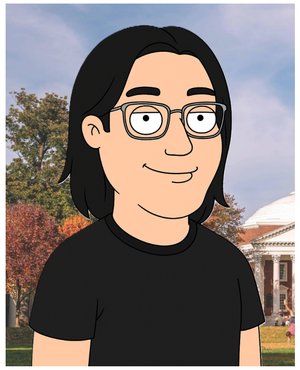
Understanding Host Network Stack Latency

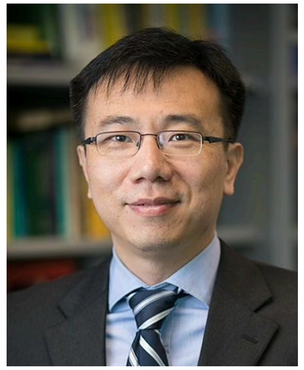


Tianyu Zuo

University of Virginia

Jaehyun Hwang

성균관대학교 SUNG KYUN KWAN UNIVERSITY(SKKU)

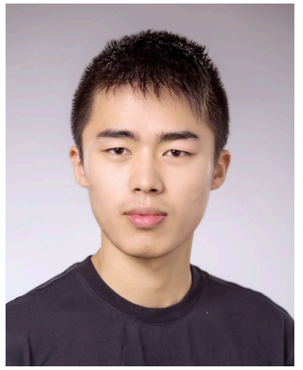


Ao Tang

Cornell University

Rachit Agarwal

Cornell University



Qizhe Cai

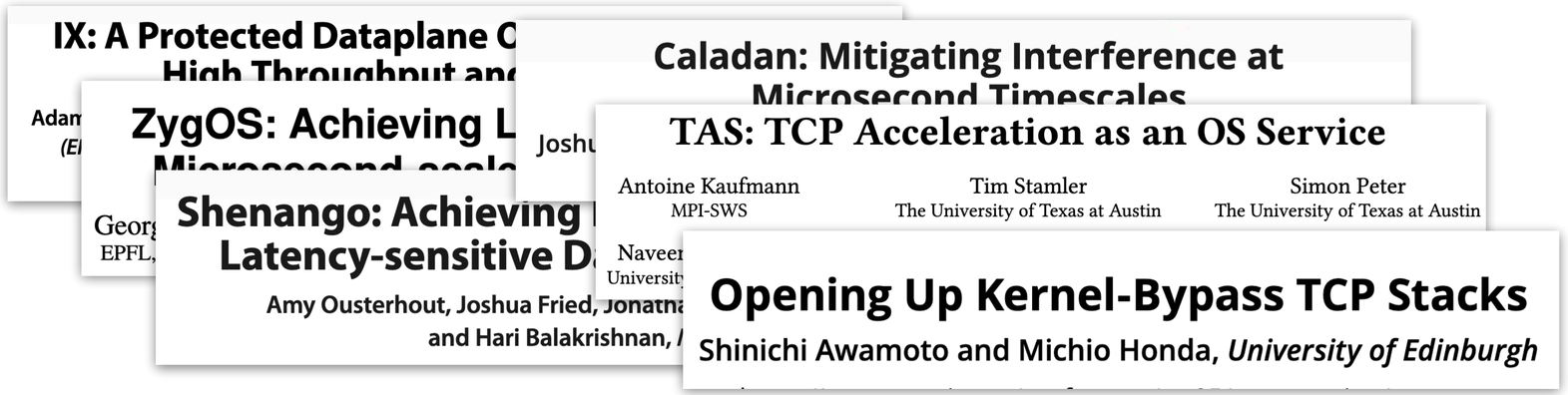
University of Virginia

Tail Latency in Host Network Stack

Tail Latency in Host Network Stack

The received wisdom of our community:

- **“Linux network stack can suffer ms-scale tail latency”**

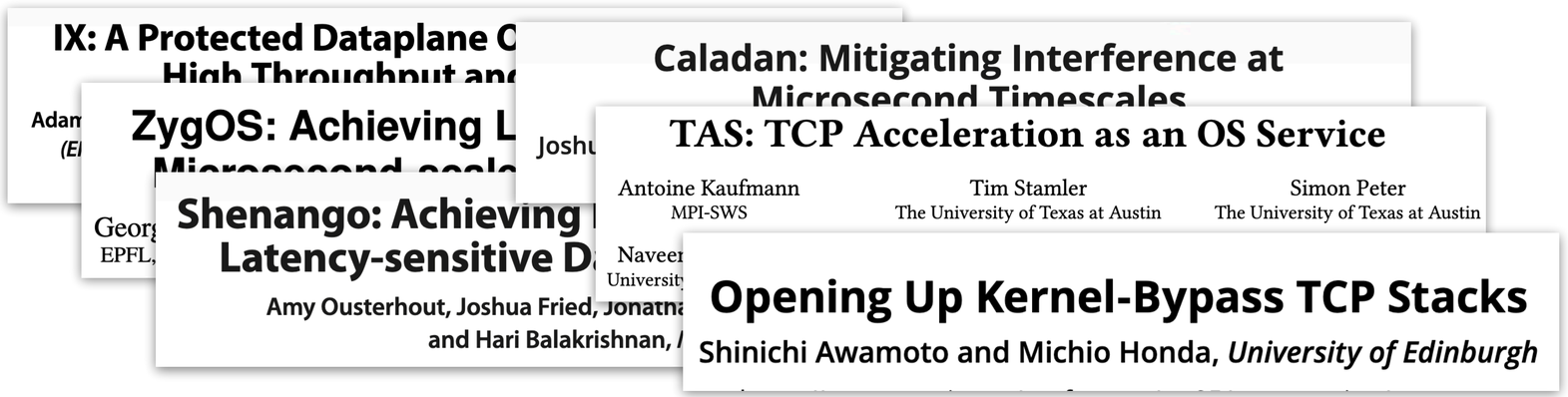


IX (OSDI 14), ZygOS (SOSP 17), Shenango (NSDI 19), Caladan (OSDI 20), TAS (Eurosys 19), Opening Up (ATC 25)

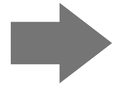
Tail Latency in Host Network Stack

The received wisdom of our community:

- **“Linux network stack can suffer ms-scale tail latency”**



IX (OSDI 14), ZygOS (SOSP 17), Shenango (NSDI 19), Caladan (OSDI 20), TAS (Eurosys 19), Opening Up (ATC 25)

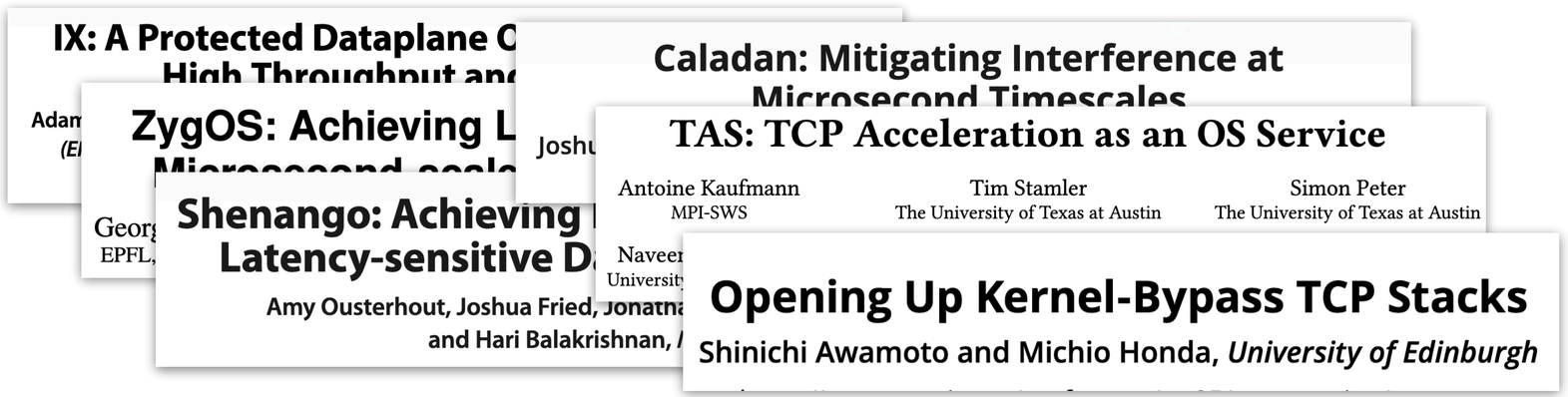


- **Operating Systems**
- **Clean-slate Network Stacks**
- **Hardware Offloads**

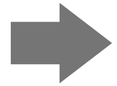
Tail Latency in Host Network Stack

The received wisdom of our community:

- **"Linux network stack can suffer ms-scale tail latency"**



IX (OSDI 14), ZygOS (SOSP 17), Shenango (NSDI 19), Caladan (OSDI 20), TAS (Eurosys 19), Opening Up (ATC 25)



- Operating Systems
- Clean-slate Network Stacks
- Hardware Offloads

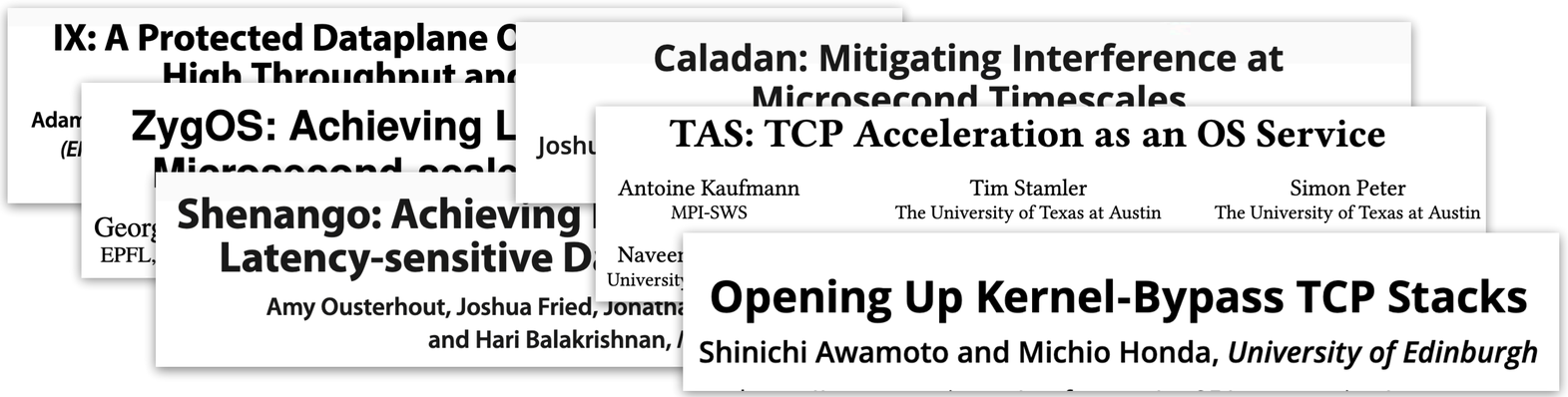
One basic question remains:

- ***Why* does Linux network stack have high tail latency?**

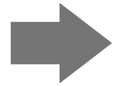
Tail Latency in Host Network Stack

The received wisdom of our community:

- **"Linux network stack can suffer ms-scale tail latency"**



IX (OSDI 14), ZygOS (SOSP 17), Shenango (NSDI 19), Caladan (OSDI 20), TAS (Eurosys 19), Opening Up (ATC 25)



- Operating Systems
- Clean-slate Network Stacks
- Hardware Offloads

One basic question remains:

- ***Why* does Linux network stack have high tail latency?**

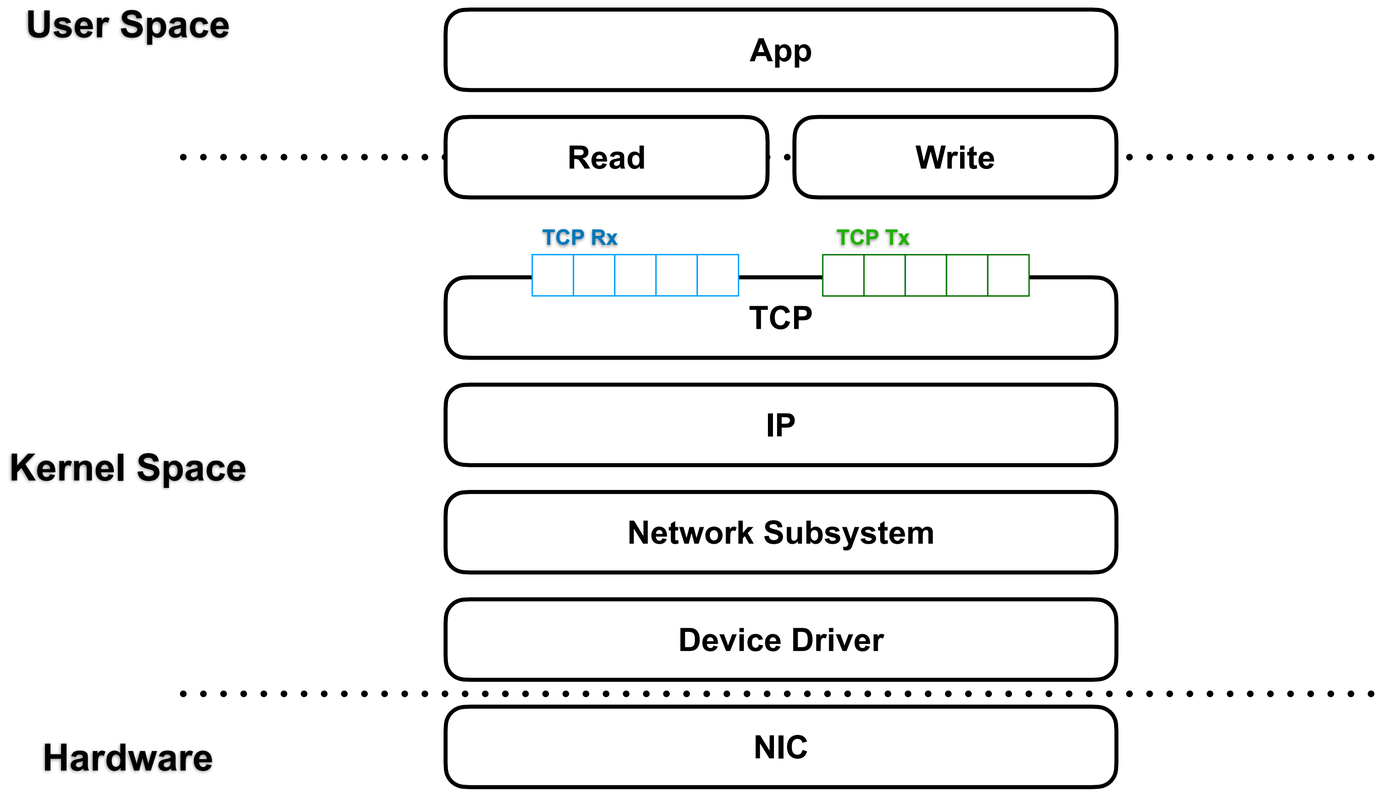
Our surprising finding:

