LBSwitch: Layer 4 Load Balancing in P4 switch

JK Lee, James Zeng

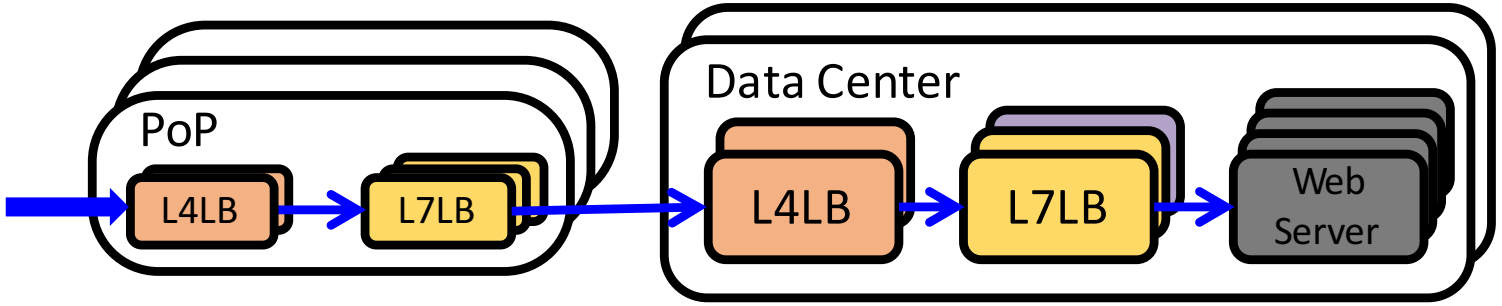
Co-work with: Rui Miao, Changhoon Kim, Minlan Yu

Agenda

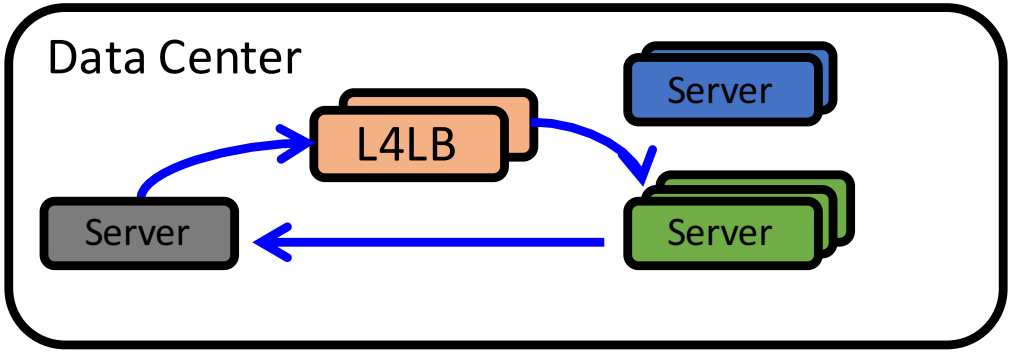
- Load Balancing at Facebook
- Insights from Facebook data
- LBSwitch: cloud-scale load balancing on switches
- Evaluations on Facebook data

