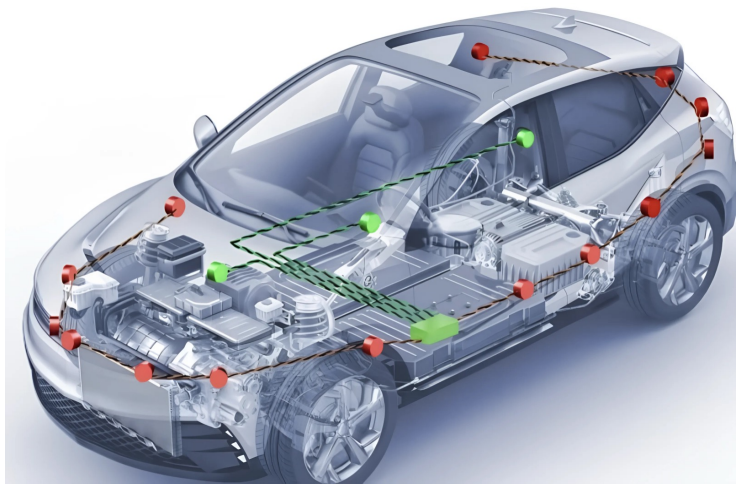
KeepON: Supporting Deterministic Traffic on Standard NICs

Chuanyu Xue, Tianyu Zhang, Andrew Loveless, Song Han

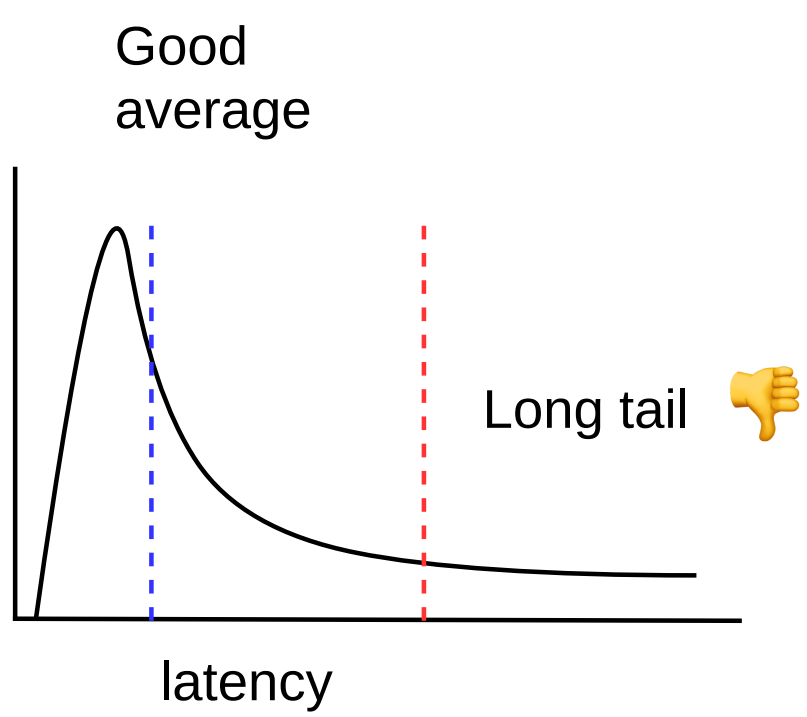
https://github.com/ChuanyuXue/KeepON-rpi

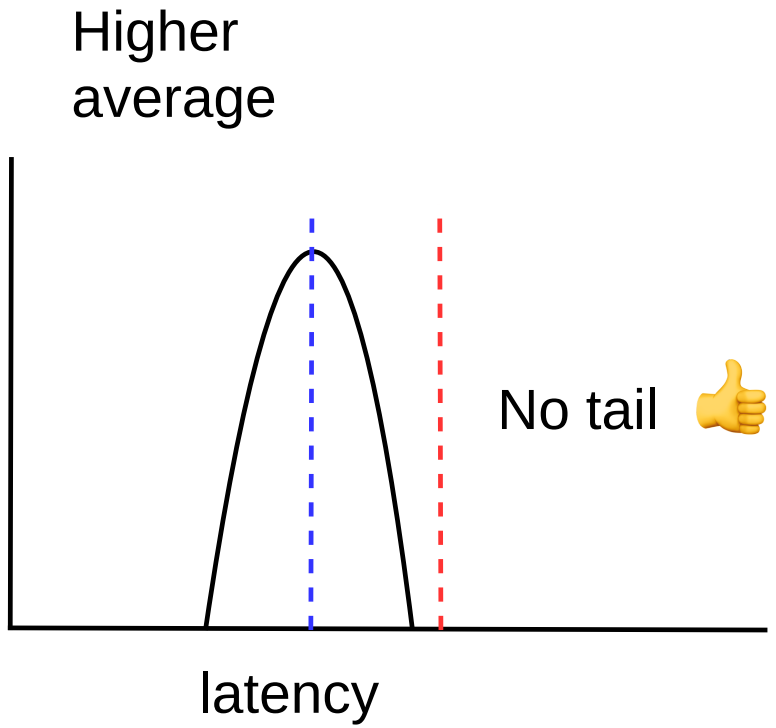
Networked mission-critical applications



- Sensors, actuators, and controllers are inter-connected through real-time network(s).
- Functional correctness and temporal correctness

Real-time networks





- The network must guarantee packet delivery with predictable latency and jitter.

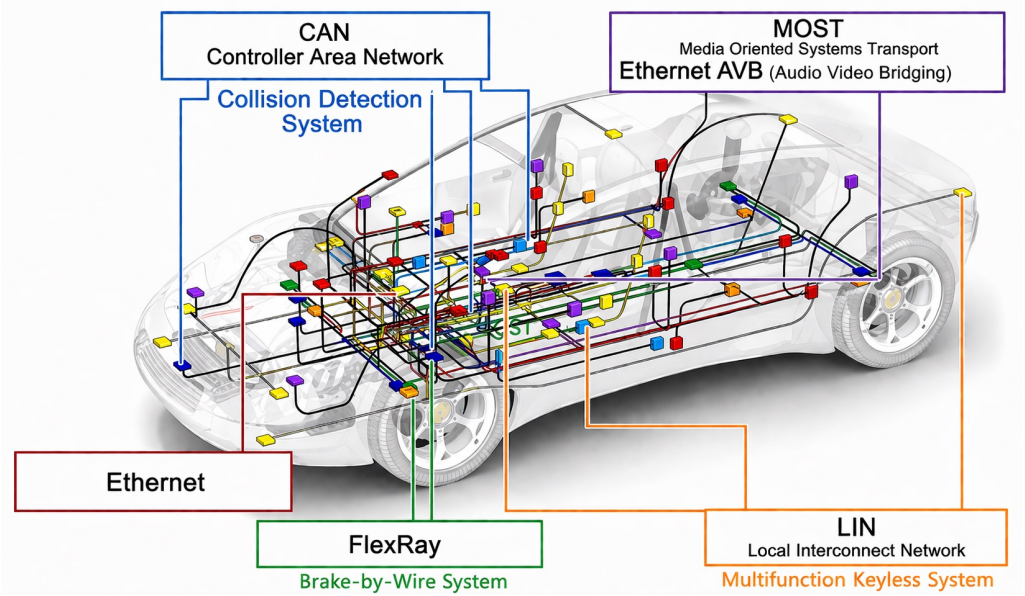
Legacy field bus → converged network

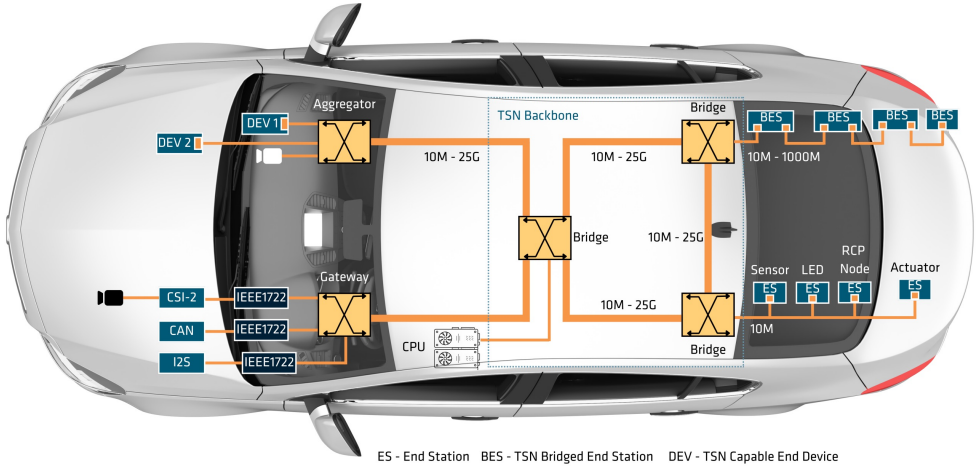
Domain-specific bus

- High cost (many cables)
- Low bandwidth (≤1Mbps)

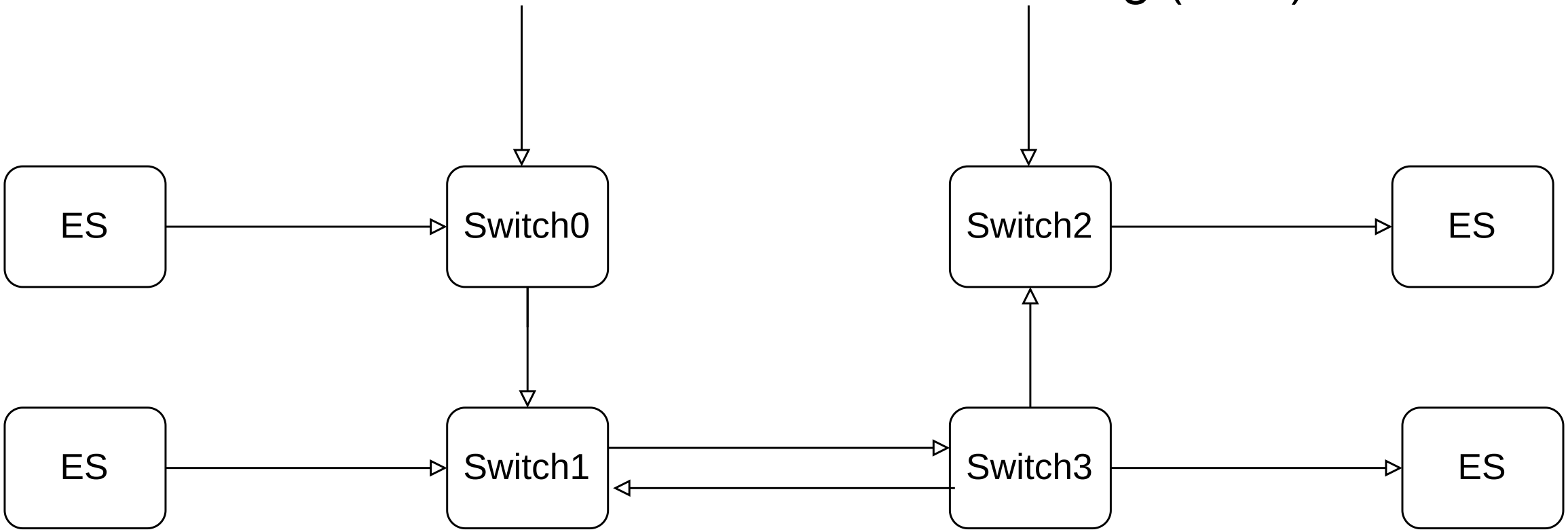
Ethernet

- Low cost (shared)
- High bandwidth (≥ 1G)



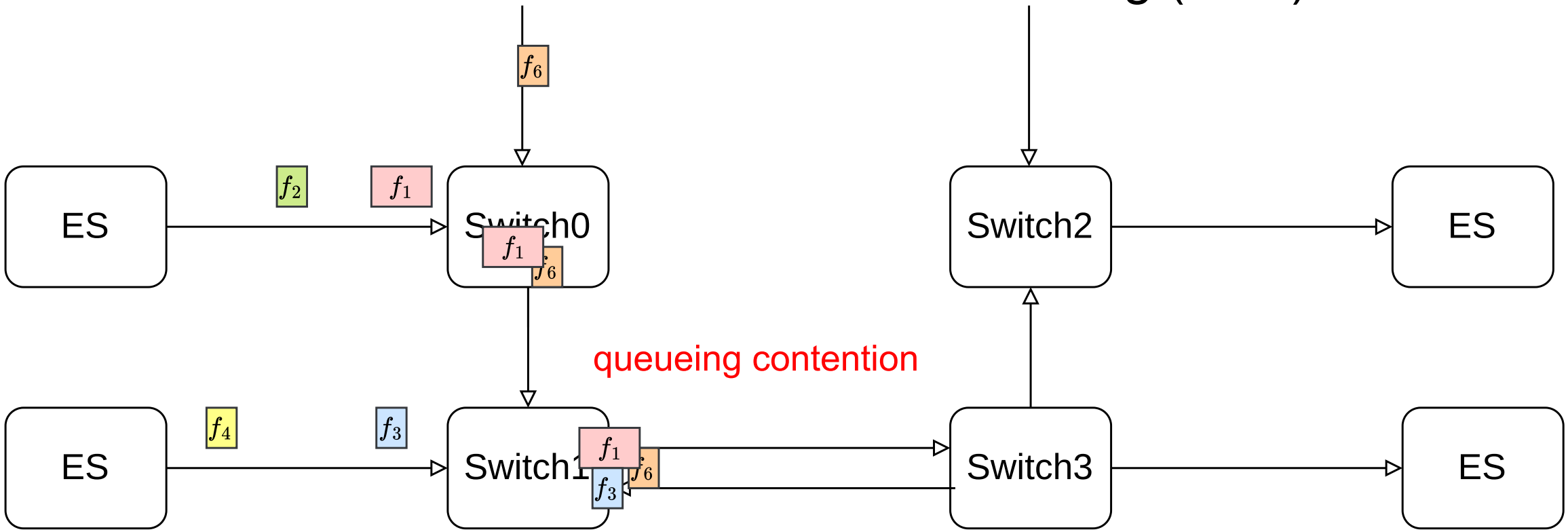


Standard Ethernet vs. Time-Sensitive Networking (TSN)



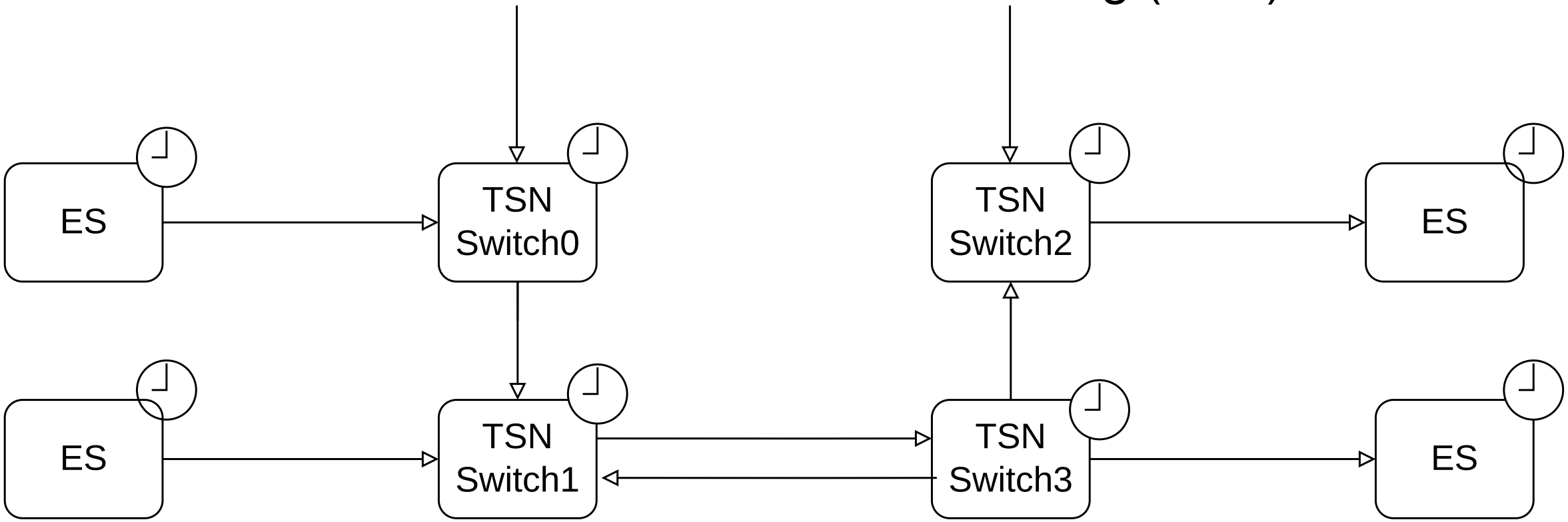
- Ethernet is best-effort
- Time-Sensitive Networking

