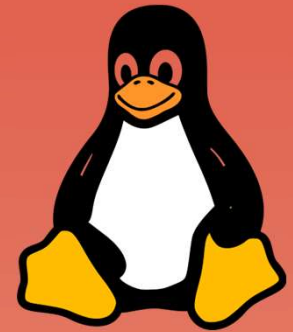
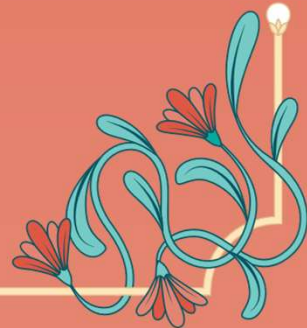




Beyond Best-Effort: Deterministic Real-Time Ethernet on Multi-Core Linux



Meghana Malladi, Daolin Qiu, Schuyler Patton



Meghana Malladi

- Senior Software Engineer at Texas Instruments (TI)
- 3+ years on RTOS and Linux Networking drivers
- Expertise in XDP on TI SoCs and networking solutions with XDP and eBPF



Daolin Qiu

- Systems Engineer at Texas Instruments (TI)
- Joined Systems Applications in 2023 focusing on real-time networking and control applications using embedded Linux on TI SoCs



Schuyler Patton

- Systems Engineer and Member Group Technical Staff at Texas Instruments (TI)
- 20+ years expertise in system designs requiring networking on embedded Linux systems

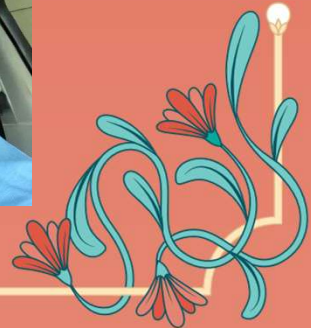
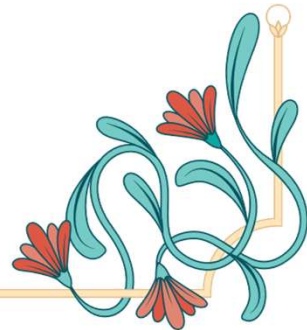
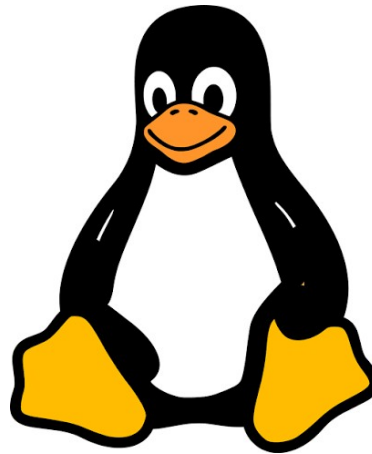


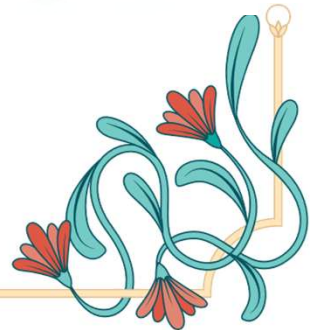
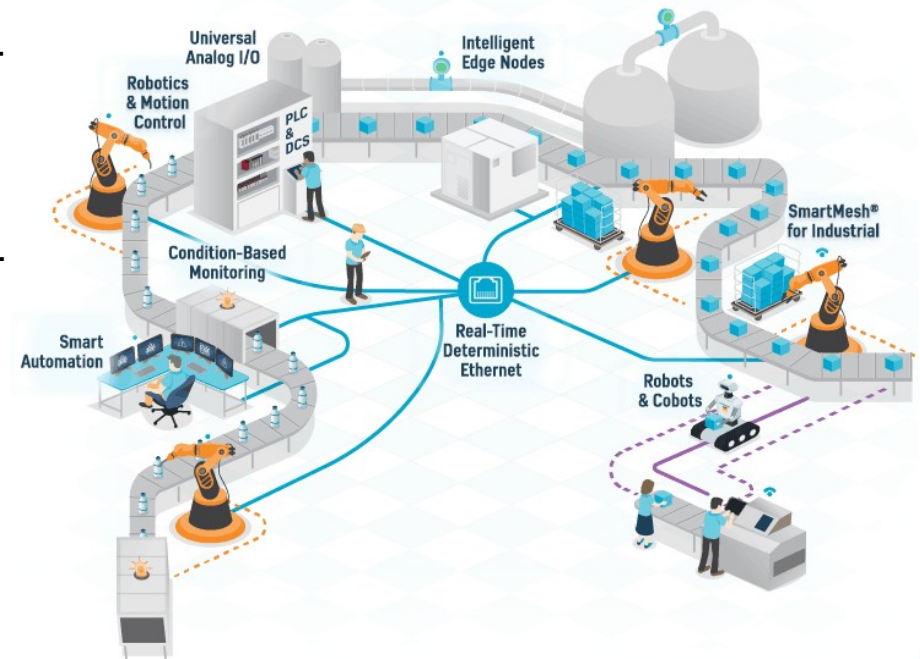
Table of Contents

1. Problem Statement
2. Approach
3. Experimental Setup
4. RTC-Testbench
5. Evaluation methodology
6. Results
 1. Baseline
 2. Traffic Steering
 3. RT tuning
7. Takeaways



Problem Statement

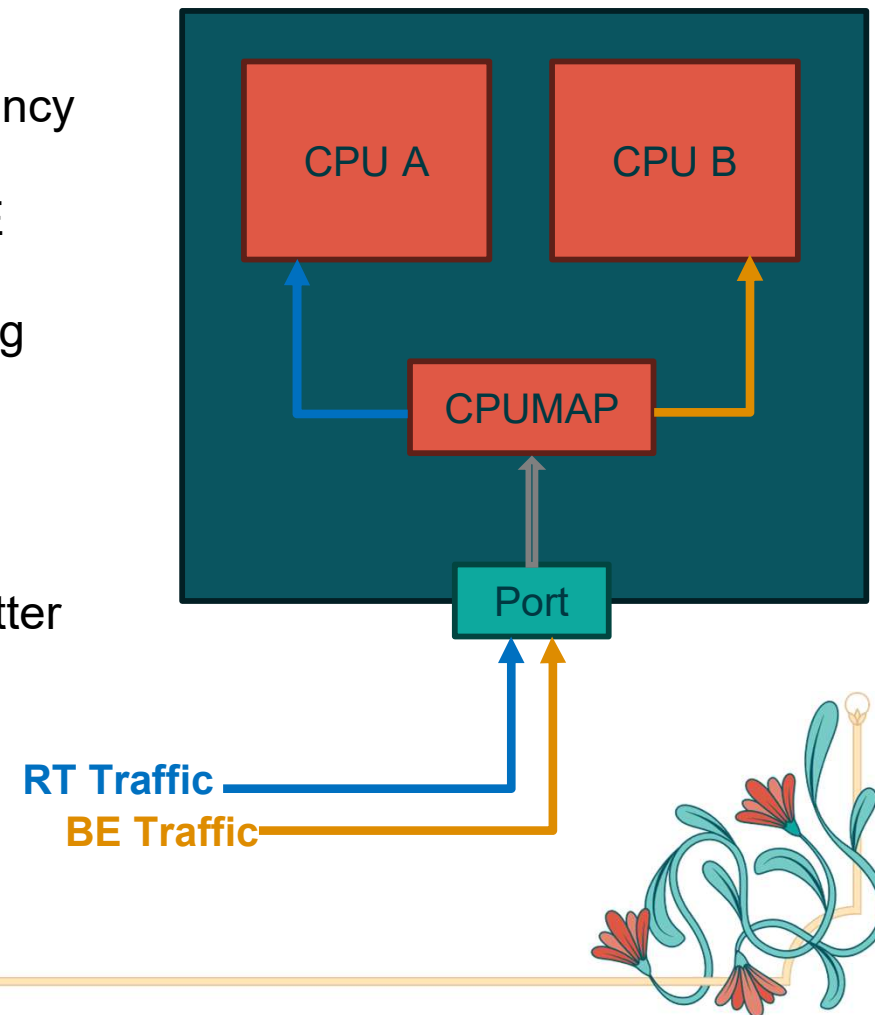
- Industrial Ethernet protocols (EtherCAT, EtherNet/IP, PROFINET) require deterministic, low-latency packet processing
- There can be use cases in industrial network grids where Best Effort traffic meets Real Time traffic, like PLC control with diagnostics in the background.
- This degrades RT performance, especially when both traffics compete for CPU resources.
- When both arrive through same ethernet port, they share the host's packet-processing resources.
- This becomes challenging when the NIC does not support or have feasible hardware packet steering.
- **Question: In the absence of hardware traffic separation, how can we improve RT performance while continuing to handle both traffic types on the same ethernet port?**