Load balancing at Facebook

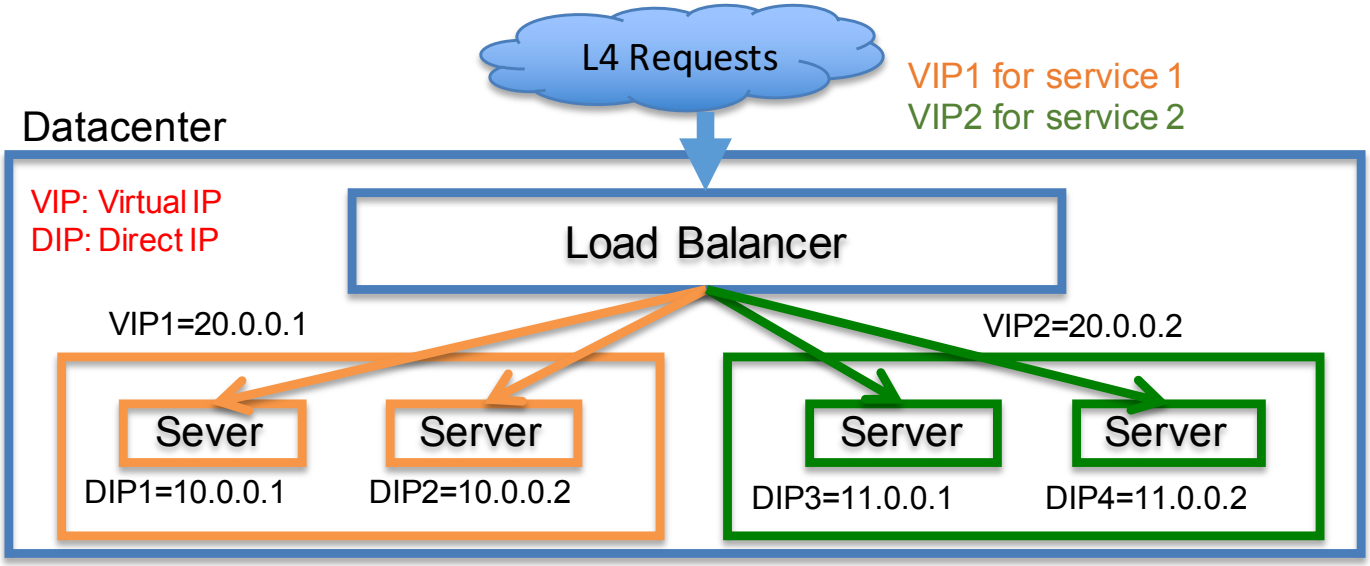
North-South traffic



East-West traffic



L4 LB Key Functions



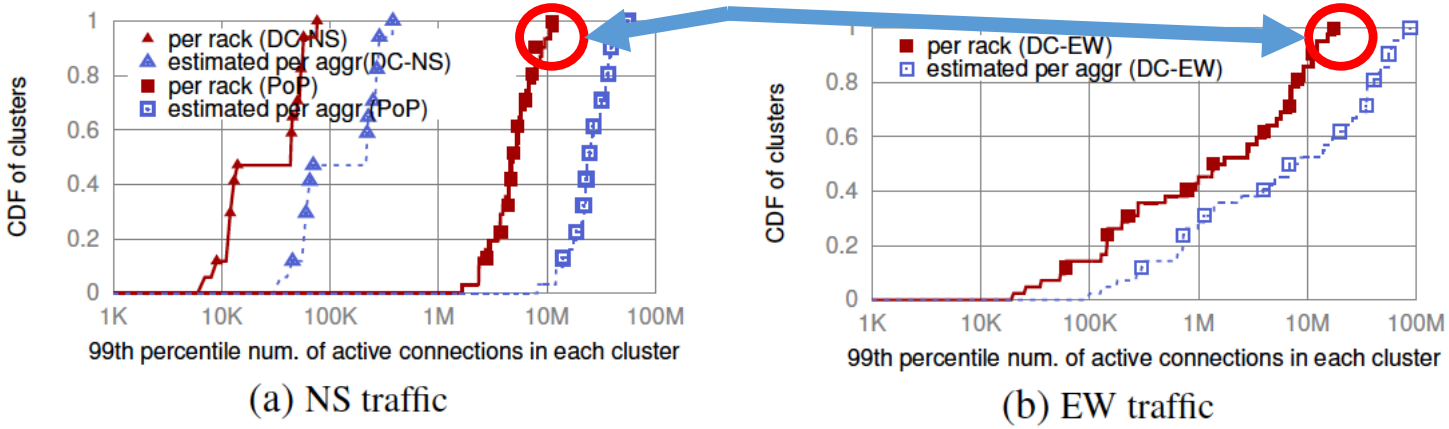
- VIP: Client-facing Virtual IP, DIP: Server IP
- Maintain VIP → DIP-pool mappings
- Given a VIP, choose an appropriate DIP from the pool, and forward packets to the DIP
- A DIP pool can change over time

Agenda

- Load balancing at Facebook
- **Insights from Facebook data**
  - **Many active connections**
  - **Frequent DIP pool updates**
  - **Frequent new connection arrivals**
- LBSwitch: cloud-scale load balancing on switches
- Evaluations on Facebook data