Standard Ethernet vs. Time-Sensitive Networking (TSN)



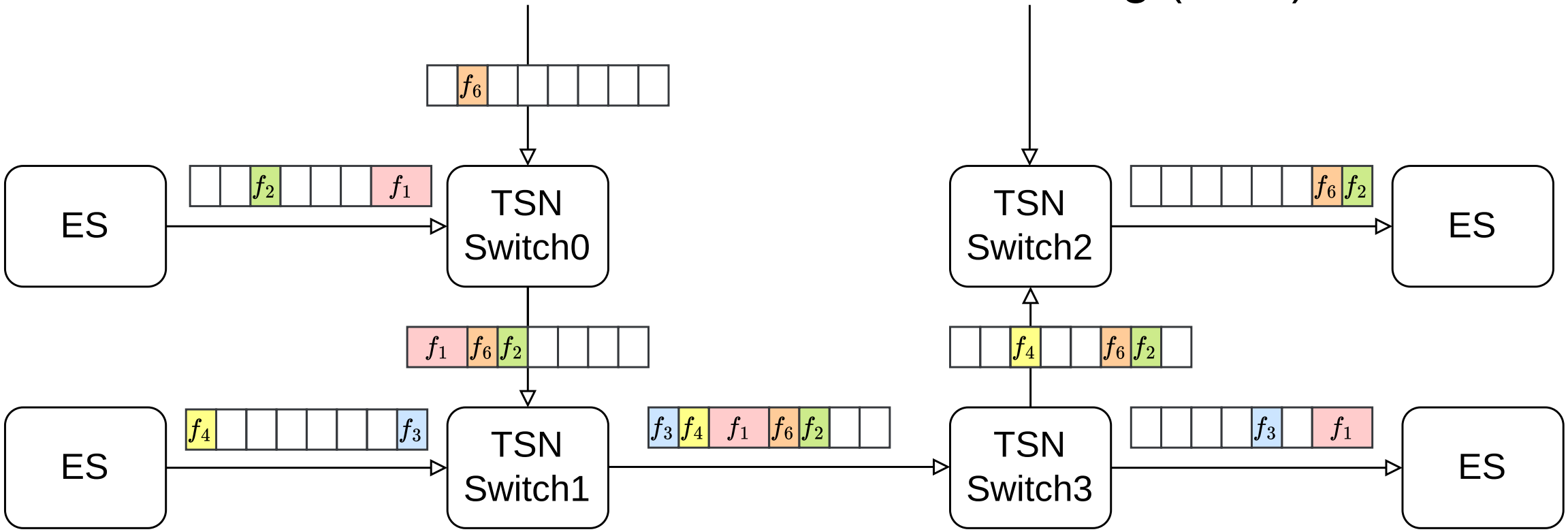
- Ethernet is best-effort → random queuing delay
- Time-Sensitive Networking

Standard Ethernet vs. Time-Sensitive Networking (TSN)



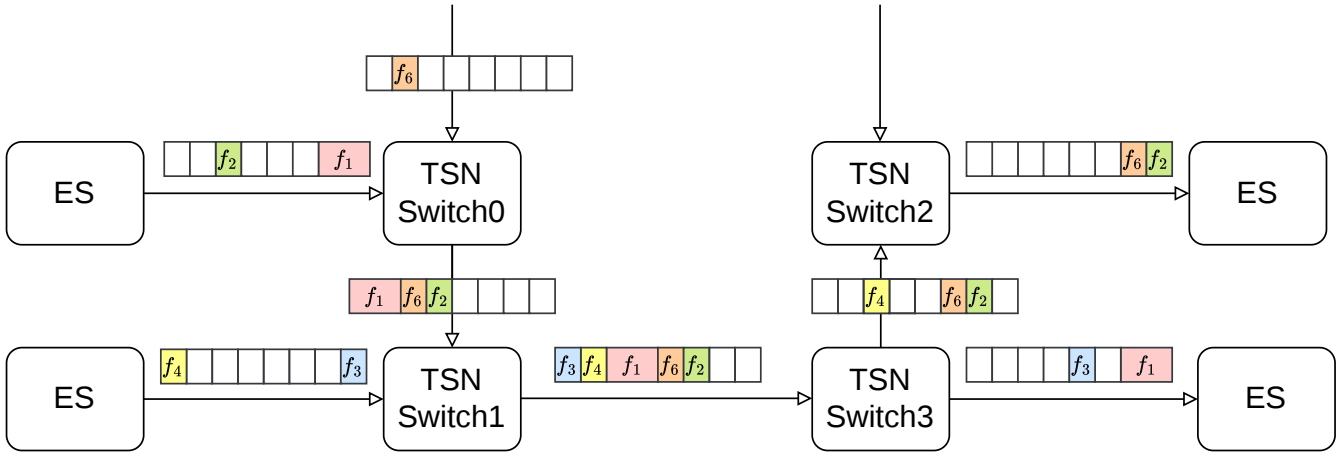
- Ethernet is best-effort → random queuing delay
- Time-Sensitive Networking
  - Network-wide time synchronization (IEEE 802.1AS)
  - Time-aware scheduling (IEEE 802.1Qbv)

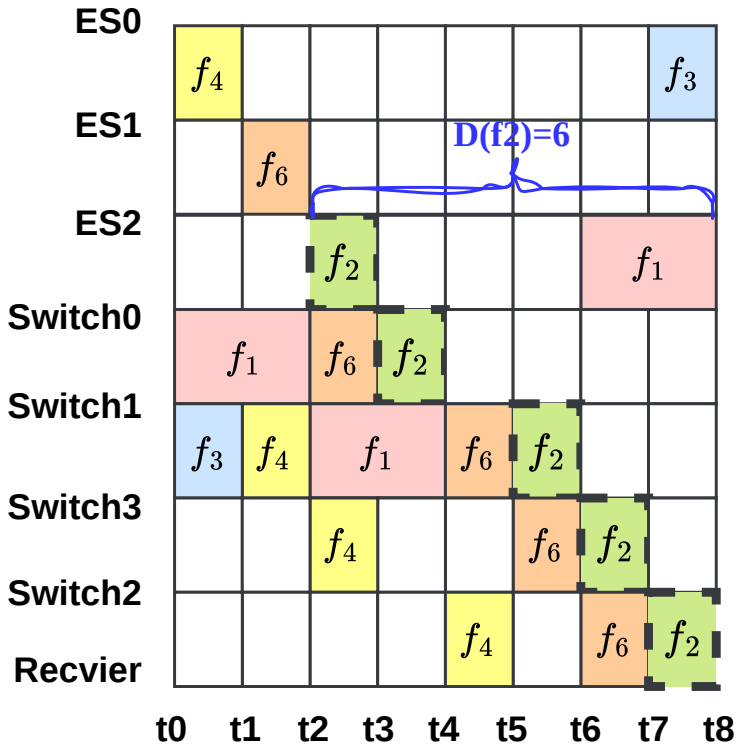
Standard Ethernet vs. Time-Sensitive Networking (TSN)



- Ethernet is best-effort → random queuing delay
- Time-Sensitive Networking
  - Network-wide time synchronization (IEEE 802.1AS)
  - Time-aware scheduling (IEEE 802.1Qbv)

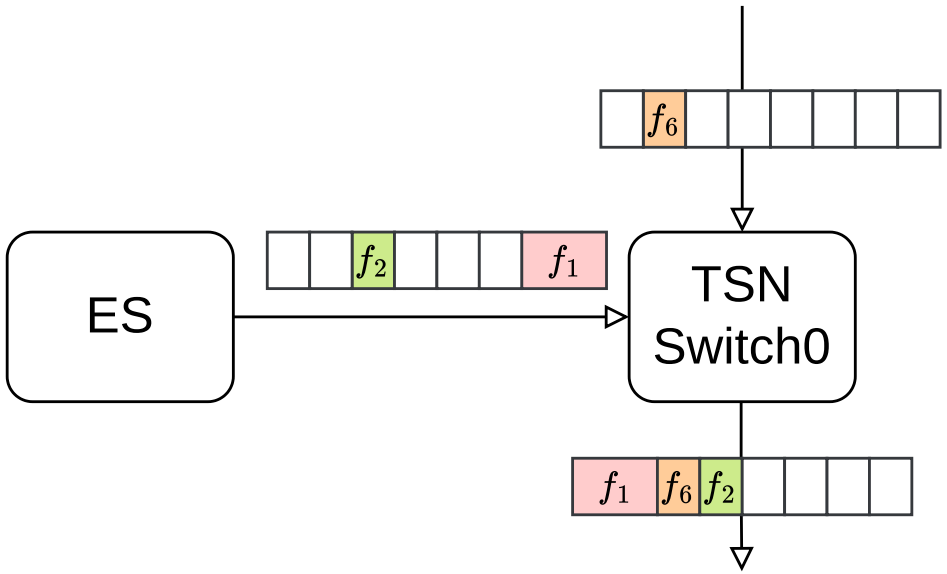
Standard Ethernet vs. Time-Sensitive Networking (TSN)

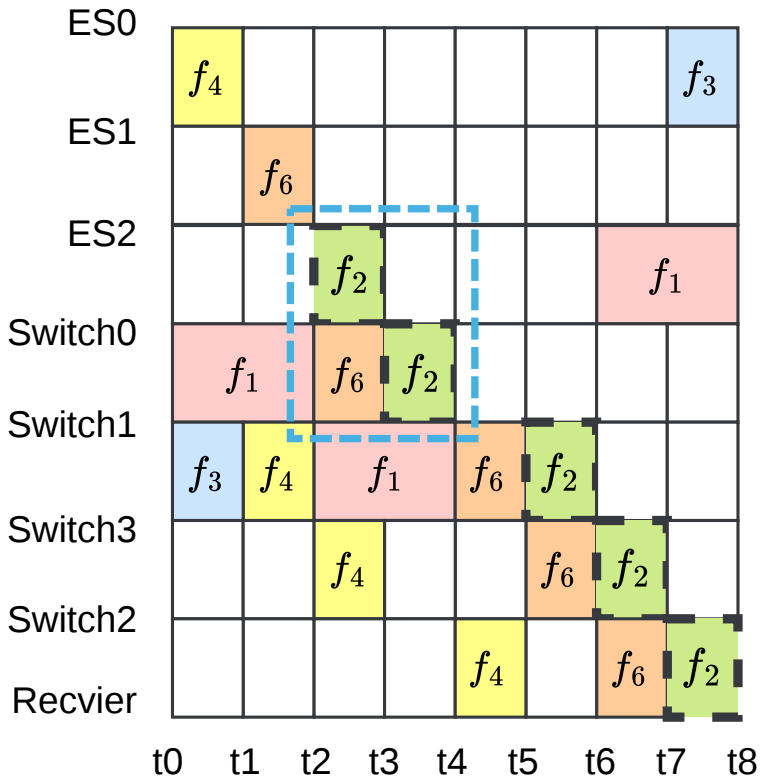




- Ethernet is best-effort → random queuing delay
- Time-Sensitive Networking
  - Network-wide time synchronization (IEEE 802.1AS)
  - Time-aware scheduling (IEEE 802.1Qbv)

End-station jitter breaks TSN determinism

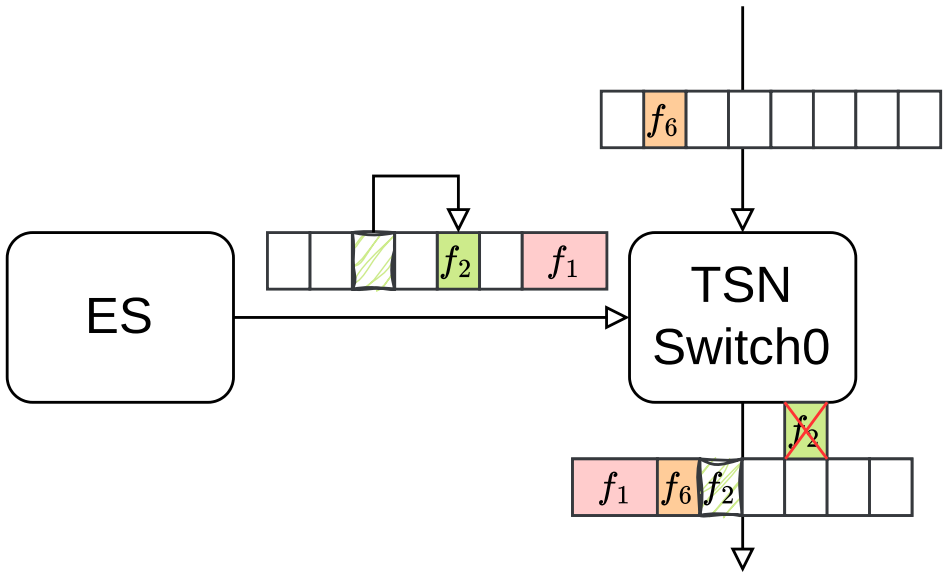


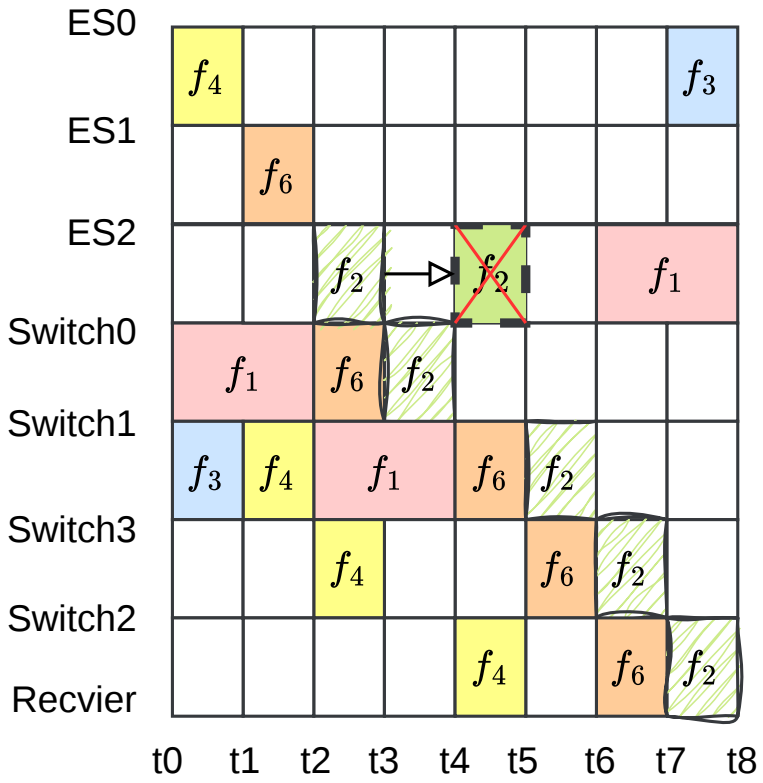


TSN assumes end-stations are also deterministic.

End-station jitter disrupts switch schedule → downstream flows miss deadlines.