Approach

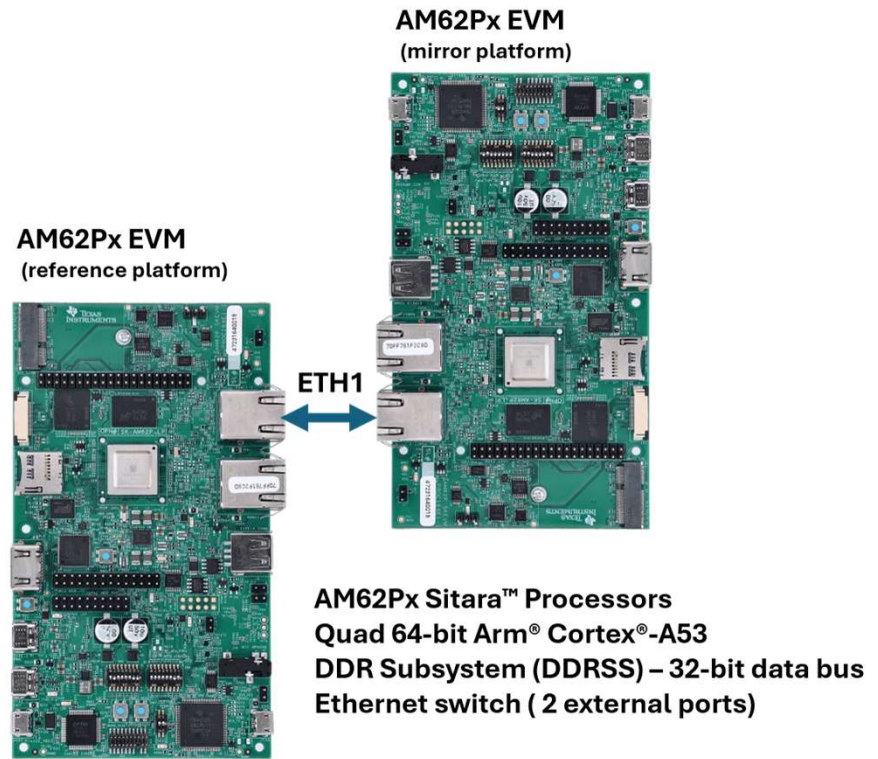
1. Establish baseline performance of RT traffic latency and jitter with BE traffic
2. Implement software-based traffic steering on BE and RT traffic
3. Apply additional RT tunings on top traffic steering
4. Compare performance:
 1. Baseline – Out of box behavior
 2. Software-based traffic steering
 3. Traffic steering + RT tunings
5. Evaluate the overall impact on RT latency and jitter

Latency = round trip time of packet
Jitter = variation in latency



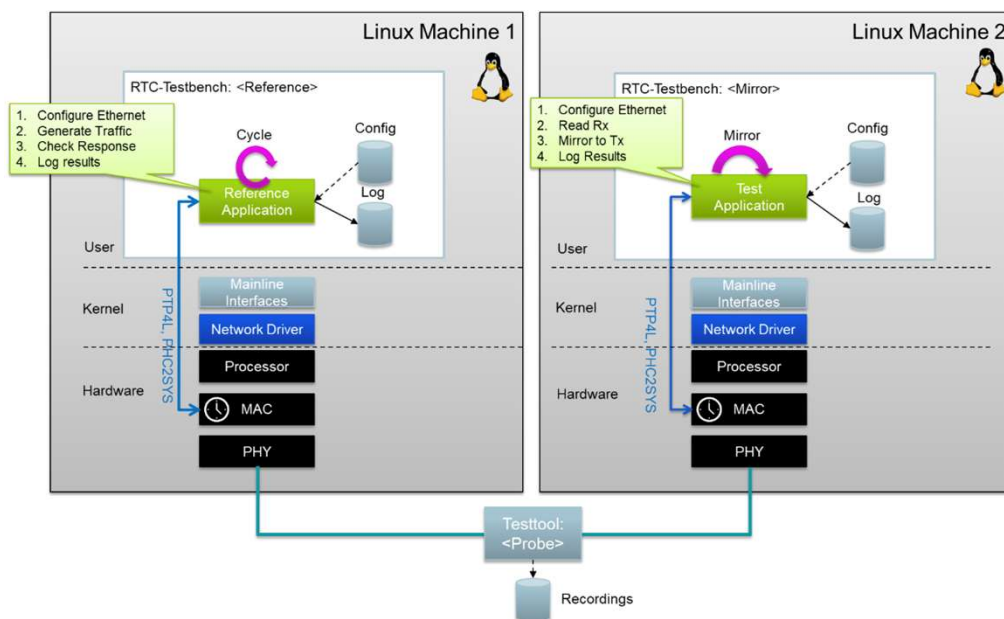
Experimental Setup

- Platform:
 - 2 x am62px-evm
 - Multi-core ARM Cortex-A53 Linux
 - PREMPT_RT
 - Kernel version: 6.12.57
 - Link speed: 1Gbps
 - RT + BE concurrently
 - Single port
 - Same HW queue
- RTC-testbench:
 - Reference and mirror
 - RT traffic: TSN PROFINET Traffic
 - BE traffic: UDP background traffic
 - Runtime duration: 5 minutes
 - Measure round-trip time (RTT)



RTC-Testbench (Linux Real-Time Communication Testbench)

- Open-source RT and NRT traffic validation tool for converged ethernet networks, with and without TSN mechanisms
- Supports wide range of industrial protocols (e.g. PROFINET, OPC UA PubSub) and traffic profiles (e.g. LLDP, UDP, generic L2)
- Each test run is driven by a YAML configuration (reference.yaml & mirror.yaml) which define the test parameters
- Basic measurements includes number of TX and RX packets, RttMin, RttAvg, RttMax, Err, Outliers
- Use histogram.txt output file to generate histogram plot of RTT results (**RTT = latency, distribution of RTT = jitter**)



