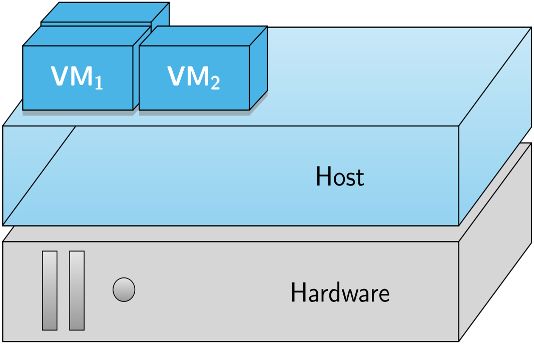
Offloading Security Services to the Cloud Infrastructure

Paul Chaignon[12], Diane Adjavon[23], Kahina Lazri[2],
Jérôme François[1], and Olivier Festor[145]

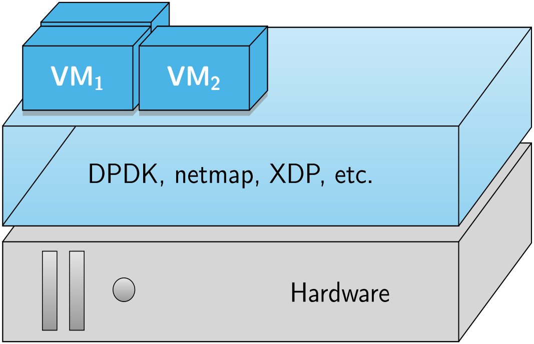
[1]Inria [2]Orange Labs [3]EURECOM [4]Telecom Nancy
[5]University of Lorraine

August 24, 2018

Cost of Hardware Memory Isolation



Cost of Hardware Memory Isolation



- Many improvements at the host layer
- Difficult to get the same performance boost in tenant domains

Offloading Security Services

Offloading Security Services

Security services as a first target for offloads

1. Filters in front of applications
   - IDS/IDP
   - Anti-DDoS
   - Rate-limiters
   - ...

Offloading Security Services

Security services as a first target for offloads

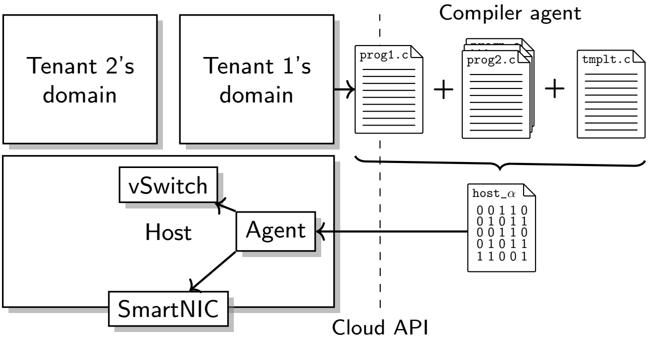
1. Filters in front of applications
   - IDS/IDP
   - Anti-DDoS
   - Rate-limiters
   - ...
2. Encode insights on the application's expected queries:
   - Frequency of queries
   - Format of queries
   - ...

Offloading Security Services

Security services as a first target for offloads

1. Filters in front of applications
   - IDS/IDP
   - Anti-DDoS
   - Rate-limiters
   - ...
2. Encode insights on the application's expected queries:
   - Frequency of queries
   - Format of queries
   - ...
3. Sometimes work in coordination with application (e.g., SYN cookies)

Design


Compiler agent
Tenant 2's domain
Tenant 1's domain
prog1.c
+
prog2.c
+
tmplt.c
vSwitch
Host
Agent
SmartNIC
host_α
0 0 1 1 0
0 1 0 1 1
0 0 1 1 0
0 1 0 1 1
1 1 0 0 1
Cloud API

Design: Isolation

- Many software solutions available:
  - Safe languages (e.g., Rust, Java, Modula-2)
  - Proof-Carrying Code [OSDI'96]
  - Software-Fault Isolation [SOSP'93]