End-station jitter breaks TSN determinism

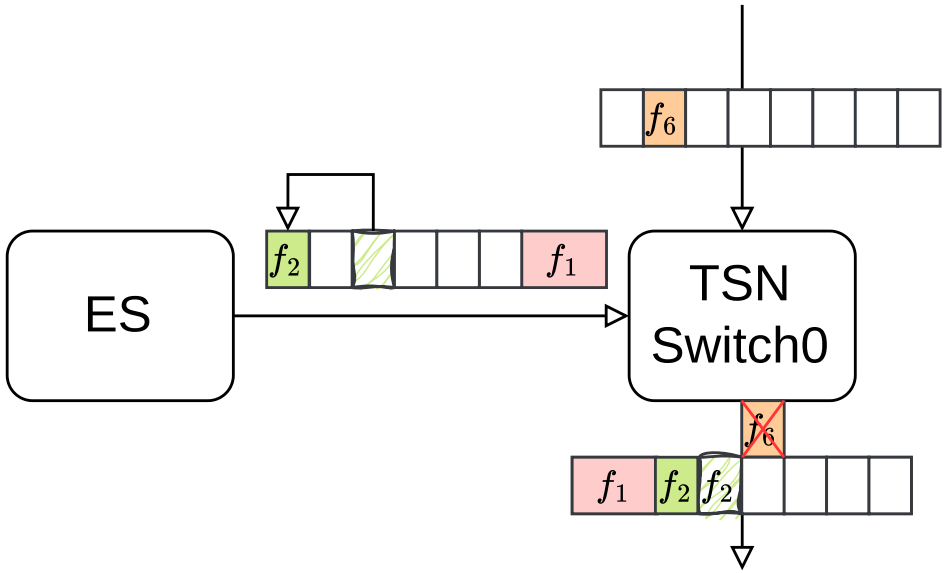


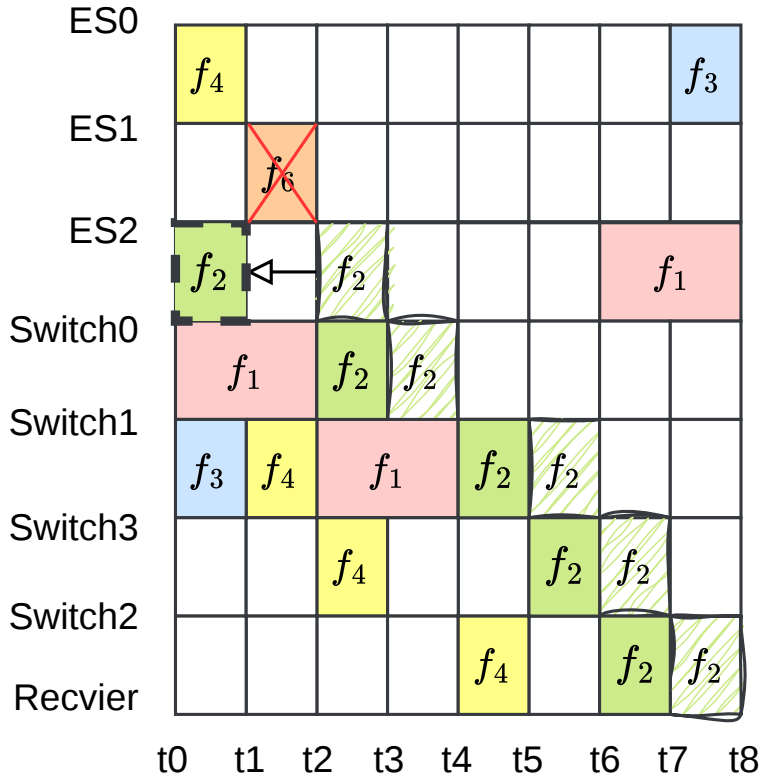


TSN assumes end-stations are also deterministic.

End-station jitter disrupts switch schedule → downstream flows miss deadlines.

End-station jitter breaks TSN determinism

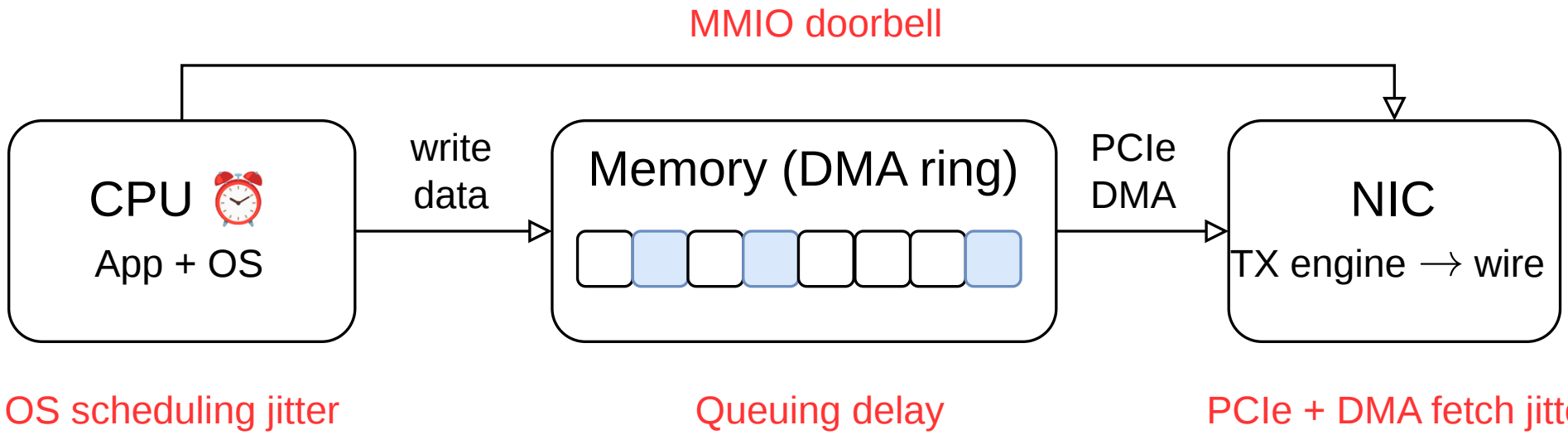




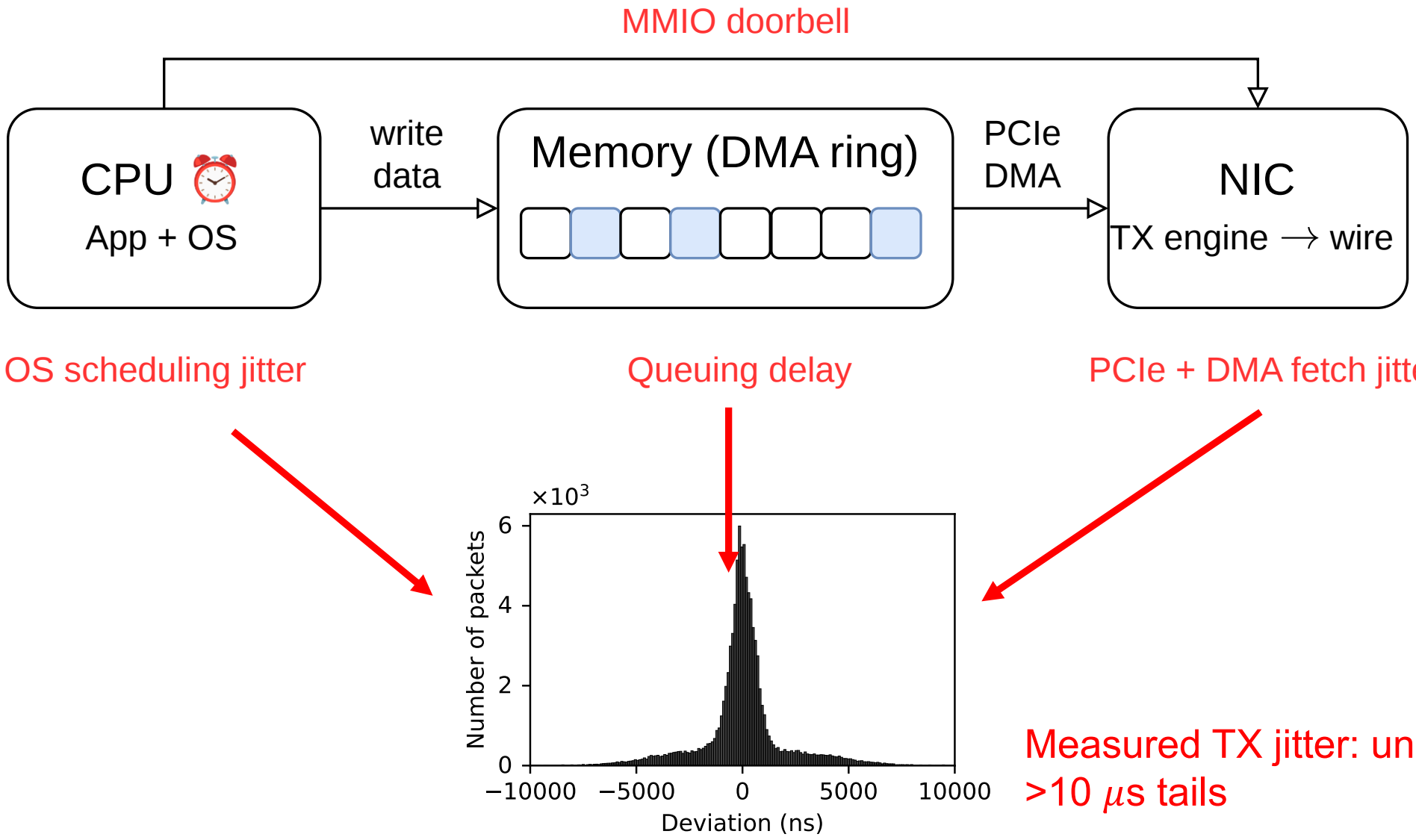
TSN assumes end-stations are also deterministic.

End-station jitter disrupts switch schedule → downstream flows miss deadlines.

Why is achieving end-station determinism hard?

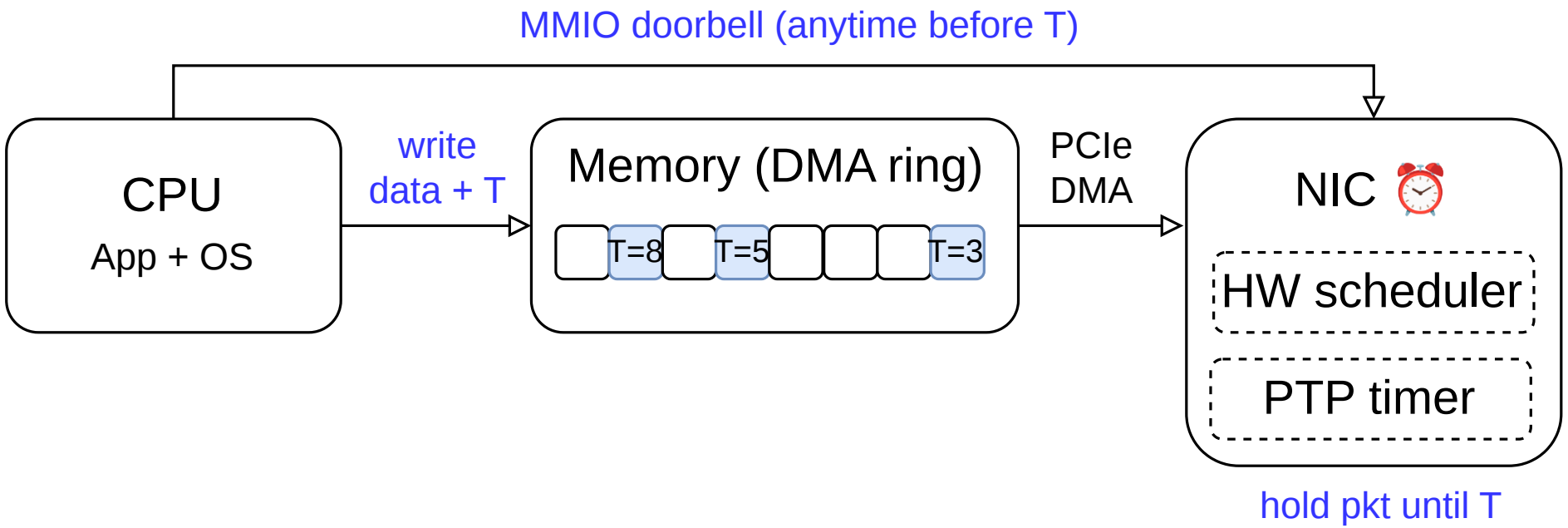


Why is achieving end-station determinism hard?



Measured TX jitter: unbounded, >10 μs tails

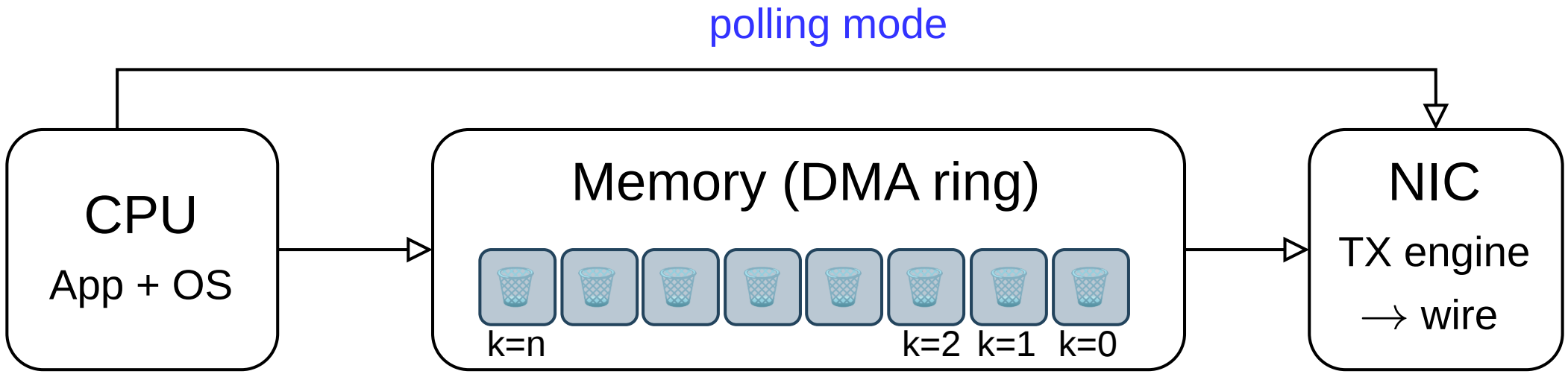
Existing solutions: hardware scheduling offloading (SO_TXTIME)



- Pro:
  - very accurate (ns-level)
- Cons:
  - limited support (13/341 drivers)
  - high cost 💰
  - legacy incompatible

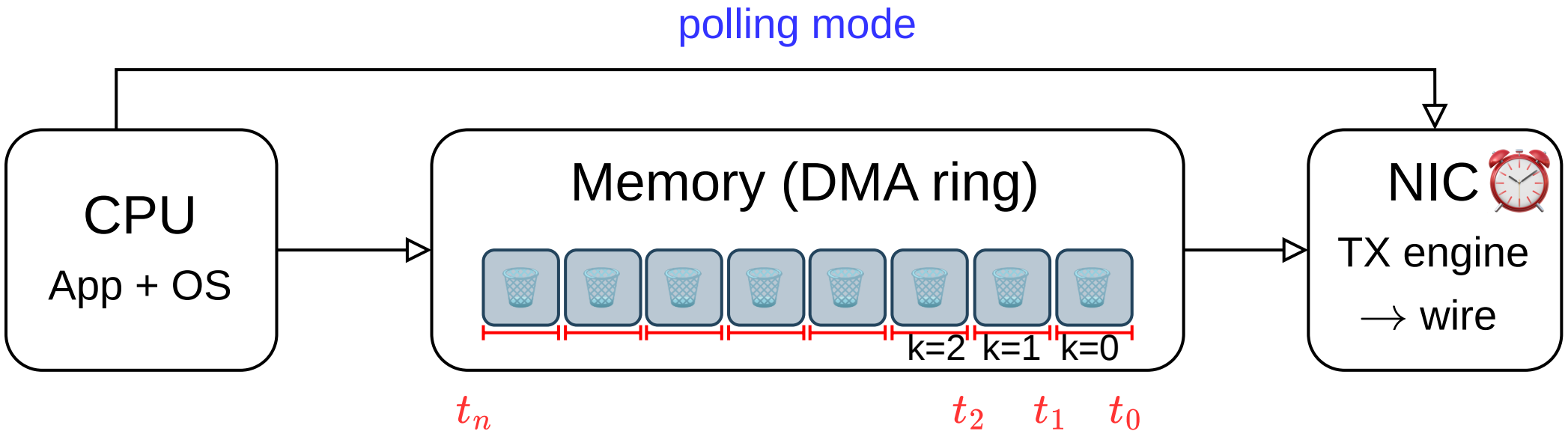
KeepON: purely software solution

- Challenge: hardware clock ⏰ is needed at NIC to bypass jitter at OS/PCIe.
- KeepON: construct clock ⏰ by fixed-size dummy packets at line-rate, packet count becomes tick