RTT = Round-trip time

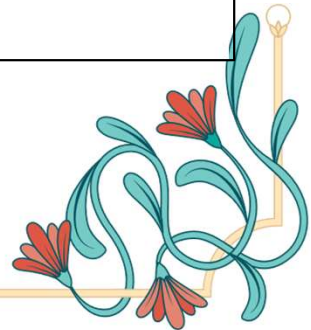


RTC-Testbench Configuration: Traffic Profile

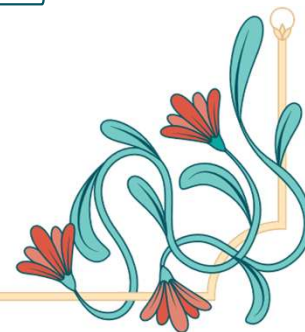
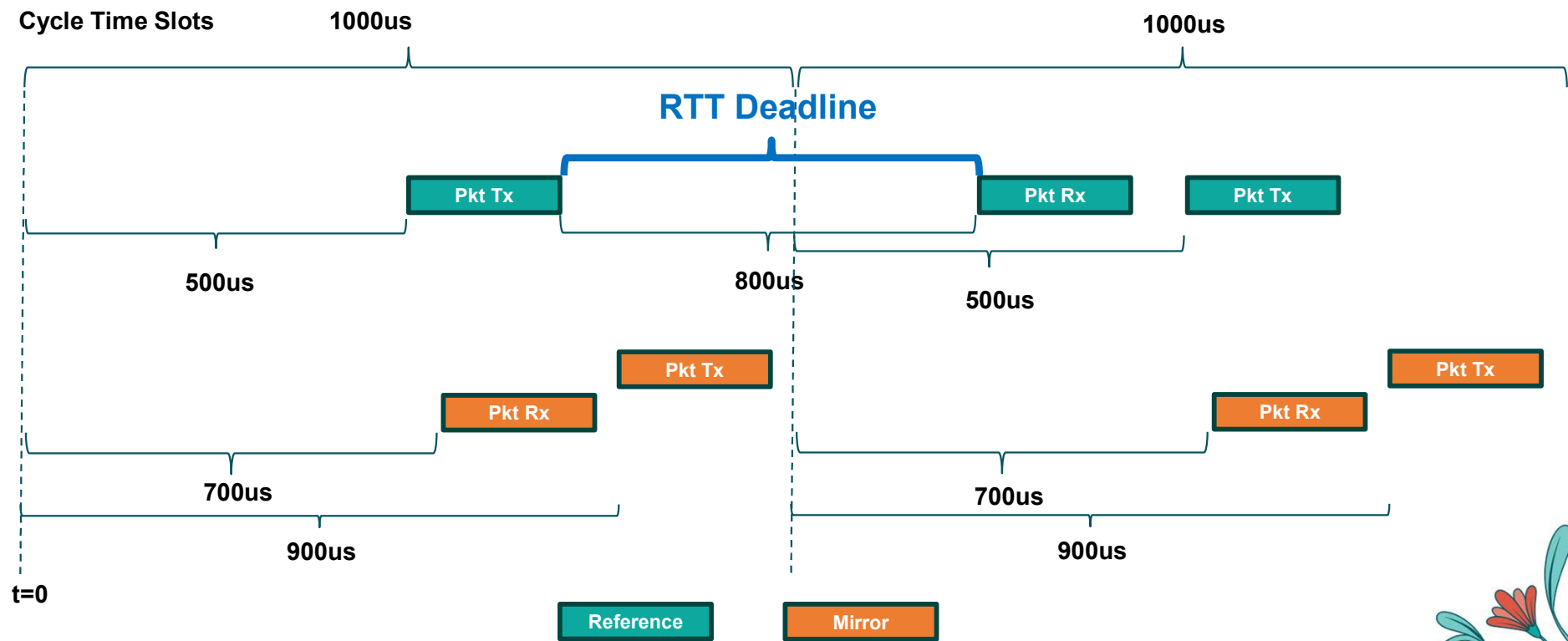
- **TSNHigh: RT traffic**
 - PROFINET
 - VLAN-tagged
 - XDP enabled: packet redirected to AF_XDP socket
 - 1 frame per 1ms cycle
 - 128-byte frame
 - ~1.024 Mbps
 - Tx/Rx thread affinity: CPU3
 - Tx/Rx thread priority: 80
- **UDPHigh: BE traffic**
 - Bursts of 51 frames per 1ms cycle
 - 64-byte frame
 - ~26.112 Mbps
 - Tx/Rx thread affinity: CPU2
 - Tx/Rx thread priority: 50

```
11 Application:
12 ApplicationClockId: CLOCK_TAI
13 # ApplicationBaseStartTimeNS: 0
14 ApplicationBaseCycleTimeNS: 1ms
15 ApplicationTxBaseOffsetNS: 500us
16 ApplicationRxBaseOffsetNS: 300us
17 ApplicationXdpProgram: xdp_kern_profinet_vid100_cpumap.o
18 #ApplicationXdpProgram: xdp_kern_profinet_vid100.o
19 TSNHigh:
20 TsnHighEnabled: true
21 TsnHighXdpEnabled: true
22 TsnHighXdpSkbMode: false # native mode "â€" required when RX_TIMESTAMP=ON (sets BPF_F_XDP_DEV_BOUND_ONLY)
23 TsnHighXdpZcMode: false # ZC required: bypasses CPSW switch fabric so VID 100 frames reach the wire
24 TsnHighXdpWakeupMode: true
25 TsnHighTxTimeEnabled: false
26 TsnHighTxTimeOffsetNS: 0
27 TsnHighTxTimeStampEnabled: false
28 TsnHighVid: 100
29 TsnHighNumFramesPerCycle: 1
30 TsnHighPayloadPattern: |
31 TsnHighPayloadPattern
32 TsnHighFrameLength: 128
33 TsnHighRxQueue: 0
34 TsnHighTxQueue: 0
35 TsnHighSocketPriority: 7
36 TsnHighTxThreadPriority: 80
37 TsnHighRxThreadPriority: 80
38 TsnHighTxThreadCpu: 3 # isolated core
39 TsnHighRxThreadCpu: 3 # isolated core
40 TsnHighInterface: eth1
41 TsnHighDestination: ae:c4:45:08:32:ec
42 UDPHigh:
43 UdpHighEnabled: true
44 UdpHighBurstPeriodNS: 1ms
45 UdpHighNumFramesPerCycle: 51
46 UdpHighPayloadPattern: |
47 UdpHighPayloadPattern
```

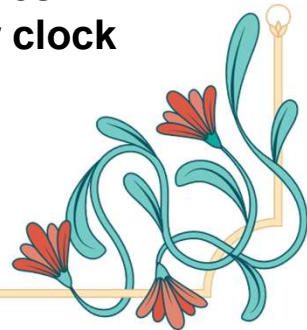
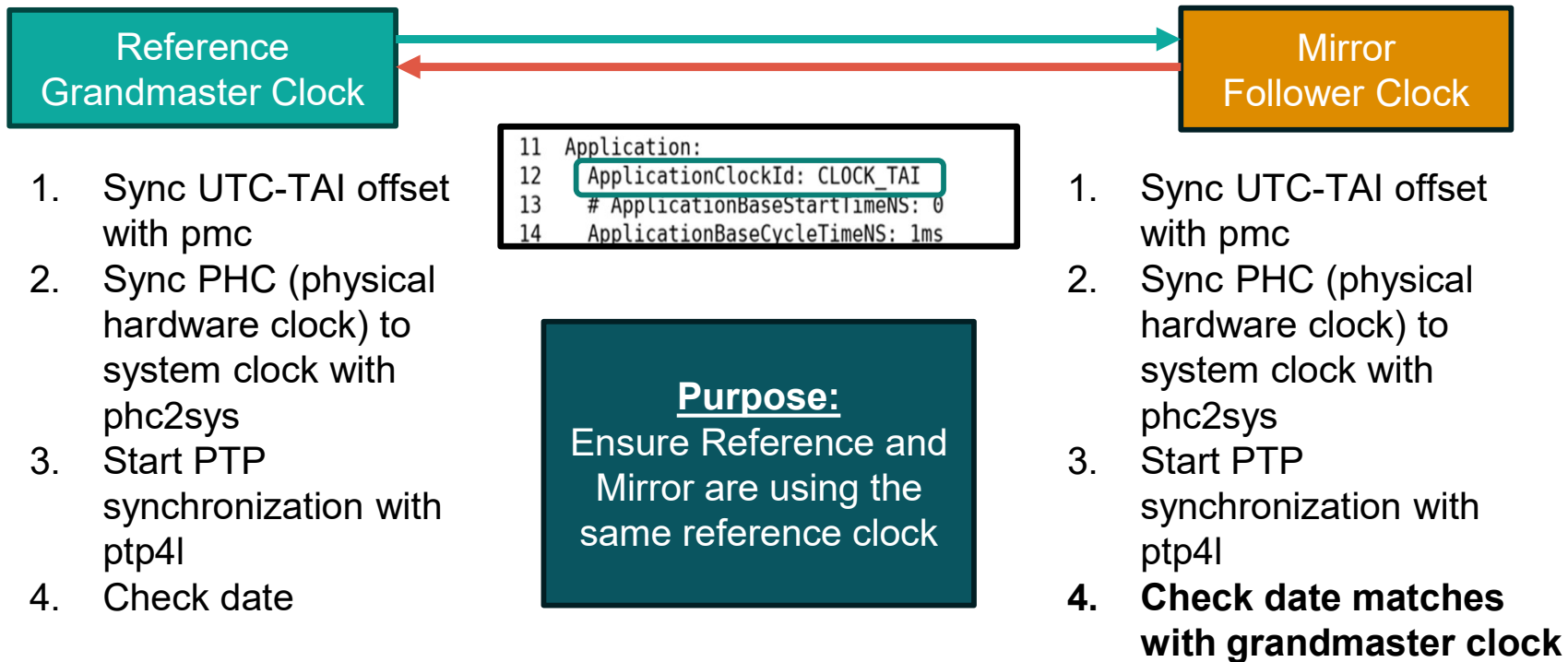
Snippet of reference.yaml