Measurement Study: Many Active Connections

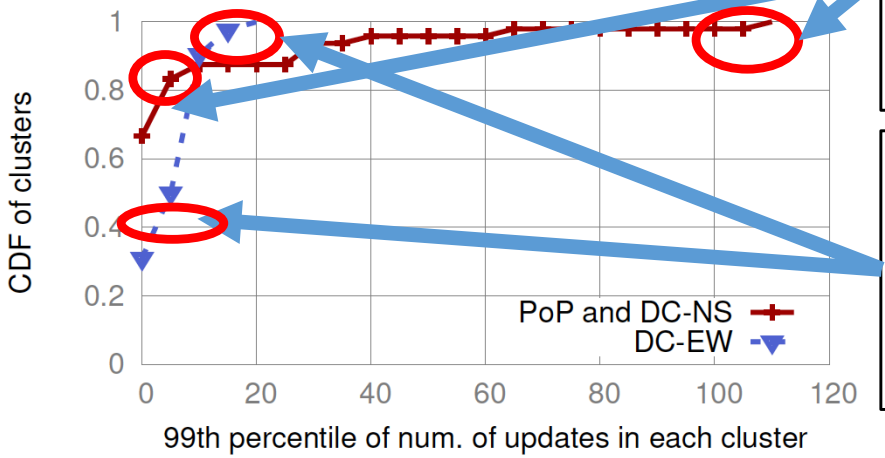
Peak clusters have 11-17M connections per rack



Implication: Each rack needs to handle 10+M connections

Frequent DIP pool updates

- Bursty updates in NS traffic, frequent updates in EW traffic
  - NS: high stability and shared DIPs
  - EW: high service dynamics



Bursty updates in NS traffic
66.7% clusters have no updates
Peak at 107 updates/min

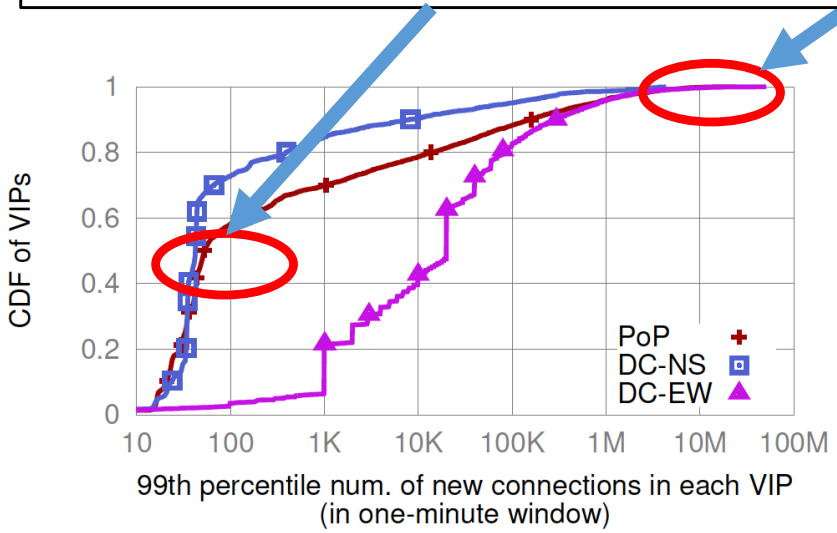
Frequent updates in EW traffic
5 updates/min in the median
16 updates/min in the peak

Implication: Need to ensure consistent VIP-DIP mappings during frequent DIP pool updates