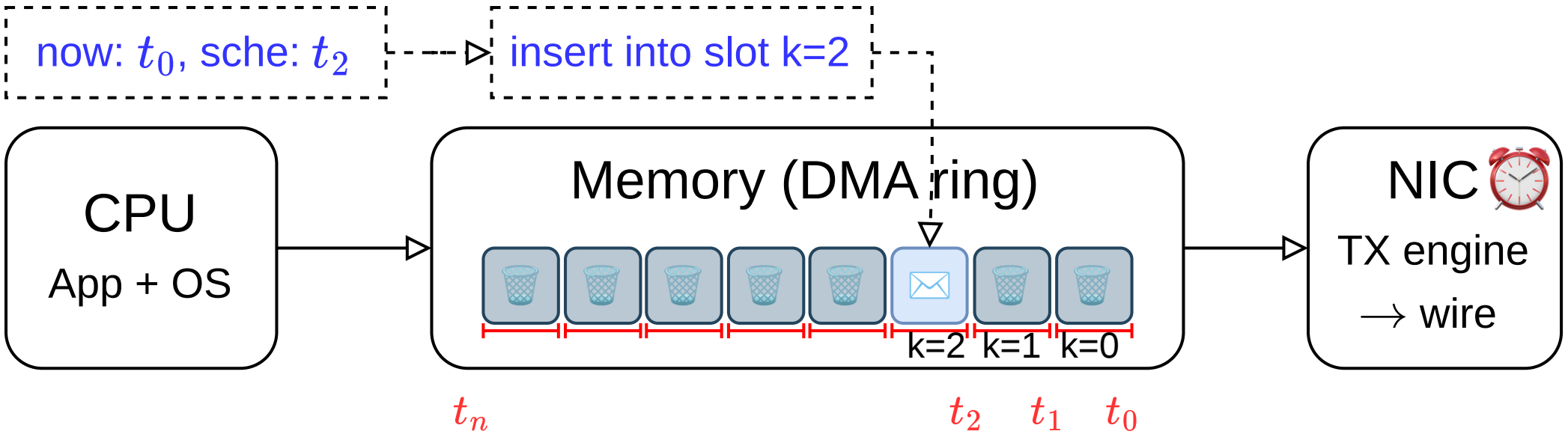
KeepON: purely software solution

- Challenge: hardware clock ⏰ is needed at NIC to bypass jitter at OS/PCIe.
- KeepON: construct clock ⏰ by fixed-size dummy packets at line-rate, packet count becomes tick



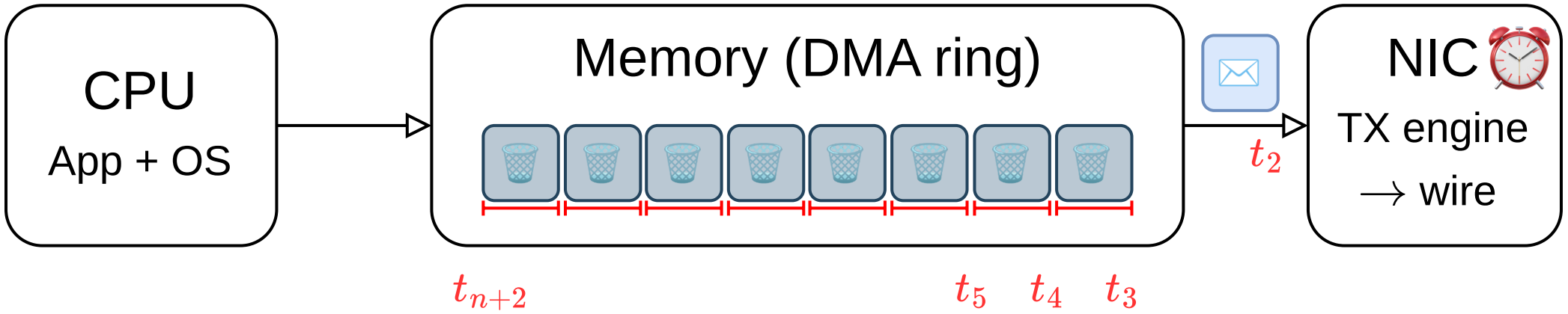
KeepON: purely software solution

- Challenge: hardware clock ⏰ is needed at NIC to bypass jitter at OS/PCIe.
- KeepON: construct clock ⏰ by fixed-size dummy packets at line-rate, packet count becomes tick

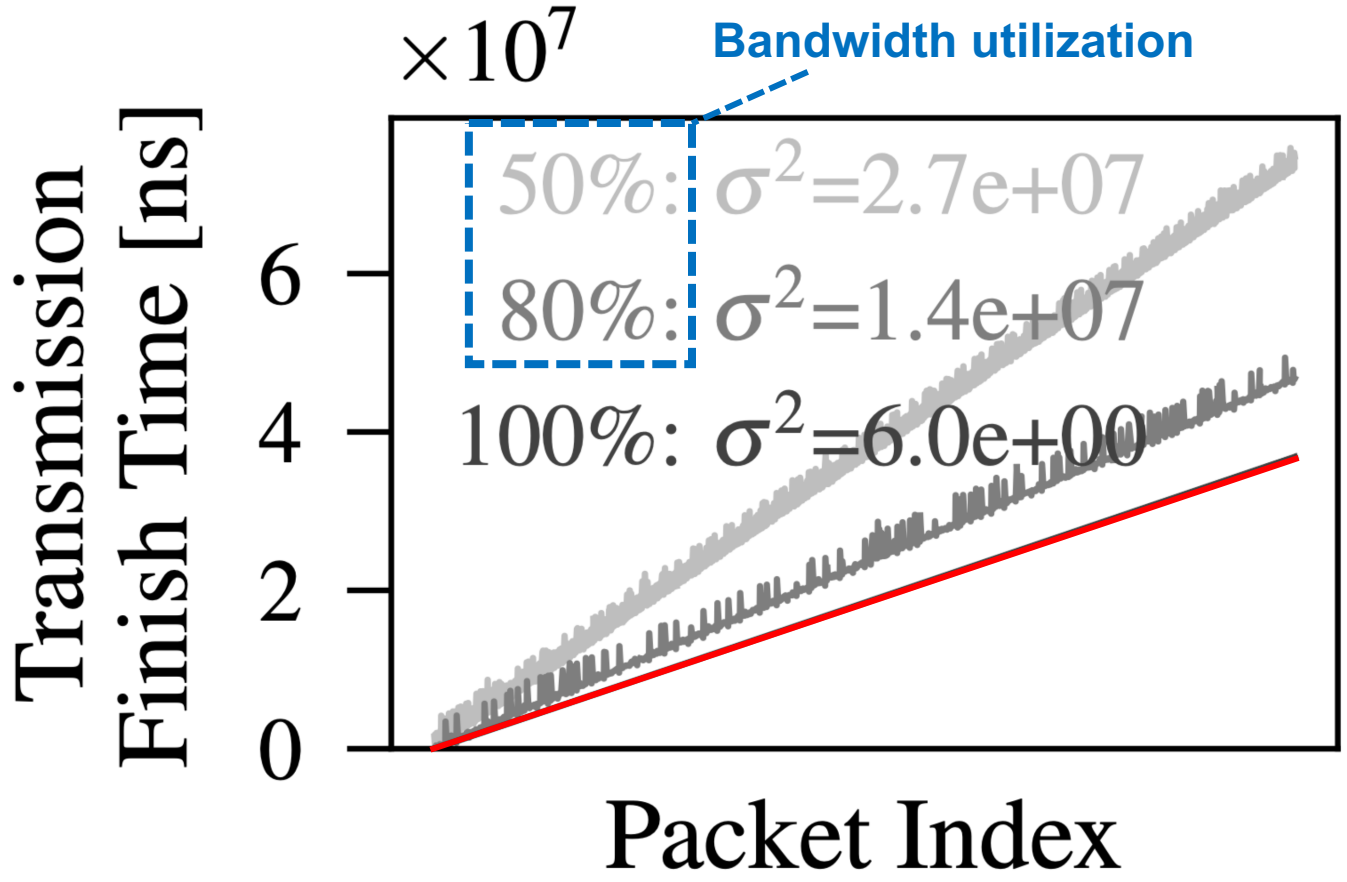


KeepON: purely software solution

- Challenge: hardware clock ⏰ is needed at NIC to bypass jitter at OS/PCIe.
- KeepON: construct clock ⏰ by fixed-size dummy packets at line-rate, packet count becomes tick

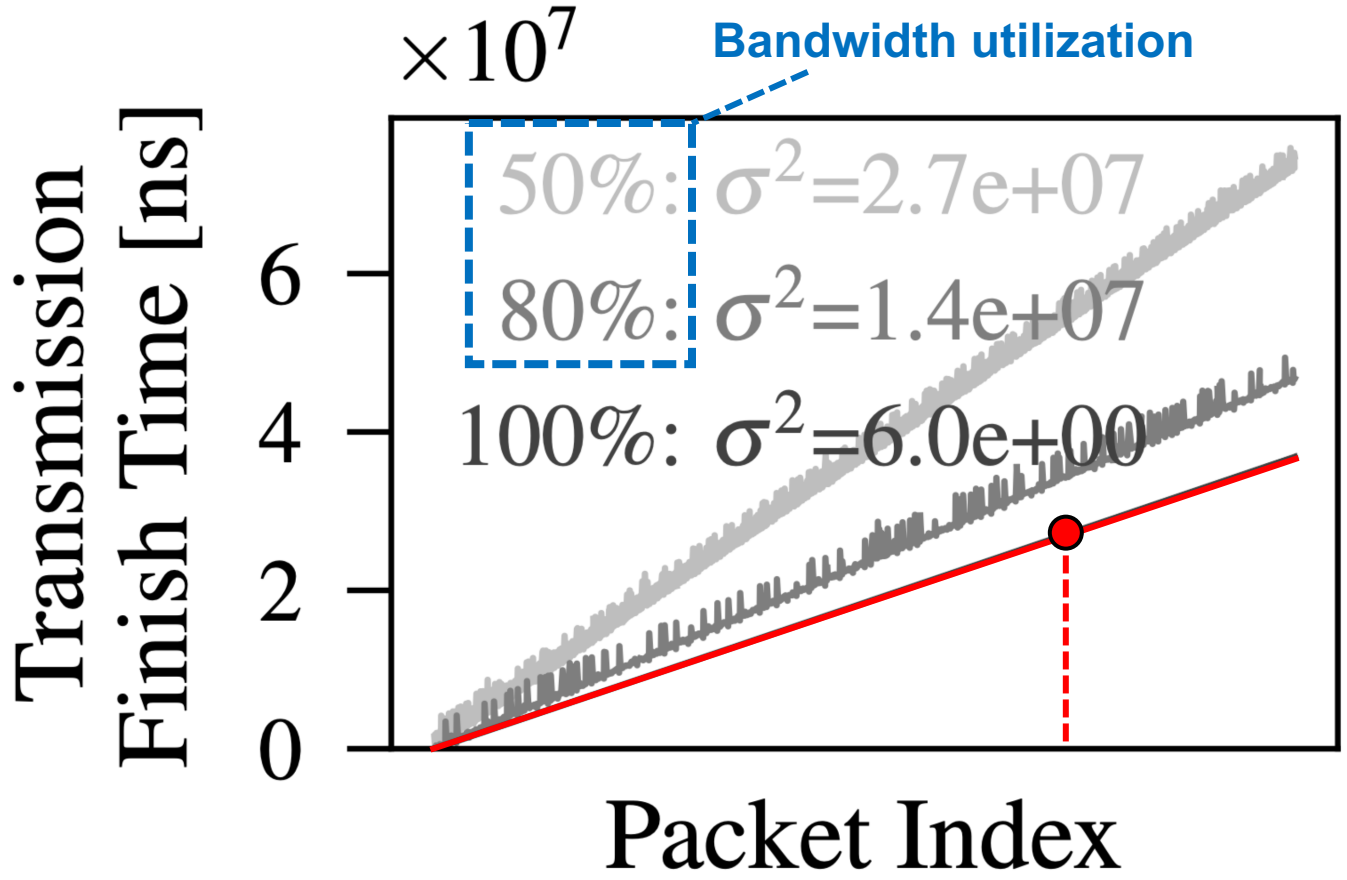


Observation 1: line-rate → predictable timing



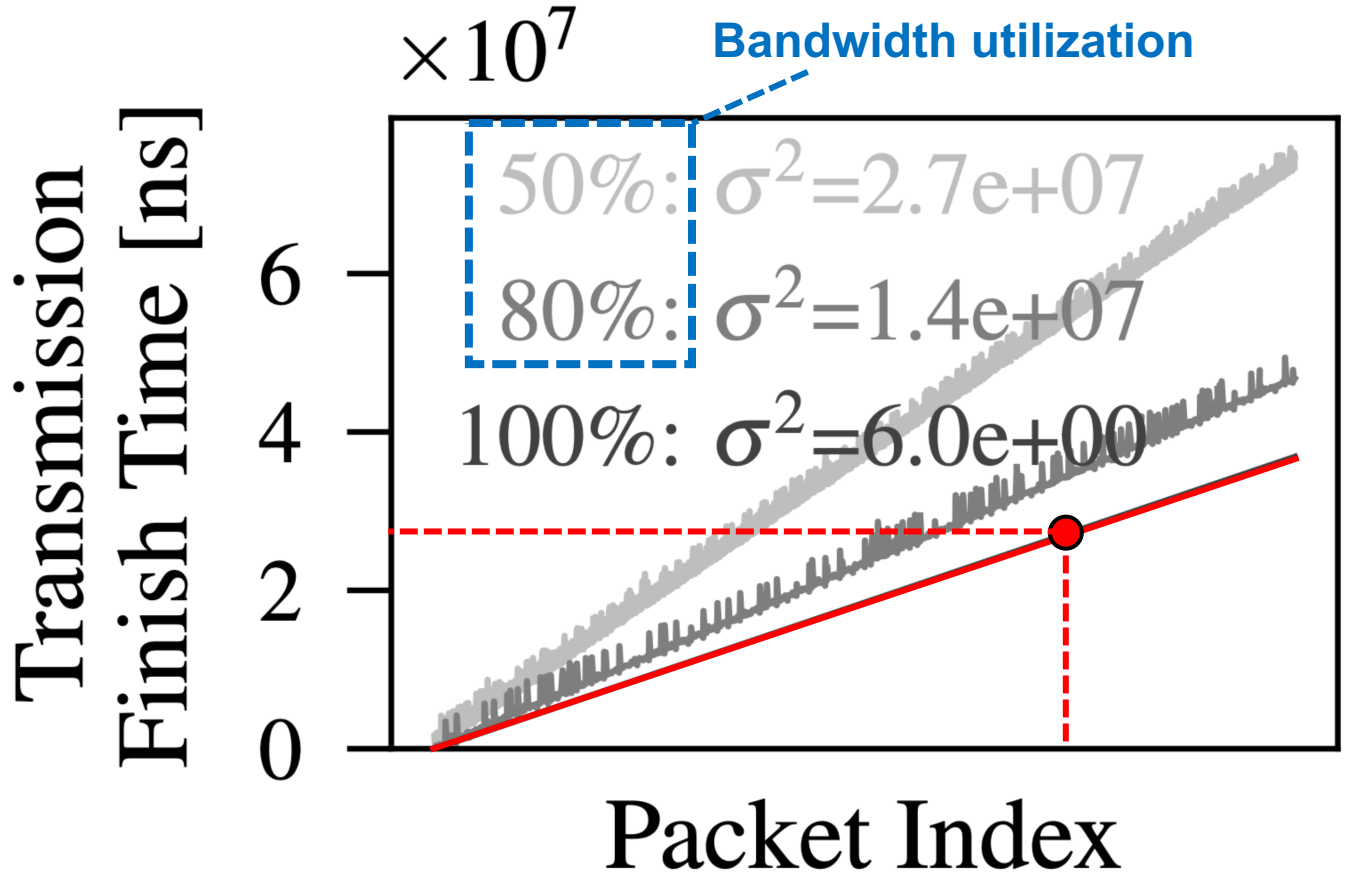

```
iperf -u -l 1500 -b 500M/800M/1G -e -i 1
```