- **The bottleneck is not the network stack and packet processing itself, but CPU scheduling!**

Methodology and Experimental Setting

To understand the root cause of high tail latency:

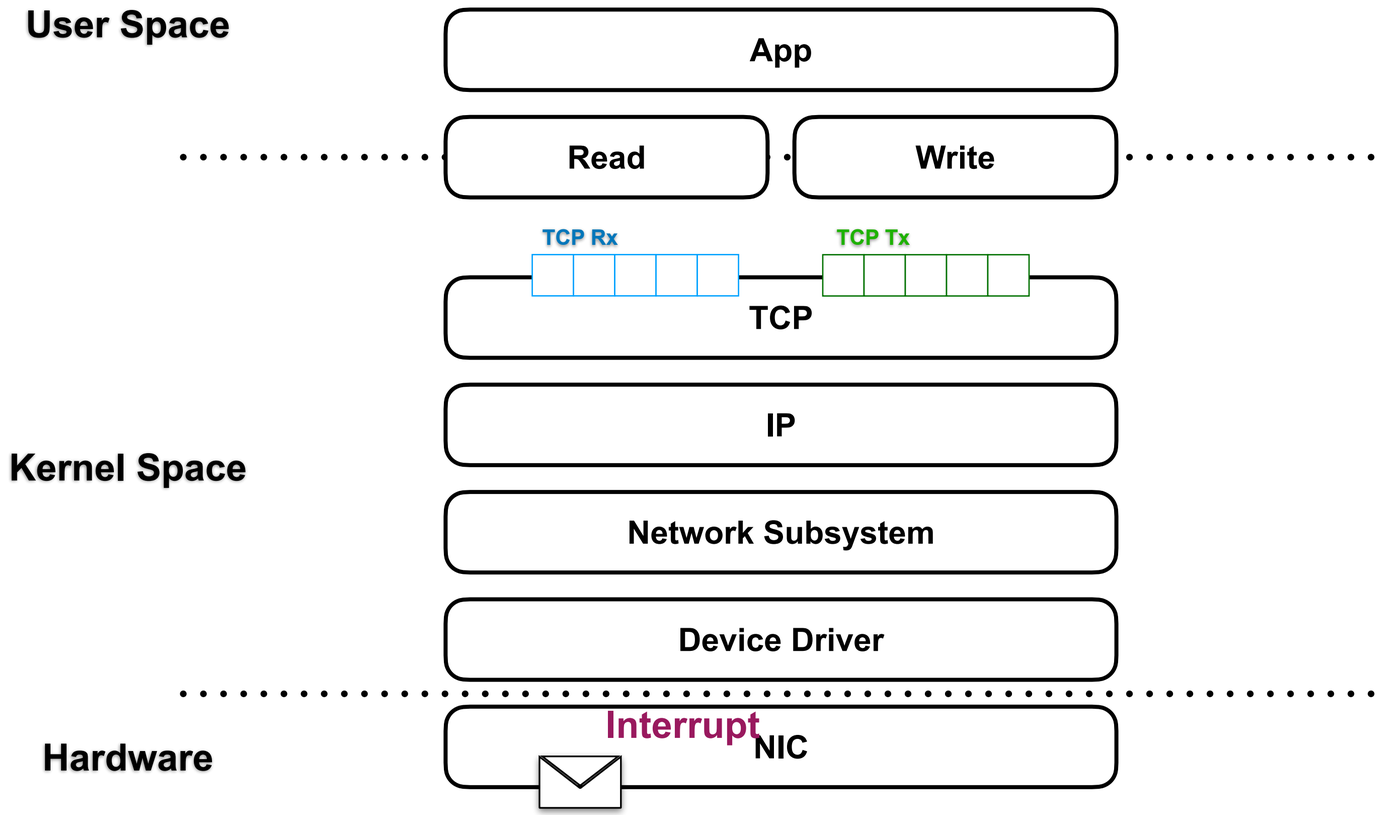
❖ **We build a 12-component RPC latency breakdown measurement infrastructure**



Methodology and Experimental Setting

To understand the root cause of high tail latency:

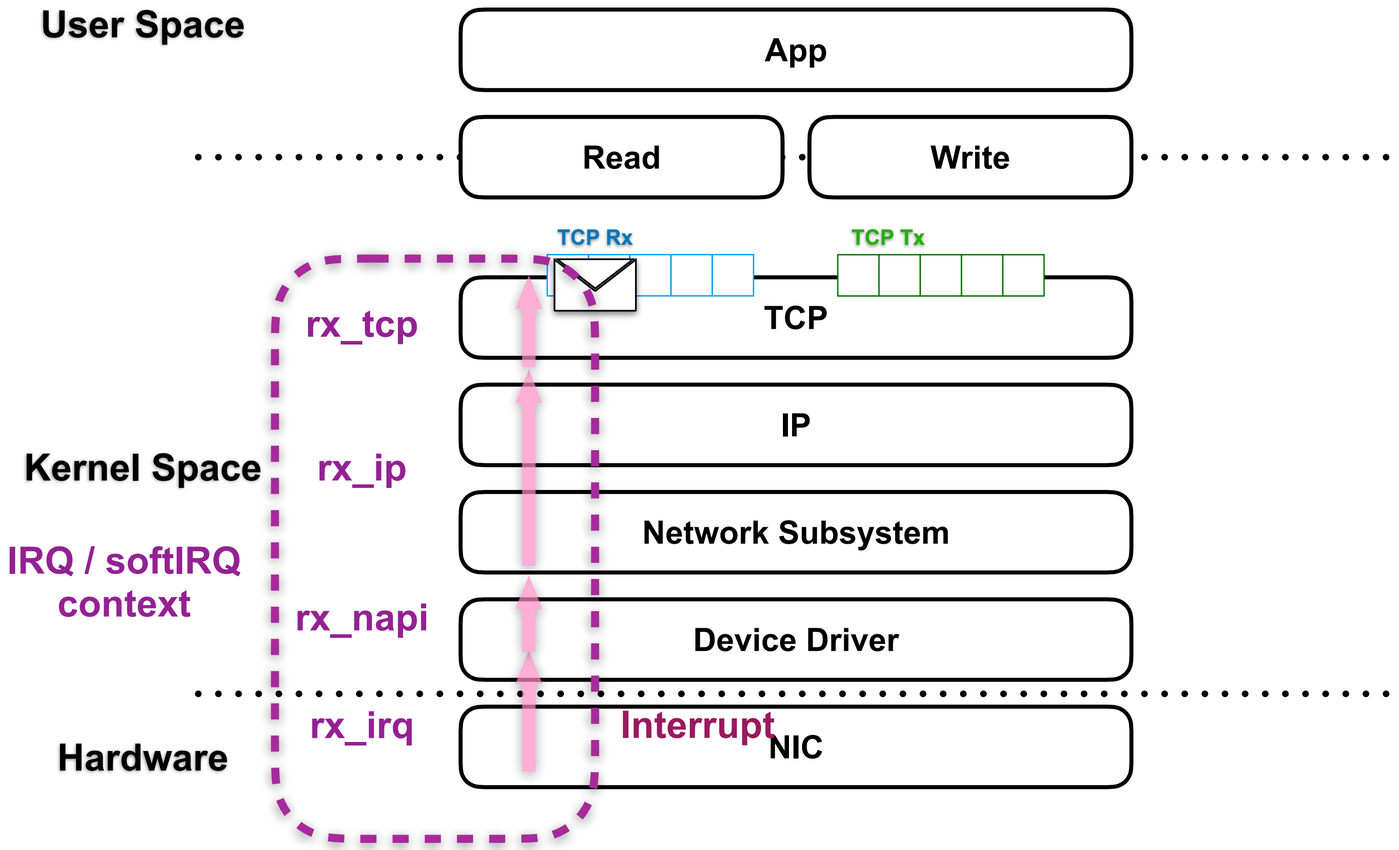
❖ **We build a 12-component RPC latency breakdown measurement infrastructure**



Methodology and Experimental Setting

To understand the root cause of high tail latency:

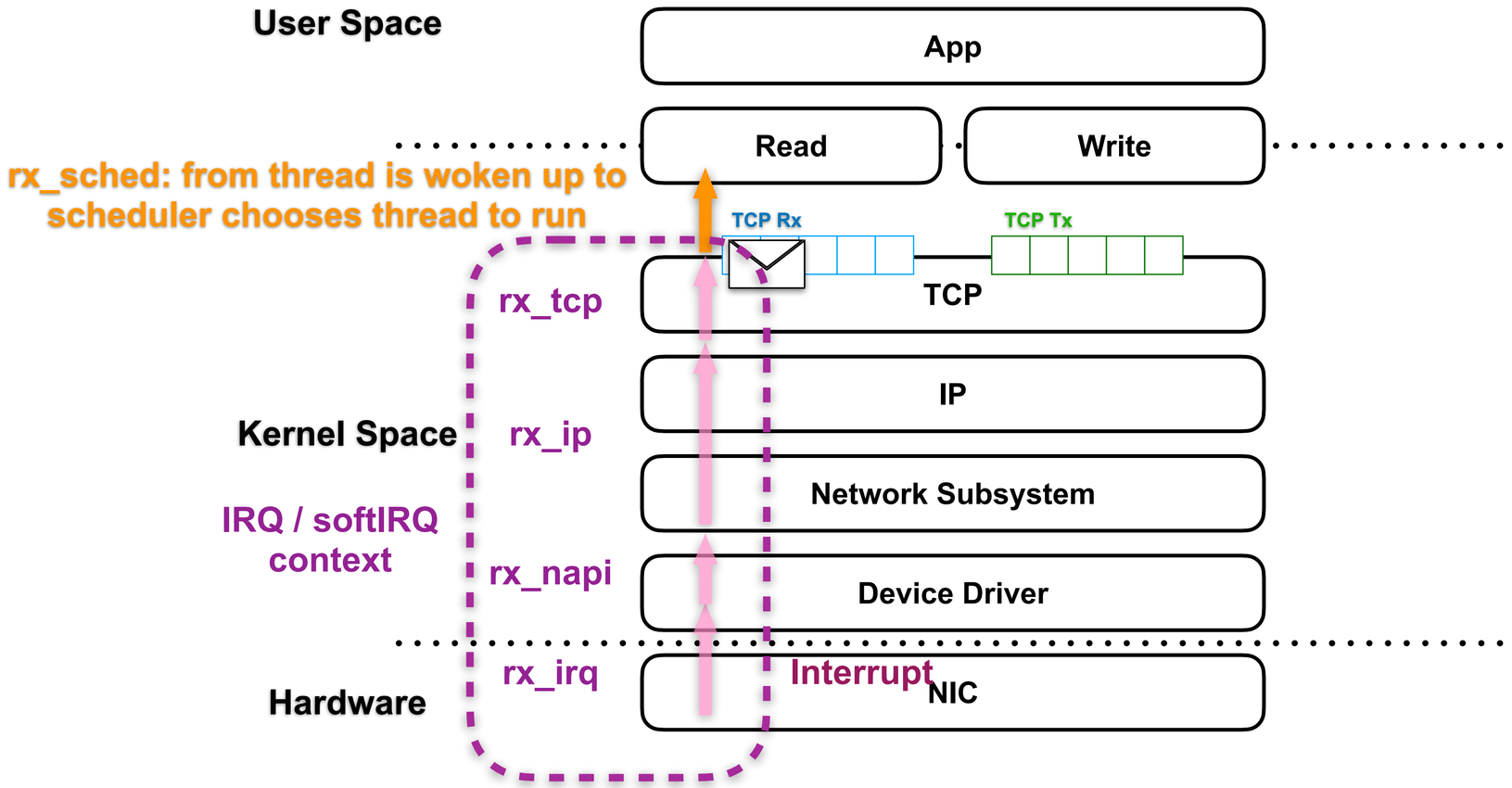
❖ **We build a 12-component RPC latency breakdown measurement infrastructure**



Methodology and Experimental Setting

To understand the root cause of high tail latency:

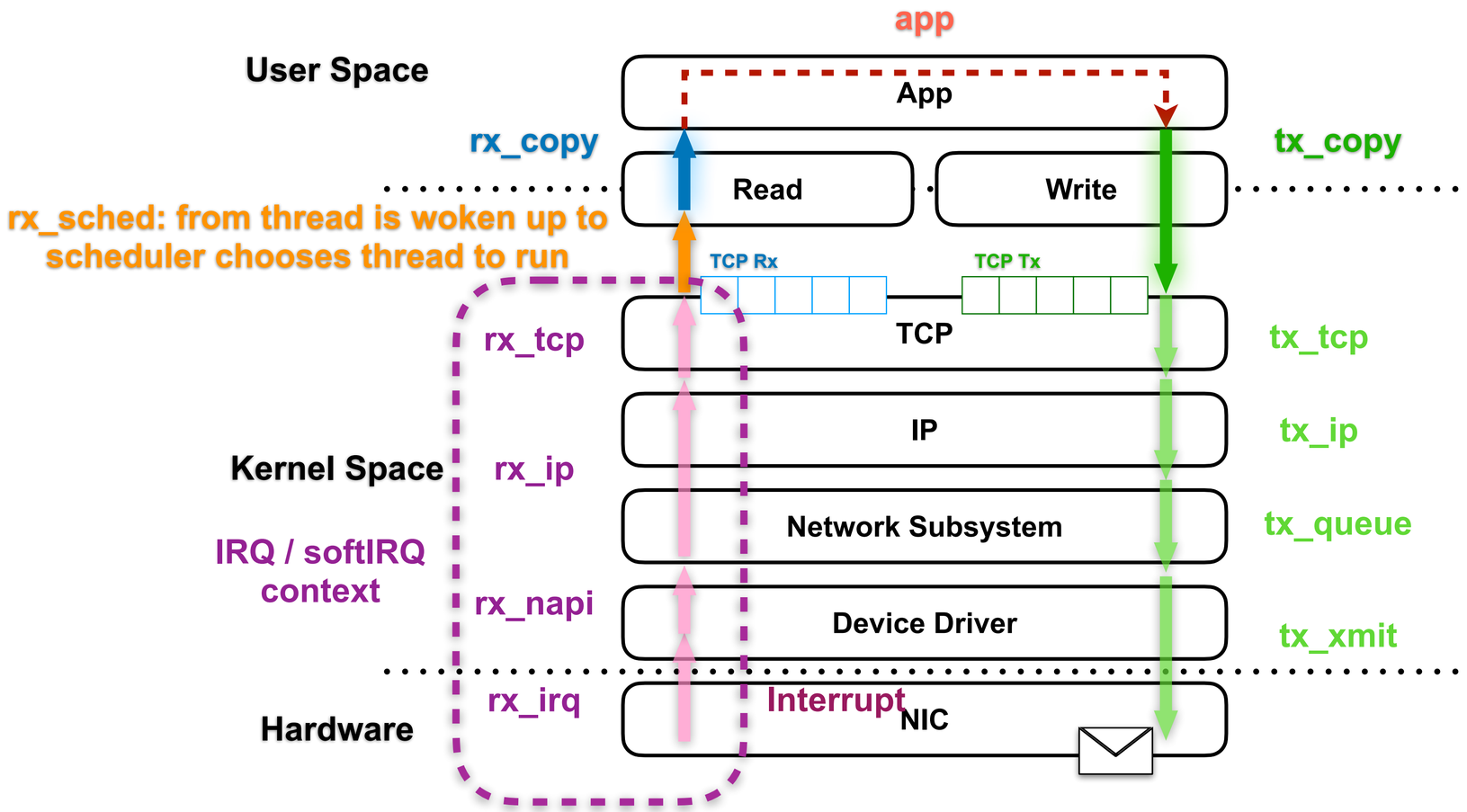
❖ **We build a 12-component RPC latency breakdown measurement infrastructure**