Frequent new connection arrival

Median at 50 connections per min

Peak at 4.4-51M per min



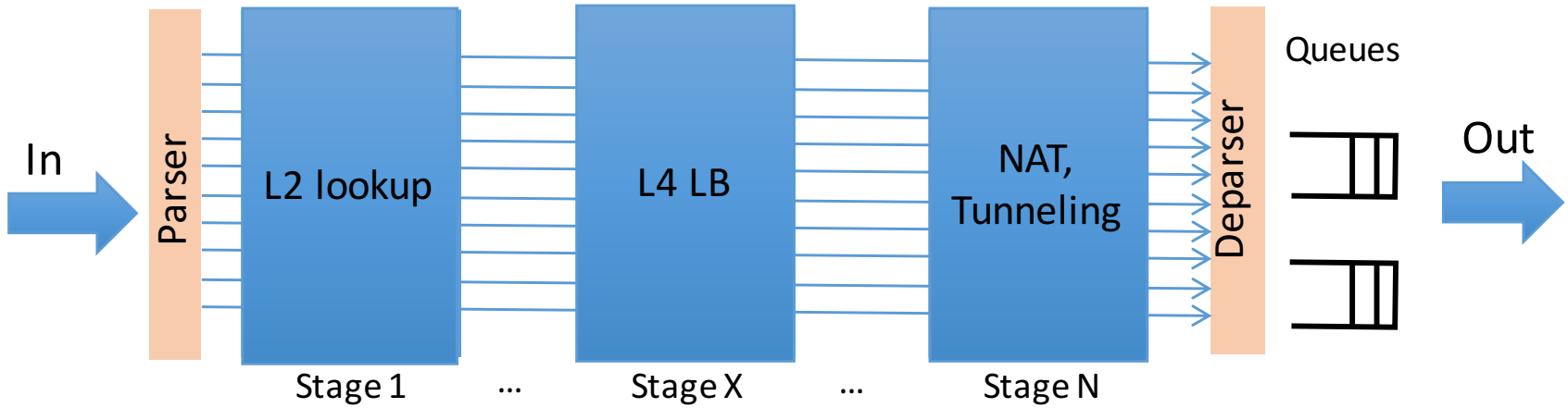
*VIP: Virtual IP

Implication: Need to handle consistent arrival of new connections

Today's L4 Load-Balancing

- Special HW middlebox
  - Hard to scale, rigid placement
- SW data plane in x86 servers
  - Many servers are needed
  - Variable latency
  - Hard to isolate performance between services, tenants

Key Idea: P4 Switch as Cloud-Scale Load-Balancer



- Benefits
  - High throughput (Tbps, Gpps), zero-latency, ubiquitous
  - Predictable performance even under availability attacks
- Challenges
  - Don't break existing connections during DIP pool update
  - Maintain millions of connection states in switch SRAM

DIP Pool Update Scenario

DIP Pool Version 1

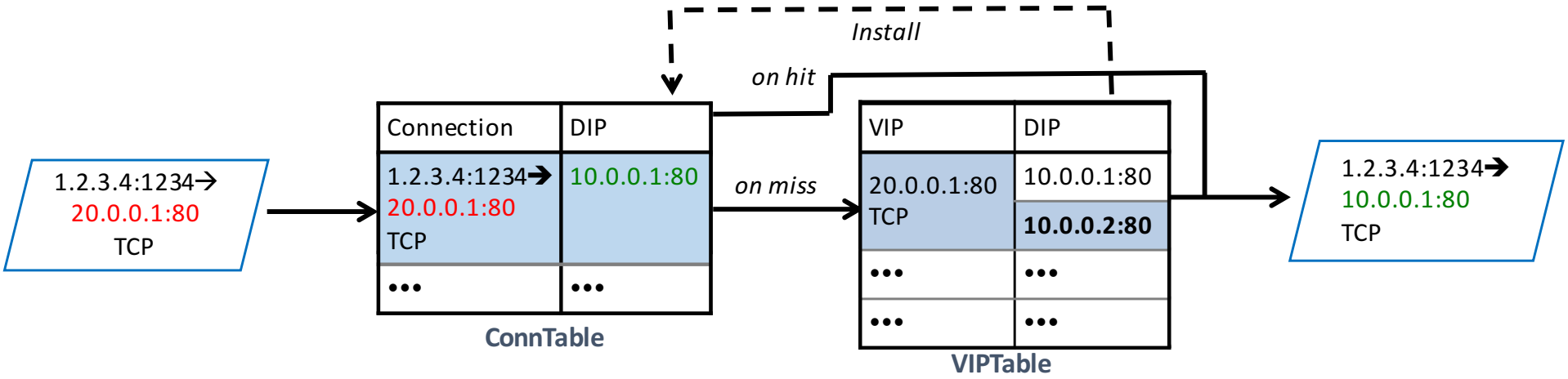
VIP	DIPs
20.0.0.1:80	10.0.0.1:80 (DIP1)