Observation 1: line-rate → predictable timing



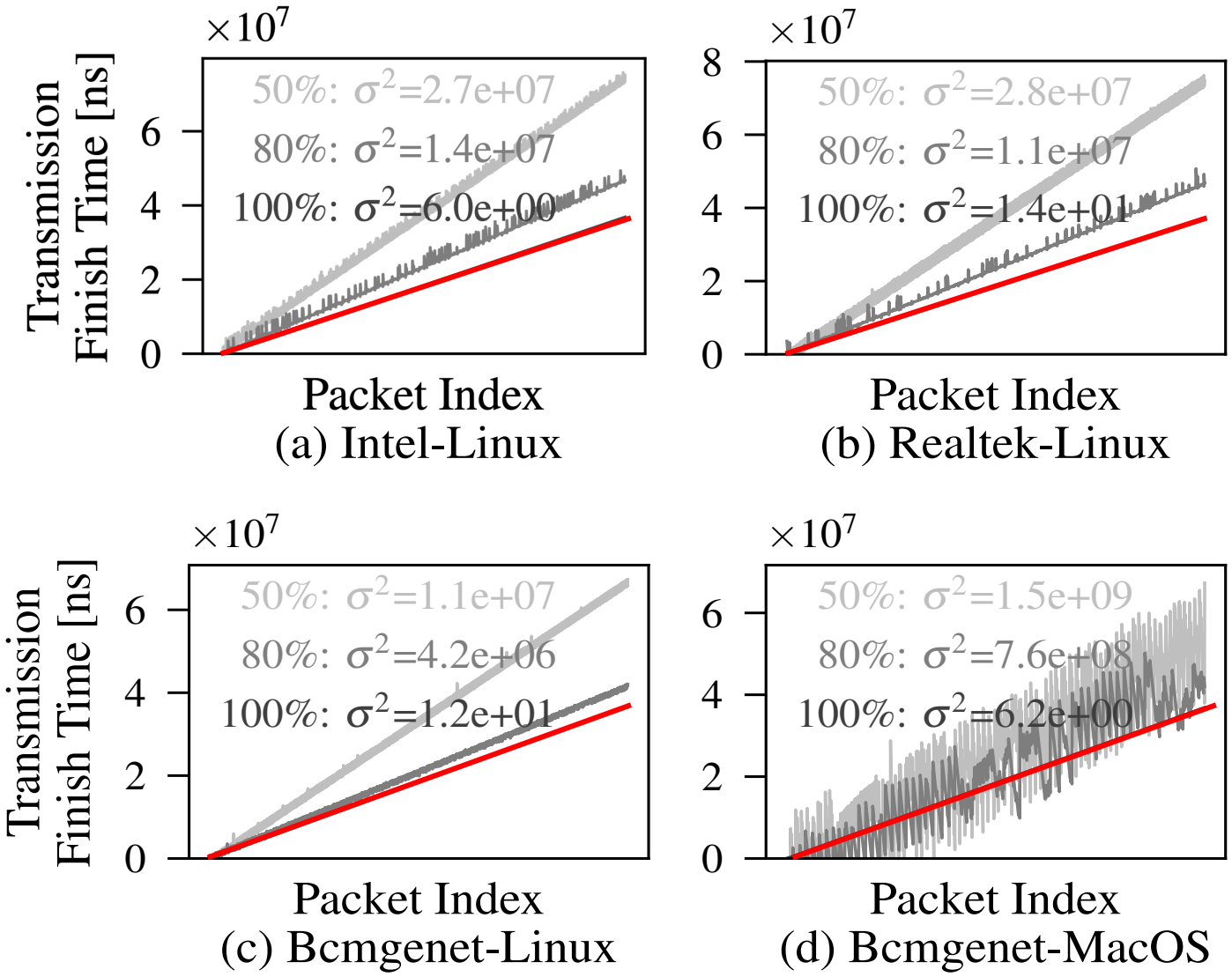

```
iperf -u -l 1500 -b 500M/800M/1G -e -i 1
```

Observation 1: line-rate → predictable timing



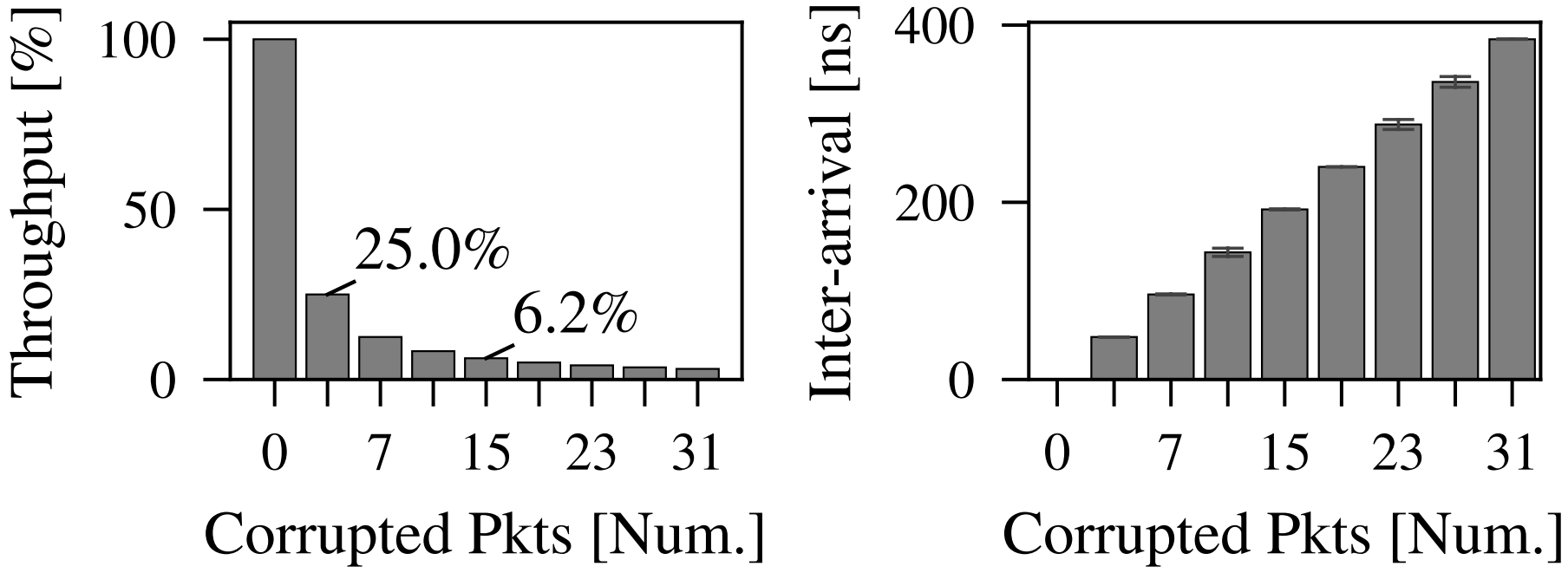

```
iperf -u -l 1500 -b 500M/800M/1G -e -i 1
```

Observation 1: line-rate → predictable timing

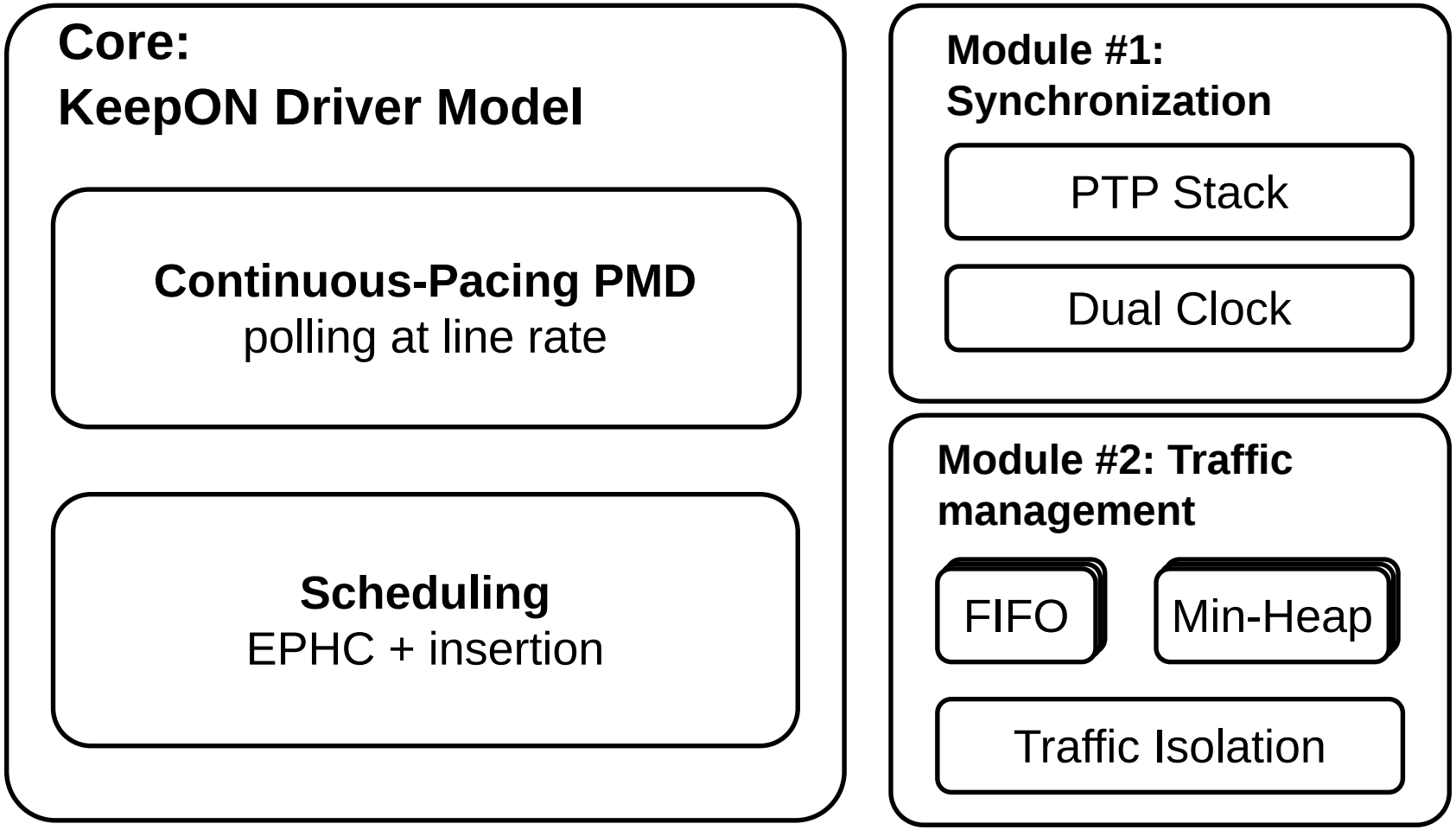


Observation 2: dummy packets are free

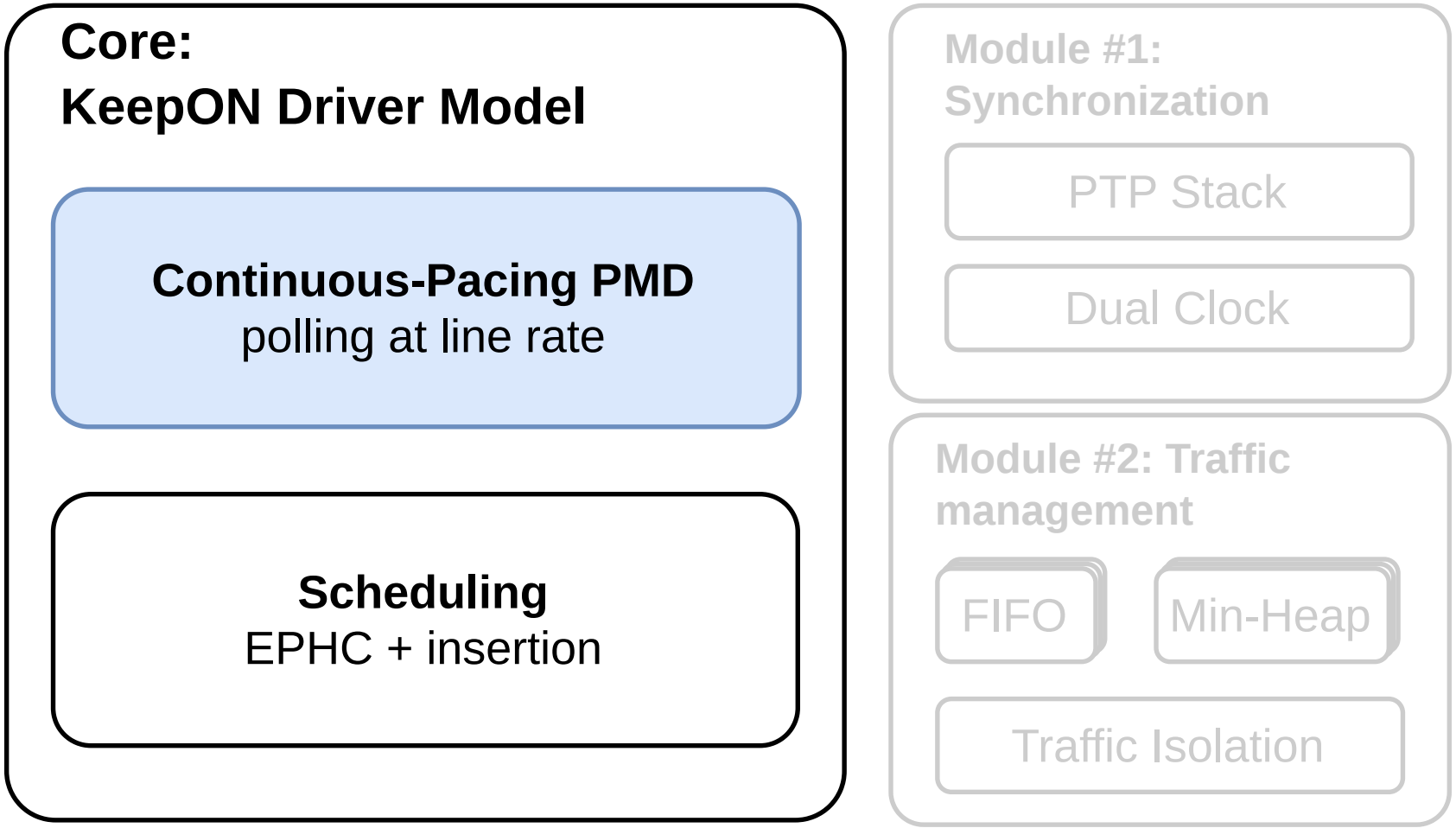
- Dummy packets: [header][payload][corrupted CRC❌]
  - Basic Ethernet function at layer-2, dropped by 1-st switch
  - No throughput overhead
  - No timing overhead



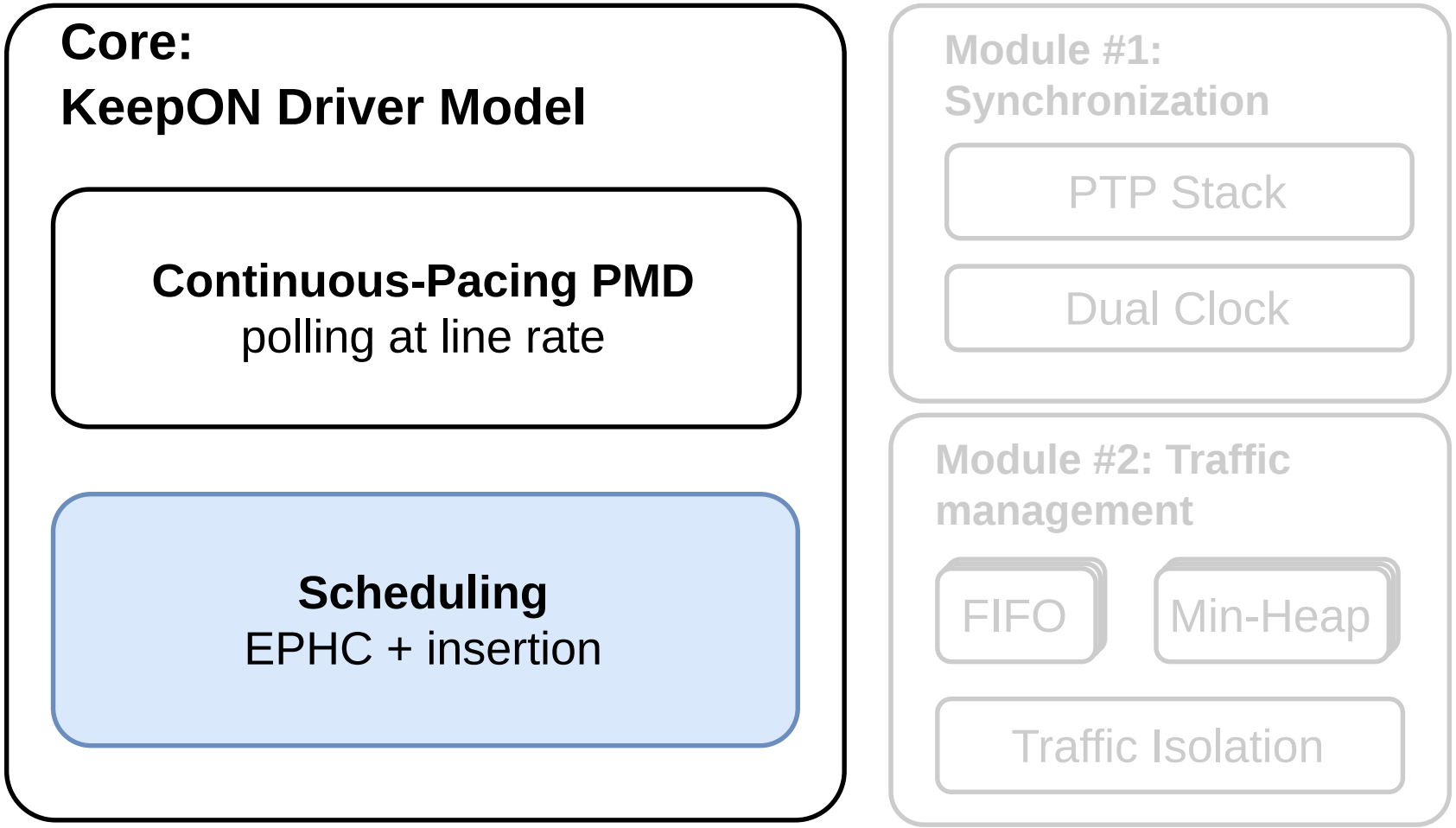
KeepON architecture



KeepON architecture

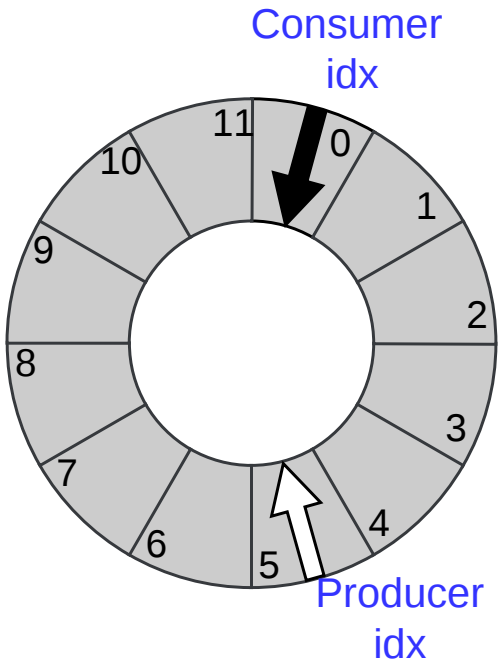


KeepON architecture



Continuous-Pacing Poll Mode Driver (CP-PMD)

