Highly-Scalable Transparent Performance Enhancing Proxy

Verizon: Jae Won Chung, Xiaoxiao Jiang, Manish Kurup, Sriram Sridhar

Mojatatu Networks: Jamal Hadi Salim, Roman Mashak

November 10, 2017

verizon

Performance Enhanced Proxy (PEP)

Improve end-to-end TCP performance over a wireless network.

TCP congestion avoidance algorithms widely used today are designed for wired networks, and may not work well on wireless environment.

New TCP congestion avoidance algorithms that may work well on wireless environments are proposed, but require thorough evaluation to be widely adopted in the Internet.

PEP can bridge two different TCP congestion avoidance algorithms; one suitable for wireless and another for wired network.

PEP Service Implementation

Technical Challenges

Fast time-to-market

Fast adaptation to emerging technology

Reduce software maintenance headache

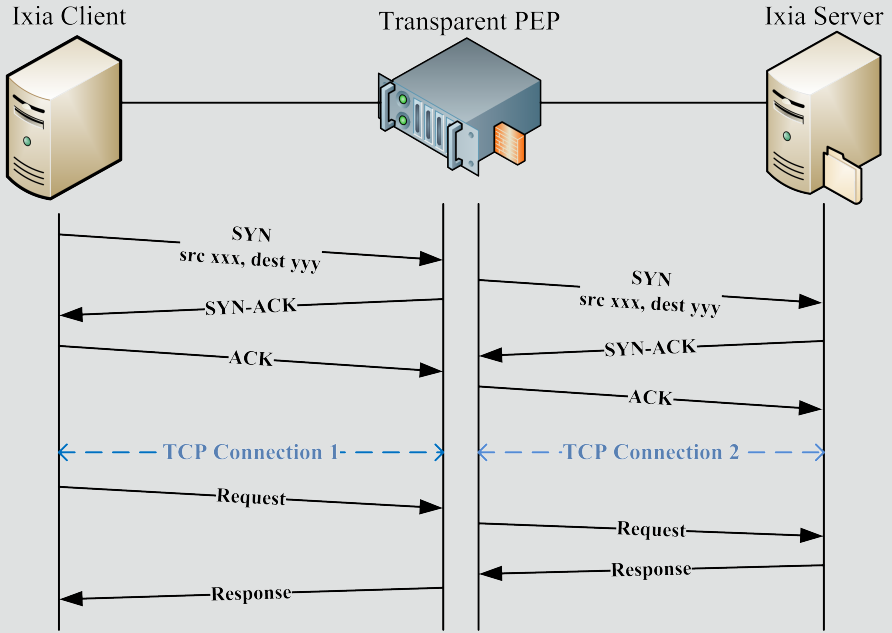
Attractive Potential Solution

Transparent PEP using

- Open source TCP proxy
- Linux TCP and networking stack
- Existing / new / home-grown TCP Congestion Avoidance Module

PEP Design & Evaluation

PEP Service PoC Setup



- **Ixia: Client and Server**
- **x86 Blade with HAProxy: Transparent PEP**
- **HAProxy: 20 processes**

Architecture	x86_64
Number of Socket	2
Cores per Socket	14
Thread per Core	2
Model Name	Intel Xeon E5-2648L@1.80GHz
NIC	Intel XL710 40GbE
Kernel	Linux 4.11.0

verizon

Design Principles

Maximize Parallel Processing

Pining packets from a pair of proxy flows to the same CPU core using receive side scaling (RSS)

Minimize Memory Access across the NUMA boundary

Reduce interrupts and context switches

Running all HAProxy instances on the same NUMA node responsible for the NIC PCI management

