Sidecar: Building Programmable Datacenter Networks without Programmable Switches

Alan Shieh
Cornell University

Srikanth Kandula
Microsoft Research

Emin Gün Sirer
Cornell University
Programming the datacenter network

- Many new problems in datacenter networks can benefit from customization on data path
- But few datacenter networks are programmable!

**Program commodity hardware**
- e.g., fast path on switch, NIC, I/O coprocessor on PC, ...
- Too much to chew off

**Adopt software-based switches**
- e.g., Click, RouteBricks, PacketShader
- Uphill battle for deployment
**Goal:** Enable operator to install new packet processing code

- Provides better visibility:
  Can run custom processing everywhere, for fraction (1-10%) of packets

- Works in existing datacenters with minor upgrades
  - Rely only on existing commodity devices
  - Use only widely available functionality

- Scales to data rates of current networks

- Similar management model and comparable TCB size
Existing platform:

**Commodity Switch**

Most switches share similar architecture:

- Line-rate, fixed functionality fast path
  - *Limited flexibility: few rules, limited actions*
- Programmable slow path
  - Use rules to divert some packets to control processor
  - *Slow, closed platform*
Existing platform:

OpenFlow switches

- Enables innovation in the control plane
  - Abstracts away complexity of programming switch
  - Switches export open, common API for low-level control
- Similar datapath as commodity switches
- But, requires cooperation from equipment vendors, perhaps new switches
- Different target applications
  - Flow granularity, hooks for flow setup/teardown, hooks into forwarding tables
  - High frequency of changes to switch
Existing platform:
Modifying end hosts

- Can run arbitrary processing on general purpose processor
- Isolated execution domain (e.g., hypervisor, AppEngine)
- Centralized control over end host software (e.g. Autopilot)
Limitations:

**Modifying end hosts**

- Limited to adding functionality at edge
  - Can’t see into the middle or change packet processing
- Isolation is a concern
  - Legacy systems: Can’t generalize to non-virtualized settings
  - Compromised hosts: Can amplify network damage
- Direct I/O: Need to modify NIC to maintain control
SideCar architecture

- Offload custom in-network packet processing to **sidecars**
  - **Steer** packets to sidecars with config changes
- Push packet classification to **end hosts**
- Sidecars have better isolation than hypervisor
  - Attack surface limited to network protocols
  - Can **spot-check** end hosts to remove trust dependencies

- Unlike OpenFlow, no software or hardware changes on switch
Realizing SideCar in today’s datacenters

How to:
Transfer packets to sidecars without modifying switches?

Implement sidecars at reasonable performance and cost?
Transferring packets to sidecars

**Marked steering**
1. Hosts mark packets
2. Switches redirect all marked packets to sidecar
Transferring packets to sidecars

Sampled steering
Switches redirect packets uniformly at random
Configuring switches for steering

Input

- Forwarding table lookup

Ruleset lookup

Drop

Output

- Switching Fabric

Marked steering:
- L2: Match by VLAN
- L3: Match by IP headers or policy routing

Sampled steering

sFlow
Realizing SideCar in today’s datacenters

How to:
Transfer packets to sidecars without modifying switches?

Implement sidecars at reasonable performance and cost?
What to use as sidecars?

**Option 1:** Commodity servers
- Can process 3-5 Gb/s
  - Standard OS & network stack, previous generation hardware I/O architecture

**Option 2:** New platforms (e.g., RouteBricks, PacketShader)
- Can process 35-40 Gb/s
  - Exploit parallelism in new hardware I/O architectures
    - Point-to-point connections, multi-queue NIC, multi-core CPU/GPU
  - Can keep pace with network scaling?
    - More cores, faster busses
## Sidecar platform suitability

<table>
<thead>
<tr>
<th>Location</th>
<th>Ingress bandwidth</th>
<th>Added cost per server port</th>
<th>Processing rate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Commodity server</strong></td>
<td>ToR</td>
<td>80 Gb/s</td>
<td>$70</td>
</tr>
<tr>
<td><strong>RouteBricks / PacketShader</strong></td>
<td>Agg/Core</td>
<td>1560 Gb/s</td>
<td>$8</td>
</tr>
</tbody>
</table>

For typical topology

3% – 25% cost increase per server port
Proof of concept applications

SideCar:
- Works with legacy switches, endpoints, practices
- Provides network visibility
- Improves isolation

Next:
Initial ideas on how to fix some datacenter problems with SideCar
Application #1: Reachability Isolation

**Problem**
Cloud datacenters run a mix of mutually untrusting tenants
- Best to put each tenant on own (virtual) private network
- Need rich, granular policy support
Application #1: Reachability isolation

**Switch-only**

ToR switches support only small, inflexible policies (100s of rules)

- **Not enough precision**

**Host-only:** Host enforces policy with send/receive filters

- **DoS, needs hypervisor**

**SideCar:** Protection w/o hypervisors

- Sample packets,
  spot-check against policy
- Quarantine violators
Application #1: Reachability isolation

Switch-only
ToR switches support only small, inflexible policies
- Not enough precision

Host-only: Host enforces policy with send/receive filters
- DoS, needs hypervisor

SideCar: Protection w/o hypervisors
- Sample packets, spot-check against policy
- Quarantine violators
Application #1: Reachability isolation

Switch-only
ToR switches support only small, inflexible policies

- **Probabilistic bound (<100s of packets per source)** on # of leaked packets & size of DoS

- **SideCar**: Protection w/o hypervisors
  - Sample packets, spot-check against policy
  - Quarantine violators

- Leverages improved isolation & legacy support

Network Policy Enforcer
Application #2: Exploit better network feedback

**Example:** Enforce network sharing between applications
- TCP-like approach: Tunnel all traffic through one control loop
  - Throughput loss from probing for network state
- SideCar approach: Use explicit feedback
  - Better throughput from measuring network state directly

Feedback
- 50% in use
- 25% unused

Leverages network visibility
Application #3: Limit SAN usage

- Disks are constrained by IOPS (seek rate) and transfer rate
- **Goal**: Achieve performance isolation across users
- Can we do this without trusting the client (e.g., legacy or direct I/O) or modifying the target array?
Rule #1: Steer command packets to gather stats

Rule #2: Spot check that command packets are marked correctly

Not marked: Punish host
Related work

- Programmable network elements
  (NetFPGA, PacketShader, RouteBricks, OpenFlow)
- Hypervisor–network integration
  (End to the middle, Open vSwitch, CloudPolice)
SideCar provides a new way to program the datacenter network

- Architectural benefits
  - Works with legacy switches, endpoints, practices
  - Network visibility
  - Better isolation

- Feasible to process a modest subset of all packets
- Designed applications that benefit from SideCar
Questions?