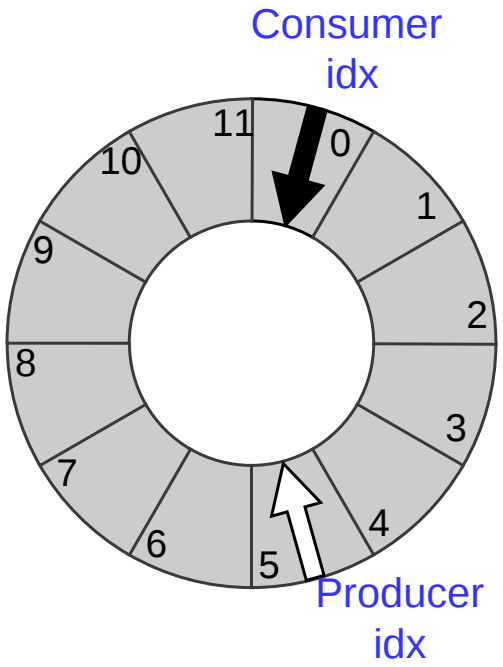
- How to keep line-rate efficiently? → Polling!



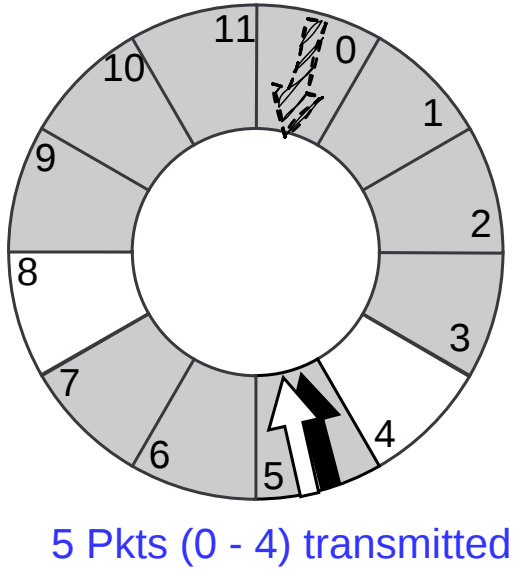
❶ Initialized with dummy packets

Continuous-Pacing Poll Mode Driver (CP-PMD)

- How to keep line-rate efficiently? → Polling!



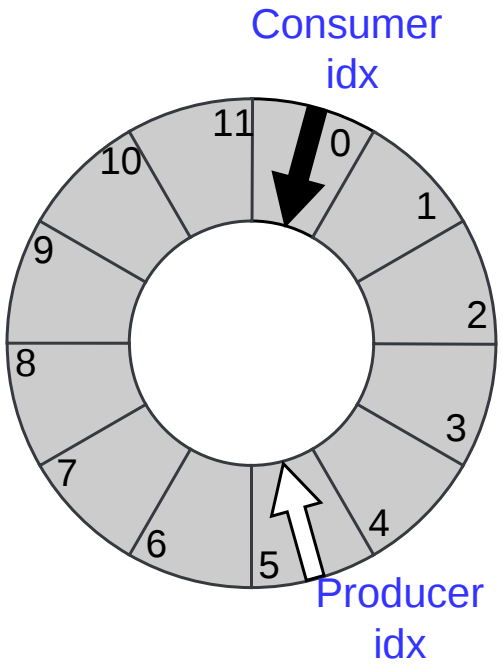
❶ Initialized with dummy packets



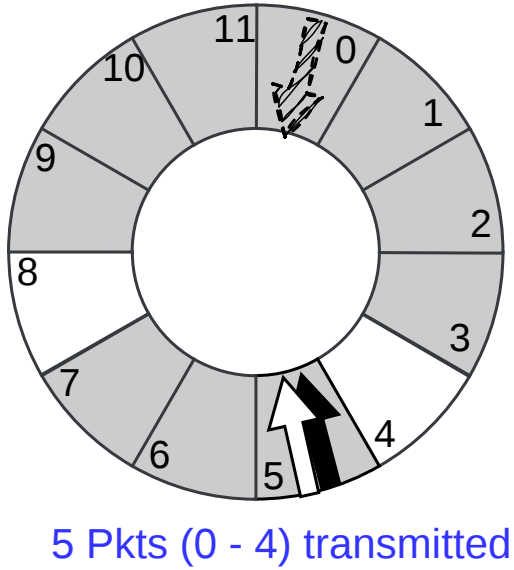
❷ Query consumer index

Continuous-Pacing Poll Mode Driver (CP-PMD)

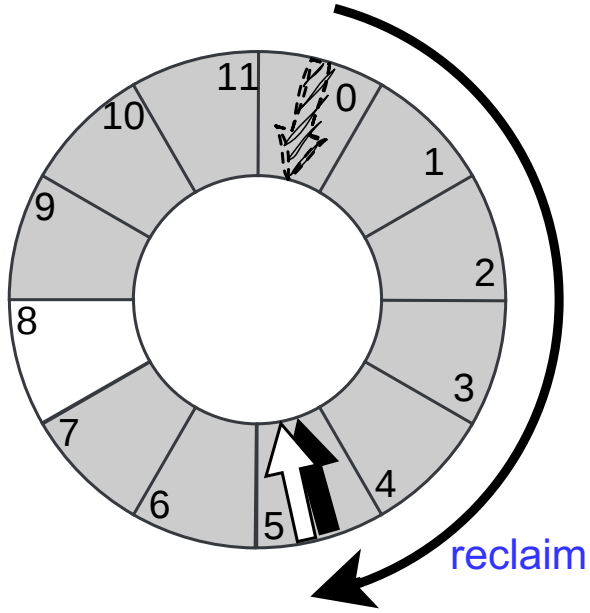
- How to keep line-rate efficiently? → Polling!



❶ Initialized with dummy packets



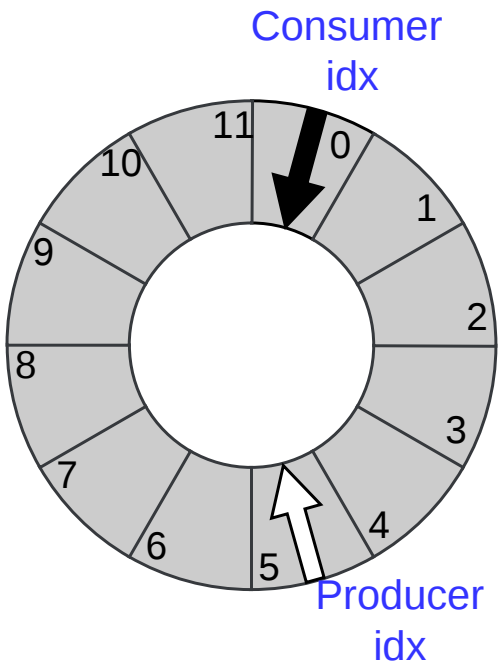
❷ Query consumer index



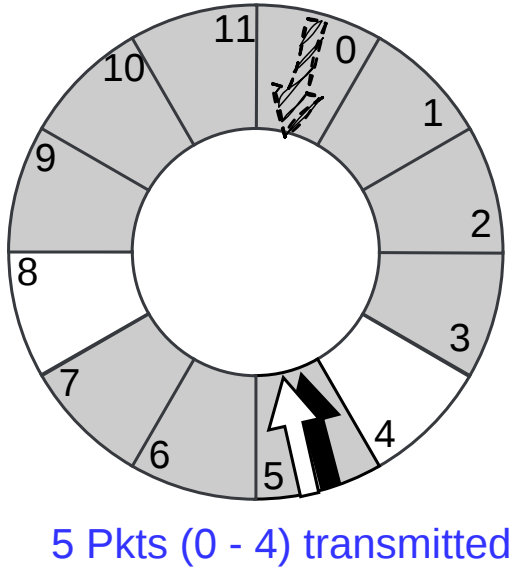
❸ Reclaim to dummy by overwrite wrong CRC code

Continuous-Pacing Poll Mode Driver (CP-PMD)

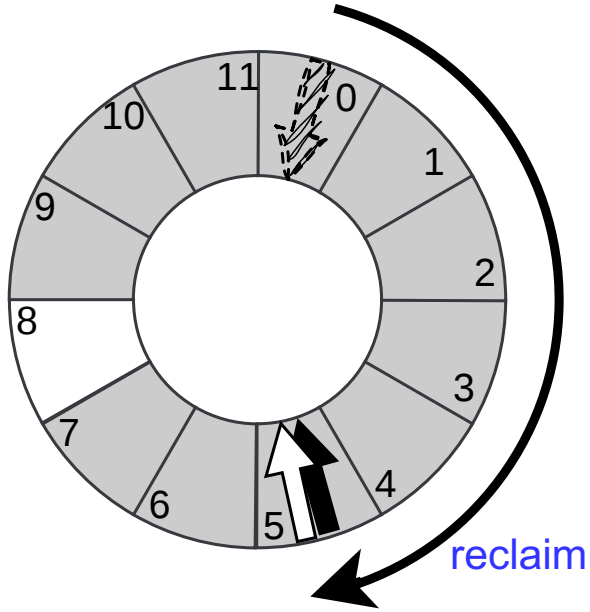
- How to keep line-rate efficiently? → Polling!
- Repeat ❷❸❹



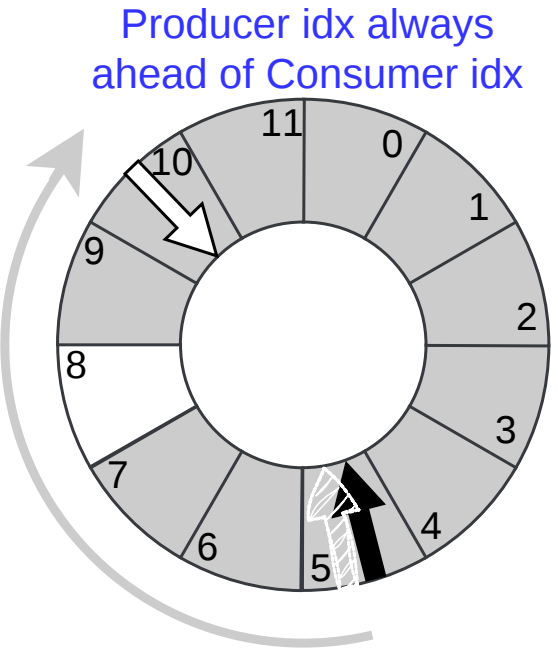
❶ Initialized with dummy packets



❷ Query consumer index



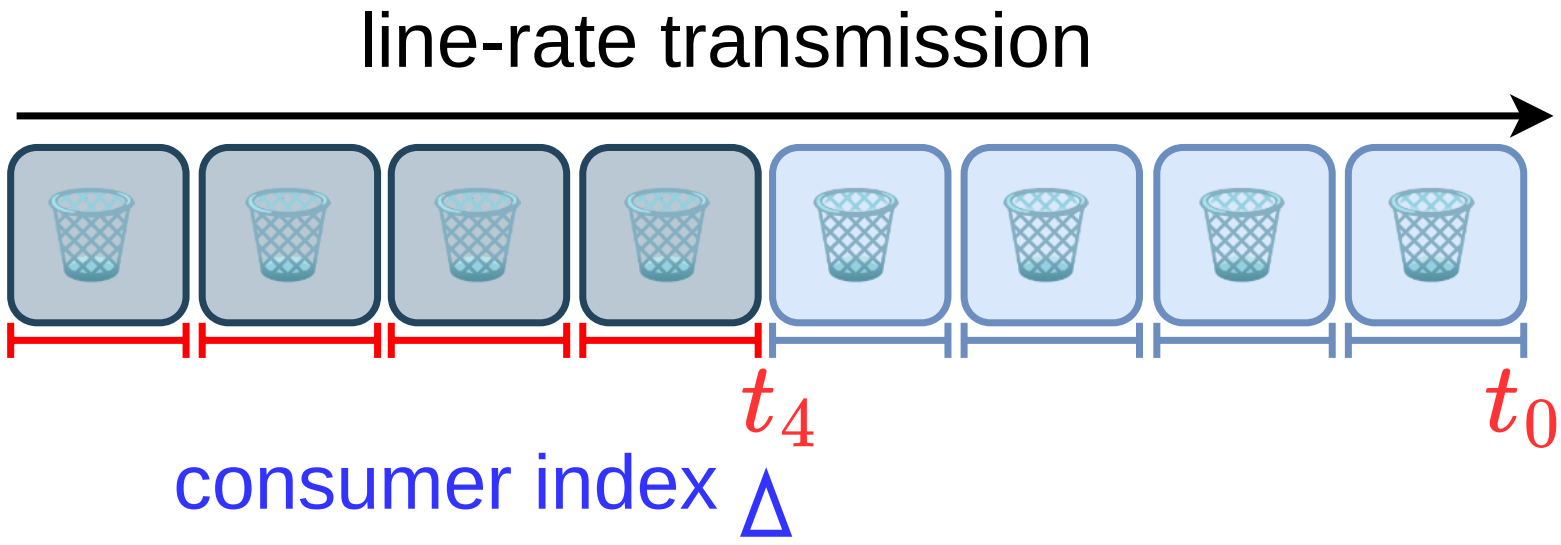
❸ Reclaim to dummy by overwrite wrong CRC code



❹ Advance producer index

Emulated PTP Hardware Clock (EPHC)

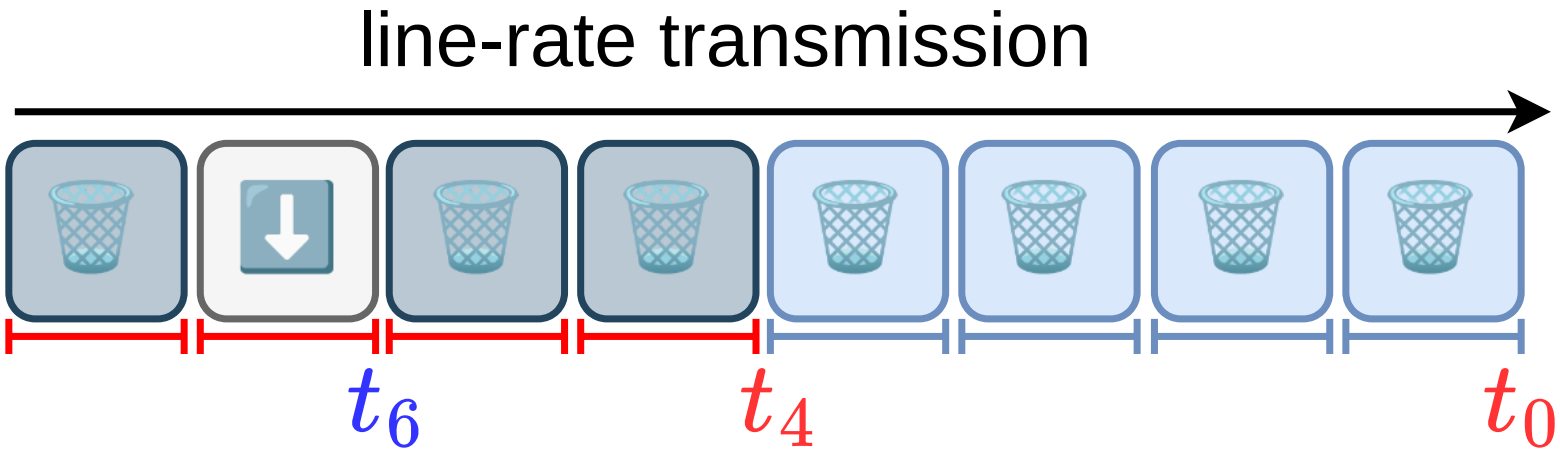
- EPHC ⏰ : time ↔ ring index
  - time = pkt count × transmission time + offset
    - ➢e.g., pkt size 200 bytes, t_0 = 0 → t_4= 4 x 200 bytes / 1Gbps = 6.4μs