Containers

Simplify transparent proxy routing and the service orchestration

Transparent PEP Overview

Host Network

VLAN for differentiating client-side and server-side traffic

TC Mirred action redirect packets based on VLAN

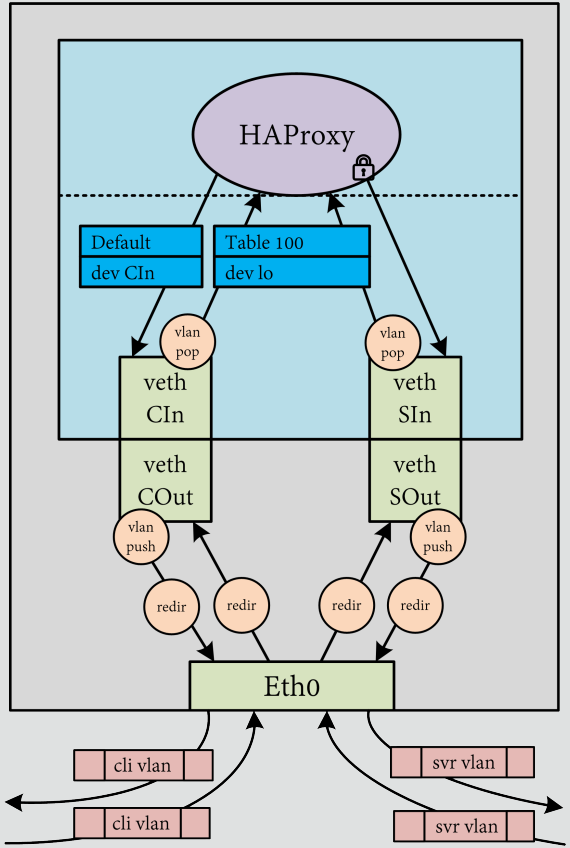
TC VLAN action strips and adds back the VLAN

Network Container

Default routing to client-side veth

HAProxy binds the server-side connection to the server-side veth

All incoming traffic is routed to loopback device via table 100



System Test Scenario

NO-NF

Kernel without Netfilter

NF-NO-IPT

Kernel with Netfilter but no iptables rule

NF-IPT

Kernel with iptables rule using TPROXY Target for NAT

System Scenarios	Netfilter	Iptables Rule	Proxy Listen Port
NO-NF	Disable	None	80
NF-NO-IPT	Enable	None	80
NF-IPT	Enable	TPROXY Target	1234

Performance Tuning Options

NIC RSS

Distributing traffic among multiple receiving queues (28 in our experiments)

Symmetrical RSS can be achieved by configuring the hash key in NIC

Splicing

Two sockets can be spliced inside kernel instead of sending traffic to the user-space proxy.

Proxy Mode (TCP/HTTP)

HTTP proxy has additional cost of parsing HTTP request/response

Proxy-NUMA Binding

Binding HAProxy processes to the cores within the same NUMA that manages the NIC PCI

Baseline Bottleneck

Baseline Performance

Baseline Environment with container and TC

128 TC graphs, 12 RSS queues, mqprio qdisc, netdev_budge_usece=4000

Qualified Metrics

Packets per Second (PPS) and Average Latency

Identified Bottlenecks

A transmit lock in the prio qdisc when contended by multiple cores

A TC action context and statistics update lock in the VLAN action

Bottlenecks Elimination

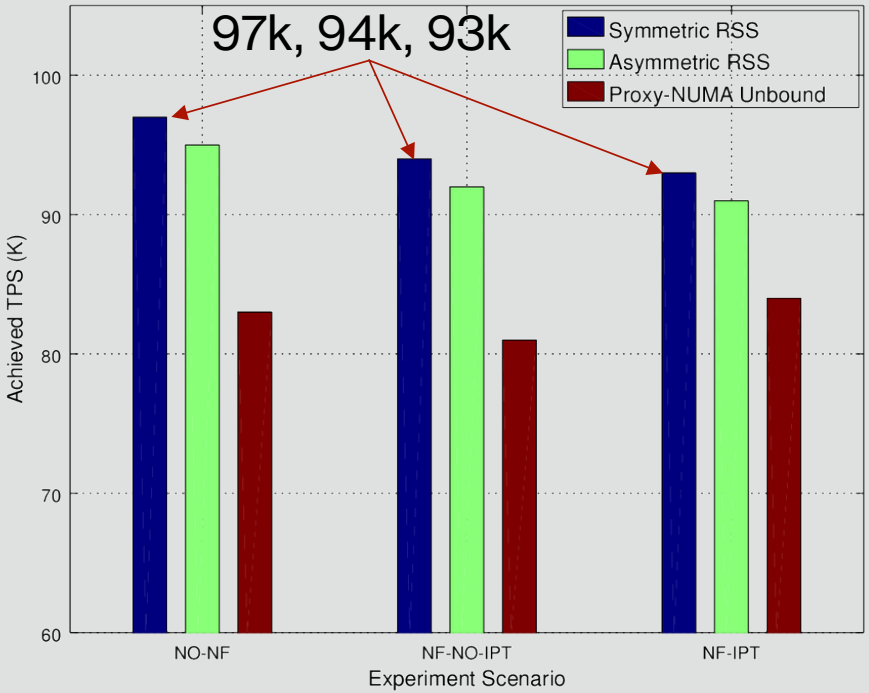
Using mqprio qdisc and modifying the TC VLAN action to use the Read Copy Update (RCU) mechanism, instead of a spinlock.

