Multitenancy for Fast and Programmable Networks in the Cloud

Tao Wang*, Hang Zhu*, Fabian Ruffy, Xin Jin, Anirudh Sivaraman, Dan Ports, and Aurojit Panda

NEW YORK UNIVERSITY, JOHNS HOPKINS UNIVERSITY, Microsoft Research

by: Group 5
Jiawei Yao, Jiayu Li, Xiaowan Guo

UNIVERSITY of WASHINGTON

1

Outline

- Paper Overview
- Background
- System Overview
- key contributions
- Related Work
- Evaluation
- Conclusions
- Critique: Strengths, Weaknesses and Evaluation
- GAPS

2

Introduction

What does today's cloud offer as a service?

- Generic compute and storage resources

- Specialized accelerators

3

Emergence of programmable network devices

- Pipeline-based programmable devices
  →In-network switches
  →At-host SmartNICs

- Enable wide-range innovations for classical networked systems
  →Consensus: NOPaxos, NetPaxos
  →Caching: NetCache, IncBricks
  →Storage: NetChain, SwitchKV

Why not offer such system as a cloud service?

- **Challenge: Need multitenancy support**
- **Benefit**
  - **Provider's aspect**
    - **Improve resource utilization**
      - One application can hardly consume all the hardware resources
      - Heterogeneous resource requirement
  - **Tenant's aspect**
    - **Enable innovations**
      - New programs can be easily tested w/o impacting basic network functionality

5

How to enable multitenancy for programmable devices?

Requirements:

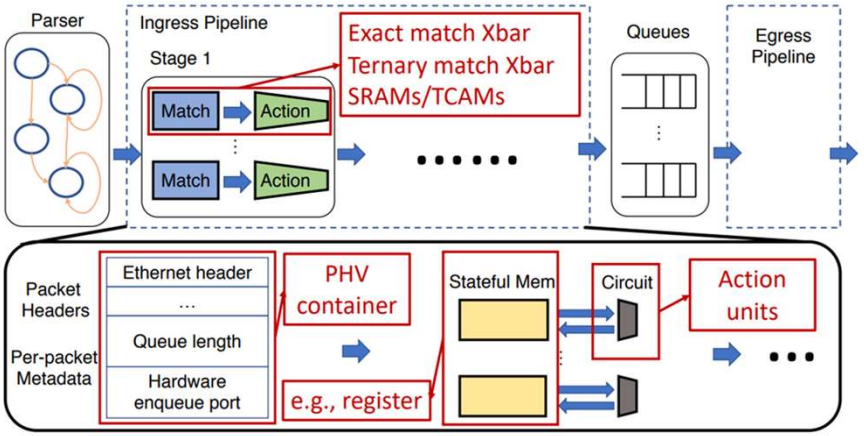
- **Resource efficiency**
  - **Little overhead**
- **Isolation**
  - **Performance**
  - **Allocated resource**

The vision of this paper:

a hybrid **compile-time** and **run-time** solution

UNIVERSITY of WASHINGTON 6

Background on programmable network devices



7

Programmable devices' characteristics

- Various types of hardware resources
  - Most are decided during compile time
- Limited run-time support
  - Hardware wirings are decided during compile time
  - No temporal scheduling (e.g., CPU scheduling)
  - No spatial reconfiguration (e.g., FPGA)

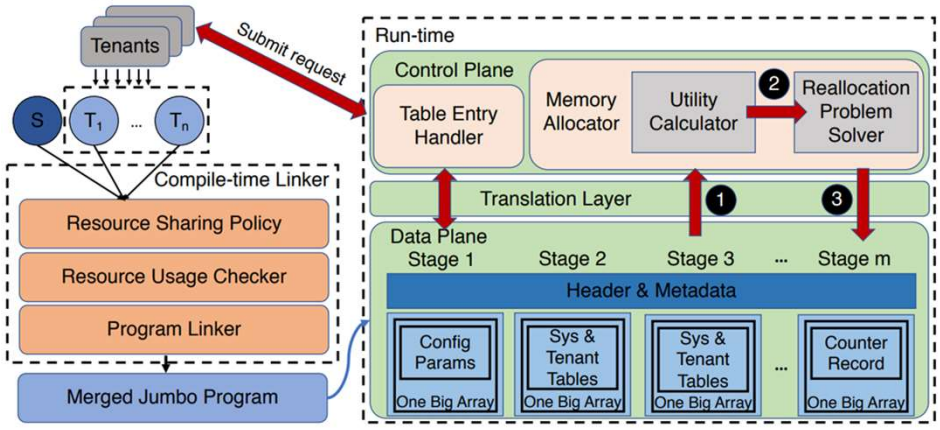
- Resource efficiency
  - Little overhead
- Isolation
  - Performance
  - Allocated resource