RTC Testbench Configuration: Packet Schedule



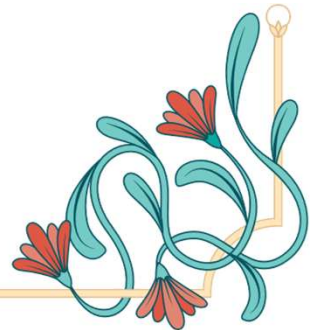
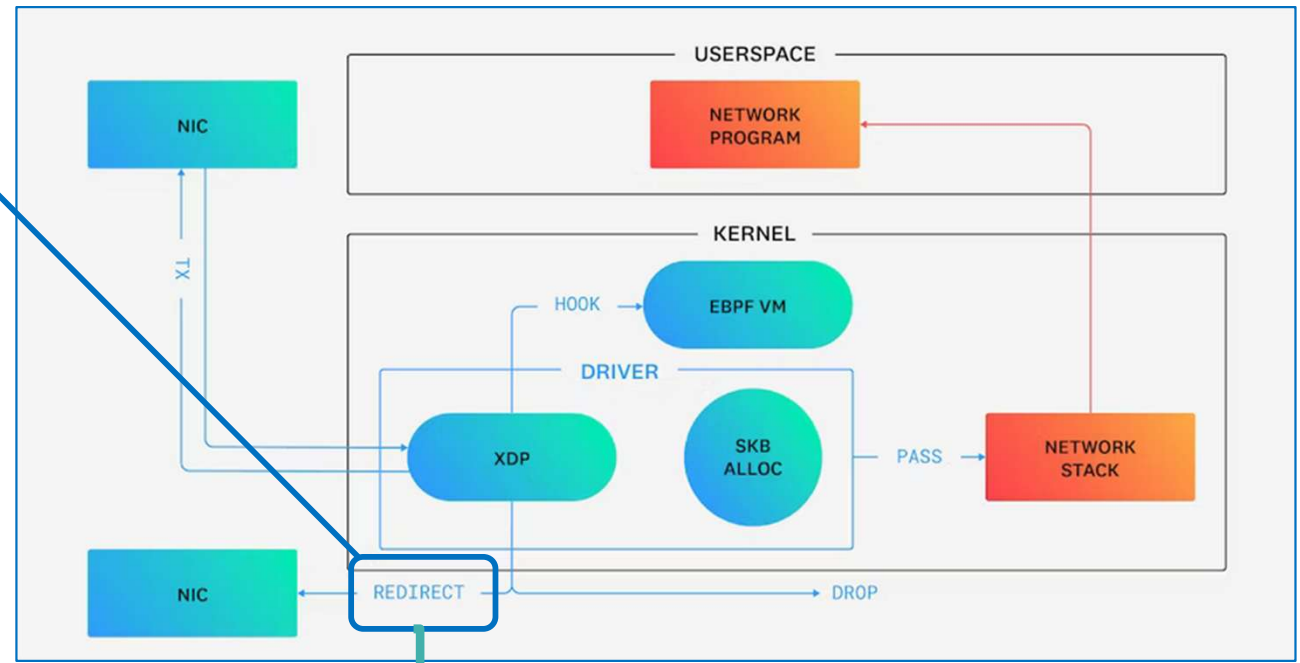
Time Synchronization



XDP + CPUMAP

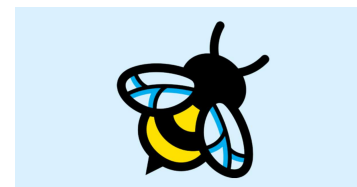
XDP_REDIRECT:
raw packets gets
redirected to different
destinations like
netdevices, sockets
and CPUS

CPUMAP: tells which
CPU to redirect to and
run kthread on the
destination CPU



Traffic Steering

- The CPU to which NIC RX queue IRQ gets steered, initially sees and executes eBPF program on that packet.
- Objective here is to scale the CPU usage across multiple CPUs
- So BPF program filters and redirects traffic accordingly to different cores
- Kernel creates cpumap/2 kthread pinned to CPU2
- After redirection, UDP frames get dequeued by the kthread on CPU2 and re-injected to the kernel network stack



eBPF logic

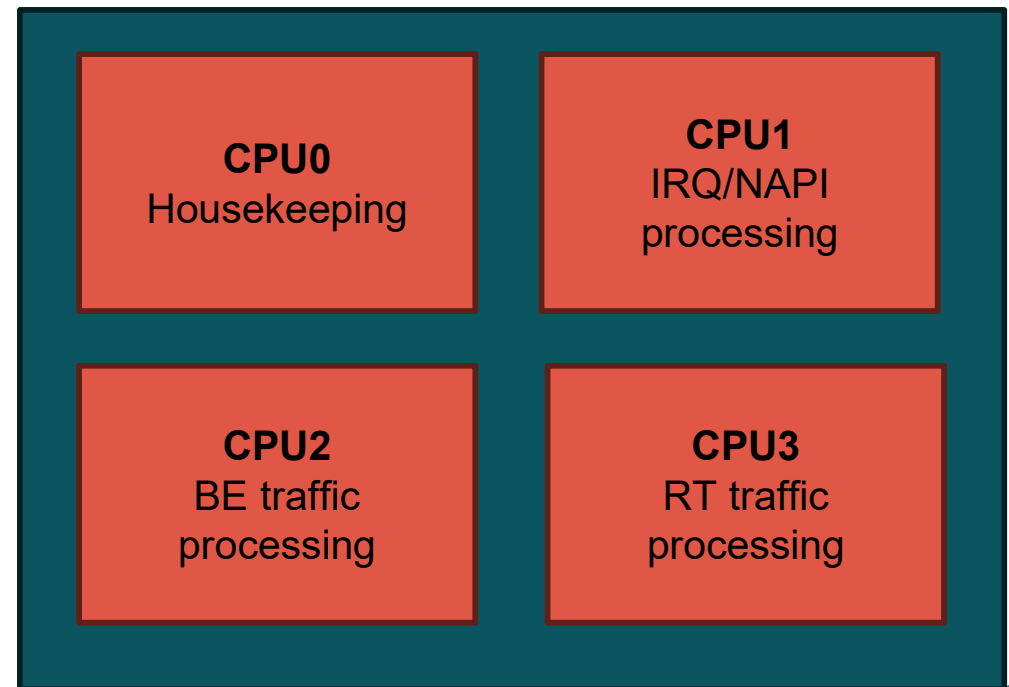
RX frame arrives on eth1

- Not VLAN (UDP/IP best-effort)
 - `bpf_redirect_map(&cpumap, cpu=2, XDP_PASS)`
gets redirected to CPU2 ring
- Profinet RT + FrameID matches known list
 - `bpf_redirect_map(&xsk_map, idx, 0)`
gets redirected to xsk socket on CPU3



RT Tunings

- Assign dedicated cores for handling different critical tasks (e.g. softirq, packet processing)
- Apply following configurations:
 1. Isolate CPU3: `isolcpus=3 rcu_nocbs=3 nohz_full=3 rcu_nocb_poll`
 2. Disable irqbalance
 3. IRQ affinity for NIC HW queues to CPU1
 4. CPUMAP redirects UDP traffic to CPU2
- Note: above list is not an exhaustive list of all tunings that can be applied

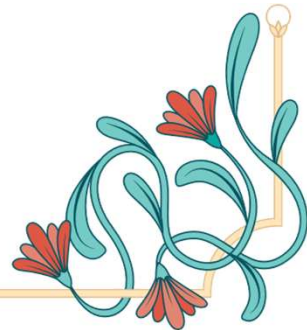


Only CPU3 is isolated, rest of all the cores have other processes running other than intended tasks shown in picture



Evaluation Methodology

Configuration	Traffic Handling	RT tuning
Config 0: No BE traffic	Out-of-Box packet processing	No additional tuning
Config 1: Baseline	Out-of-Box packet processing	No additional tuning
Config 2: Software-based Traffic Steering	XDP + CPUMAP	No additional tuning
Config 3: Config 2 + RT Tunings	XDP + CPUMAP	CPU isolation, IRQ affinity etc