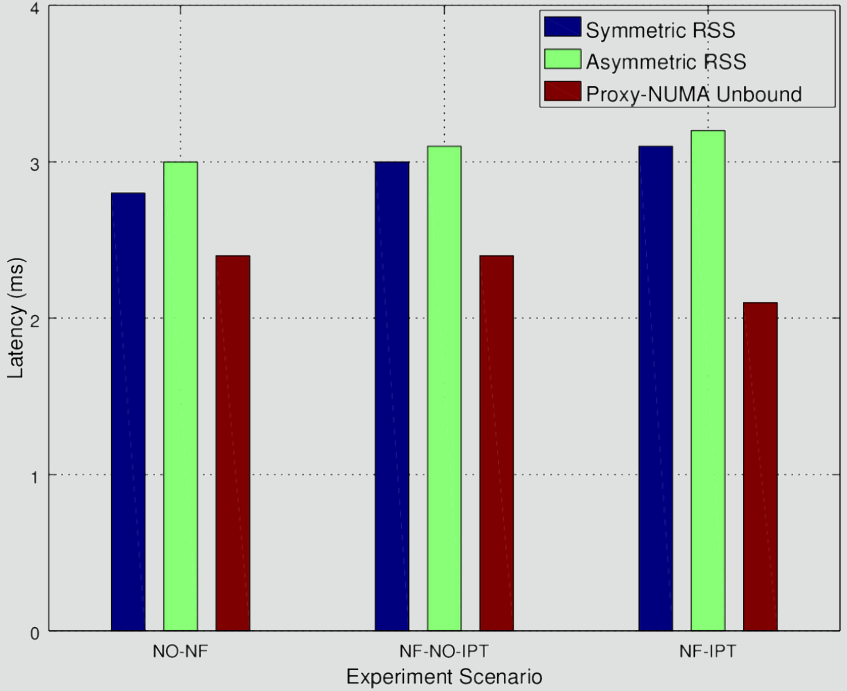
Results after Bottleneck Removal

Packet Size (B)	Rx PPS	Tx PPS	Rx Mbps	Tx Mbps	Rx Avg. Latency	Tx Avg. Latency
78	10M	6.8M	6,240	4,261	873	873
256	8.9M	7.7M	18,285	15,736	883	883
800	3M	3M	19,417	19,417	34	34
900	2.7M	2.7M	19,483	19,483	97	190
1500	1.6M	1.6M	19,680	19,680	105	106

