ExoPlane: An Operating System for On-Rack Switch Resource Augmentation

Daehyeok Kim
Microsoft and UT Austin

Vyas Sekar
Carnegie Mellon University

Srinivasan Seshan
Two trends in in-network computing

**Increasing number of applications:** Academia & industry proposes many innovative applications [1]

**Increasing workload size:** Number of concurrent flows and traffic volume keep increasing (e.g., millions of concurrent flows) [2]

Is in-network computing ready for its prime time?

Problem: Serving concurrent stateful apps on a switch

Example scenario in a datacenter:
Four apps (VPN gateway, NAT, ACL, Monitor) on a switch

Root cause: Limited switch resources
E.g., 10s MB of SRAM ≪ Million flow entries
Possible solutions and limitations

- More switches
  - Expensive
  - Hard to extend

- A beefier switch

- Optimizing applications
  - Resource-efficient design (e.g., using sketches)
  - Not generally applicable
Case for on-rack switch resource augmentation

On-rack resource augmentation:
A switch + resource on external devices

- Programmable
- Larger resources
  E.g., a few GB of DRAM

Cost efficient
Easy to extend
What do we need for realizing it?

We need an OS \([AD'12]\)!

- Providing abstractions of resources
- Managing shared resource between apps
- Facilitating the sharing of resources at runtime

What should an “operating model” be?

A single place to process

Where to process?
Strawman model 1: App pinning

Pin an app to one device and process packets on that device

+ Low performance overhead
+ Low resource overhead

- Resource underutilization

Packet drops due to insufficient memory
Strawman model 2: Full disaggregation

An app running on multi-devices and processing a packet on multi-devices

- Flow 1
- Flow 2
- Flow 3

Switch data plane

- Key: 5-tuple Stateful FW
- Key: SrcIP Pkt Counter
- Key: dstIP Forward

External device data plane

- Key: 5-tuple Stateful FW
- Key: SrcIP Pkt Counter
- Key: dstIP Forward

+ High resource utilization
- High performance overhead
- High resource overhead
Candidate model: Packet pinning

An app running on multi-devices and processing a packet on a single device

Switch data plane
- Key: 5-tuple Stateful FW
- Key: SrcIP Pkt Counter
- Key: dstIP Forward

External device data plane
- Key: 5-tuple Stateful FW
- Key: SrcIP Pkt Counter
- Key: dstIP Forward

How to ensure that all necessary state is available on a device?
Our approach: Packet pinning + Union key-based flow management

Union key: a union of key types of application objects

Switch data plane

- Key: 5-tuple
  - Stateful FW
- Key: SrcIP
  - Pkt Counter
- Key: dstIP
  - Forward

External device data plane

- Key: 5-tuple
  - Stateful FW
- Key: SrcIP
  - Pkt Counter
- Key: dstIP
  - Forward

Key insight: Skewness of flow key distribution
E.g., 6% of flow keys takes 90% of total traffic

By placing popular keys on the switch, it can process most of the traffic while the remaining is processed at an external device.
ExoPlane design overview

- Device information
- Cross-app requirements
- Objective functions

Switch programs

Developer

Network operator

ExoPlane planner
Optimal resource allocation

Merged program

Infinite resource abstraction

ExoPlane runtime environment
“Packet pinning model”
Challenge 1: Correctness under workload changes

1. New flows arrive → Insert entries of the flow
2. Flow popularity changes → Insert (evict) entries of popular (unpopular) flows
Challenge 1: Correctness under workload changes

1. New flows arrive → Insert entries of the flow
2. Flow popularity changes → Insert (evict) entries of popular (unpopular) flows
Problem: Incorrect state eviction

Switch control plane

1. Key: 5-tuple Stateful FW
2. Key: SrlIP Pkt Counter
3. Key: dstIP Forward

Switch data plane

4. UKey: 5-tuple Flow manager
5. Key: 5-tuple Stateful FW

Flow 2

Order matters!

Entry deleted

Packet dropped!