Design: Isolation

- Many software solutions available:
  - Safe languages (e.g., Rust, Java, Modula-2)
  - Proof-Carrying Code [OSDI'96]
  - Software-Fault Isolation [SOSP'93]
- We use the BPF interpreter
  - Relies on ahead-of-time verification of programs through static analysis
  - Tailored for packet processing (limited ISA, limited computational power)

Design: CPU Fairness

1. Guarantee each tenant its fair share of the CPU time
2. Work-conserving allocation: not wasting CPU time

Design: CPU Fairness

- Run-to-completion model common across packet processing frameworks
  - Packets processed by a single thread, on a single core
  - Reduces the number of expensive context switches

Design: CPU Fairness

- Run-to-completion model common across packet processing frameworks
  - Packets processed by a single thread, on a single core
  - Reduces the number of expensive context switches

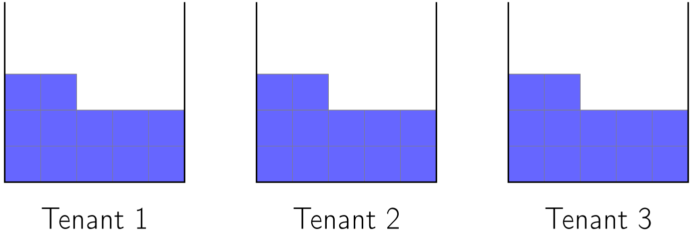
- Preemptive CPU schedulers break this model

- Current approach is to dedicate entire cores to programs [Andromeda @NSDI'18] [NetBricks @OSDI'16]
  - Inefficient use of resources
  - Requires demultiplexing in hardware NIC