DIP Pool Version 2

VIP	DIPs
20.0.0.1:80	10.0.0.1:80 (DIP1) **10.0.0.2:80 (DIP2)**

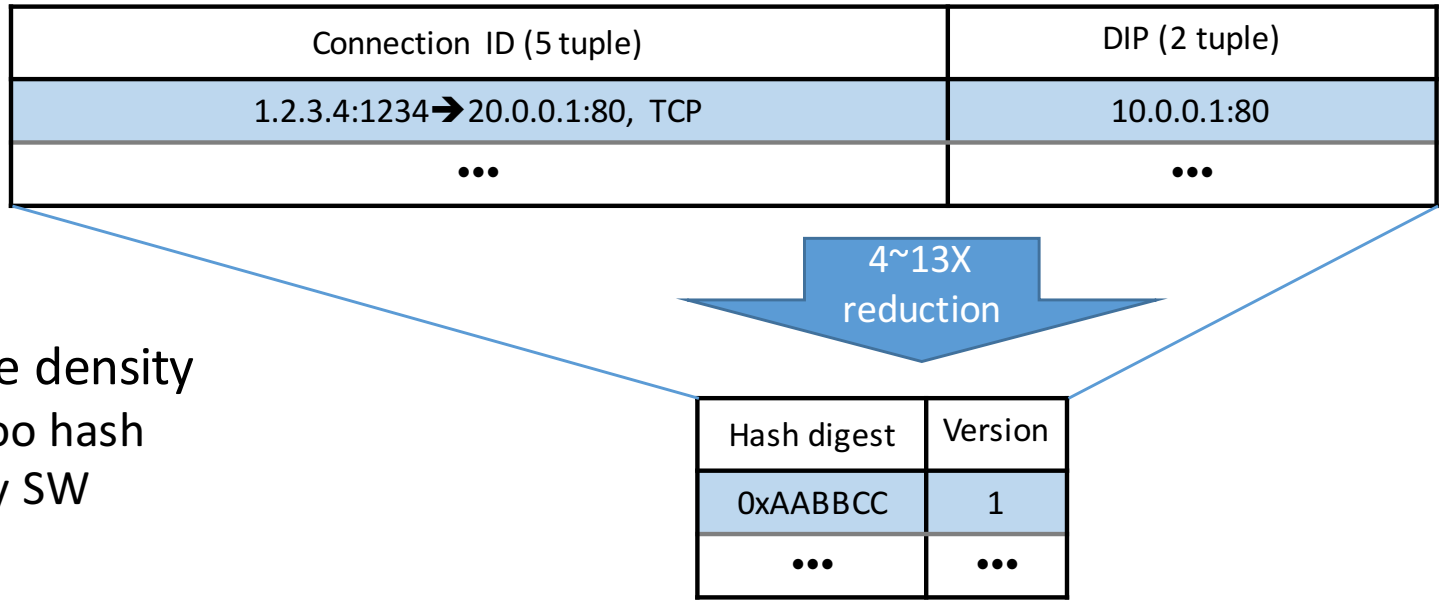
DIP2 addition

- Existing connections to DIP1 shouldn't be affected by DIP2 addition



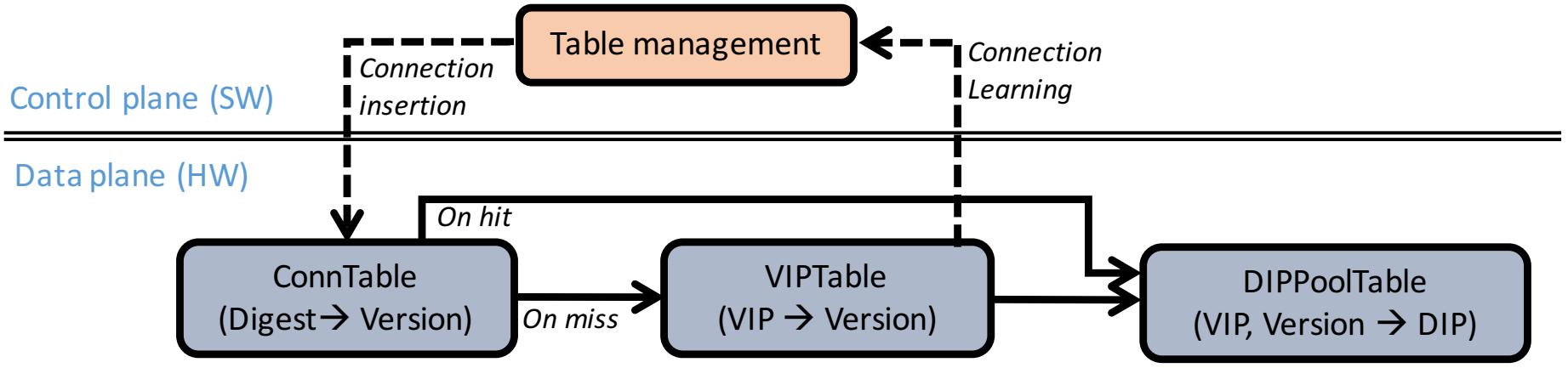
