Stateful Layer-4 Load Balancing in Switching ASICs

Jeongkeun Lee
Barefoot Networks

Rui Miao
University of Southern California

Changhoon Kim
Barefoot Networks

Minlan Yu
Yale University

Hongyi Zeng
Facebook

CCS CONCEPTS

• **Networks** → **Programmable networks**; *Network management*; *Data center networks*;

KEYWORDS

Load balancing; Programmable switches

ACM Reference format:
Jeongkeun Lee, Rui Miao, Changhoon Kim, Minlan Yu, and Hongyi Zeng. 2017. Stateful Layer-4 Load Balancing in Switching ASICs. In *Proceedings of SIGCOMM Posters and Demos '17, Los Angeles, CA, USA, August 22–24, 2017,* 3 pages.
DOI:10.1145/3123878.3132012

1 INTRODUCTION

In this demo, we show that a large number of software based load balancer servers can be replaced by a single modern switching ASIC, potentially reducing the cost of load balancing by two orders of magnitude. Today, large data centers typically employ hundreds or thousands of servers (or around 4% of their data center compute resources [1]) to load-balance incoming traffic over application servers. These software load balancers (SLBs) map packets destined to a service (with a virtual IP address, or VIP), to a pool of servers tasked with providing the service (with multiple direct IP addresses, or DIPs) [1, 4]. An SLB is stateful; *it must always map a connection to the same server, even if the pool of servers changes and/or if the load is spread differently across the pool.* This property is called **per-connection consistency** or **PCC**. The challenge is that the load balancer must keep track of millions of connections simultaneously.

Until recently, it was not possible to implement a load balancer with PCC in a commercial off-the-shelf (COTS) switching ASIC, because high-performance switching ASICs typically cannot maintain per-connection states with PCC guarantee. Newer switching ASICs provide resources and primitives to enable PCC at a large scale. We present a system, called SilkRoad, which is defined in a 400 line P4 program, and when compiled to a state-of-the-art switching ASIC, it can load-balance millions of connection simultaneously at line rate. Our conference paper [2] details the solution design and evaluates its scale and cost by running trace-driven simulations on the data collected from a large web service provider. The paper also shows the feasibility of storing millions of connection states into a switching ASIC. On top of that, the present two-page abstract shows the full cycle of connection and DIP pool management, explaining the co-design between switch data plane (ASIC) and control plane (software) to enable PCC in switching ASICs.

SIGCOMM Posters and Demos '17, August 22–24, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5057-0/17/08...$15.00
DOI:10.1145/3123878.3132012

2 SILKROAD DESIGN AND OPERATION

L4 load balancing is basically a mapping function from a connection (i.e., the source and VIP IP addresses, the protocol ID, and the port numbers) to a DIP; SilkRoad needs a table that selects a DIP when it sees the first packet of a new connection. A DIP pool is managed for each VIP (each service); we maintain the VIP-to-DIPPool mappings in VIPTable. To provide PCC, another table is needed to remember the selected DIP for each new connection so that the following packets of the connection will be consistently mapped to the same DIP, even when the DIP pool changes or the traffic load (new connections) is spread differently across the pool. ConnTable is introduced to remember the per-connection mapping states.

ConnTable is placed before VIPTable. The first packet of a new connection will miss ConnTable and then it must immediately get a DIP assigned by VIPTable and be forwarded to the selected DIP by the switching ASIC. Thus, SilkRoad runs a hash over the connection 5-tuple to choose a DIP out of the current DIP pool, the current set of servers backing up the service represented by the VIP.

To guarantee PCC, the connection-to-DIP mapping entry must be quickly programmed into ConnTable so that the subsequent packets of the connection will match the entry in ConnTable and get forwarded to the same DIP with the first packet. However, in high-speed switching ASICs, inserting a new entry into an exact-match table typically requires a complex cuckoo-hashing algorithm running on the switch management CPU [2, 3]. A new connection arrival event must be *learned* by the CPU, which then programs a new entry into ConnTable to store the connection-to-DIP mapping in the switching ASIC. Thus, LearnTable is introduced in SilkRoad to trigger connection learning to the CPU when a new connection arrives and gets a DIP from VIPTable.

Similar to L2 MAC learning, this L4 connection learning through CPU can take about one millisecond.

Figure 1 shows the overall architecture of SilkRoad, depicting the control flow between the various tables. The figure shows a couple of additional indirections in the mapping from connection to DIP and two more tables: DIPPoolTable and TransitTable. This is to tackle two major design challenges: 1) scaling ConnTable and 2) guaranteeing PCC under DIP pool changes.

ConnTable scaling: we observed up to ten million connections handled by a top-of-rack (ToR) switch in the data center of the large web service provider [2]. To fit millions of connection states within tens of MBs of switch SRAM, we reduce the size of a connection entry of ConnTable: match field and action data. We propose to store a small hash of a connection 5-tuple rather than the full 5-tuple to reduce the match field size, while our design mitigates the impact of false positives introduced by the hash. For example, 37 bytes of a 5-tuple of an IPv6 connection can be reduced down to a 16-bit hash digest.