8

A hybrid compile-time and run-time solution

- Compile-time program linker
  - Target generic resources (e.g., SRAMs/TCAMs, action units, etc.)
  - But static
- Run-time memory allocator
  - Target stateful memory
  - But limited

System overview

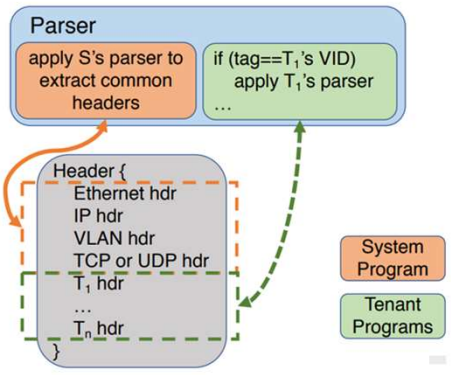


Goals of compile-time linker

- Restrict resource usage
- Provide isolation
  - Ensure tenant program does not inference with others'
  - Ensure no infinite packet resubmitting
  - Ensure no loop forwarding configuration

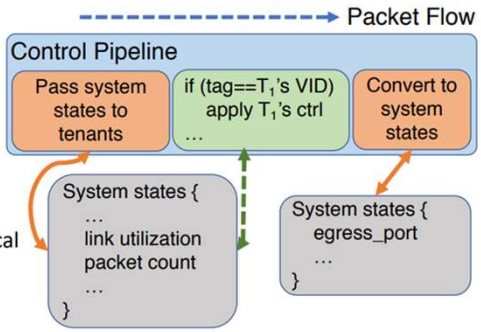
Parser

- Fixed packet format
  - Eth, VLAN, IP, TCP or UDP header followed by custom headers
- System program
  - Extract common headers
- Tenant Programs
  - Extract tenant-defined headers



Control (ingress and egress) pipeline

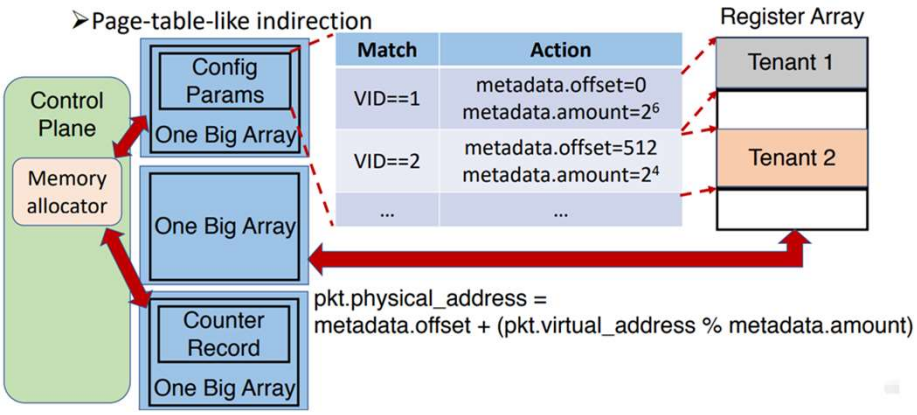
- Feed-forward packet flow
  - "Sandwich" architecture
    - write-then-read half
    - read-then-write half
- System program
  - Interact with tenant programs
  - E.g., pass system states
  - Convert virtual addresses to physical ones



13

Run-time memory allocator

- Page-table-like indirection



14

Key contribution

- design a hybrid compile-time and run-time solution to multitenancy
- efficient resource utilization
- isolation of tenant programs