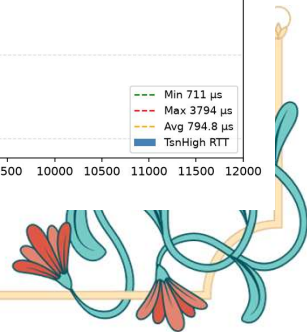
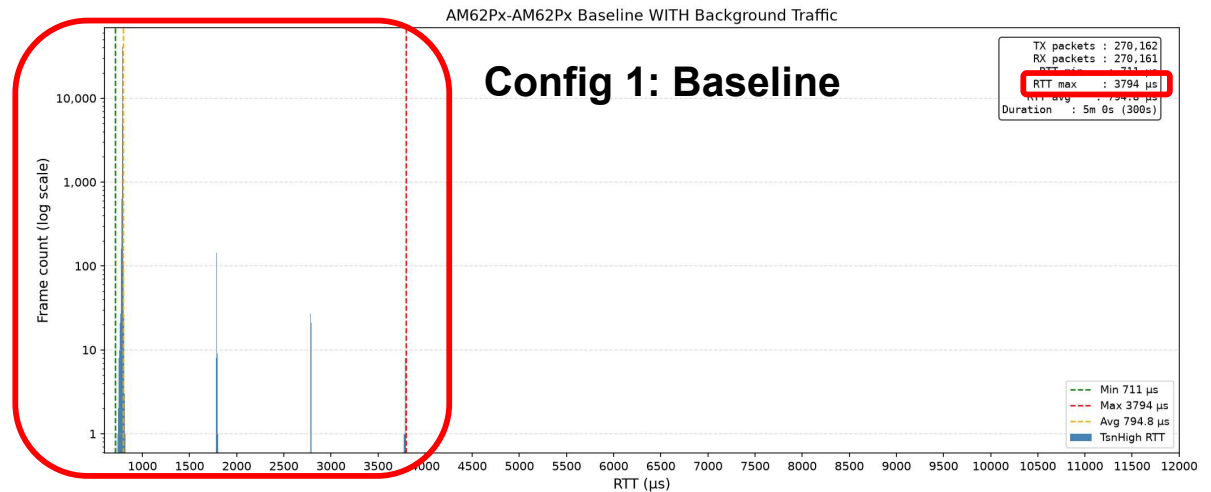
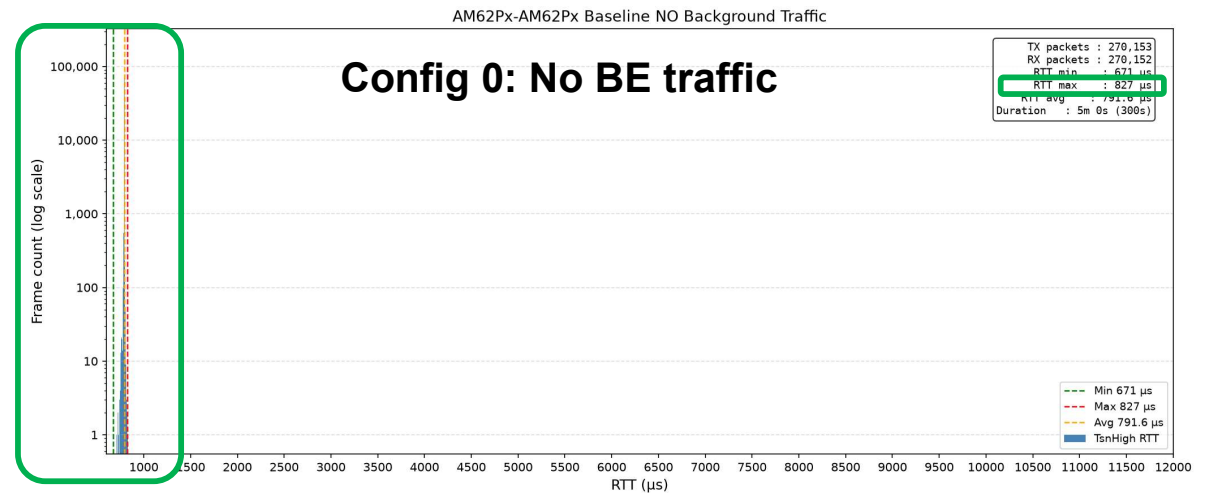


Results

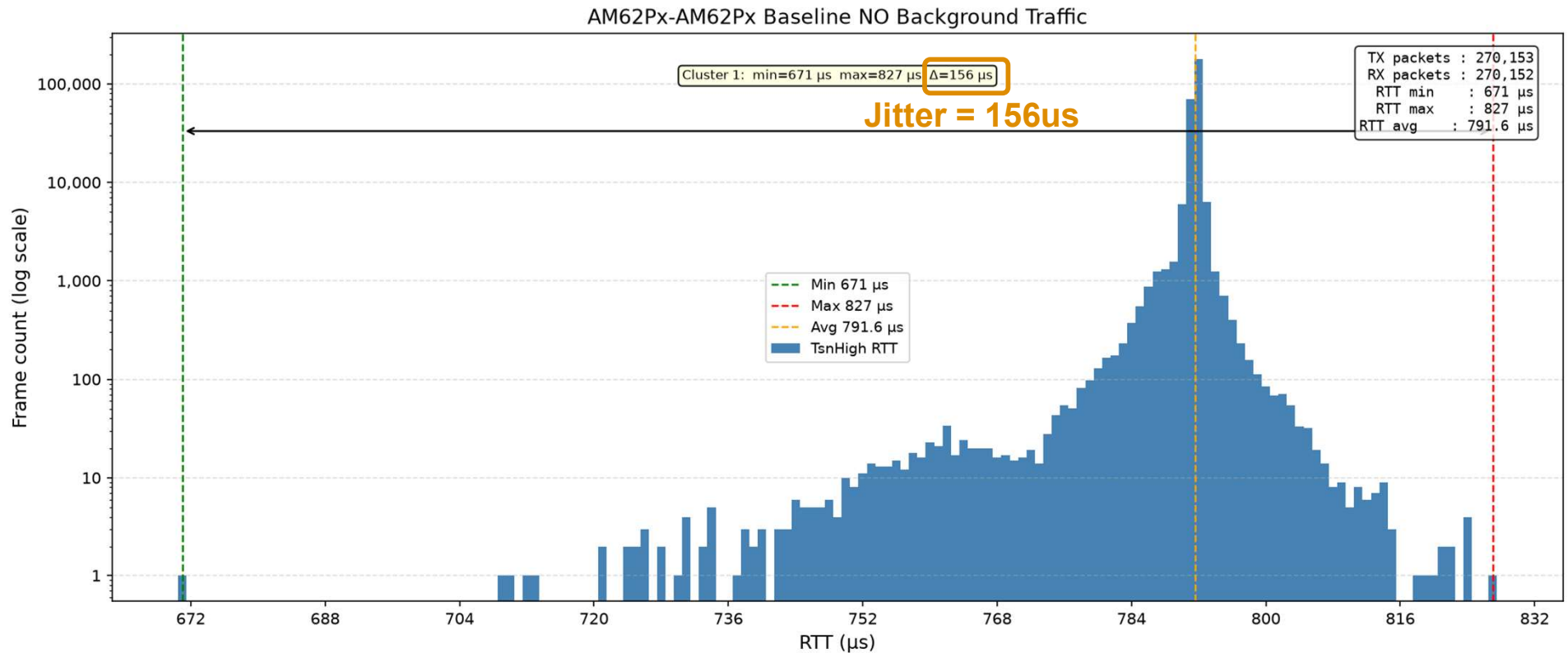


Baseline Results

- Histogram plots of RT traffic RTT
 - **RTT Deadline 800us**
 - Outliers are packets with RTT greater than 2x cycle time (2ms)
 - **Config 0:**
 - No major missed deadlines when no BE traffic present
 - **Config 1:**
 - Outliers are present when BE traffic introduced
 - Many packets missed deadline by +1000us, +2000us, and +3000us
- **BE traffic has large negative impact on RT traffic latency**