Methodology and Experimental Setting

To understand the root cause of high tail latency:

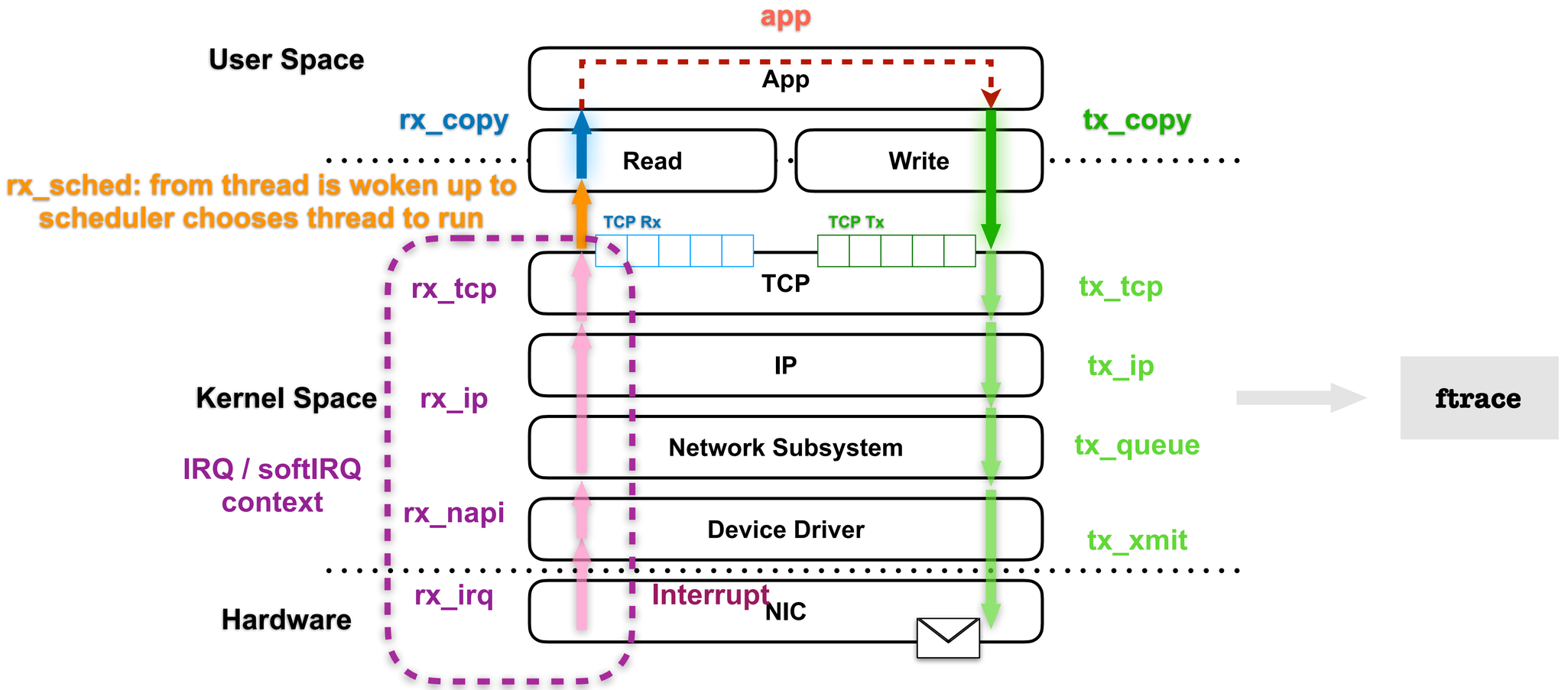
❖ **We build a 12-component RPC latency breakdown measurement infrastructure**



Methodology and Experimental Setting

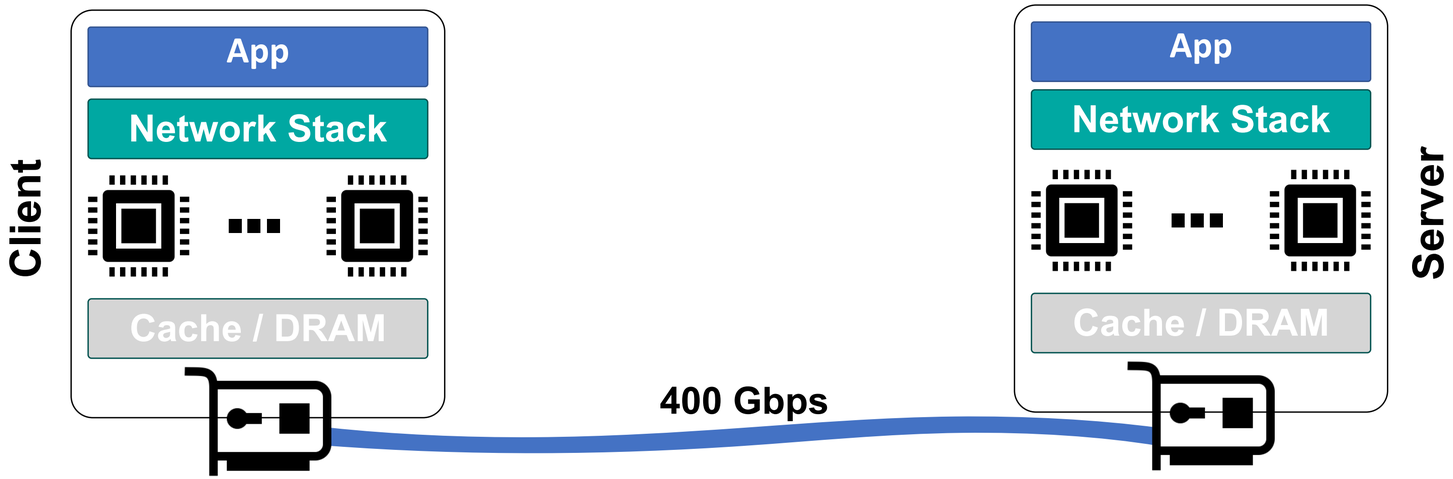
To understand the root cause of high tail latency:

❖ **We build a 12-component RPC latency breakdown measurement infrastructure**

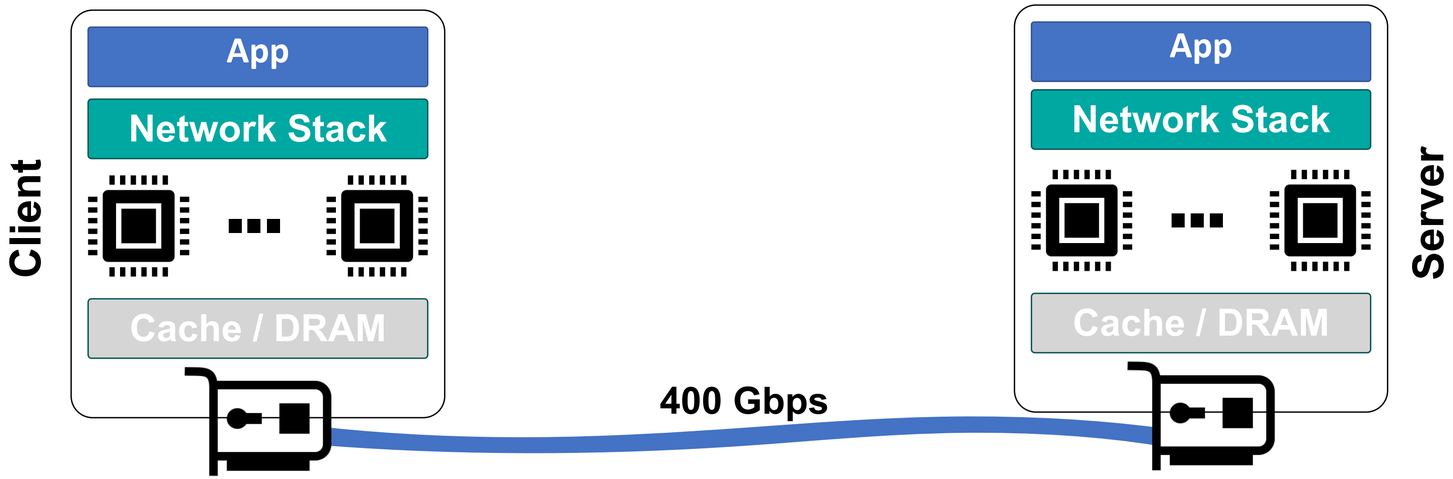


Methodology and Experimental Setting

Methodology and Experimental Setting



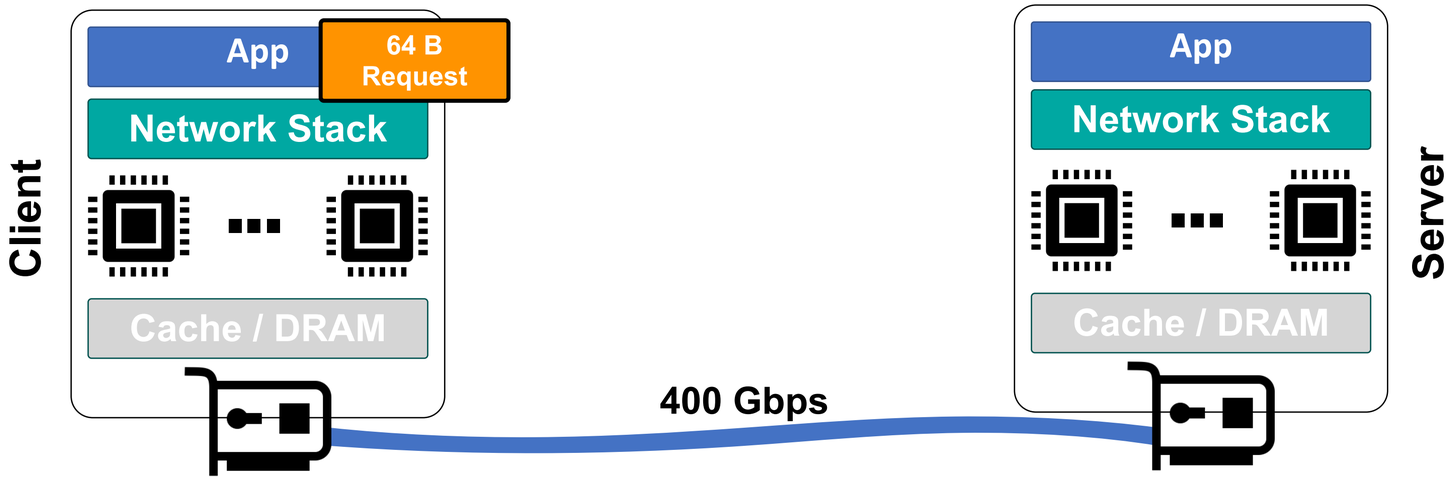
Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

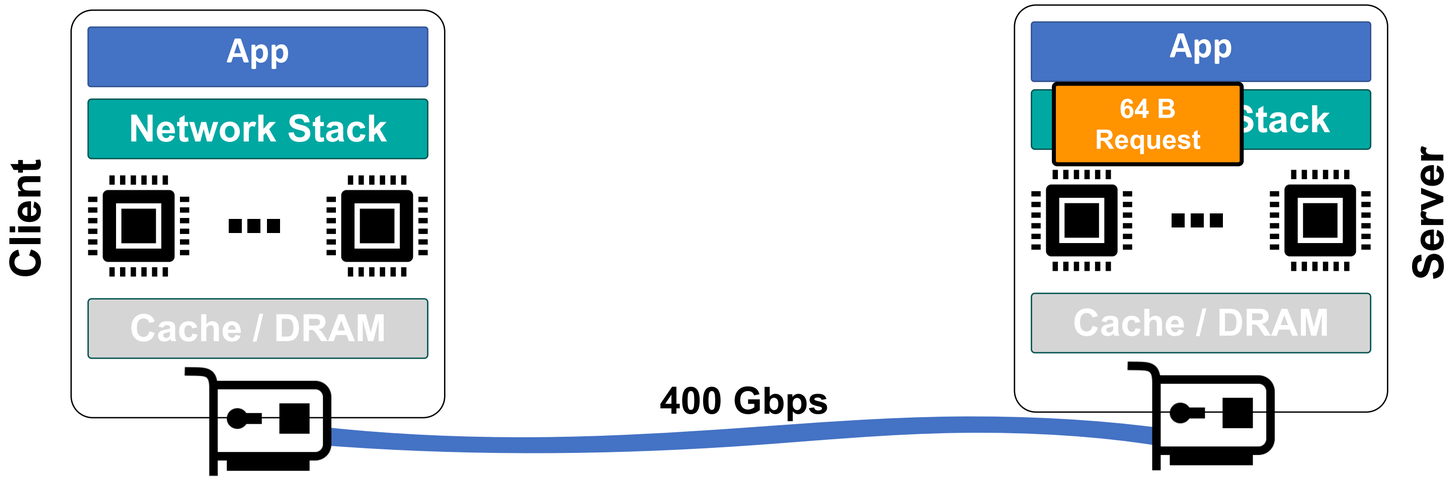
Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

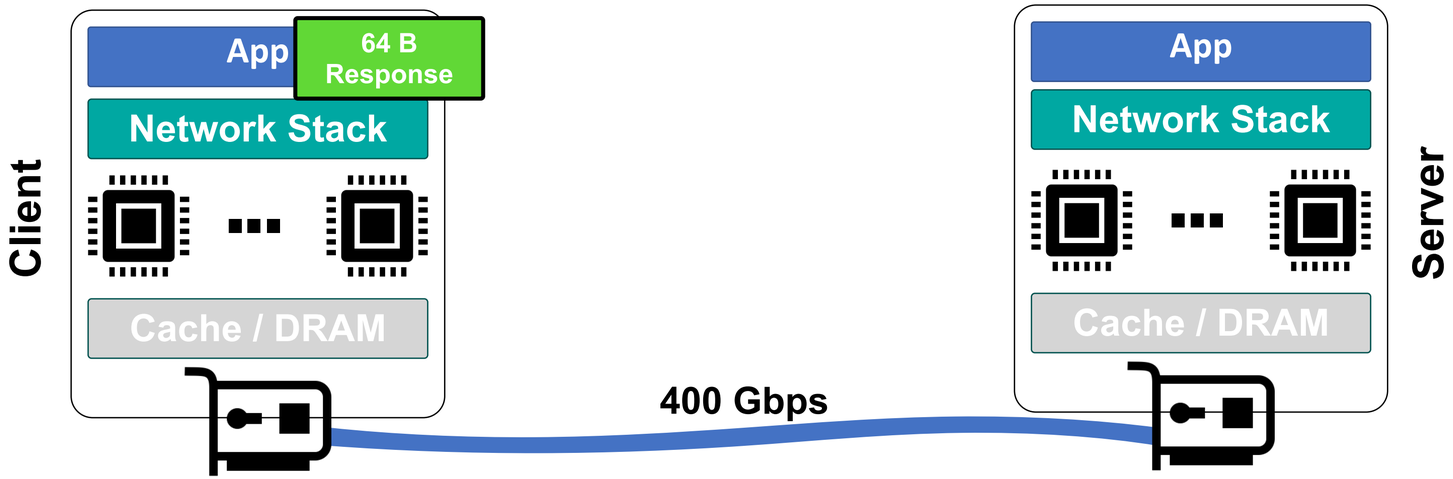
Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

