table. However, it is a perfect feature that can be used for a load balancer. We show that it is possible to create a POG such that the result is uniformly random.

Let $k' \in U$ be an arbitrary key randomly chosen from the key universe. $i' = h_a(k')$ and $j' = h_b(k')$ can be considered as uniform random values. Then, for any l-bit value $\mathtt{r}$, query on the POG returns $\mathtt{r}$ with probability

$$P_{\mathtt{r}} = \Pr[\tau(k) = \mathtt{r}] = \frac{1}{m_a m_b} \sum_{t=0}^{2^l-1} E_A(t) E_B(t \oplus \mathtt{r})$$

Here, $E_A(t)$ is the number of elements with value equal to t in array A, and $E_B(t \oplus \mathtt{r})$ is that of $t \oplus \mathtt{r}$ in array B.

Rebalance operation: One useful property of the l-POG is that although it is a memory-efficient data structure, there is a certain level of redundancy in the encoding of the query results. Consider a subset of keys $C = \{k_1, k_2, \cdots, k_t\} \subset S$, where their corresponding edges form a *connected component* in G. For all i and j values where $i = h_a(k)$, $j = h_b(k)$ for some $k \in C$, and an arbitrary l-bit integer x, execute this operation: Let $A[i] \leftarrow A[i] \oplus x$ and $B[j] \leftarrow B[j] \oplus x$. After such operation, the $\tau(k)$ value is *not* changed for any $k \in C$. This is to say that we are able to *modify the values in A and B, while not changing the $\tau(k)$ values* for any $k \in S$.

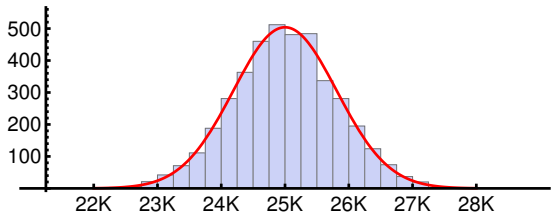

Figure 3: Histogram of $\tau(k)$ occurrence frequency for $100M$ random queries and 2^{12} possible values. Curve: normal distribution with parameters $\mu = 25K$ and $\sigma = 830$.

In addition, some $A[i]$ and $B[j]$ elements are not associated with any $k \in S$ (we call them isolated elements), and we are able to modify them without changing $\tau(k)$ values for $k \in S$.

Hence, we are able to adjust the distribution of $\tau(k')$ for arbitrary queries (in our application, k' is the ID of a stateless packet). We are able to *rebalance* the P_r ratios by assigning all isolated elements in A and B as some random values, and execute the above operation for all connected components in G with arbitrary random x values. We call this a *rebalance* operation on the POG. The actual distribution of P_r values depends on what sort of random number is used in such operation. We can use different random number generators to get different $\tau(k')$ distributions. For example, when uniform random l-bit integers are used in such operation, it generates a distribution as if $\tau(k')$ is a uniform random l-bit integer. This is shown in Figure 3, where we compute $\tau(k')$ values for $100M$ random keys on a 12-POG that is "rebalanced" using random 12-bit integers. The curve shows a normal distribution with parameters $\mu = 25$K and $\sigma = 830$. The very small standard deviation σ suggests $\tau(k')$ can be treated as a random number, although the σ value is larger than the theoretical value.

2.3 SDLB update

We may deal with multiple types of network dynamics including: **1)** Flow addition and leave of the network. **2)** MEC host capacity change. MEC hosts are heterogeneous and may be resource-limited. Hence the capacity of a MEC host to serve mobile packets may change frequently. **3)** MEC host join and leave. MEC hosts may be built on existing devices or terminals deployed for other purposes [26, 27]. Hence their churn rate is much higher than that of cloud servers. **4)** State migration. When a MEC host runs out of capacity or intends to leave the network, the state stored on it need to be migrated to other hosts.

In case 1), if there is arrival of new flows, SDLB-DP does not need to be updated to include the new flow information to guide the processing of the future packets of these flows. It is because the result $\tau(k)$ is deterministic for a same packet ID. However, when a MEC host creates state for a flow/application/device, it will notify the SDLB controller. The controller will maintain the packet ID to MEC host relationship according to the state.