Scale to Millions of Connections

- Reduce connection entry size
  - Store hash digest instead of 5 tuple
  - Store DIP-pool version instead of DIP



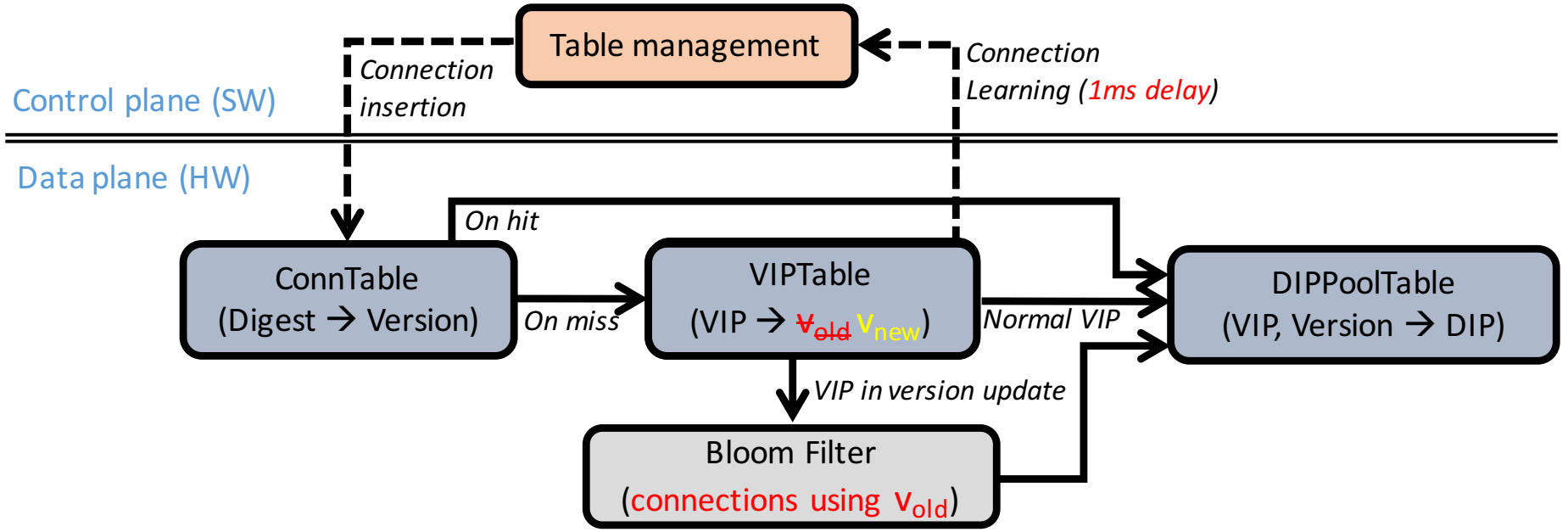
- Achieve high table density
  - Multi-stage cuckoo hash
  - Table insertion by SW