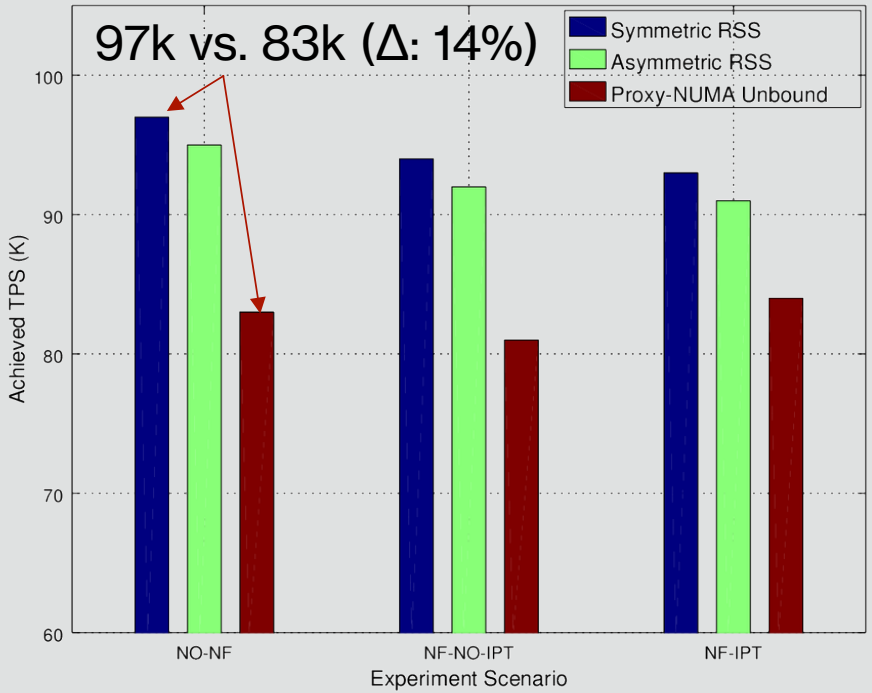
Results

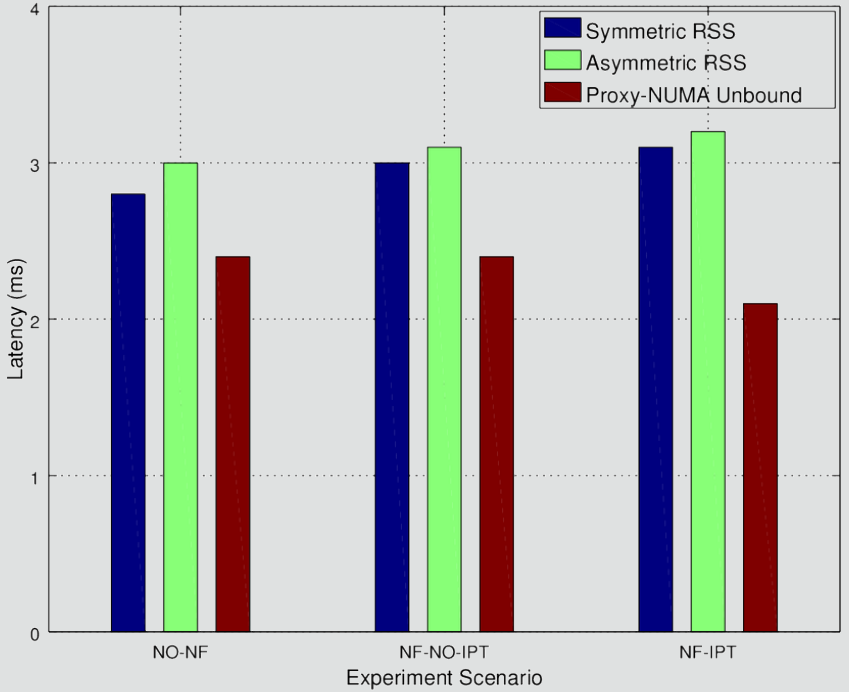
Impact of Netfilter and IPTables