To reduce the action data bits of a connection entry, we store a DIP pool *version* rather than the actual DIPs and intelligently reuse version numbers across a series of DIP pool updates. From the large web service provider data, we observed that a 6-bit version number is big enough to handle most DIP pool update scenarios [2]. We found that most inbound connections are short-lived; each DIP pool and its version do not need to last for long. Using a 6-bit version field reduces the action data size to 1/24 in case the DIPs are IPv6 (16B IP address + 2B TCP port).

Since we introduce another level of indirection (connection to *version* to DIP pool), we maintain the version-to-pool mappings in a new table called DIPPoolTable, which maintains the active DIP pools for each VIP.

PCC under DIP pool changes: the connection learning and entry insertion through CPU can take around one millisecond, which is long enough that a second packet of the connection can hit the load-balancing switch in modern datacenters with sub-ms RTT. If VIPTable updates its DIP pool version in-between the first and second packet arrivals, the second packet can be mapped to a new version, and thus a different DIP, violating PCC. To solve this, we introduce TransitTable, which is a simple bloom filter, to remember the set of connections that should be mapped to an old DIP pool version when VIPTable updates its DIP pool version. We minimize the size of the bloom filter using a 3-step update process [2].

Our co-design of the switch data plane (P4 program for the ASIC) and the control plane (software running on the CPU) ensures that every recently started connection is either programmed in ConnTable or remembered by TransitTable, thus protected from a DIP pool version change in VIPTable. Right after each version change, all the packets that miss ConnTable retrieve both old and new versions from VIPTable and then are checked by TransitTable to see if the packets hit the bloom filter. If hit, they use the old version; if miss they use the new version.

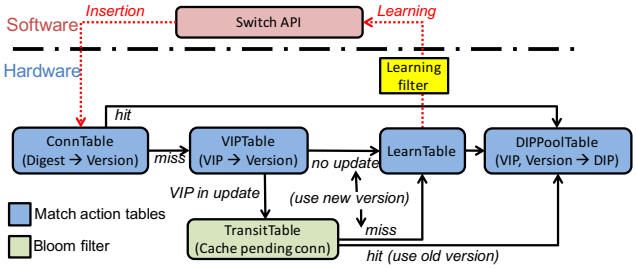

Figure 1: **System architecture.**

Connection and version management: When a DIP pool is updated, we create a new DIP pool by applying the change to a copy of the original DIP pool. We then assign a new version to the new pool and program VIPTable to map new incoming connections to the newest DIP pool version. Once a DIP pool is created and has active connections that still use it, the DIP pool is never changed in order to provide consistent hashing to the active connections. A connection 'uses' a DIP pool if the connection arrives when the pool was the newest, and thus VIPTable maps the connection to the pool.

A DIP pool is destroyed when all the connections that use it are timed-out and deleted from ConnTable. We employ idle timeout support in switching ASICs: each connection entry in ConnTable is associated with a tiny aging counter, which is reset by every incoming packet matching the entry. The ASIC periodically increments the counters and detects counter overflows, upon which entry timeout events are notified to the CPU.

When the pool is destroyed, the version of the pool is also released and returned to a ring buffer so it can be reassigned to a newly created DIP pool. The switch software tracks the connection-to-pool mappings and manages DIP pool creation/deletion and as well as the ring buffer that stores available version numbers. Our full paper [2] discusses a few ways to reduce the number of DIP pools (hence versions) that a switch needs to maintain at the same time.

3 EVALUATION

We built our prototype by adding about 400 lines of code into the P4 reference switch [2]. We have also implemented a control plane in switch software that handles new connection arrival events and connection timeout events. The software runs the cuckoo hash algorithm to insert or delete connection entries in ConnTable. In addition, the control plane performs 3-step PCC update for DIP pool updates. The connection handler and DIP pool manager are written in about 1000 lines of C code.

For the system evaluation, we create tens of thousand TCP connections starting and ending every second and observe them load-balanced to multiple DIP servers at line-rate. DIP pool updates happening in parallel to the connection arrivals do not cause connection remapping.

REFERENCES

[1] Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith, Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney, Wentao Shang, and Jinnah Dylan Hosein. 2016. Maglev: A Fast and Reliable Software Network Load Balancer. In *NSDI*.

[2] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *ACM SIGCOMM'17*.

[3] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* (2004).

[4] Parveen Patel, Deepak Bansal, Lihua Yuan, Ashwin Murthy, Albert Greenberg, David A Maltz, Randy Kern, Hemant Kumar, Marios Zikos, and Hongyu Wu. 2013. Ananta: Cloud scale load balancing. In *ACM SIGCOMM Computer Communication Review*.