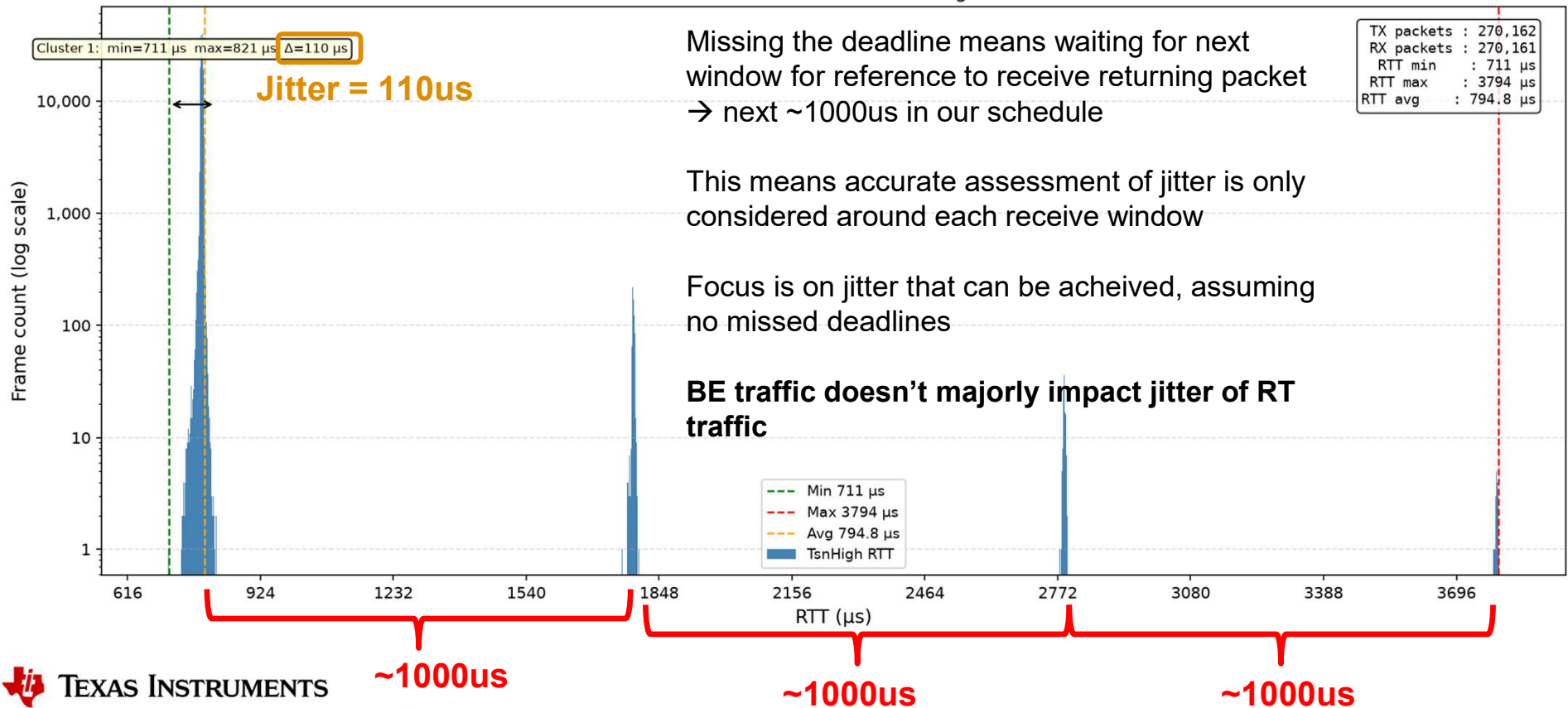


Config 0: No Background Traffic

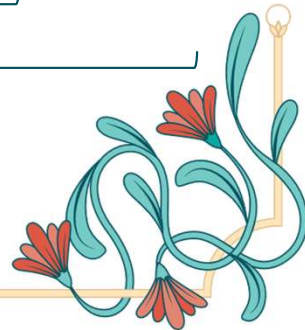
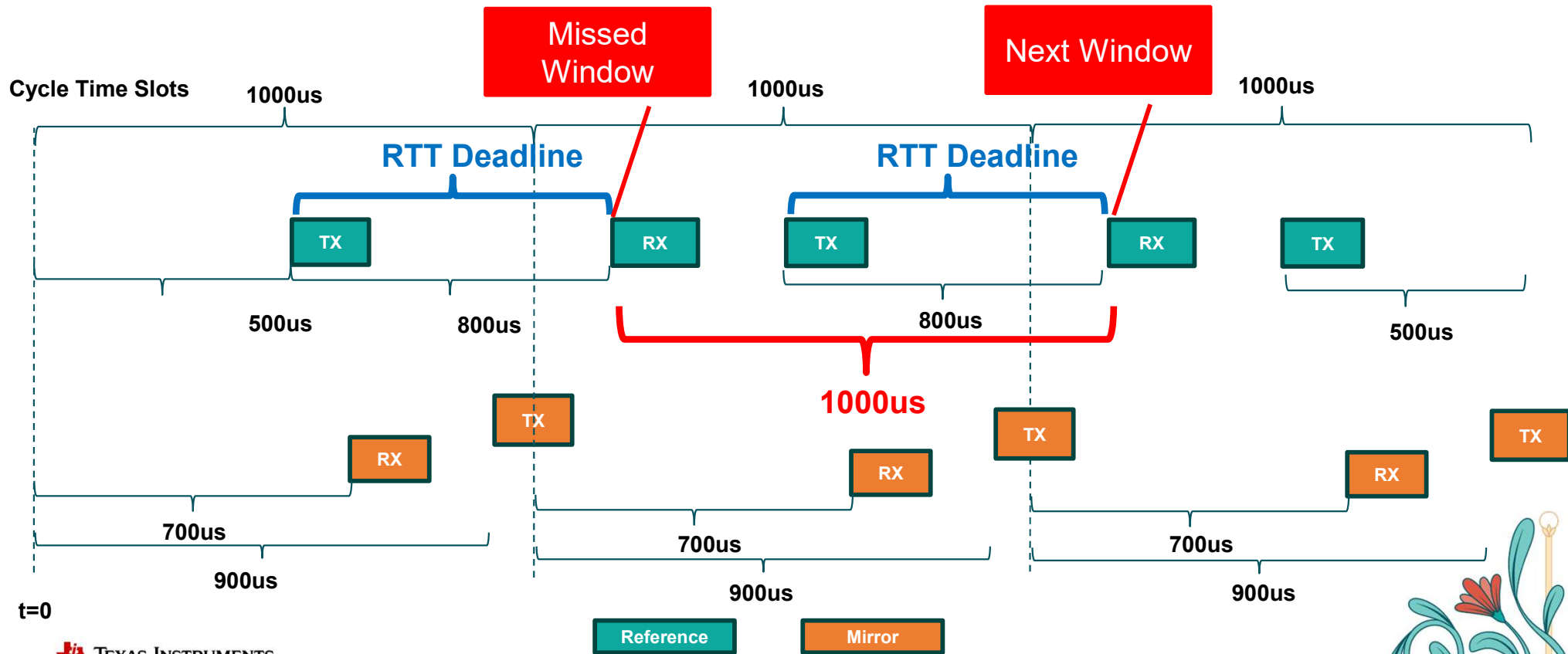


Config 1: Baseline

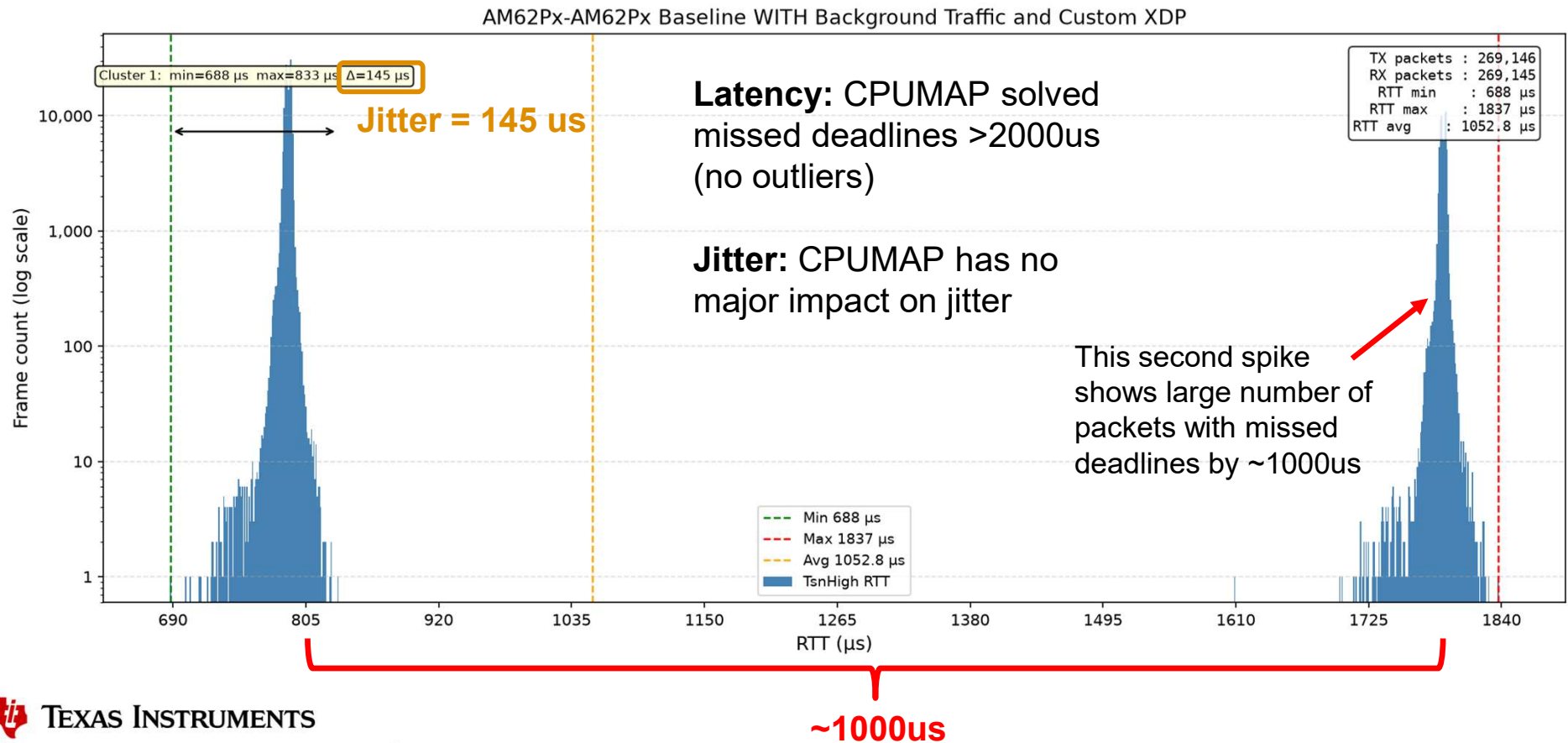
AM62Px-AM62Px Baseline WITH Background Traffic