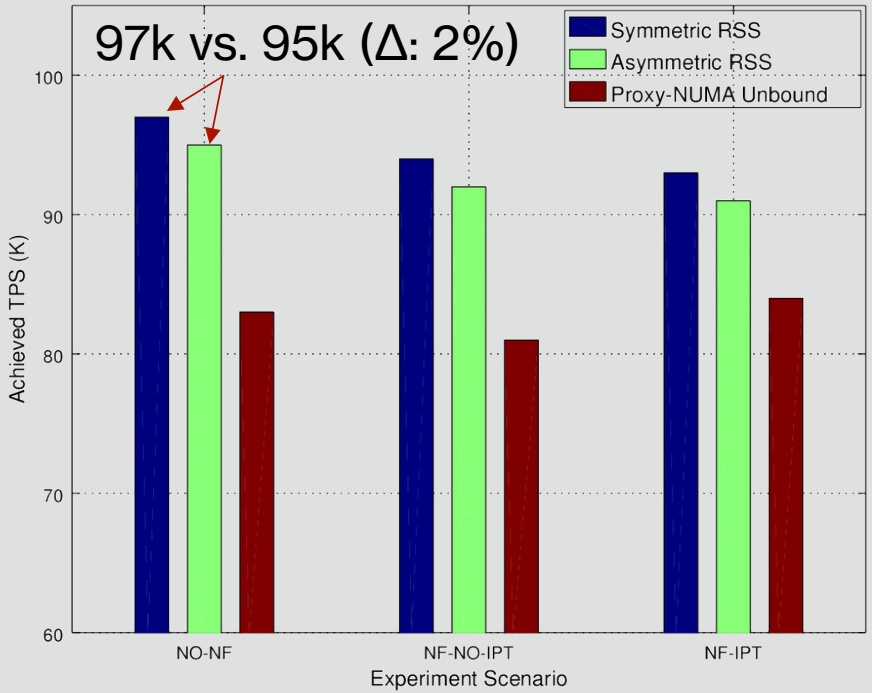
Impact of Proxy – NUMA Binding

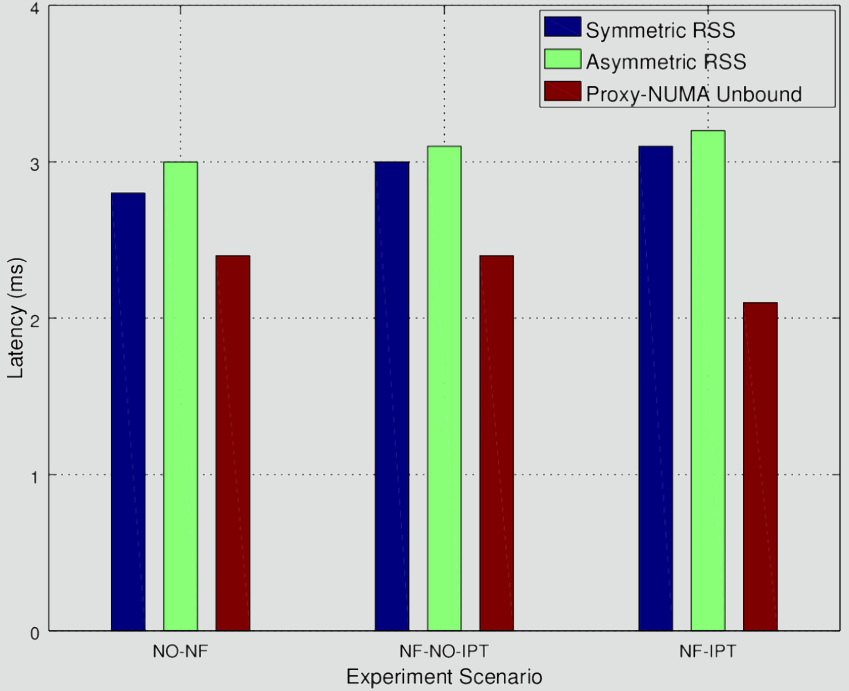