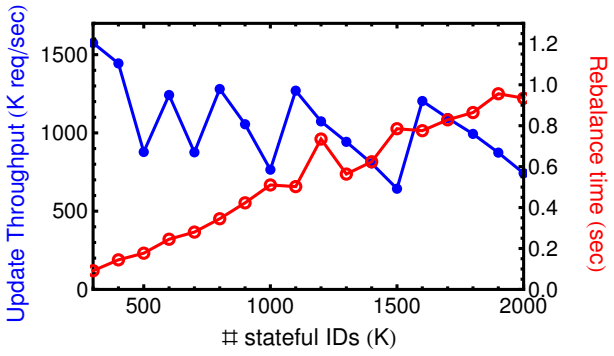

Figure 4: Update efficiency of SDLB

We start to introduce update mechanisms by introducing the "merge" operation on the SDLB-DP. We assume 2^l is far greater than the number of MEC hosts. Consider any two packets k_1 and k_2 with $\tau(k_1) \neq \tau(k_2)$ but they are assigned to the same host, i.e., $HT[\tau(k_1)] = HT[\tau(k_2)]$. We can execute an update on the POG so that $\tau'(k_2) \leftarrow \tau(k_1)$. This operation "releases" the element $HT[\tau(k_2)]$. After such operation, changing the value of $HT[\tau(k_2)]$ will not affect any stateful packets. Hence, $HT[\tau(k_2)]$ can be used as a buffer for future SDLB-DP updates.

Case 2) can be treated by modifying the HT table. To arrange more packets to a host, SDLB assigns more elements in HT to the corresponding host ID. This can be achieved by changing the values of the "released" elements in HT and it does not affect any stateful packets.

For case 3), in addition to assign the capacities, when a host leave, it may requests to migrate packets with key k to another host, which results in case 4). To modify the host assignment of a particular packet with key k, SDLB modifies the POG value so that $\tau'(k)$ points to a "released" element in the HT and assigns $HT[\tau'(k)]$ to the new host ID.

3 EVALUATION

In this section, we evaluate the performance of the SDLB. We compare SDLB with a widely-adopted approach for network load balancer. That approach first perform a lookup in a hash table that maintains the ID to host mapping for stateful packets. If the lookup fails then the load balancer knows the packet is stateless and uses consistent hashing to assign a host for it. We measure the memory size, update speed, and data plane throughput of SDLB and the compared approach (referred as HashTable in the following). In our implementation, we use HashMap that uses a self-balancing binary search tree. In our experiments we assume the packet IDs are 64-bit values while the MEC Host IDs are 16-bit integers.

3.1 Memory efficiency

We compute the memory space used by SDLB-DP and the HashTable approach in Table 1. We show three typical types of packet IDs: (1) HashValue, in which the load balancer computes a hash value as the digest of the metadata (e.g,

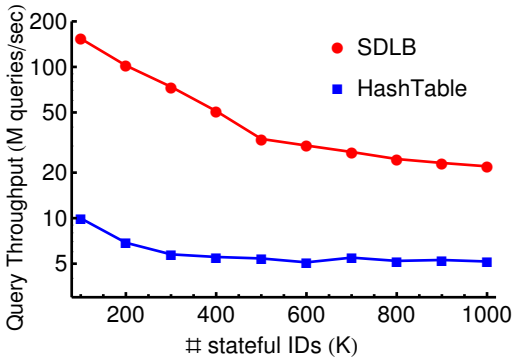

Figure 5: Data plane throughput vs. Number of stateful IDs.

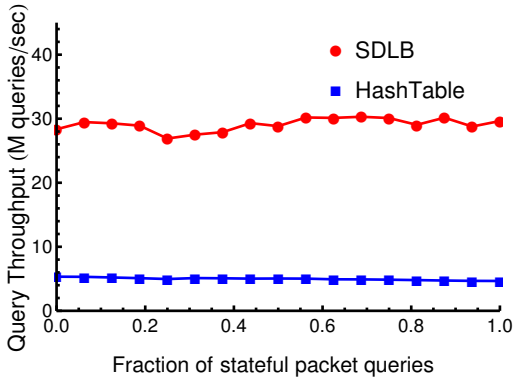

Figure 6: Data plane throughput vs. Fraction of stateful ID queries.

URL). (2) 5-tuple, it refers to the src/dst IP addresses and port numbers, and protocol number of a IP packet. (3) OpenFlow matching fields, which is a 356-bit value. SDLB-DP uses much smaller memory space than HashTable does. Note that the POG data structure does not maintain a copy of the keys in its memory, while the HashTable must maintain a copy of the keys to identify hash conflicts. Hence, the memory space of SDLB-DP grows approximately in proportion to the number of stateful packet IDs. As a comparison, the space size of the HashTable grows with both the number of stateful IDs and the length of the keys.