Related Work

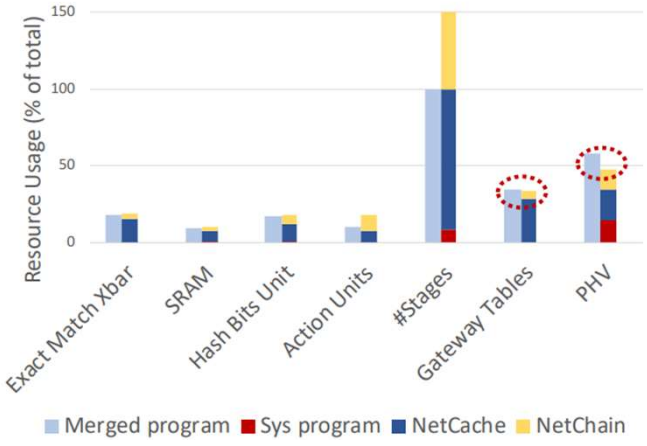
- Hyper4 and HyperV
  - virtualize software switches by declaring primitive tables and actions to emulate tenant programs
  - incur high HW overhead
- P4Visor
  - enables lightweight virtualization through program analysis
  - not enough isolation for multi-tenants
- daPipe
  - common system program and compile-time merging
  - incremental P4 programming
  - does not consider run-time dynamics memory reallocation

Compile-time program linker correctness

- Resource usage on Tofino
- Packet-level validation on PTF
  - Sys program
  - Basic parsing and forwarding logics
  - [SOSP'17] NetCache
  - [NSDI'18] NetChain
- Overhead
  - Additional gateway tables to check which program to be executed
  - Additional tag-along PHV containers

17

Compile-time program linker correctness

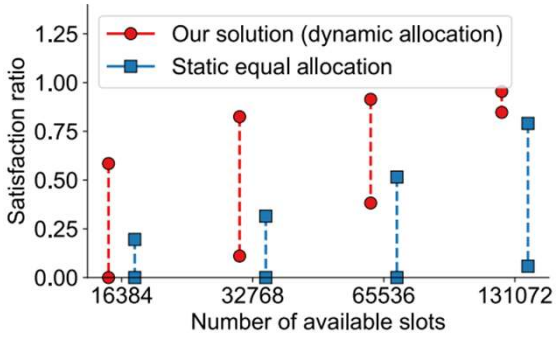


18

Run-time memory allocator efficiency

- Experimental Setting
  - 64 tenants submit 1-min heavy hitter detection task against source IP address within its /6 subnets
  - 10-min CAIDA trace replay
- Evaluation metric
  - Utility: memory hit ratio
  - Satisfaction: time fraction w/ utility > 0.9
  - We show the mean and 5th percentile

Run-time memory allocator efficiency



Conclusion

➢A hybrid solution for multi-tenancy support

➢Compile-time linker: general but static

➢Run-time memory allocator: dynamic but limited

Critique: Strengths

- Resource efficiency
  - Overhead is negligible.
- Isolation
  - Apply in the context where each tenant runs its own program and is distrustful of others tenants
- Targets pipeline-based ASICs and multitenant
- A hybrid solution
  - Consider dynamics and developed run-time memory reallocation mechanism

Critique: Weaknesses

- Only considered a model where consumers submit entire programs to run on programmable devices.
  - higher-level API? simpler, less device-specific programming? lightweight isolation, e.g. paravirtualization?
- As a device OS, failed to provide other common services for applications
  - such as, abstractions for scalable and fault-tolerant storage
- In the evaluation part, only 3 testcases performance were displayed.
  - Also, lacked comparisons with other systems and architectures

Critique: Evaluation

- Three Metrics: isolation, resource efficiency, run-time efficiency
  - It mentioned that other architectures already provided some support for multitenancy in the form of OSes, but failed to show any performance comparisons in any aspect
- Easy to repeat/reproduce test
  - Picked 5 shared tutorial programs with NetCache architecture to create the test programs.

Identify GAPS

★ **Future Work**
- Seek new hardware design, both **general** and **dynamic.**

  For example, HW supports for partial reconfiguration, or dedicated hardware for IP & MAC address translation.
- In the program linker, consolidate among tenants programs during merging to conserve hardware resources.

★ **Open Problems**
- Diversity of resource types makes it hard to determine an efficient and fair policy for partitioning among tenants. Tenants may lie about requirements to get higher allocations.

Reference

1. https://www.usenix.org/system/files/hotcloud20_paper_wang.pdf
2. https://www.usenix.org/system/files/hotcloud20-paper74-slides-wang.pdf


QUESTION ?
UNIVERSITY of WASHINGTON
27