Emulated PTP Hardware Clock (EPHC)

- EPHC ⏰ : time ↔ ring index
  - time = pkt count × transmission time + offset
    - ➢e.g., pkt size 200 bytes, t_0 = 0 → t_4= 4 x 200 bytes / 1Gbps = 6.4μs
  - ring index = [(scheduled time – offset) / transmission time] % ring size

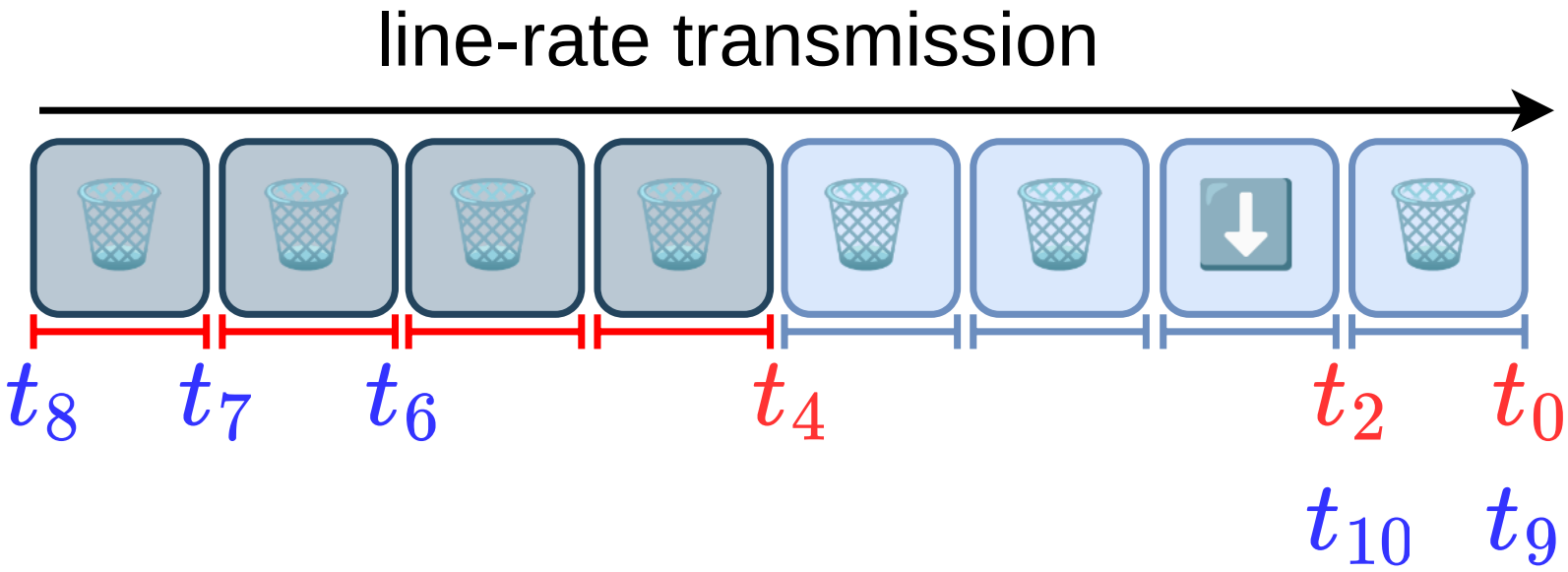
now: t_4, sche: t_6



Emulated PTP Hardware Clock (EPHC)

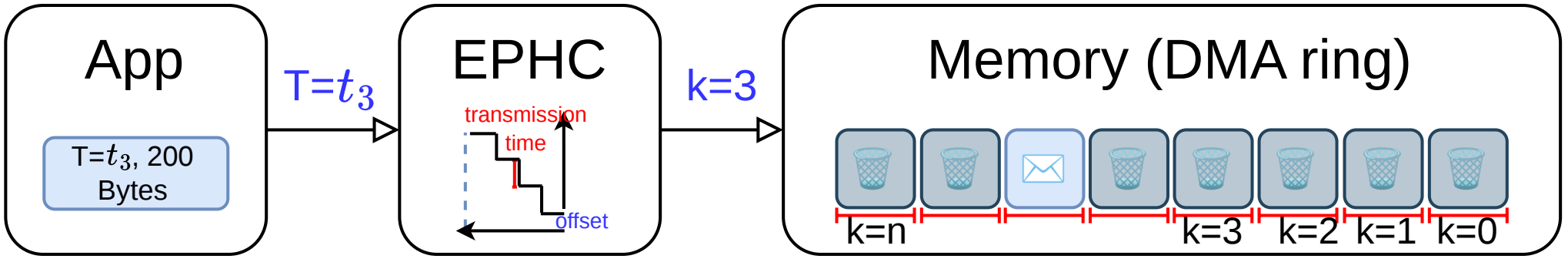
- EPHC ⏰ : time ↔ ring index
  - time = pkt count × transmission time + offset
    - ➢e.g., pkt size 200 bytes, t_0 = 0 → t_4= 4 x 200 bytes / 1Gbps = 6.4μs
  - ring index = [(scheduled time – offset) / transmission time] % ring size

now: t_4, sche: t_{10}



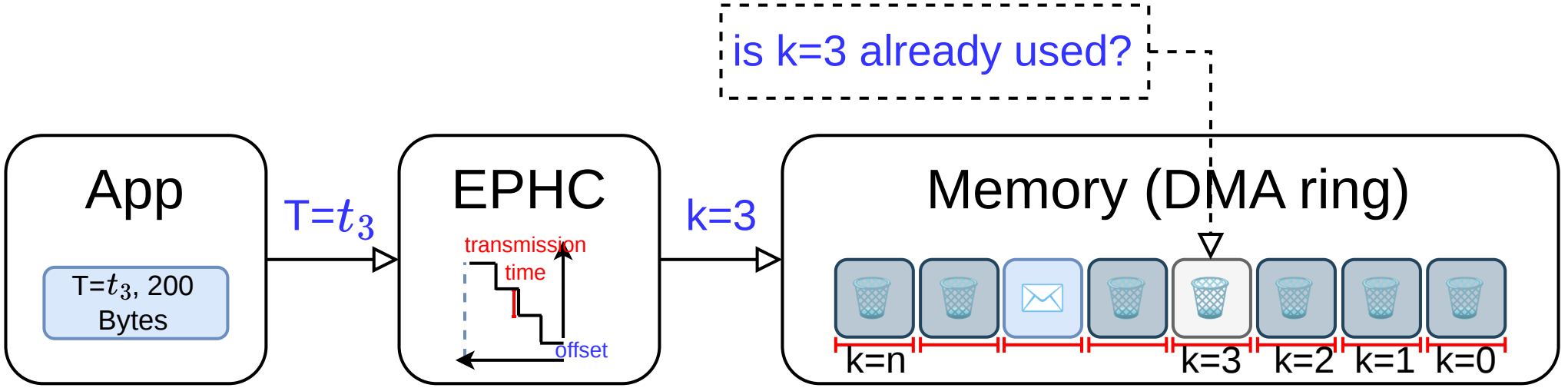
Scheduled packet insertion

- Step1: calculate index ← scheduled EPHC time



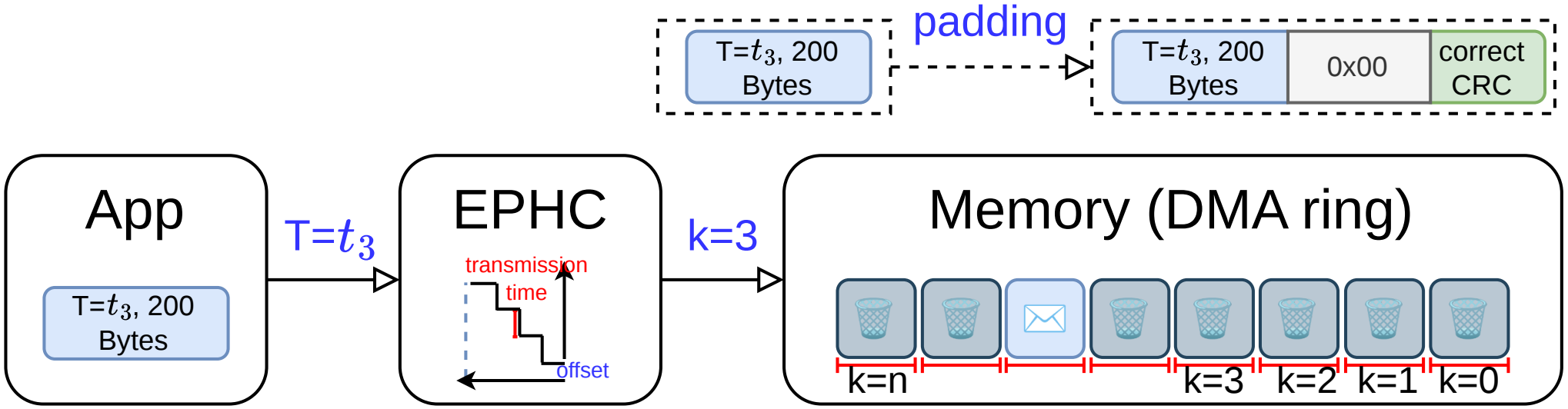
Scheduled packet insertion

- Step1: calculate index ← scheduled EPHC time
- Step2: validate if index is feasible (wrong CRC)



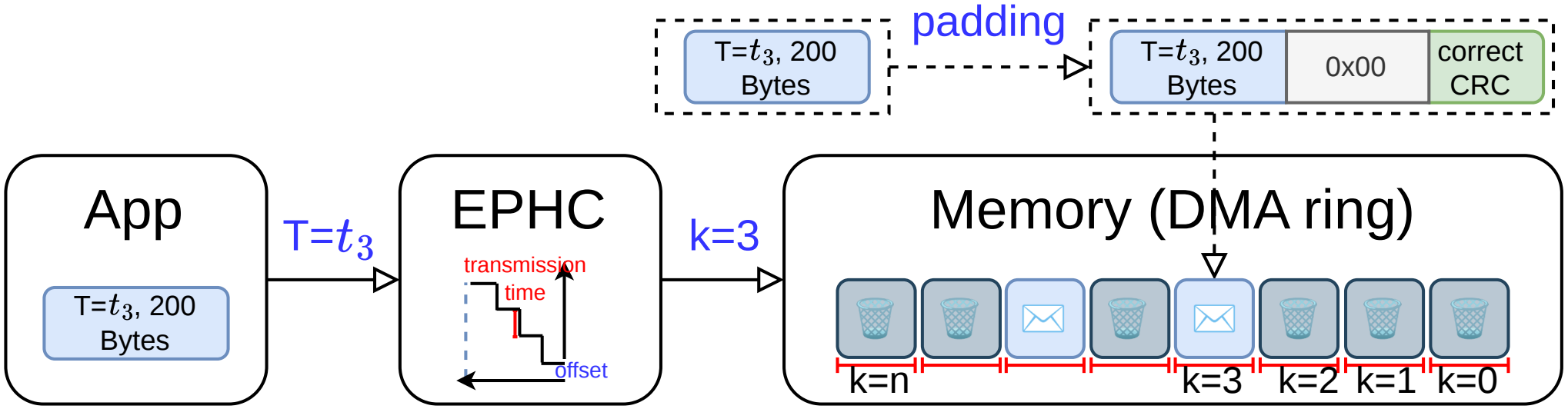
Scheduled packet insertion

- Step1: calculate index ← scheduled EPHC time
- Step2: validate if index is feasible (wrong CRC)
- Step3: pad the real data into fixed-size, and reset correct CRC code



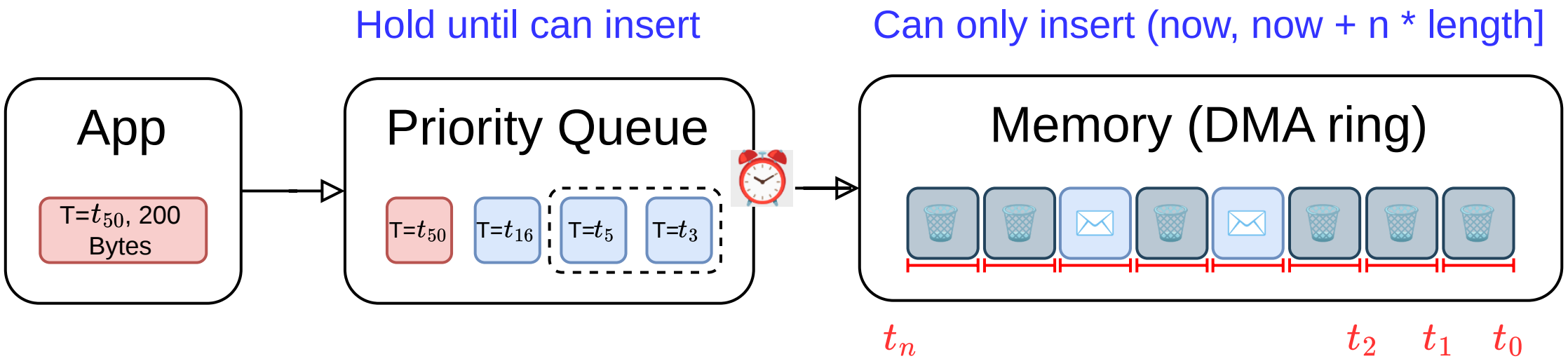
Scheduled packet insertion

- Step1: calculate index ← scheduled EPHC time
- Step2: validate if index is feasible (wrong CRC)
- Step3: pad the real data into fixed-size, and reset correct CRC code
- Step4: overwrite dummy packet by real data



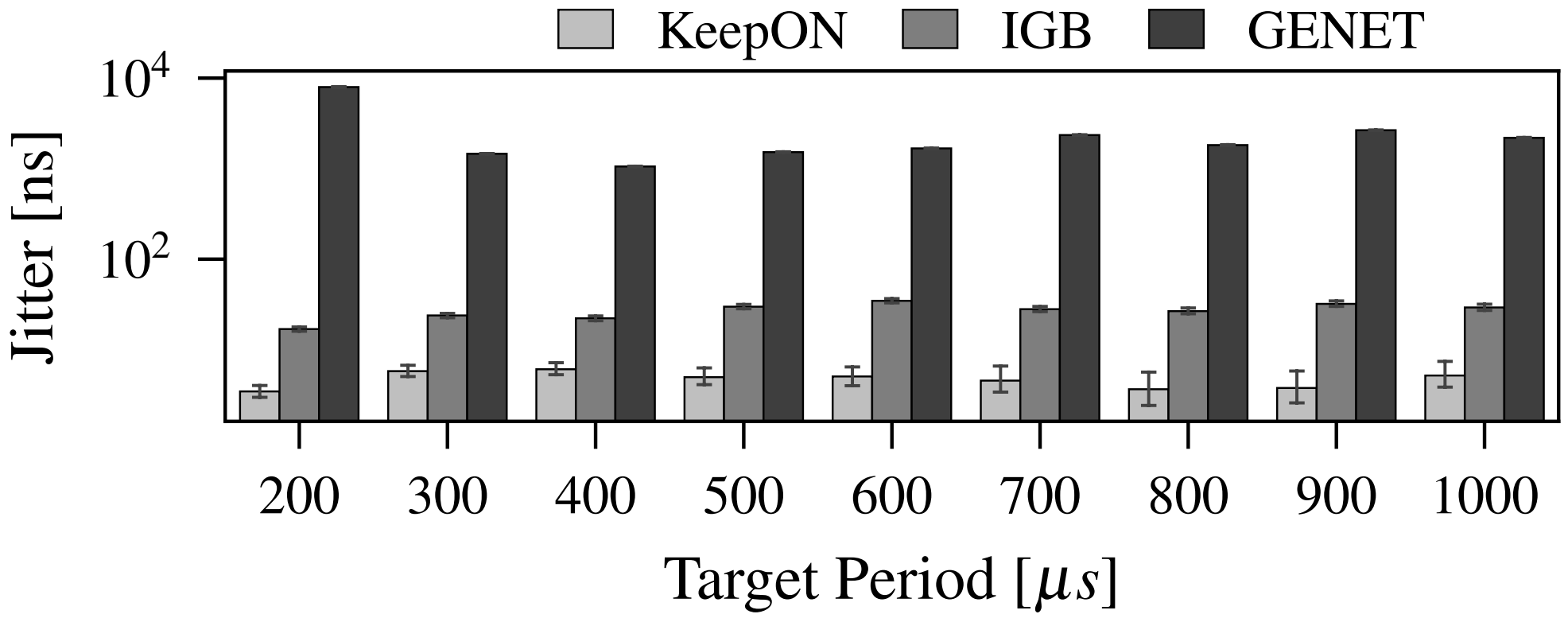
Packet pre-buffering

- Step1: calculate index ← scheduled EPHC time (e.g., $T = t_{50}$ out of range)
- Step2: validate if index is feasible (wrong CRC)
- Step3: pad the real data into fixed-size, and reset correct CRC code
- Step4: overwrite dummy packet by real data