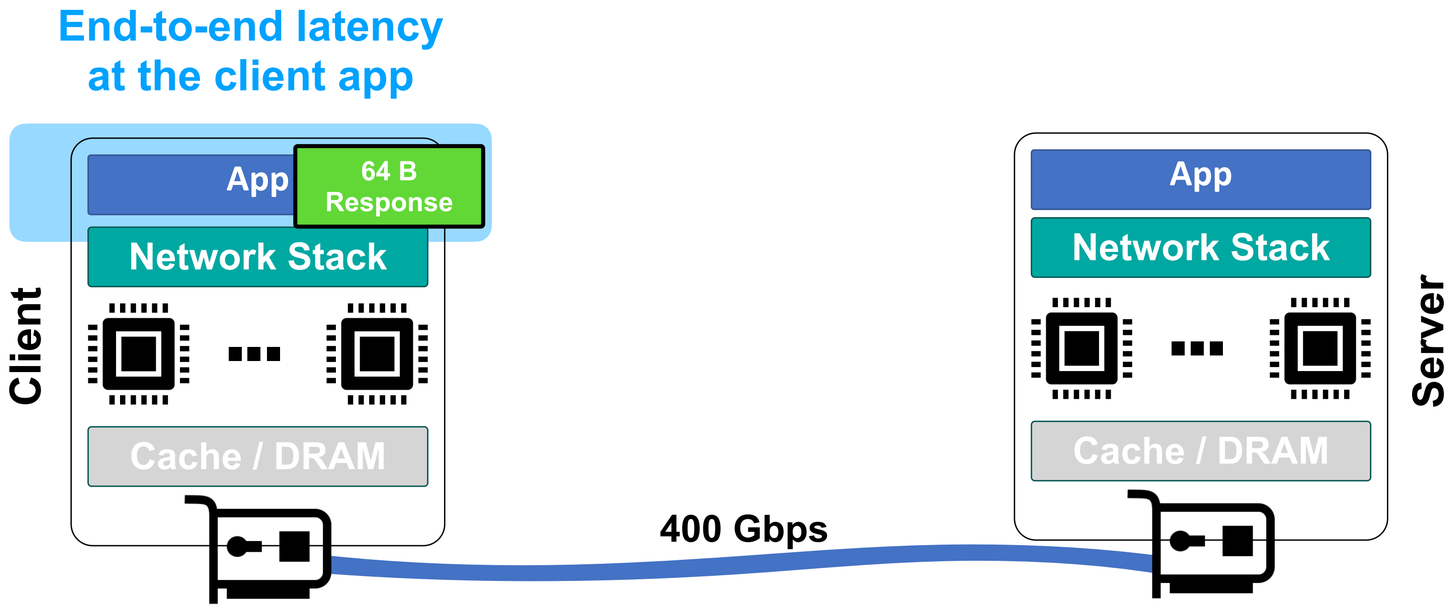
Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

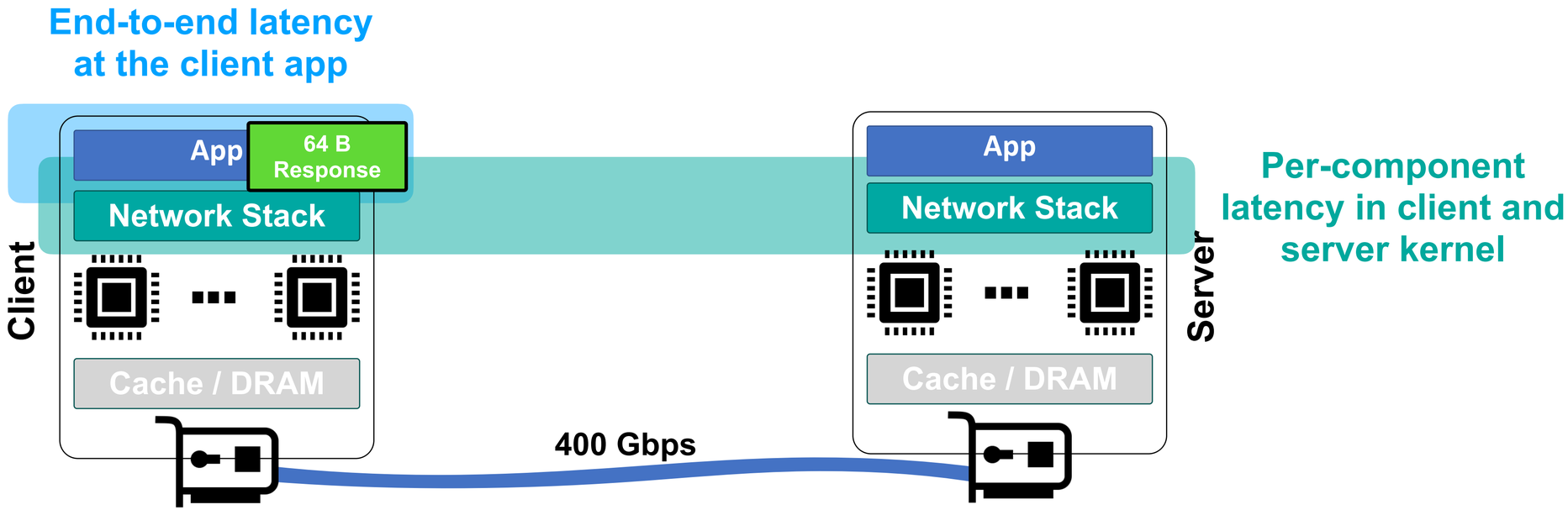
Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

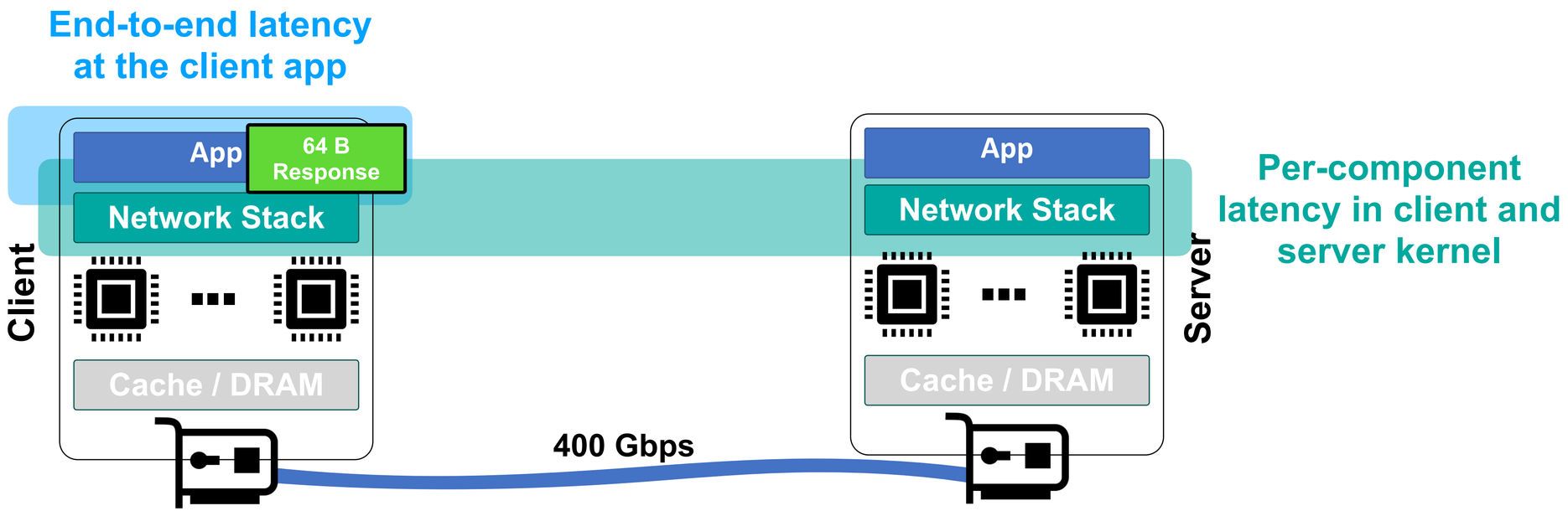
Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

Methodology and Experimental Setting



✔Enable Hyper-Threading, TSO/GRO, aRFS, and Jumbo Frame

✔ Enable Dynamic Interrupt Moderation (DIM) after CPU bottleneck

More experiment settings in the paper:

- numbers of CPU cores, threads, and in-flight
- RPC sizes, traffic patterns
- …

Four Main Lessons from Our Study

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

✦ Choose your parallelism carefully

- More in-flight requests per thread beat more threads

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

✦ Choose your parallelism carefully

- More in-flight requests per thread beat more threads

See our paper!

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

✦ Choose your parallelism carefully

- More in-flight requests can be much more latency-efficient than more connections

See our paper

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

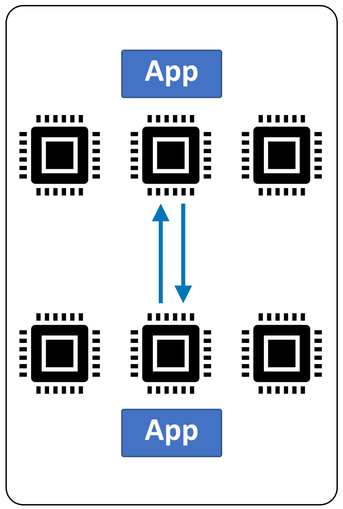
✦ Choose your parallelism carefully

- More in-flight requests can be much more latency-efficient than more connections

See our paper

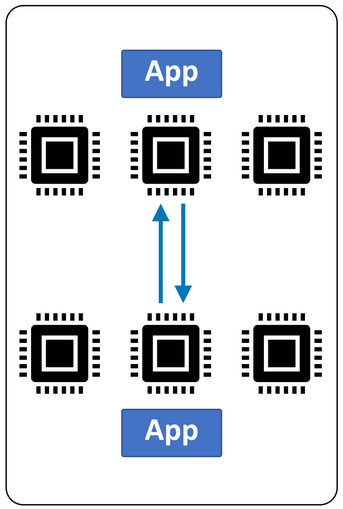
An Isolated Thread Already Achieves Low Tail Latency

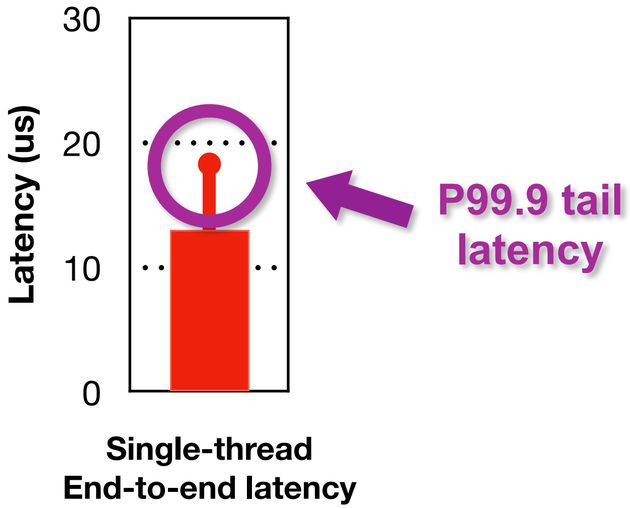
We start with a single thread in a single core, with a single request in flight:



An Isolated Thread Already Achieves Low Tail Latency

We start with a single thread in a single core, with a single request in flight:

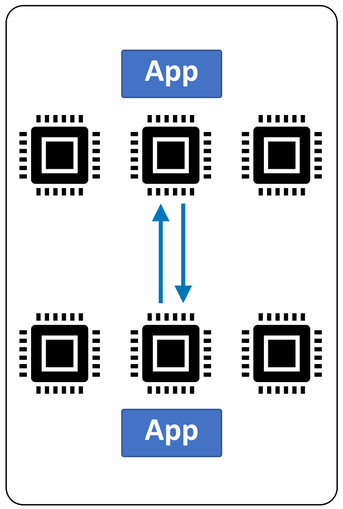


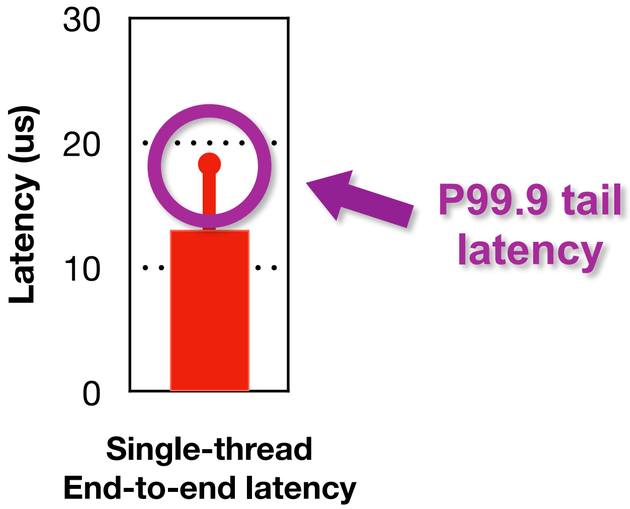


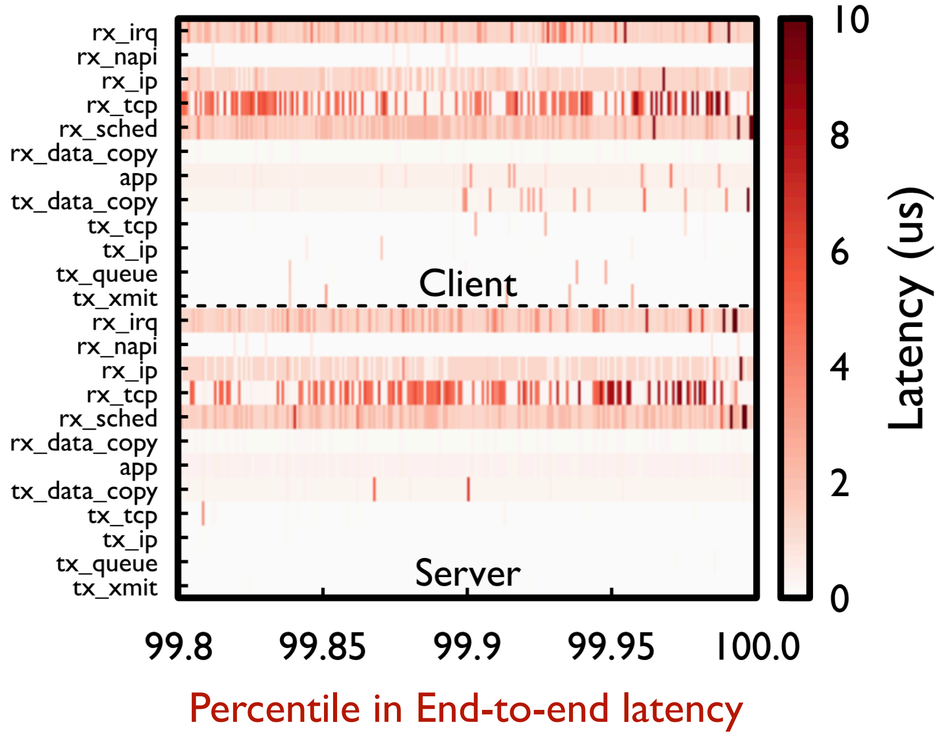
❖ **Linux is able to achieve only ~19us tail latency without resource contention**