Similar issue can happen for insertion!
Our solution: Two-phase state update

Flow 2

Phase 1

1. UKey: 5-tuple
Flow manager

2. waits for $T_{\text{flush}}$

Phase 2

3. Key: 5-tuple
Stateful FW

4. Switch data plane

5. Key: SrcIP
Pkt Counter

6. Key: dstIP
Forward

Switch control plane
Challenge 2: Synchronizing data plane-updatable states

Multiple copies of an object entry can be updated at different places

- Updated at a high rate
  - Buffer & sync does not work!

- Entries with the same SrcIP should be synchronized

Flow 1
Flow 2
Flow 3

UKey: 5-tuple Flow manager
Key: 5-tuple Stateful FW
Key: SrcIP Pkt Counter
Key: dstIP Forward

Stateful FW
Key: 5-tuple Stateful FW
Key: SrcIP Pkt Counter
Key: dstIP Forward

External device data plane
Bounded inconsistency via periodic synchronization

Observations on data plane-updatable state
- Approximate or statistical information
- Mergeable values

Our approach: bounded-inconsistency mode via periodic synchronization

Switch control plane

- Tracking remote changes ($\delta$)

Switch data plane

- Key: SrcIP
- Pkt Counter

- ① Snapshot

- ③ Merge $\delta$

External device control plane

- Tracking remote changes ($\delta$)

Exernal device data plane

- Key: SrcIP
- Pkt Counter

- ① Snapshot

- ③ Merge $\delta$

② Exchange

$<$Snapshot, Metadata$>$
Challenge 3: Meeting requirements across apps

App-specific requirements (e.g., affinity to the switch)

Developer

Network operator

– Cross-app requirements
– Objective functions

How to find an “optimal” resource allocation that satisfies all requirements?
Finding optimal resource allocation using ILP

Developers
- Switch program codes
- App-specific requirements

Network operator
- Device information
- Cross-app requirements
- Objective functions

Profiler
- Resource footprint
- Packet processing latency
- Compatibility matrix

Optimal resource allocation
Encode & solve resource allocation ILP

Objective:
- Min. Expected Latency

Subject to:
- Resource constraint
- Compatibility constraint
- Workload assignment

App merger

Loaded to the switch and external devices
Putting it all together

ExoPlane provides an **infinite resource abstraction** to applications

- Developers
- Network operator

ExoPlane planner
- Merged programs
- Optimal resource allocation using ILP
- Packet pinning operating model
- Two-phase state management
- Periodic state synchronization

ExoPlane runtime environment
Implementation and evaluation setup

- Profiler & merger based on open-source P4 compiler frontend
- Resource allocator using Gurobi

ExoPlane planner
Merged programs

Tofino-based programmable switch
4 x Netronome Agilio CX smart NICs

ExoPlane runtime environment

- Data plane: P4
- Control plane: Python/C++

Ensemble of four apps in two scenarios

Developers
Network operator
Does packet-pinning model work well?

Aggregate throughput (Gbps)

- VPN
- VPN+NAT
- VPN+NAT+ACL
- VPN+NAT+ACL+Monitor

- App pinning
- ExoPlane (Packet pinning)

Throughput rates:
- VPN: 43.8%
- VPN+NAT: 69.3%
- VPN+NAT+ACL: 45.3%
How does ExoPlane work under dynamic workload?

Throughput drops due to insufficient capacity at the NIC

Switch + a single external device

Switch + 4 x external devices
Limitations and future work

Supporting non-P4 programmable external devices

Supporting other types of resources on external devices

Enabling rapid runtime resource reallocation

What-if analysis of benefits from resource augmentation
Summary

Limited on-chip resources prevent concurrent stateful apps on programmable switches

ExoPlane provides OS abstractions for switch resource augmentation
- Packet pinning operating model
- Two-phase state management
- Periodic state synchronization
- Optimal resource allocation using ILP

Realizes resource augmentation with minimal performance and resource overhead
- Effectiveness of the packet pinning model
- Adapt to workload changes
- Low and predictable per-packet processing latency