RTC Testbench Configuration: Packet Schedule

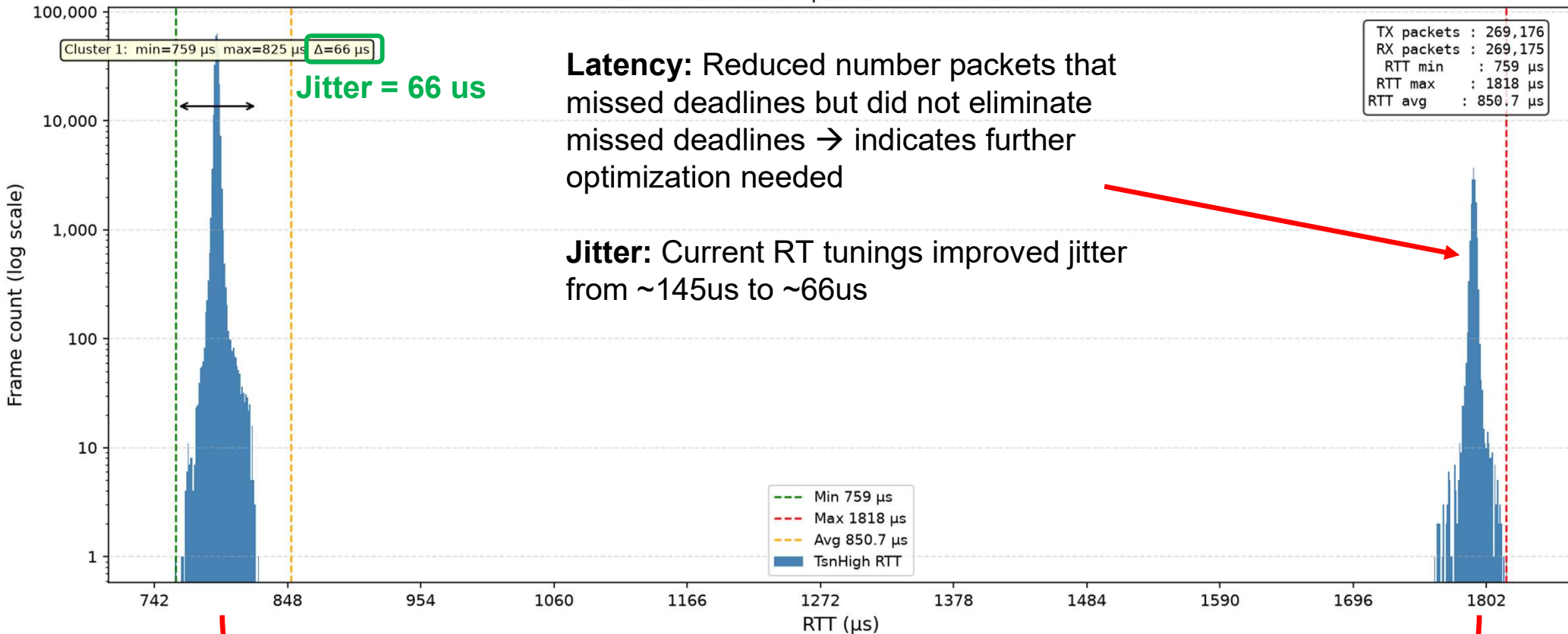


Config 2: Software-based Traffic Steering



Config 3: Config 2 + Real-Time Tunings

AM62Px-AM62Px Optimal with Custom XDP



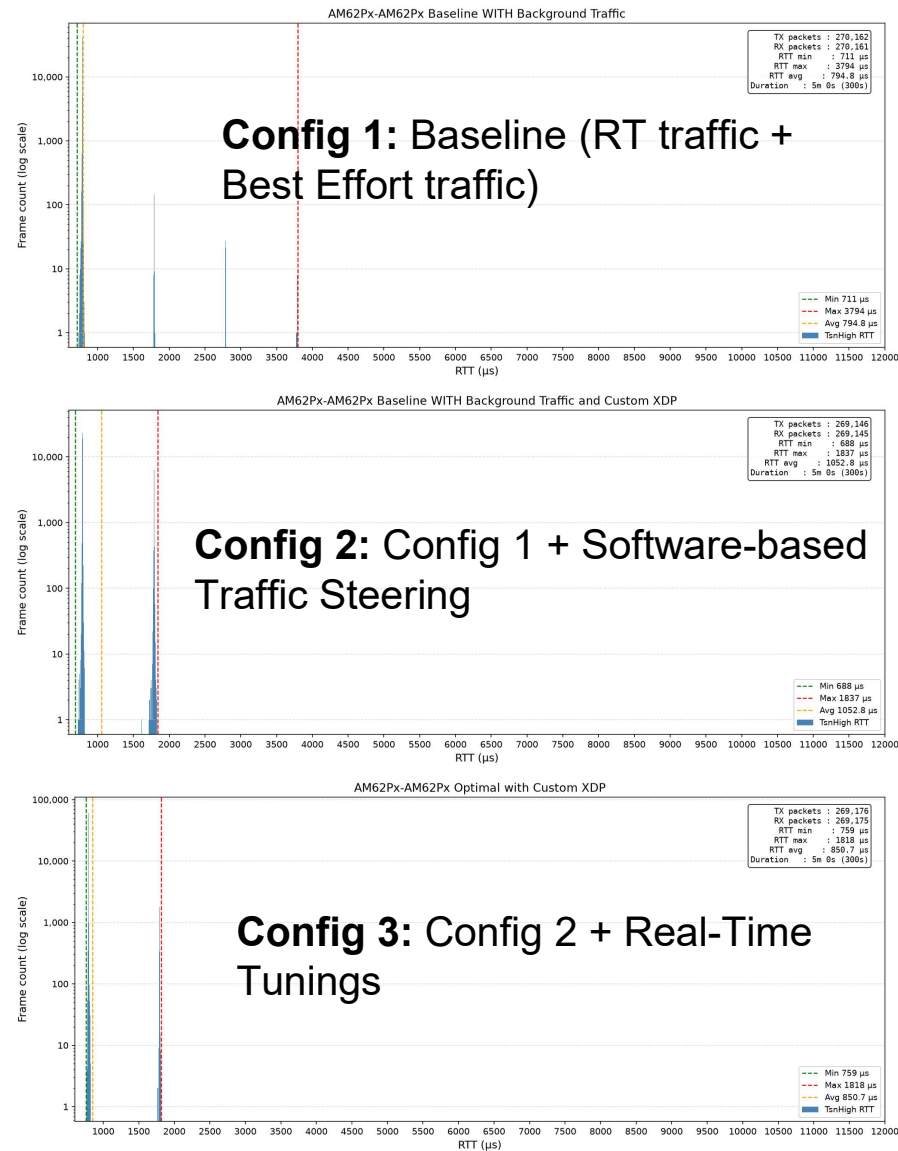
Final Comparison

Highlights

- Software-based traffic steering removed >2000us missed deadlines
- Simple real-time tunings reduced jitter by ~54% and reduced number of ~1000us missed deadlines

Lowlights

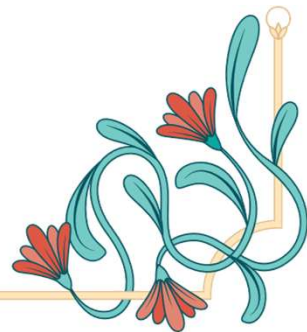
- Missed deadlines by ~1000us are not eliminated using a pure software-based traffic steering solution
- Hardware traffic steering/separation (e.g., NIC HW TX/RX queue separation) needed to fully eliminate missed deadlines



Takeaways

Question: “When hardware-based traffic separation is not feasible, can software-based traffic steering combined with Linux RT tuning reduce the interference from best-effort traffic and improve real-time performance?”

- BE traffic has large negative impact on RT traffic latency but not necessarily on jitter
- Software-based traffic steering is a solution to reduce RT traffic missed deadlines by > 2 cycles but not > 1 cycle
- Using dedicated CPU cores for critical tasks has positive impact on RT traffic latency and jitter
- Complete RT and BE traffic separation still needed from NIC HW TX/RX queues necessary to further reduce missed deadlines > 1 cycle



About us: TI Embedded Processing and Open Source



Upstream FIRST!