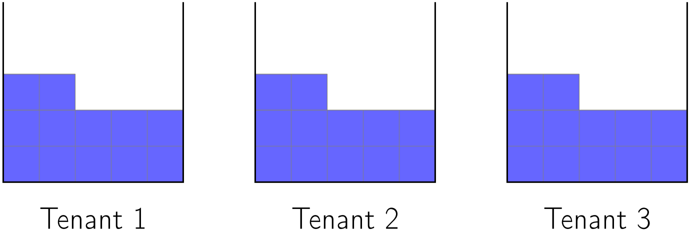
Design: CPU Fairness

Indirectly limit the CPU consumption by
limiting the number of processed packets

Design: CPU Fairness

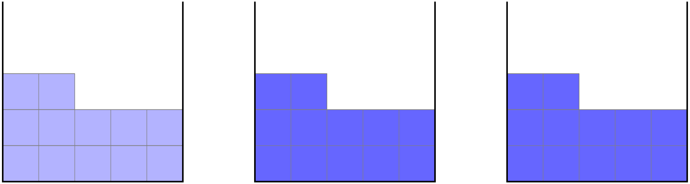


Design: CPU Fairness



Packet for tenant 1 arrives; costs 12 to process

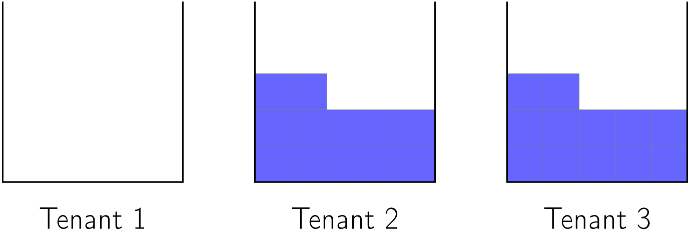
Design: CPU Fairness



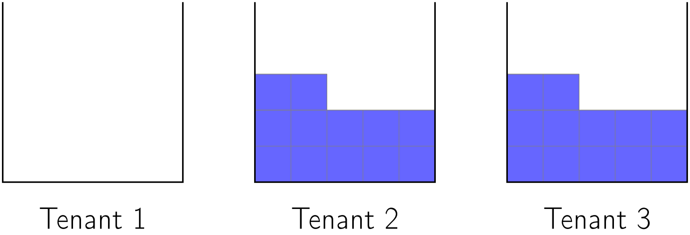
Tenant 1 Tenant 2 Tenant 3

Packet for tenant 1 arrives; costs 12 to process

Design: CPU Fairness

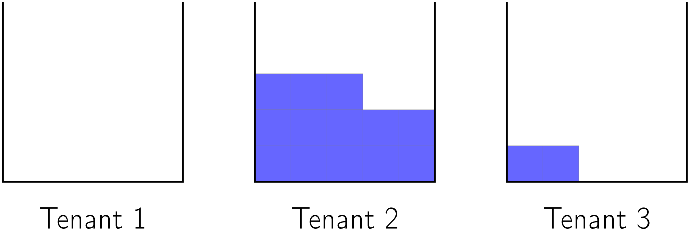


Design: CPU Fairness



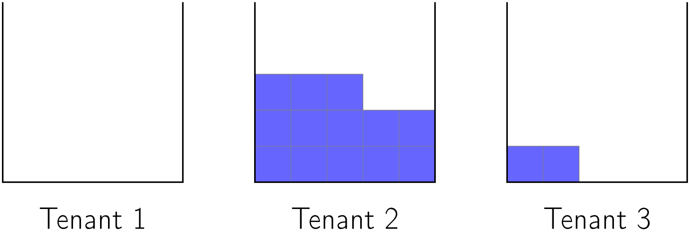
Packet for tenant 1 arrives; we drop it

Design: CPU Fairness



Generate new tokens every Δt

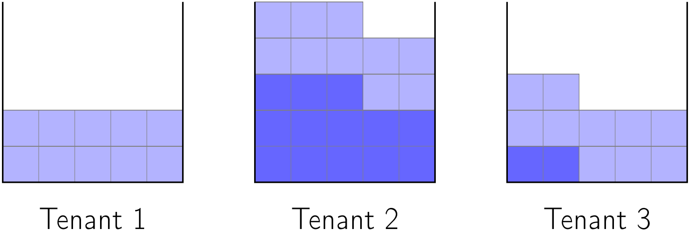
Design: CPU Fairness



Generate new tokens every Δt

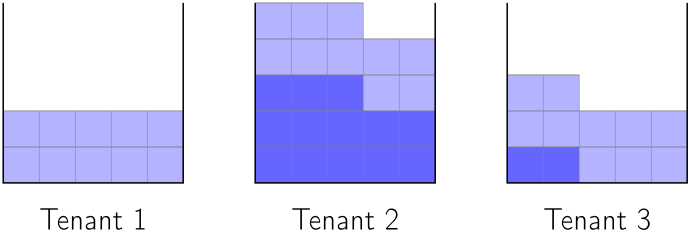
t_1: +30 tokens to distribute

Design: CPU Fairness



Generate new tokens every Δt
t_1: +30 tokens to distribute

Design: CPU Fairness

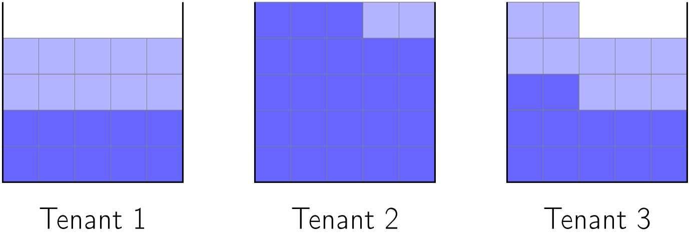


Generate new tokens every Δt

t_1: +30 tokens to distribute

t_2: +30 tokens to distribute

Design: CPU Fairness

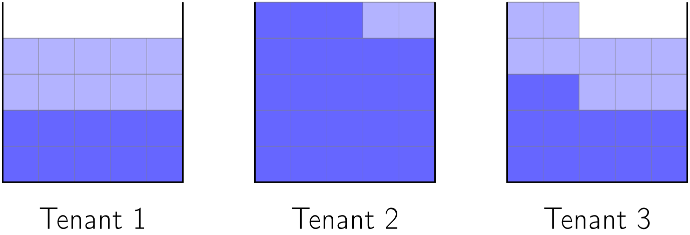


Generate new tokens every Δt

t_1: +30 tokens to distribute

t_2: +30 tokens to distribute

Design: CPU Fairness



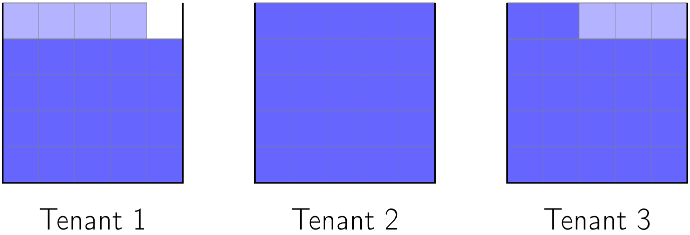
Generate new tokens every Δt

t_1: +30 tokens to distribute

t_2: +30 tokens to distribute

t_2': +8 tokens to distribute