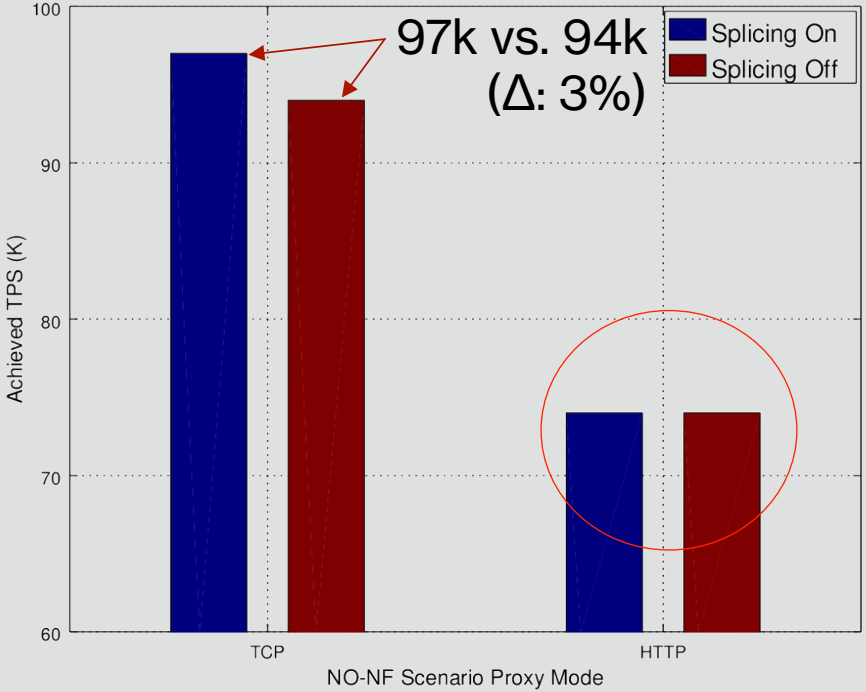
verizon

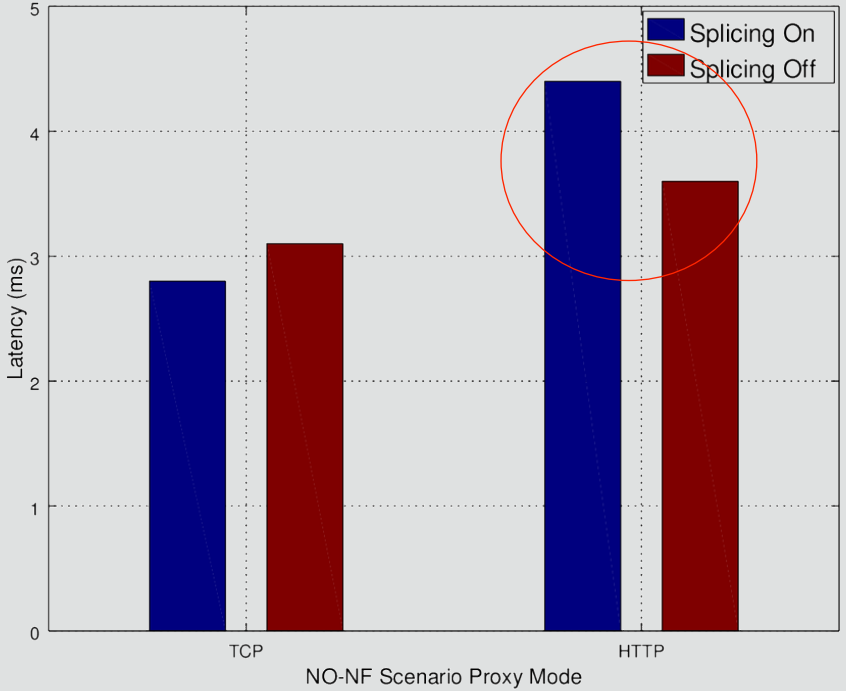
Impact of RSS – Symmetric vs. Asymmetric