LBSwitch Design



- **Data plane**: connection tracking, version mapping, DIP selection at line-rate
- **Control plane**: connection learning & insertion, DIP pool updates
- No connection stalled by SW (no slow path)

Protect Connections during DIP-Pool Update



- Connection learning & insertion by CPU is slow
- 2nd packet of the new connection may arrive before ConnTable insertion
- 100s of connection arrivals in 1ms (per VIP), vulnerable to version update: v_{old} to v_{new}
- Solution: use bloom filter to cache connections recently mapped to v_{old}

P4 Prototype

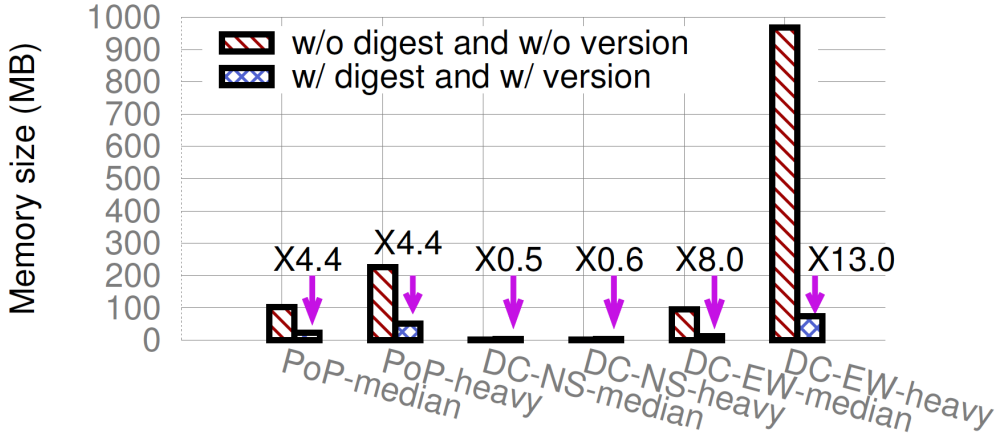
- Data plane
  - ~300 lines of P4 code into switch.p4
  - *register_read/write* primitives to implement bloom filter

- Control plane
  - ~800 lines of C code in switch_api

```
control process_l4l {
    apply(l4l_conn_table) {
        on_miss {
            apply(l4l_vip_table);
            if (l4l_metadata.update_status == USE_CACHE
                    and l4l_metadata.cache_status == CACHED) {
                apply(l4l_use_old_version);
            }
            else if (l4l_metadata.vip_traffic == TRUE) {
                apply(l4l_learn_conn);
            }
        }
    }
    if (l4l_metadata.vip_traffic == TRUE) {
        apply(l4l_dippool_table);
    }
}
```

Evaluations

- Re-play Facebook traces: connection & DIP pool update events
- Scale
  - Up to 13X reduction of ConnTable (Bigger reduction w/ IPv6)
  - 80MB SRAM for 17M connections



- Protect existing connections during DIP pool updates
  - w/o caching: up to 1400 connections break per VIP, per update
  - w/ caching: zero connection break
  - Few KB SRAM for bloom filter caching

Conclusion & Next Steps

- Cloud-scale L4 LB is feasible in P4 switches
- Plan to open-source P4 and switch_api codes
- Deployment scenarios
  - Hybrid deployment of LBSwitch (L4) + SLBs (L7)
    - P4 switch as a building block of decomposed VNF
  - Server health monitoring in LBSwitch data plane
- Support OpenStack LBaaS (Load-Balancing-as-a-Service)
  - Change deployment model: VM-centric to switch-centric
  - Tenant isolation: management and performance

Thank you

Questions

- Provide consistency across pipelines, under routing/ECMP changes?
  - Yes, by keeping hash function consistent across pipelines
- Can quickly react to server failures while guaranteeing consistency?
  - Yes, by resilient hashing in HW
- How to manage and update VIPs, DIP-pools?
  - Plan to support OpenStack LBaaS
  - L4 only, not L7