Design: CPU Fairness



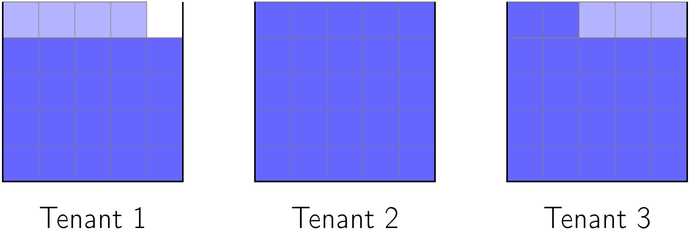
Generate new tokens every Δt

t_1: +30 tokens to distribute

t_2: +30 tokens to distribute

t_2': +8 tokens to distribute

Design: CPU Fairness



Generate new tokens every Δt

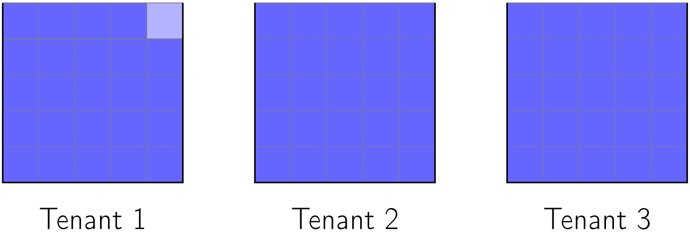
t_1: +30 tokens to distribute

t_2: +30 tokens to distribute

t_2': +8 tokens to distribute

t_2'': +1 tokens to distribute

Design: CPU Fairness



Generate new tokens every Δt

t_1: +30 tokens to distribute

t_2: +30 tokens to distribute

t_2': +8 tokens to distribute

t_2'': +1 tokens to distribute

Design: Accounting for CPU Usage

- First timestamp read on packet arrival
- Second timestamp read once packet is processed, depending on action:
  - Transmitted => Hook on return of transmit function
  - Sent to tenant domain => Hook after packet handoff
  - Dropped => Hook on return of free function

Evaluations: Implementation and Example Offloads

1. TCP proxy
   - Answers with SYN cookies using Linux's algorithm
   - 1 hash table lookup + SipHash algorithm + addresses swapping
   - Retransmits SYNs, drops invalid SYN+ACK, sends to tenant otherwise
2. DNS rate limiter
   - Check queried domain + token bucket
   - Parse DNS query + 2 memory accesses
   - Drops packet or sends to tenant

Evaluations: Performance Gain

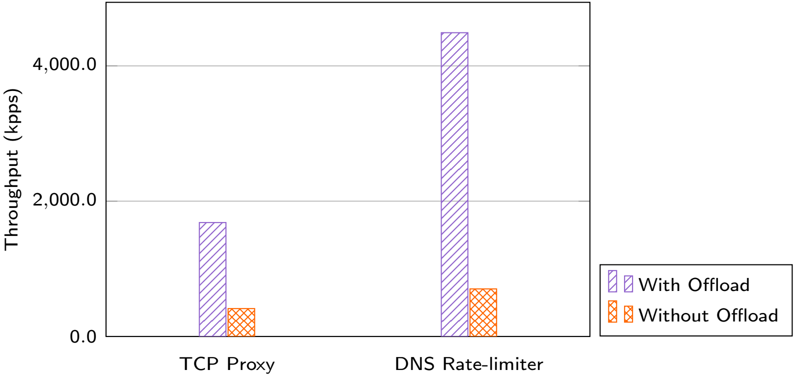


Figure: Packet processing performance with and without offload.