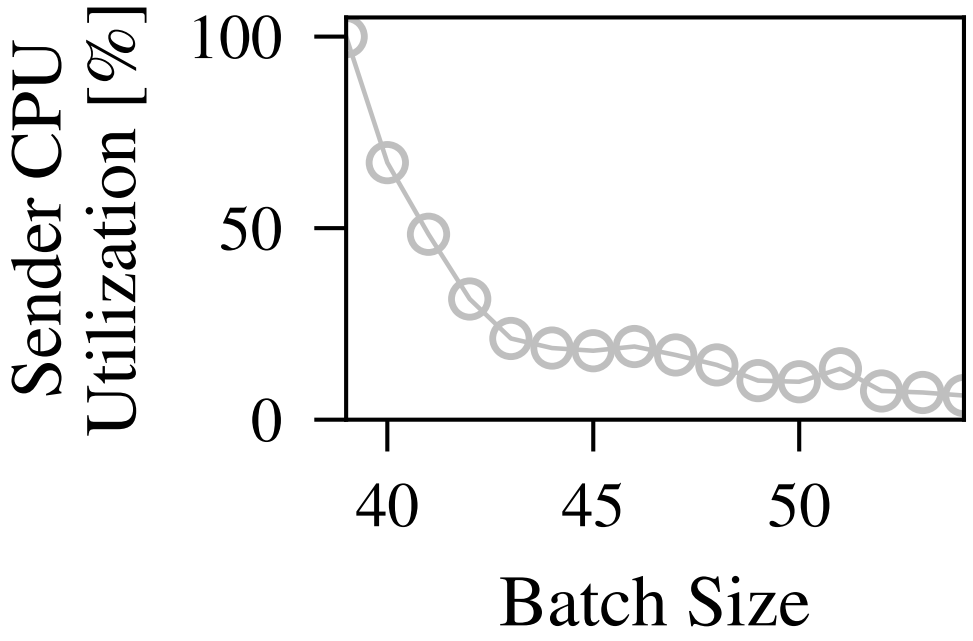
Evaluation: TX determinism

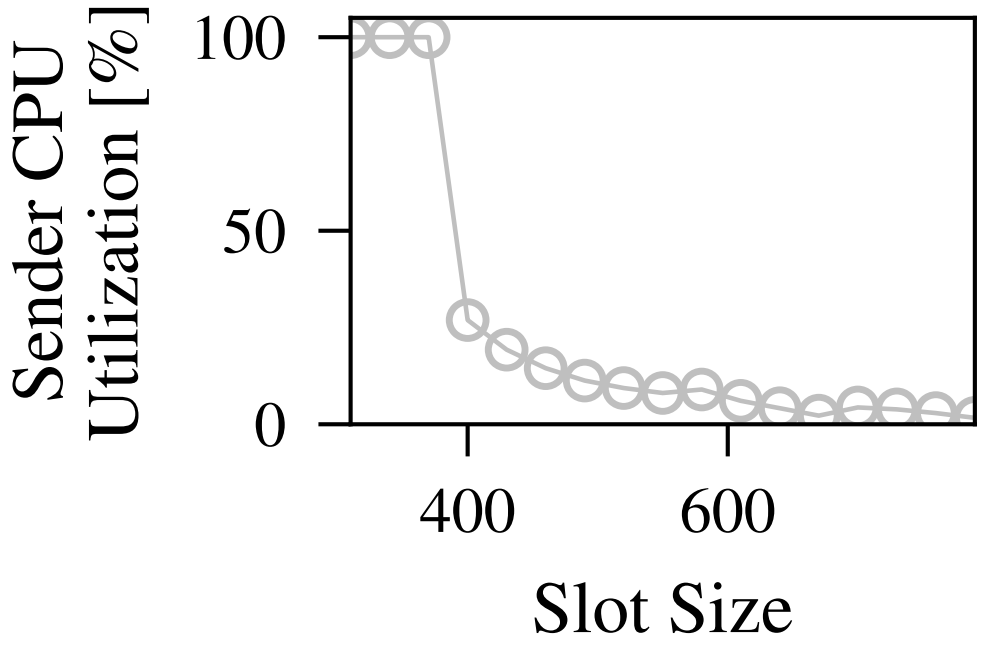
- KeepON achieves even better accuracy than IGB's hardware offloading
  - KeepON: 3.7ns, Hardware offloading (IGB): 7.7ns, Software: ~5μs



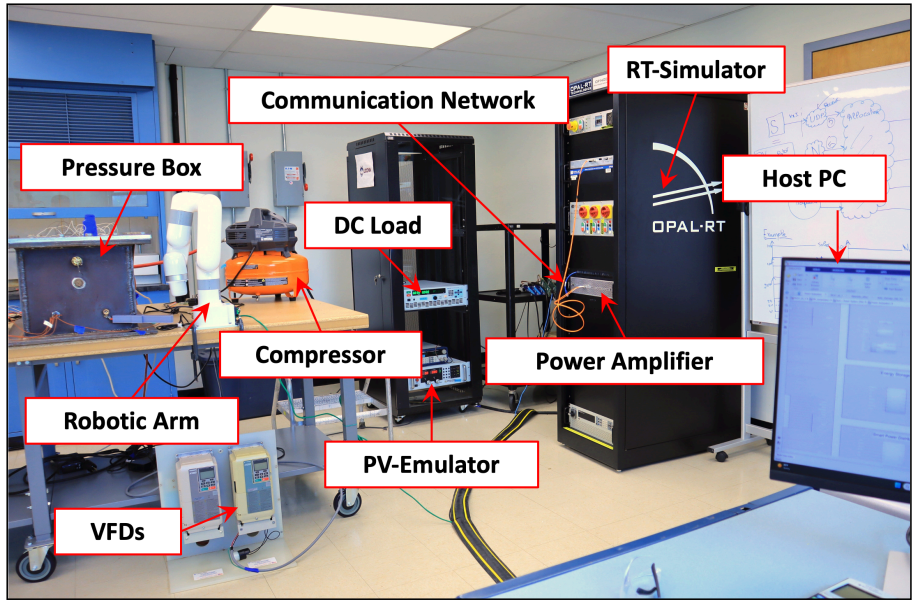
Evaluation: tunable system overhead

- ≦10% CPU ← Large PMD batch-size and slot-size to keep line-rate.

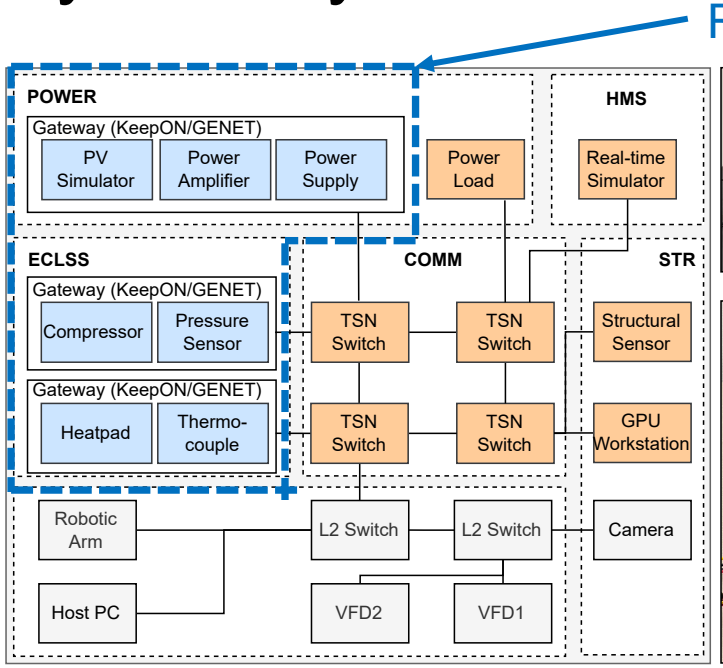




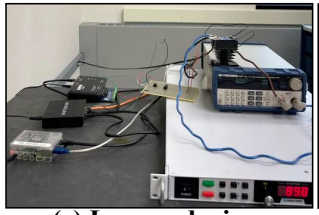
Case study: NASA RETHi Cyber-Physical Testbed



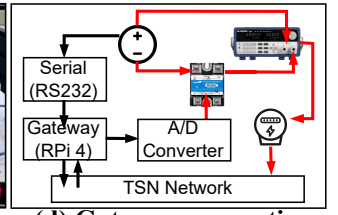
(a) Overview of Cyber-Physical Testbed



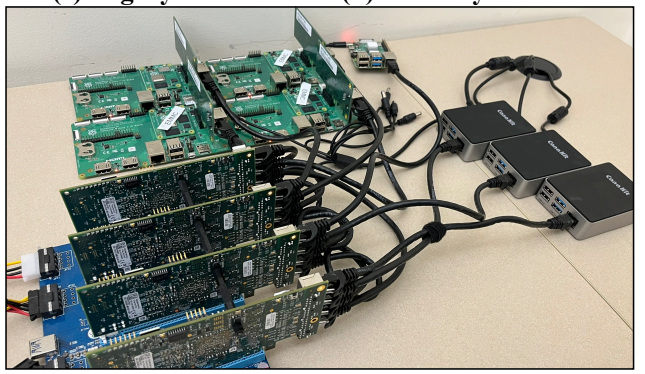
(b) Heterogenous Network topology



(c) Legacy devices

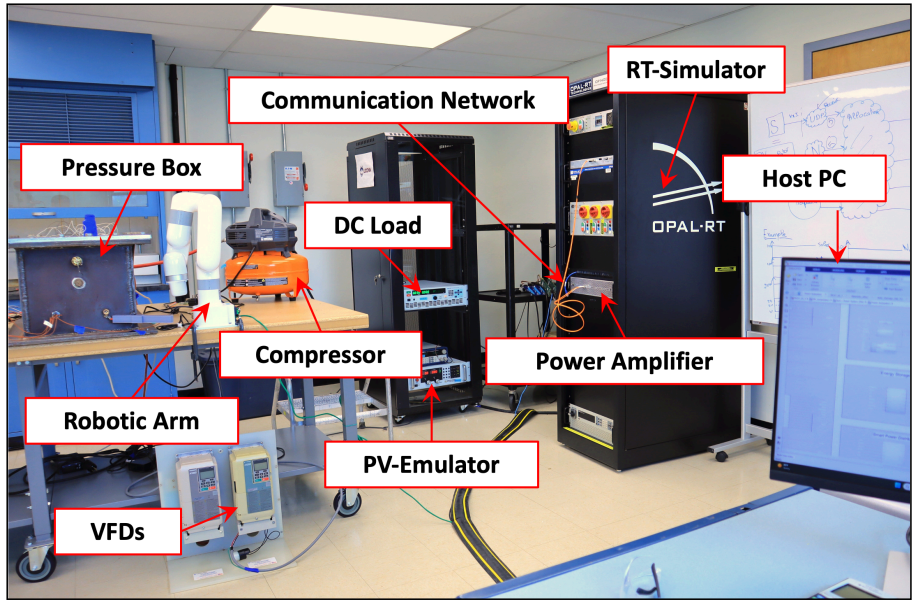


(d) Gateway connection

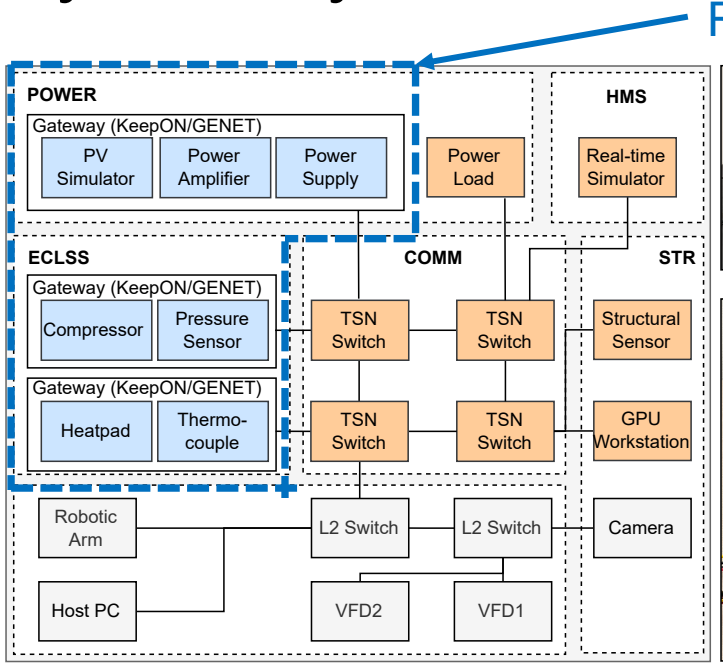


(e) TSN testbed

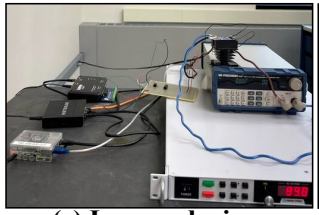
Case study: NASA RETHi Cyber-Physical Testbed



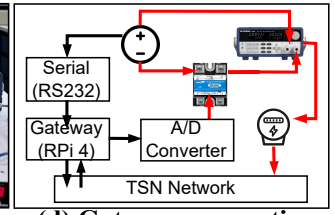
(a) Overview of Cyber-Physical Testbed



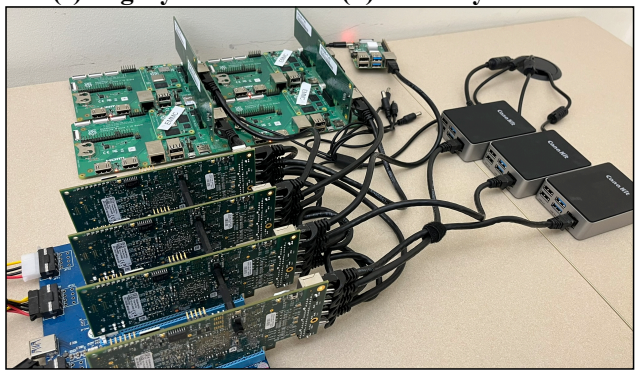
(b) Heterogenous Network topology



(c) Legacy devices



(d) Gateway connection



(e) TSN testbed

- 12 critical flows, 4 switches, 1ms period
  - GENET: Delays spike to >1ms (Missing deadlines)
  - KeepON: Meeting deadline

FlowId	GENET Delay [μs]			KeepON Delay [μs]		
	Mean	P99	Max	Mean	P99	Max
1	8.8	8.8	8.8	8.8	8.8	12.7
2	15.5	15.6	19.7	12.4	12.4	16.5
3	29.2	**1008.8**	**1008.8**	12.4	12.4	12.4
4	**936.7**	**1010.5**	**1010.5**	12.1	12.1	12.1
7	**980.6**	**1009.2**	**1009.2**	12.4	12.4	12.4
8	10.3	10.5	12.4	8.5	8.5	8.6
12	5.2	5.2	5.2	5.2	5.2	5.2

Conclusion

- Software-based deterministic networking on standard NICs
- Accurate scheduling on par with hardware solutions (ns-level accuracy)
- Practical path to real-world deployment

chuanyu.xue@uconn.edu

https://github.com/ChuanyuXue/KeepON-rpi

The work is supported in part by the National Science Foundation Grant CNS-2008463, CNS-2432533 and the NASA STRI Resilient Extraterrestrial Habitats Institute (RETHi) under grant number 80NSSC19K1076.