An Isolated Thread Already Achieves Low Tail Latency

We start with a single thread in a single core, with a single request in flight:



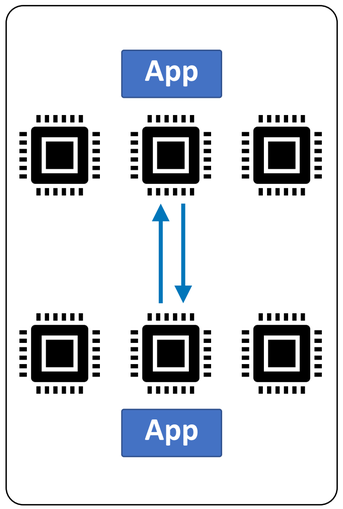


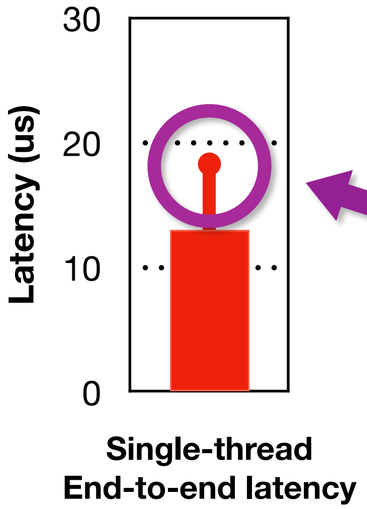


❖ **Linux is able to achieve only ~19us tail latency without resource contention**

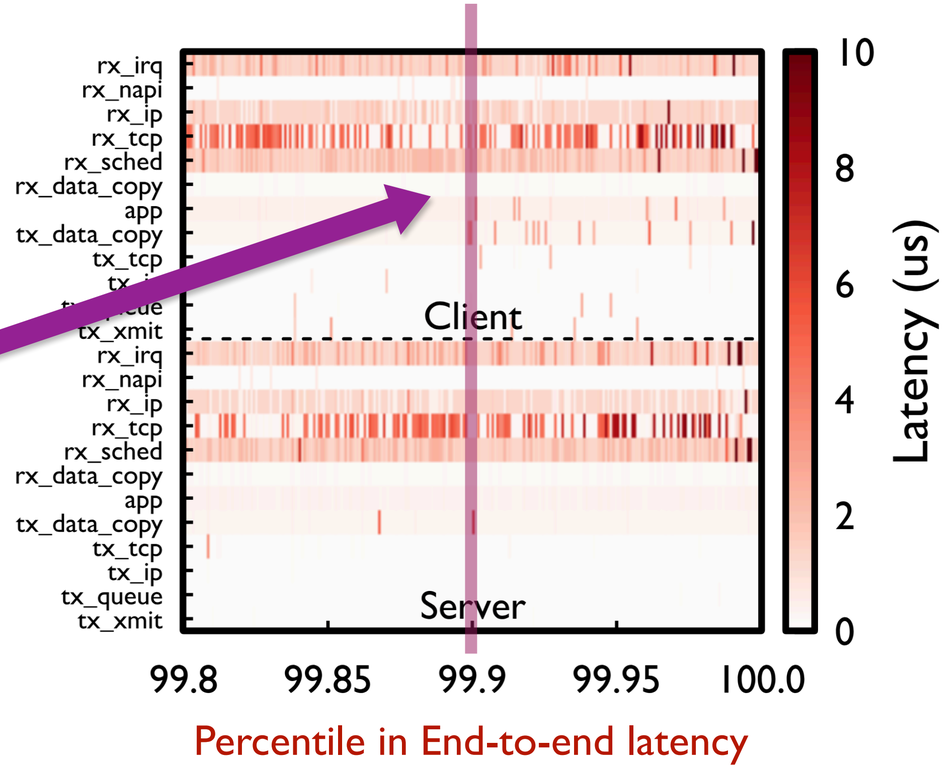
An Isolated Thread Already Achieves Low Tail Latency

We start with a single thread in a single core, with a single request in flight:





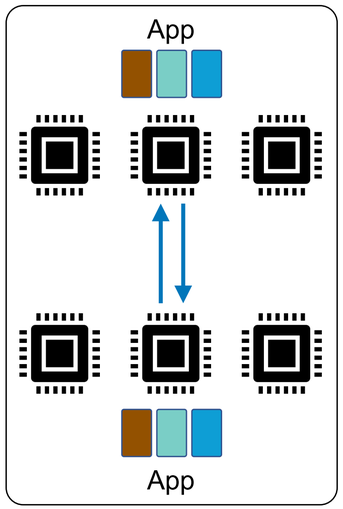
P99.9 tail latency



❖ **Linux is able to achieve only ~19us tail latency without resource contention**

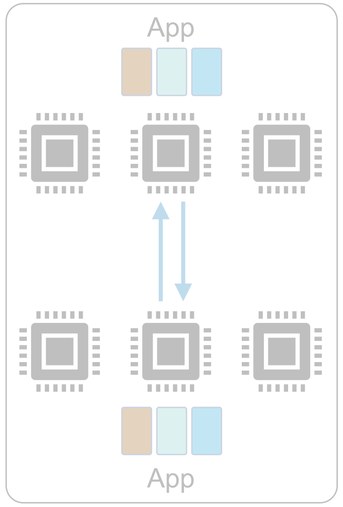
Rx_sched Latency Dominates Tail Latency

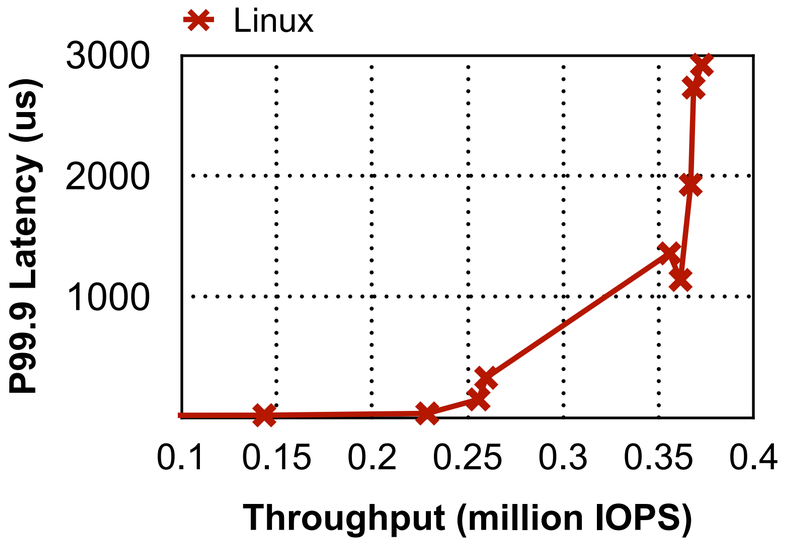
We increase the number of threads (one connection per thread) in one CPU core:



Rx_sched Latency Dominates Tail Latency

We increase the number of threads (one connection per thread) in one CPU core:

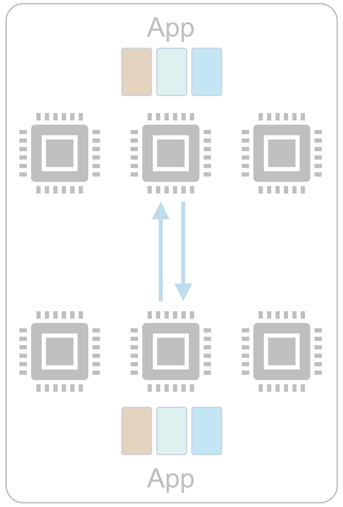


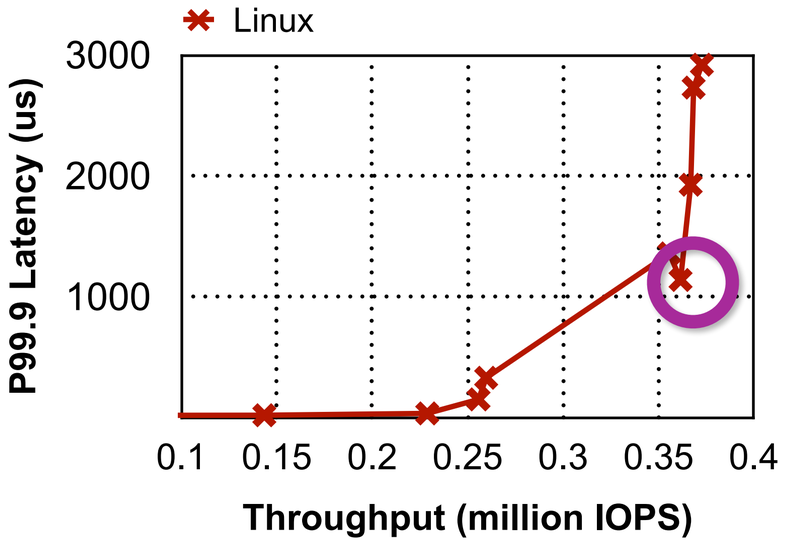


Rx_sched Latency Dominates Tail Latency

We increase the number of threads (one connection per thread) in one CPU core:

- **Linux reaches ~1100us P99.9 tail latency at knee point, with 24 threads in one logic core**

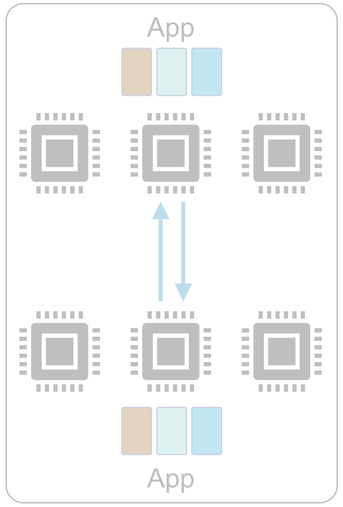


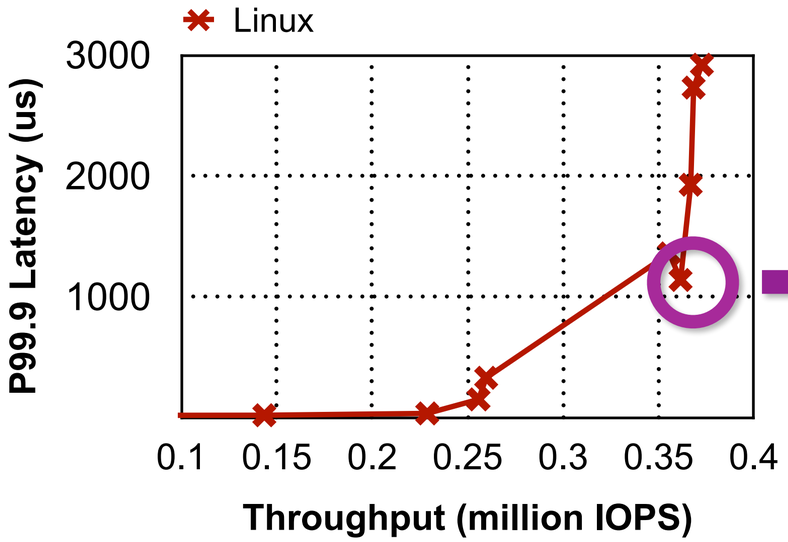


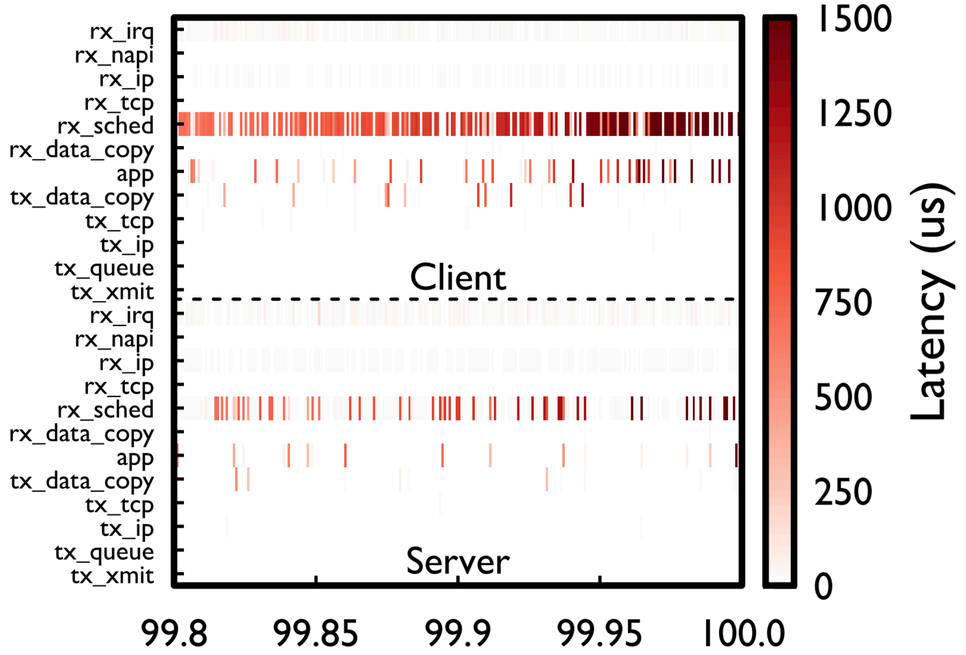
Rx_sched Latency Dominates Tail Latency

We increase the number of threads (one connection per thread) in one CPU core:

- **Linux reaches ~1100us P99.9 tail latency at knee point, with 24 threads in one logic core**