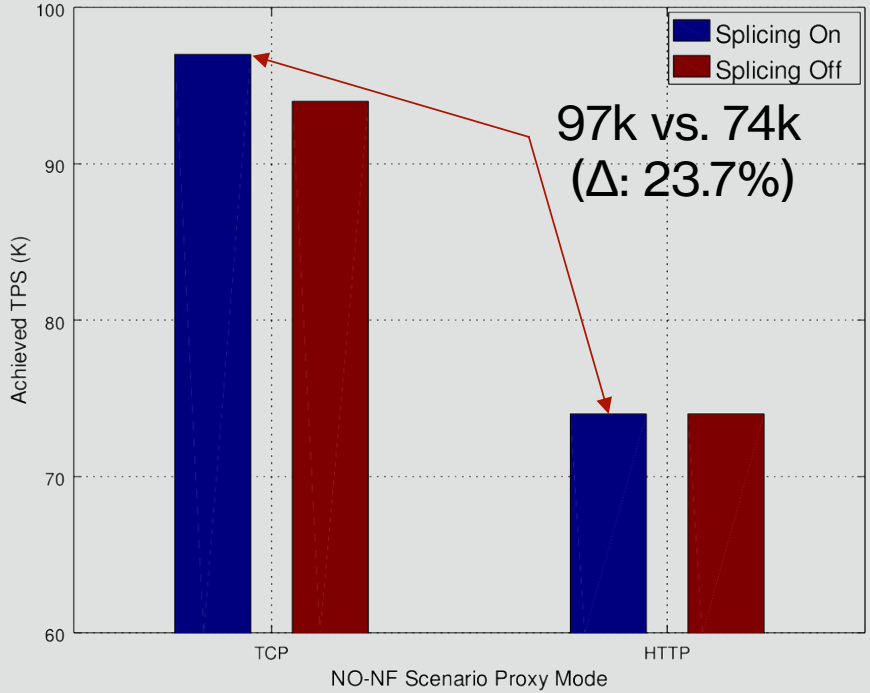


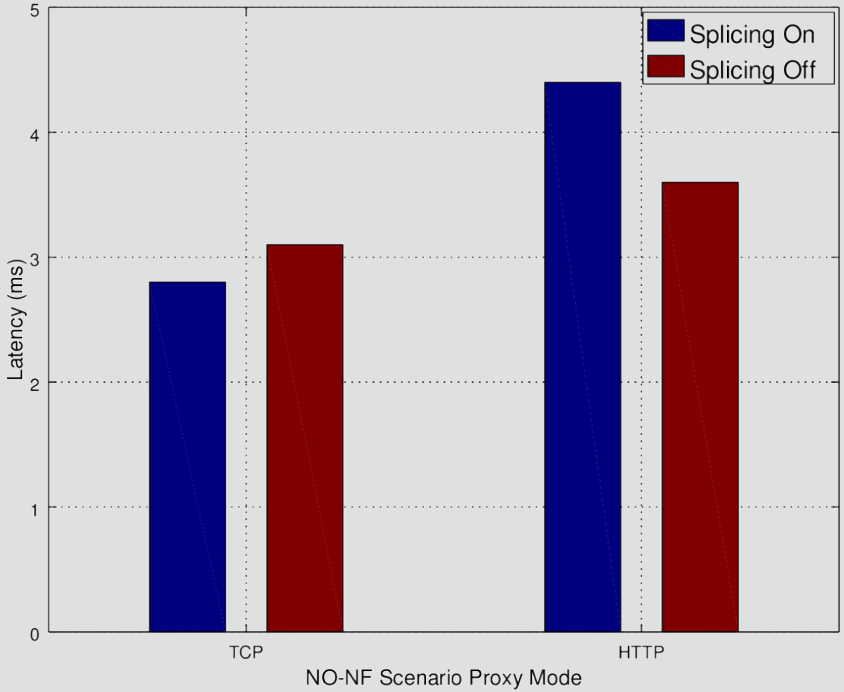
Impact of Splicing





Impact from Proxy Mode





verizon

Conclusions

Conclusions

Build a Highly Scalable Transparent PEP on Linux

Using open-sourced HAProxy

Simple but efficient

Our transparent PEP Achieved Closed to 100K TPS

Netfilter is disabled

With 14 core CPUs, Hyper-Thread (28 RSS Queues)

8K object size

Evaluates Major Performance Tuning Design

Symmetric RSS, Splicing, Process-NUMA Binding, TCP/HTTP Proxy Mode

Future Work

Evaluate transparent PEP performance on more realistic traffic models

TC scaling as a tool for containerized service orchestration

XDP scaling as a mean to enforce service bypass rules

Acknowledgement

Thanks to our colleagues for providing insight and expertise that assisted this project.

Damascene Joachimpillai

Mark Richardson

Anh Quach

Rekha Sundararajan

Thank you.