- ✓ Upstream FIRST mentality!
- ✓ Decades of contribution and collaboration
- ✓ Focus on long term, sustainable and quality products
- ✓ Upstream & opensource ecosystem in device architecture
- ✓ Ingrained culture to give back to the community



Q&A

Contact Us

Meghana Malladi (m-malladi@ti.com)

Daolin Qiu (d-qiu@ti.com)

Schuyler Patton (spatton@ti.com)



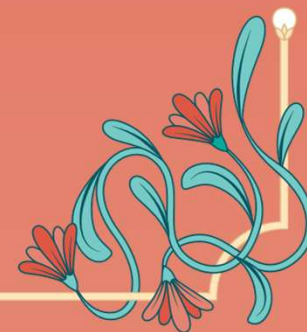
Learn more about TI products

<https://www.ti.com/processors>

<https://www.ti.com/tool/SK-AM62P-LP>

<https://www.ti.com/product/AM62P>

More info about TI software

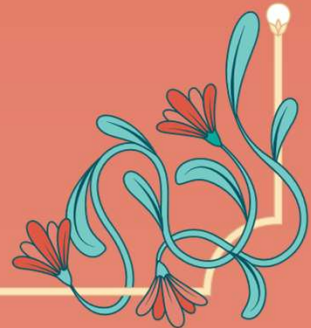


THE LINUX FOUNDATION



**EMBEDDED
LINUX
CONFERENCE**

EUROPE



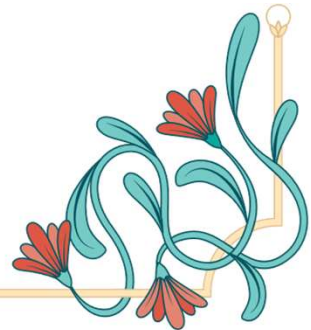


Appendix



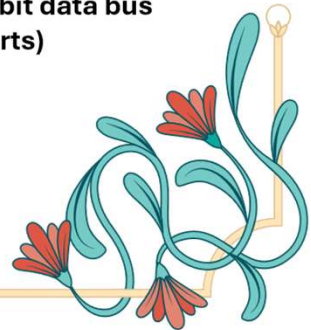
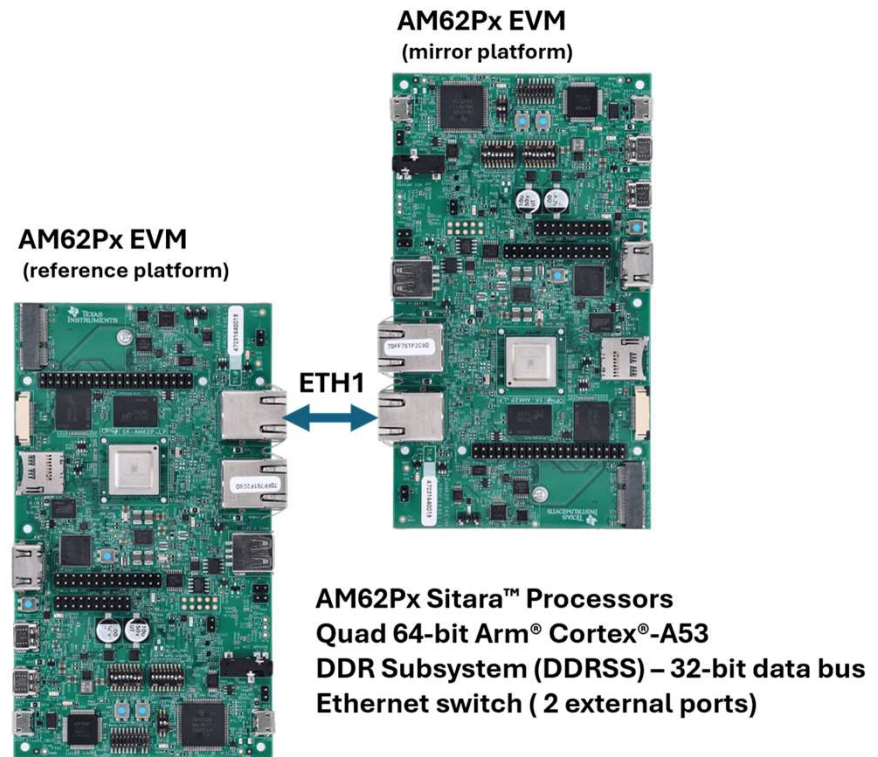
Next Steps

- Implement RX classifiers logic in TI NIC driver to filter incoming traffic based on VLANs (instead of MAC address) to separate HW RX/TX queues
- Test out theory of separating traffic through HW RX/TX queues eliminates ~1 cycle missed deadlines
- Investigate traffic separation from TX perspective so that entire round trip data path is separation to different cores



Why we chose TI AM62Px SoC?

- From the current Texas Instruments Sitara SoC family, the AM62Px has the best overall Linux cyclicttest performance at ~47us max interrupt latency
- Cyclicttest is a Linux tool used to repeatedly measure the difference between a thread's intended wake-up time and the time at which it actually wakes up from to provide statistics from the system's latencies (is a measure of RT performance)
- AM62Px offers more cores and higher DDR bitrate out of the SoC family that gives advantages when it comes to RT-Linux tuning



Verifying Traffic Steering

```
root@sk-am62p:~# ps -eo pid,psr,comm | grep 'cpumap/2'  
1317  2 cpumap/2/map:12
```

Baseline mpstat Average (per CPU)

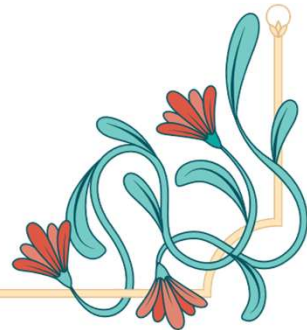
CPU	USR	SYS	SOFT	IRQ	IDLE
CPU0	1.95%	4.53%	39.33%	0.00%	54.20%
CPU1	10.32%	1.23%	0.84%	0.00%	87.61%
CPU2	12.89%	48.87%	32.11%	0.00%	6.14%
CPU3	55.83%	1.41%	0.00%	0.00%	42.75%

CPUMAP mpstat Average (per CPU)

CPU	USR	SYS	SOFT	IRQ	IDLE
CPU0	1.00%	5.50%	51.06%	0.00%	42.44%
CPU1	2.56%	20.21%	26.95%	0.00%	50.27%
CPU2	9.97%	78.29%	0.72%	0.00%	11.02%
CPU3	10.51%	2.90%	0.99%	0.00%	85.59%

CPU2: Shift in %sys from 48.87% to 78.29%

%sys: CPU usage at system/kernel level



Time Synchronization

- PTP (Precision Time Protocol, IEEE 1588) synchronizes clocks across a network to sub-microsecond accuracy.
- gPTP (Generalized PTP, IEEE 802.1AS) is the profile used for Time-Sensitive Networking (TSN) — it underlies deterministic, low-latency Ethernet communication.
- One clock in the network is elected the Grandmaster (GM); all other clocks synchronize their local time to it via periodic Sync / Follow_Up and Announce messages.

Master / Slave Roles in This Setup

Determined by the Best Master Clock Algorithm (BMCA) comparing priority1

gPTP.cfg — Reference Platform

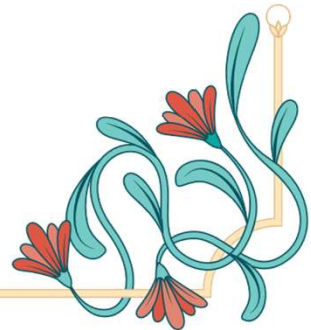
Role: MASTER (Grandmaster)

- priority1 = 100 (lower value → wins BMCA)
- priority2 = 248
- gmCapable = 1

mirror_gPTP.cfg — Mirror Platform

Role: SLAVE

- priority1 = 240 (higher value → loses BMCA)
- priority2 = 248
- gmCapable = 1 (capable, but not elected)



XDP + CPUMAP

- What is XDP (eXpress Data Path) ?
 - Framework in Linux kernel which executes eBPF (extended Berkeley Packet Filter) programs at the earliest point in the network driver
 - Instead of processing the packet in the networking stack, eBPF program parses the packet and passes a verdict on what to do with the packet
 - How this helps ?
 - Eliminates the need to pass packet to networking stack
 - Early decision made by the eBPF program
 - We will focus on XDP_REDIRECT verdict for our investigation
 - This verdict causes the raw packet frame to be redirected to different destinations
 - The CPUMAP tells which CPU to redirect to and run which kthread on the destination CPU