Evaluations: Overhead from CPU Accounting Probes

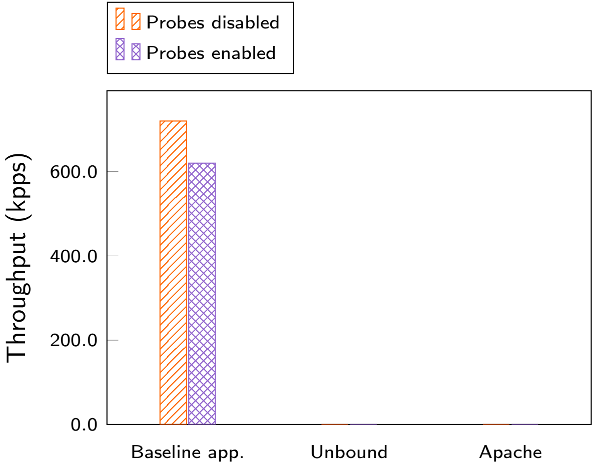


Figure: Packet processing performance with and without probes. Throughput in requests per seconds for Apache only.

Evaluations: Overhead from CPU Accounting Probes

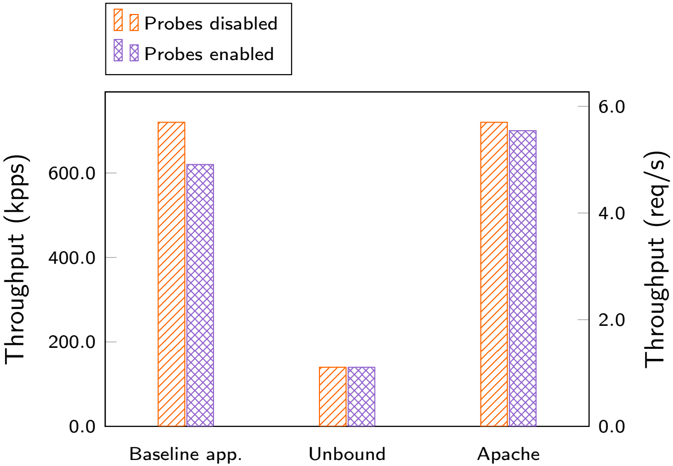


Figure: Packet processing performance with and without probes. Throughput in requests per seconds for Apache only.

Evaluations: Preemptive Scheduler

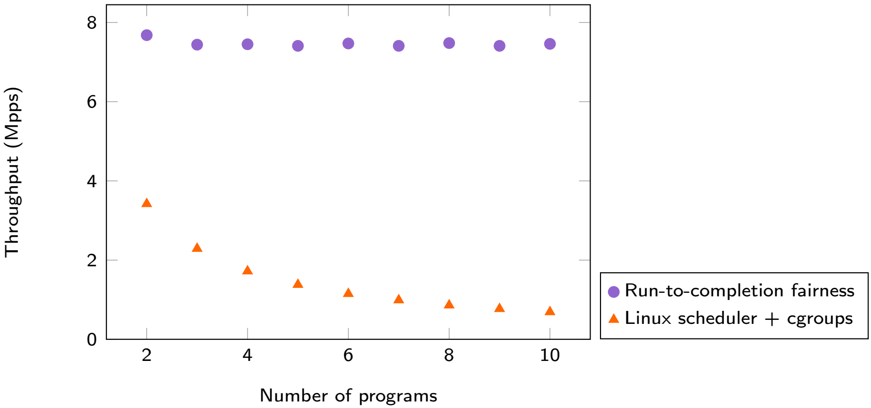


Figure: Packet processing performance under different fairness mechanisms.

Conclusion

- Offload security services using BPF for safety
- New run-to-completion fairness mechanism

- Need to trace CPU time for each packet
  - But small per-packet cost compared to app. processing
- Large performance improvement thanks to offload
  - But depends on I/O library used