Table 1: Memory size comparison

ID type	# stateful entries	SDLB	HashTable
HashValue (64b)	96K	640K	1.41M
5-tuple (104b)	256K	1.63M	5.25M
OpenFlow (356b)	1M	6.13M	72M

3.2 Update speed of SDLB

We measure the update speed of SDLB and show it in Figure 4. In each experiment, we randomly execute update operations on the SDLB controller. The numbers of stateful packet addition and deletion are set equal. The blue curve shows the throughput of SDLB updates. **SDLB is able to support about one million update requests per second.** We observe the update throughput varies against the number of stateful packet IDs. SDLB reaches higher throughput when the number of packets are close to values like 2^n or 3×2^n for some integer n. This is in consent with the properties of the POG data structure discussed in [28].

We also measure the efficiency of the "rebalance" operation discussed in Section 2.2.3. The red curve in Figure 4 shows that the time used to rebalance the SDLB grows linearly to the number of packet IDs. It takes less than one second to rebalance when there are 2M stateful IDs.

3.3 Data plane throughput

We compare the data plane throughput of SDLB and HashTable. In Fig. 5, we vary the number of stateful IDs maintained by SDLB and the HashTable and measure the query throughput of both approaches. Experimental result shows that SDLB-DP has a ¿10x higher throughput than the HashTable when the number of stateful IDs is less than 300K. It still reaches at least 4x better throughput when the number of stateful IDs grows. The throughput of both approaches decreases as there are more IDs. This is because the experiment is carried on a commodity desktop computer, and the size of both data structures grows linearly to the number of stateful IDs. With 1000K names, SDLB is able to fit into the L3 cache of the CPU. HashTable fails to fit into the cache for 500K names. In summary, SDLB is much faster than HashTable. We also measure the data plane throughput under different types of traffic. We fix the number of stateful IDs as 600K and vary the fraction of stateful queries. Fig 6 shows that such fraction does *not* affect the query throughput of SDLP.

4 DISCUSSION

Each MEC host needs to know its working load because of two reasons. 1) Each MEC host needs to provide provable and controllable resource/performance isolation between the original applications and mobile devices. 2) The SDLB controller needs the information to set the weight of the MEC host adaptively according to the available resources. The vanilla solution by analyzing resource consumptions for all MEC traffic through existing monitoring APIs (*e.g.* Intel Performance Counter Monitor API [25]) is expensive and not scalable. Meanwhile, empirical studies [12, 13, 20] have shown that the network traffic is dominated by a small fraction of *elephant flows*. We argue that tracking elephant flows on the MEC host is one cheap and efficient method for load estimation. Elephant flow detection can be achieved very efficiently. For instance, Myopia [12] leverages count-min sketch [4] to measure flow sizes for its provable tradeoff between space and accuracy of flow size estimation. If the size of a flow exceeds a threshold, an elephant flow is identified. Moreover, since

the set of elephant flows substantially overlap stateful flows, the elephant flow information can be further exploited for coarse-grained tasks such as state migration in the presence of overloading.

5 RELATED WORKS

MEC and Fog. The idea of shifting storage and computing from clouds to the edge of the network has been proposed by many similar concepts such as mobile edge computing (MEC) [9, 17], fog computing [3, 24, 26], edge computing [23] and mobile cloud computing (MCC) [7]. Fog and edge computing have interchangeable definitions, both of which allow heterogeneous devices on the path to the remote cloud to offer storage and computing resources. MEC on the other hand relies on servers owned by mobile providers (*e.g.* Cloudlet [5]) behind Radio Access Network (RAN). MCC focuses more on the federation of the cloud, proximate mobile computing entities and a plethora of mobile devices.

Software Load Balancer. HAProxy [8] and Linux Virtual Server (LVS) [15] are the mostly used open-source software load balancers. Since HAProxy operates on Layer 7, it is able to perform complicated tasks on traffic flows such as SSL authentication and traffic regulation. The proposed SDLB and LVS focus on Layer 4. Due to its simplicity, LVS is cheap and extremely fast. Meglev [6] has been used as Google's network load balancer since 2008, which is able to achieve line-rate throughput. Compared to Meglev, LVS is not as optimized in term of performance because it is designed for portability. Maglev relies on a server cluster hence cannot be applied to MEC/Fog. In contrast, SDLB is general-purpose and portable. Duet [22] is a hybrid software and hardware load balancer with all the benefits of software load balancers as well as enjoys low latency and high throughput. Duet relies on the recently provided APIs for fine-grained control over ECMP and tunneling functionality on commercial switches. It cannot be run at general-purpose platforms.

6 CONCLUSION

We present the design of a fast and dynamic software load balancer for MEC and Fog, called SDLB. SDLB is built on a new data structure named POG whose core algorithm is minimal perfect hashing. Experimental results show that SDLB is faster by 4x to 10x and uses less than 50% memory compared to existing solutions. In addition SDLB provides fast updates. We believe POG has the potential to become a fast and memory-efficient solution for software-based networking in future applications.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers. Xin Li and Chen Qian were supported by National Science Foundation Grants CNS-1701681 and CNS-1717948.

REFERENCES

[1] Djamal Belazzougui, Paolo Boldi, Rasmus Pagh, and Sebastiano Vigna. 2009. Monotone minimal perfect hashing: searching a sorted table with O (1) accesses. In *Proc. of ACM SODA*.

[2] Eugen Borcoci. 2016. Fog Computing, Mobile Edge Computing, Cloudlets - which one?. In *SoftNet Conference*.

[3] Mung Chiang and Tao Zhang. 2016. Fog and IoT: An Overview of Research Opportunities. *IEEE Internet of Things Journal* (2016).

[4] G. Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[5] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. 2010. MAUI: making smartphones last longer with code offload. In *Proc. of ACM MobiSys*.

[6] Diana E Eisenbud *et al.* 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *Proc. of USENIX NSDI*.

[7] H. T. Dinh, C. Lee, D. Niyato, and P. Wang. 2013. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing* 13, 18 (2013).

[8] HAProxy. 2017. HAProxy - the Reliable, High Performance TCP/HTTP Load Balancer. (2017). http://www.haproxy.org/

[9] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile Edge Computing A key technology towards 5G. *European Telecommunications Standards Institute White Paper* (2015).

[10] Citrix Systems Inc. 1999-2017. Netscaler. (1999-2017). https://www.citrix.com/

[11] Loadbalancer.org Inc. 2017. Loadbalancer. (2017). https://www.loadbalancer.org/

[12] Xin Li and Chen Qian. 2015. Low-Complexity Multi-Resource Packet Scheduling for Network Function Virtualization. In *Proc. of IEEE INFOCOM*.

[13] Xin Li and Chen Qian. 2016. An NFV Orchestration Framework for Interference-free Policy Enforcement. In *Proceedings of IEEE ICDCS*.

[14] Lixia Zhang *et al.* 2010. Named data networking (ndn) project. *NDN Tech. Rep.* (2010).

[15] LVS. 2017. the Linux Virtual Server project. (2017). http://www.linuxvirtualserver.org/

[16] B. S. Majewski, NC Wormald, G Havas, and ZJ Czech. 1996. A Family of Perfect Hashing Methods. *Comput. J.* 39, 6 (jun 1996), 547–554.

[17] Milan Patel *et al.* 2014. Mobile-edge computing introductory technical white paper. *Mobile-edge Computing industry initiative* (2014).

[18] Jennifer Rexford Nick Feamster and Ellen Zegura. 2013. The road to SDN: An intellectual history of programmable networks. *ACM Queue* (2013).

[19] Nick McKeown *et al.* 2008. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM CCR* 38, 2 (mar 2008), 69–74.

[20] I A Rai, G Urvoy-Keller, and E W Biersack. 2003. Analysis of LAS scheduling for job size distributions with high variance. In *Proc. of ACM SIGMETRICS*.

[21] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. 2012. MobilityFirst: A Robust and Trustworthy MobilityCentric Architecture for the Future Internet. *Mob. Comput. Commun. Rev.* (2012).

[22] Rohan Gandhi *et al.* 2014. Duet: Cloud scale load balancing with hardware and software. *Proc. of ACM SIGCOMM* (2014).

[23] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. 2016. Edge computing: Vision and challenges. *IEEE Internet of Things Journal* 3, 5 (2016).

[24] Luis M. Vaquero and Luis Rodero-Merino. 2014. Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM CCR* (2014).

[25] Thomas Willhalm, Roman Dementiev, and Patrick Fay. 2017. Intel Performance Counter Monitor. (2017). https://software.intel.com/en-us/articles/intel-performance-counter-monitor

[26] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. 2015. Fog Computing: Platform and Applications. In *Proc. of IEEE HotWeb*.

[27] Shanhe Yi, Zhengrui Qin, and Qun Li. 2015. Security and Privacy Issues of Fog Computing: A Survey. In *Proc. of WASA*.

[28] Ye Yu, Djamal Belazzougui, Chen Qian, and Qin Zhang. 2017. A Concise Forwarding Information Base for Scalable and Ultra-Fast Name Lookups. In *Proc. of IEEE ICNP*.

[29] Ye Yu, Chen Qian, and Xin Li. 2014. Distributed Collaborative Monitoring in Software Defined Networks. In *Proc. of ACM HotSDN*.