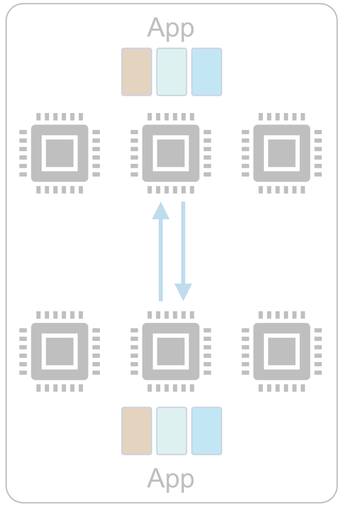


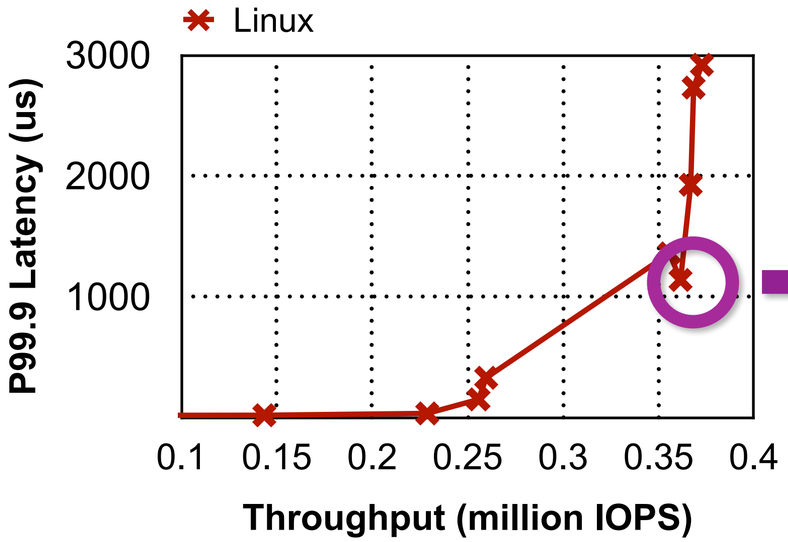


Rx_sched Latency Dominates Tail Latency

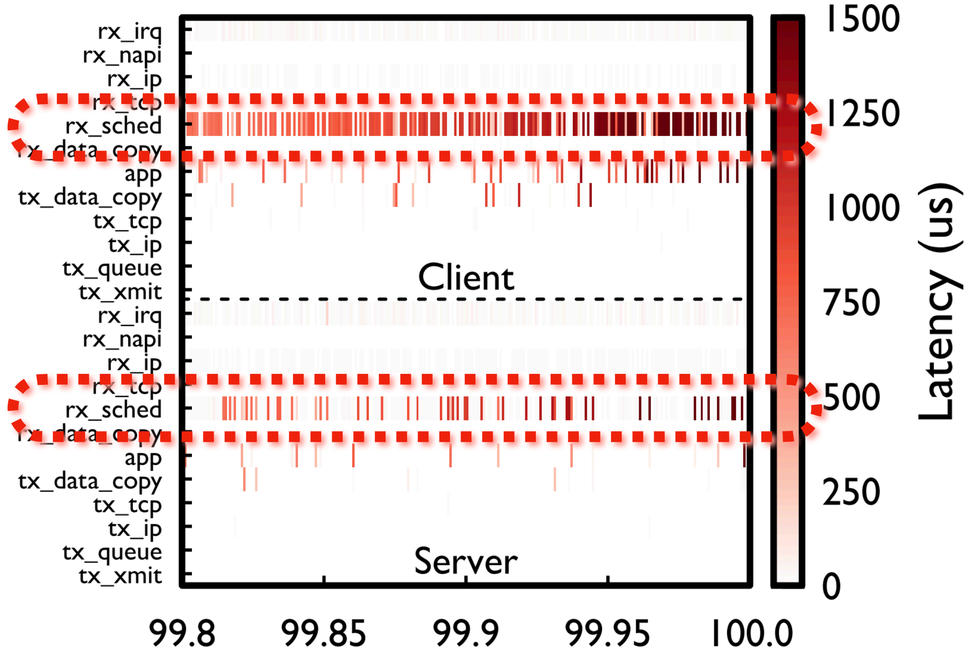
We increase the number of threads (one connection per thread) in one CPU core:

- **Linux reaches ~1100us P99.9 tail latency at knee point, with 24 threads in one logic core**





rx_sched: from thread woken up to scheduler choose thread to run

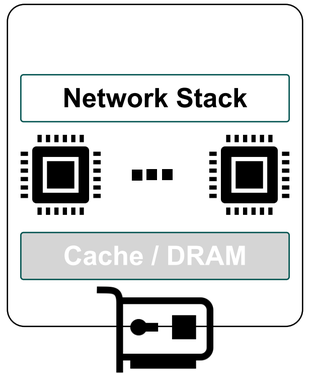


- **Rx_sched latency is the dominating factor in tail latency under contention**

Linux's Scheduler and Inaccurate IRQ Time Accounting

To understand the rx_sched latency, we need to understand the Linux scheduler:

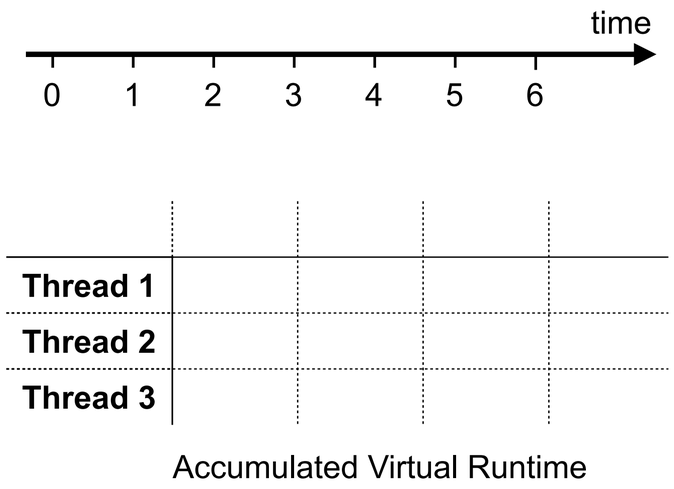
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**

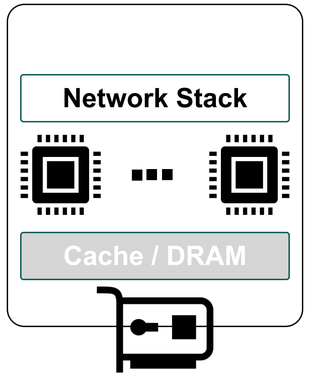


Linux's Scheduler and Inaccurate IRQ Time Accounting

To understand the rx_sched latency, we need to understand the Linux scheduler:

- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**

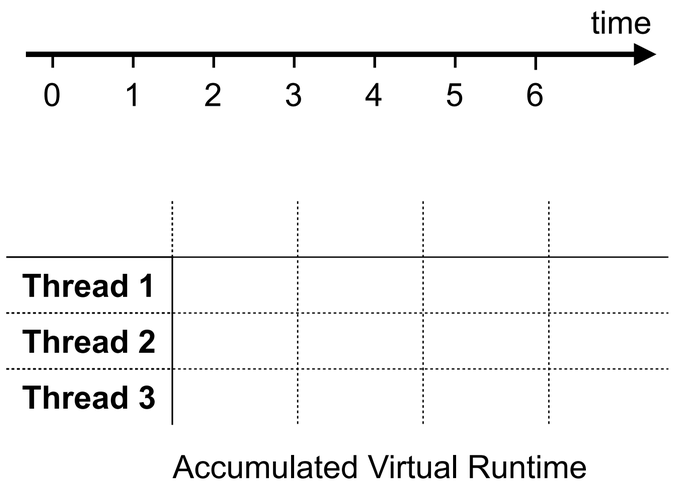


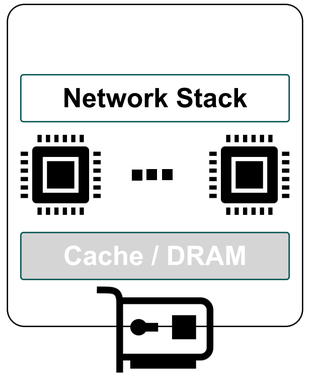


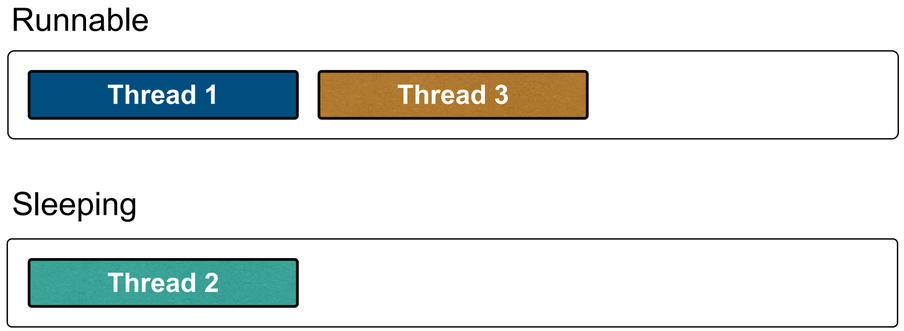
Linux's Scheduler and Inaccurate IRQ Time Accounting

To understand the rx_sched latency, we need to understand the Linux scheduler:

- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



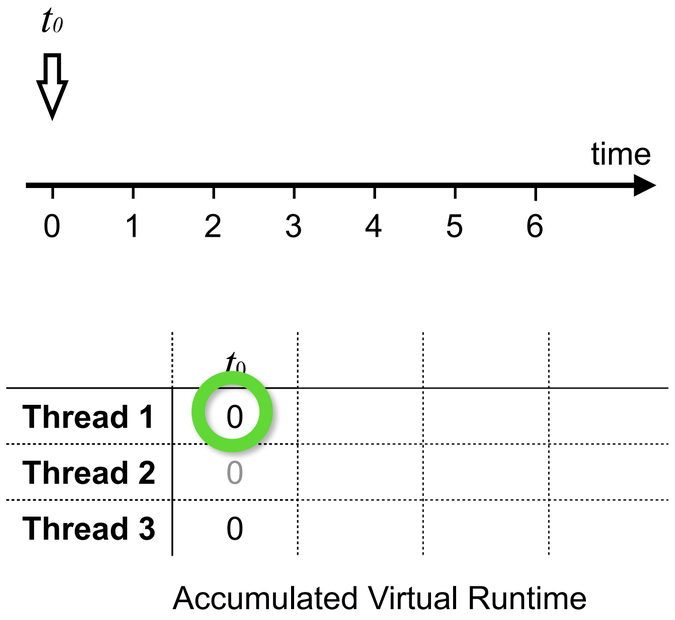


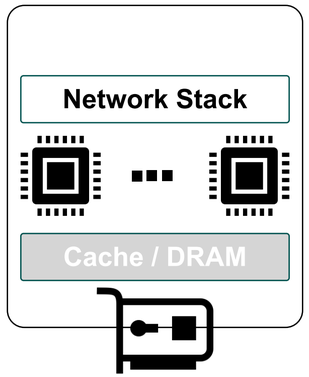


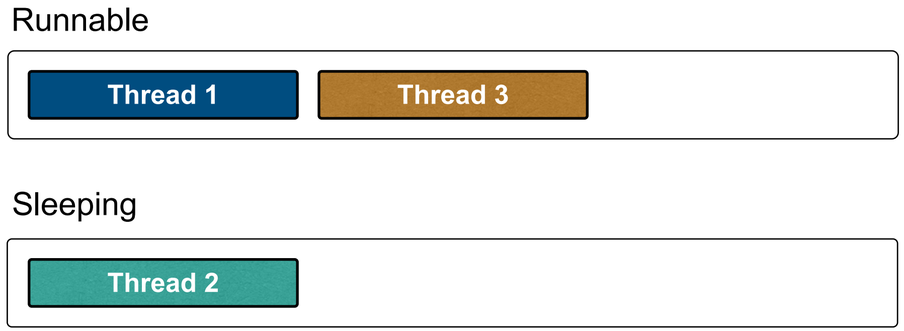
Linux's Scheduler and Inaccurate IRQ Time Accounting

To understand the rx_sched latency, we need to understand the Linux scheduler:

- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



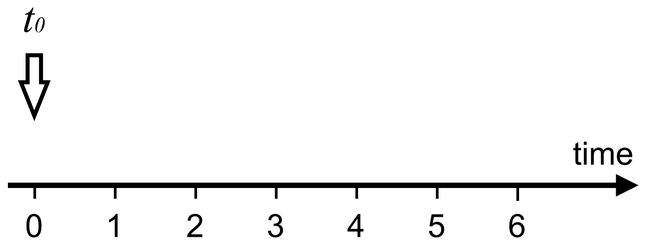




Linux's Scheduler and Inaccurate IRQ Time Accounting

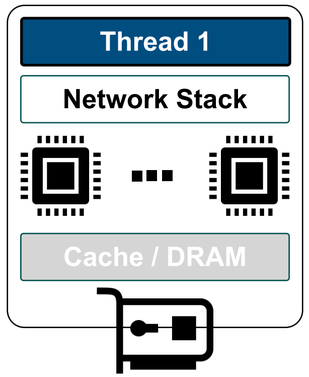
To understand the rx_sched latency, we need to understand the Linux scheduler:

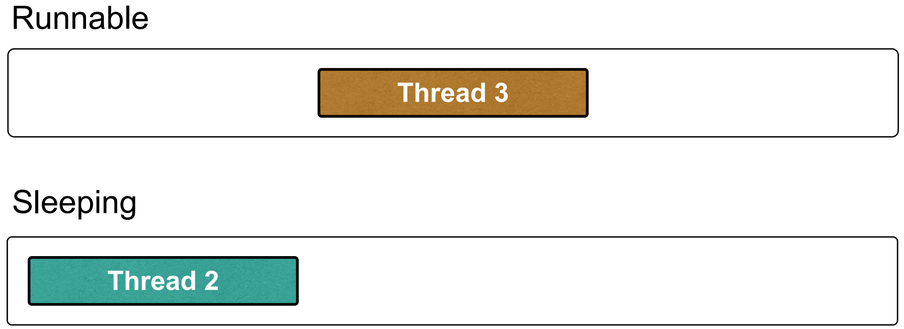
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0			
Thread 1	0			
Thread 2	0			
Thread 3	0			

Accumulated Virtual Runtime

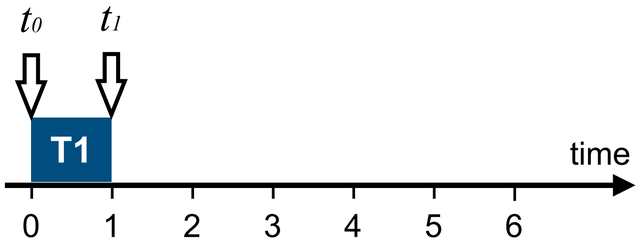




Linux's Scheduler and Inaccurate IRQ Time Accounting

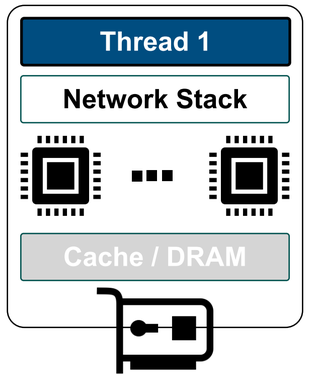
To understand the rx_sched latency, we need to understand the Linux scheduler:

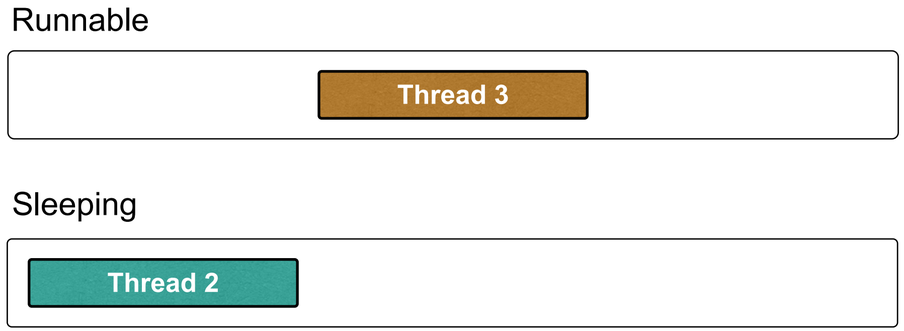
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1		
Thread 1	0	1		
Thread 2	0	0		
Thread 3	0	0		

Accumulated Virtual Runtime

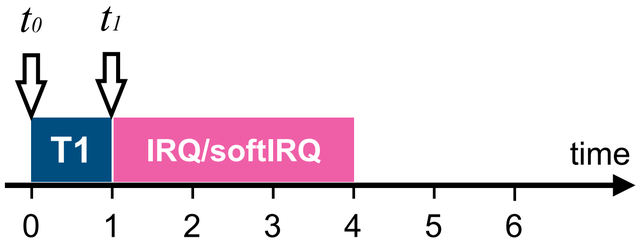




Linux's Scheduler and Inaccurate IRQ Time Accounting

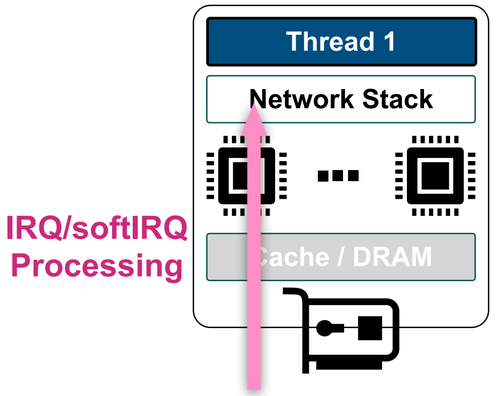
To understand the rx_sched latency, we need to understand the Linux scheduler:

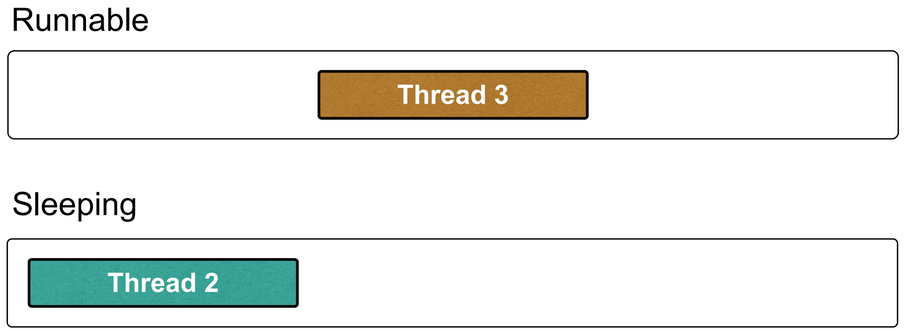
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1		
Thread 1	0	1		
Thread 2	0	0		
Thread 3	0	0		

Accumulated Virtual Runtime

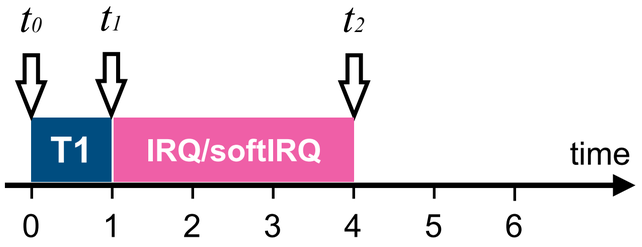




Linux's Scheduler and Inaccurate IRQ Time Accounting

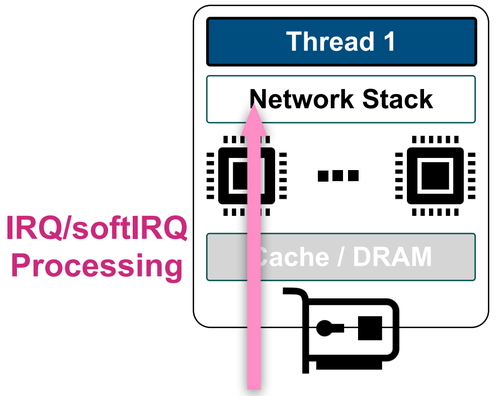
To understand the rx_sched latency, we need to understand the Linux scheduler:

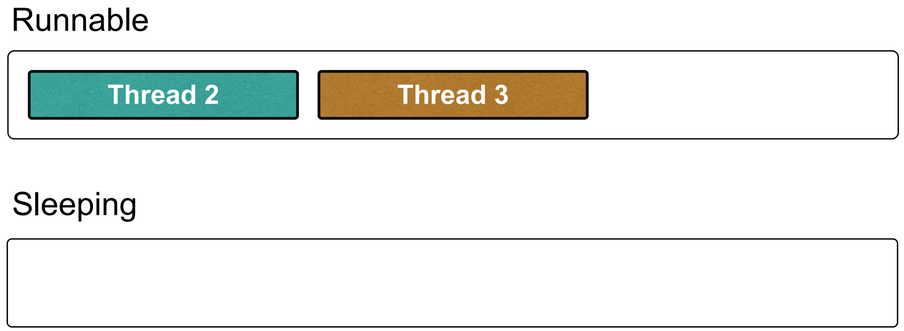
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1		
Thread 1	0	1		
Thread 2	0	0		
Thread 3	0	0		

Accumulated Virtual Runtime

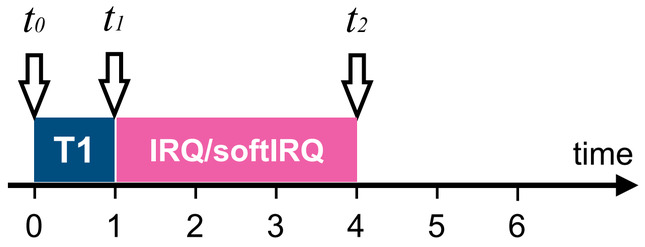




Linux's Scheduler and Inaccurate IRQ Time Accounting

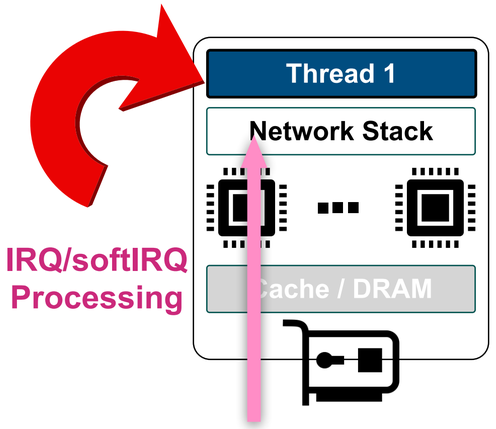
To understand the rx_sched latency, we need to understand the Linux scheduler:

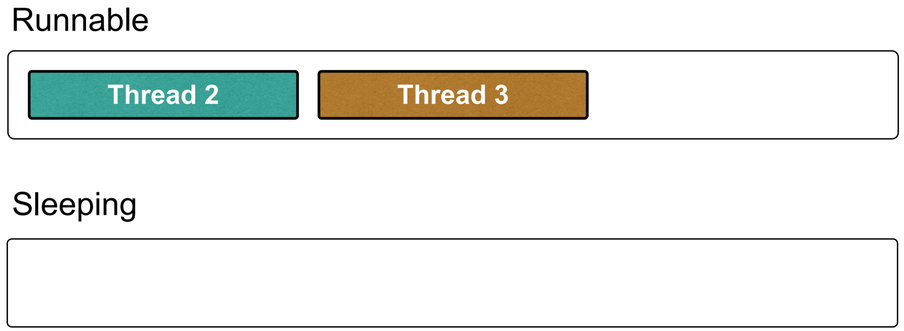
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1
Thread 1	0	1
Thread 2	0	0
Thread 3	0	0

Accumulated Virtual Runtime



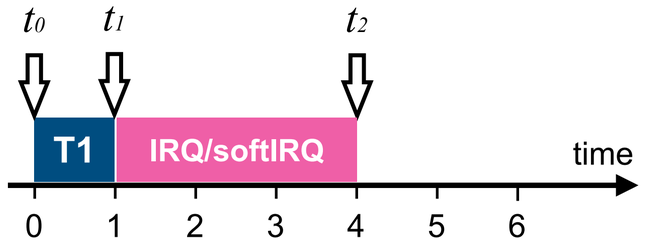


- **IRQ time is mistakenly accounted to the interrupted victim thread, inflating its virtual runtime**

Linux's Scheduler and Inaccurate IRQ Time Accounting

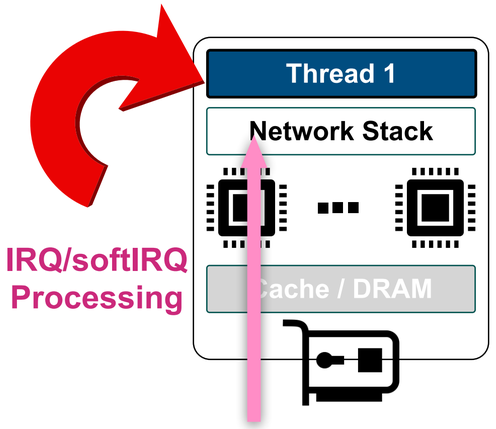
To understand the rx_sched latency, we need to understand the Linux scheduler:

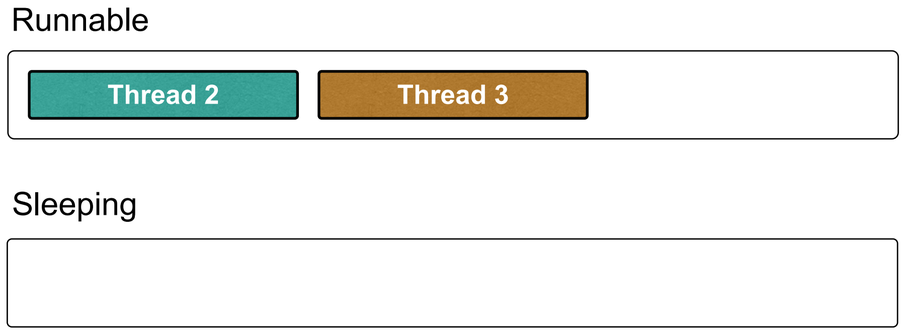
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1	t_2
Thread 1	0	1	4 (1)
Thread 2	0	0	0
Thread 3	0	0	0

Accumulated Virtual Runtime



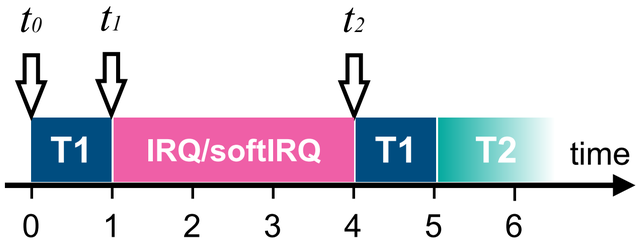


- **IRQ time is mistakenly accounted to the interrupted victim thread, inflating its virtual runtime**

Linux's Scheduler and Inaccurate IRQ Time Accounting

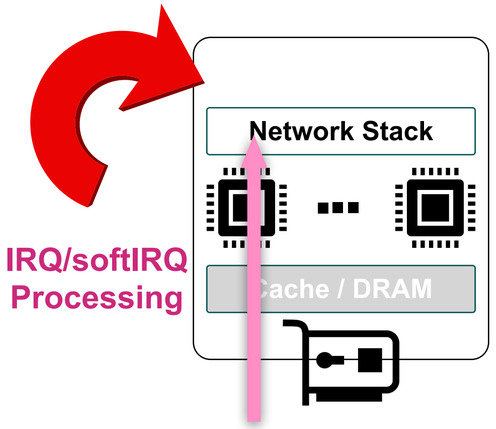
To understand the rx_sched latency, we need to understand the Linux scheduler:

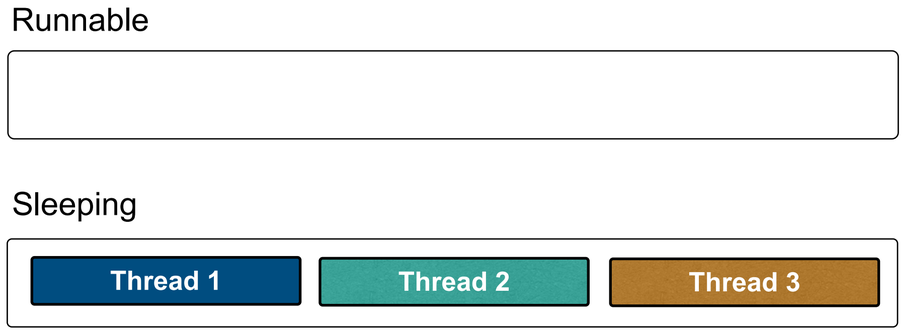
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1	t_2
Thread 1	0	1	4 (1)
Thread 2	0	0	0
Thread 3	0	0	0

Accumulated Virtual Runtime



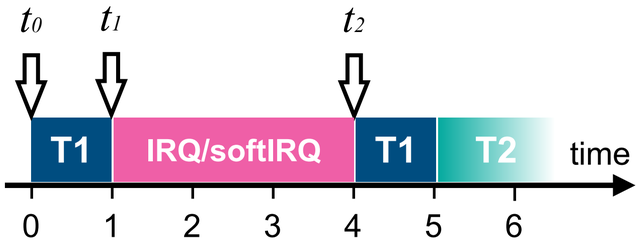


- **IRQ time is mistakenly accounted to the interrupted victim thread, inflating its virtual runtime**

Linux's Scheduler and Inaccurate IRQ Time Accounting

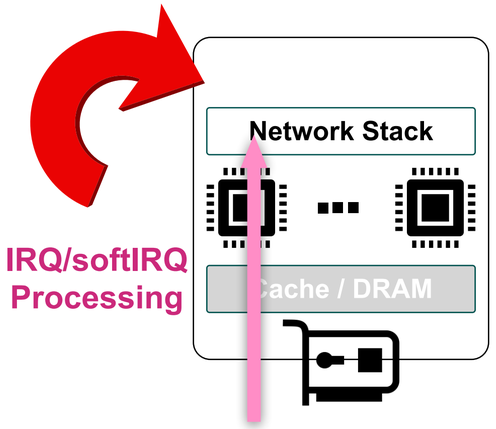
To understand the rx_sched latency, we need to understand the Linux scheduler:

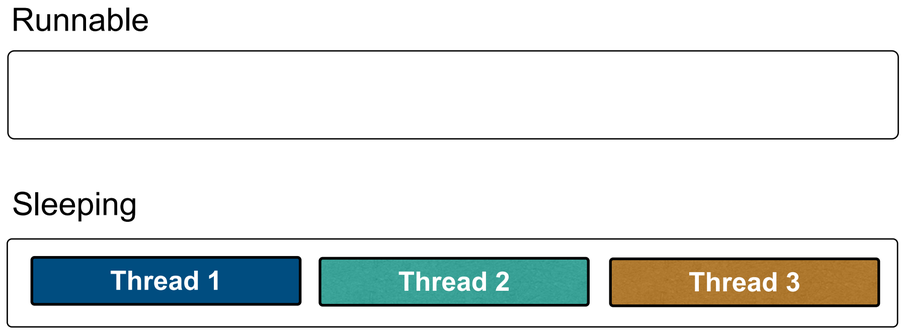
- **Linux's CFS (Completely Fair Scheduler) prioritizes the thread with smallest (virtual) runtime**



	t_0	t_1	t_2	Sleep
Thread 1	0	1	4 (1)	5 (2)
Thread 2	0	0	0 +2	2
Thread 3	0	0	0 +2	2

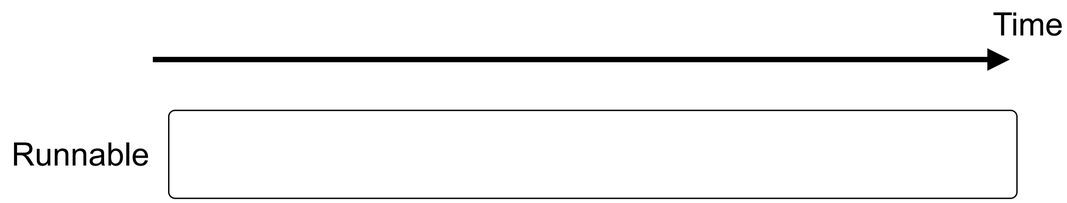
Accumulated Virtual Runtime



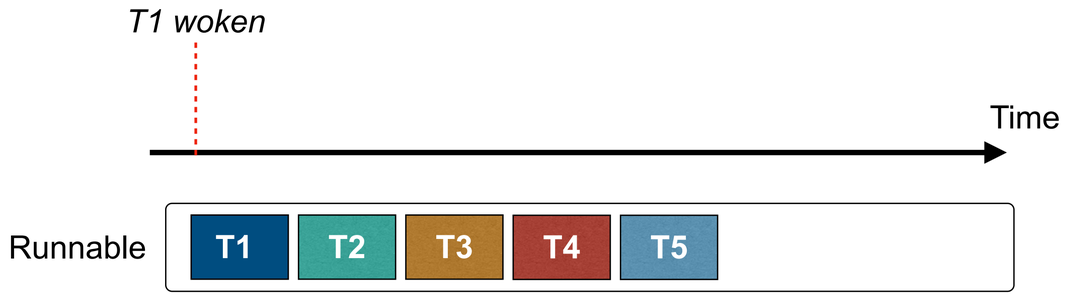


- **IRQ time is mistakenly accounted to the interrupted victim thread, inflating its virtual runtime**

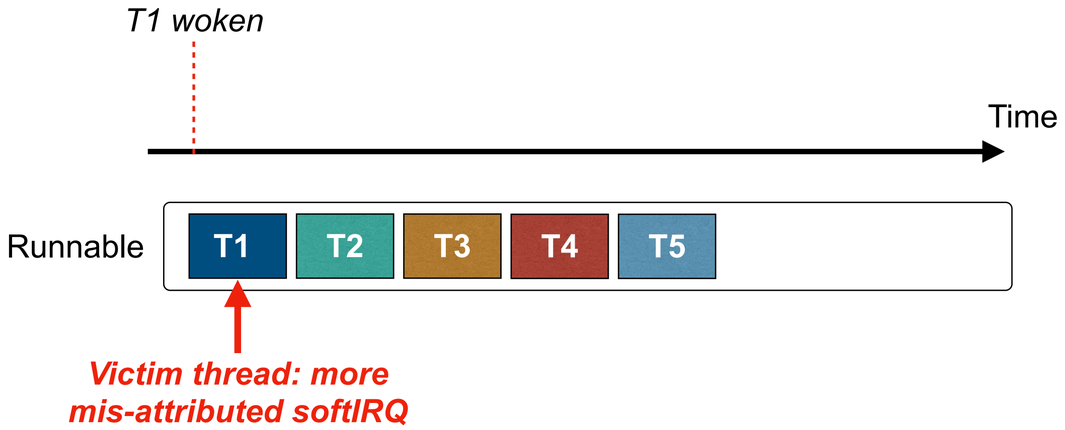
Inaccurate Time Accounting is the Reason for High Rx_sched



Inaccurate Time Accounting is the Reason for High Rx_sched



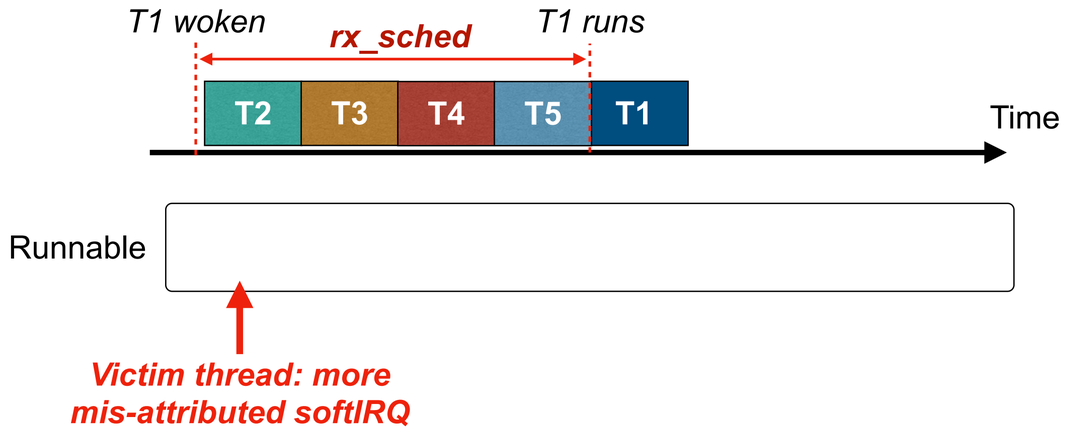
Inaccurate Time Accounting is the Reason for High Rx_sched



Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

Accumulated Virtual Runtime

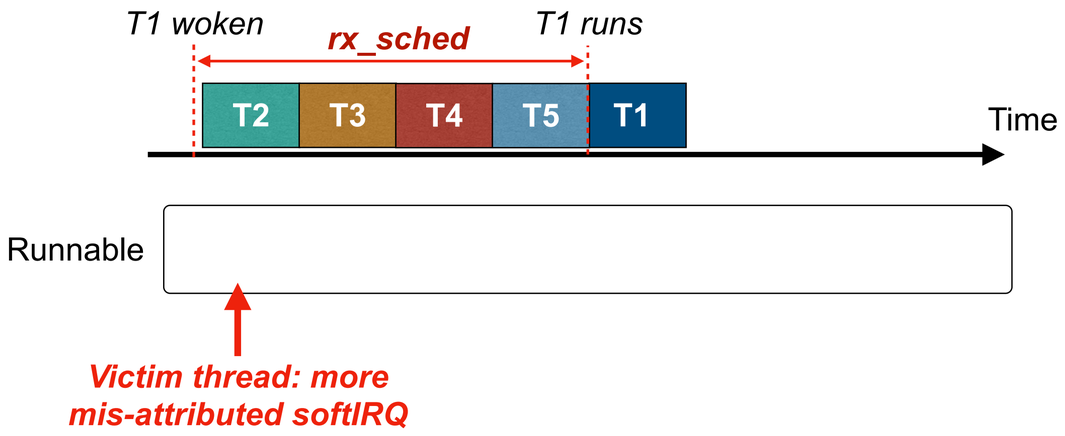
Inaccurate Time Accounting is the Reason for High Rx_sched



Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

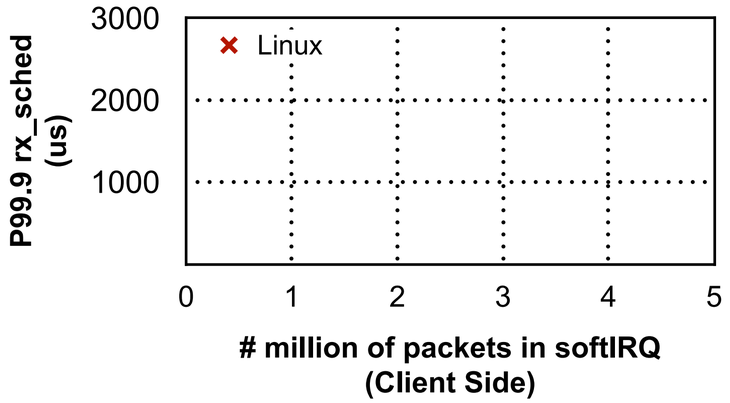
Accumulated Virtual Runtime

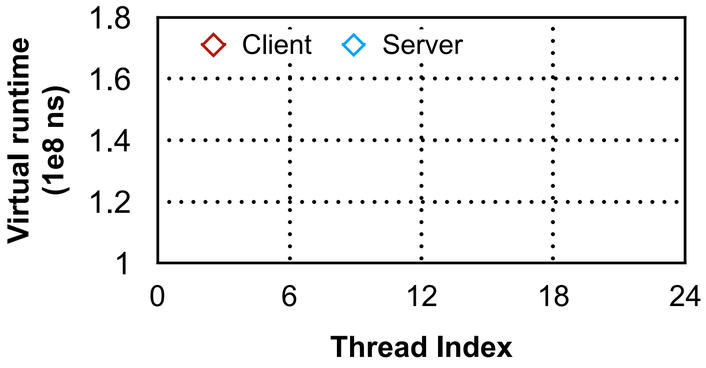
Inaccurate Time Accounting is the Reason for High Rx_sched



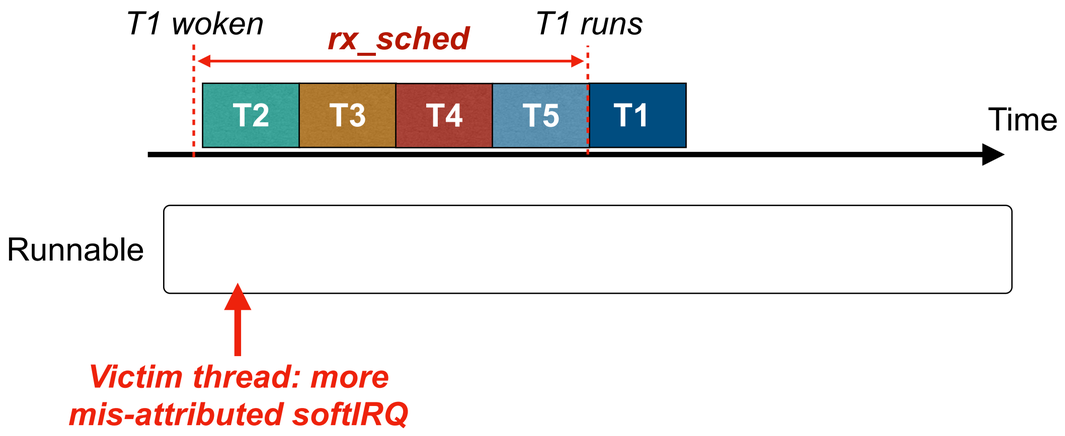
Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

Accumulated Virtual Runtime



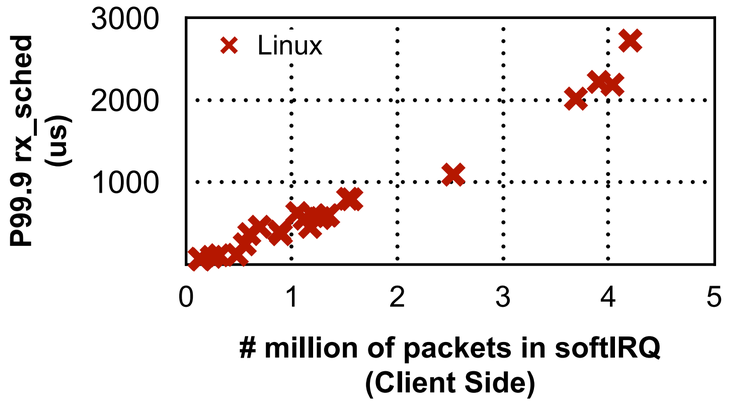


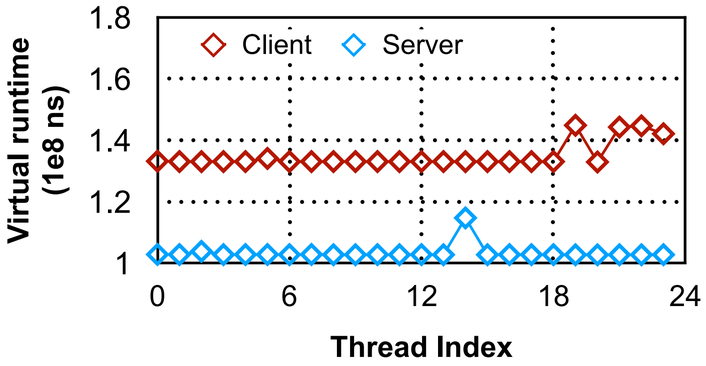
Inaccurate Time Accounting is the Reason for High Rx_sched



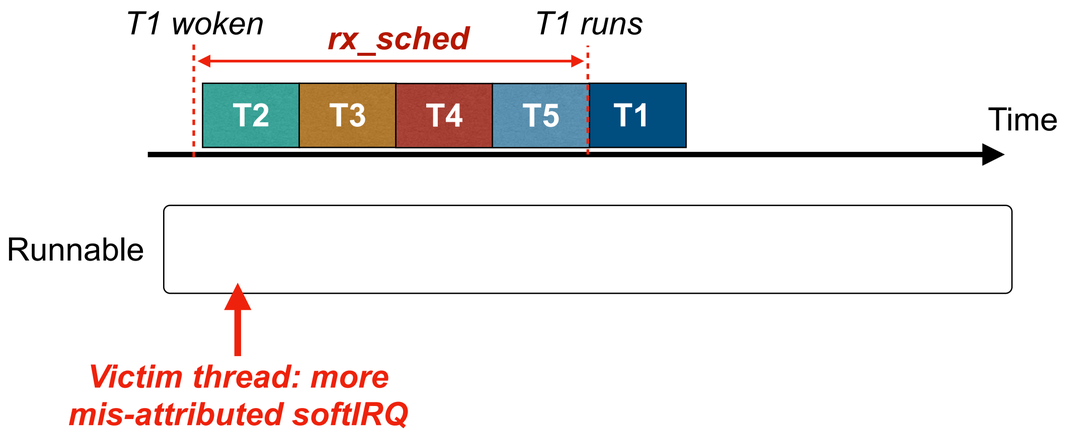
Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

Accumulated Virtual Runtime



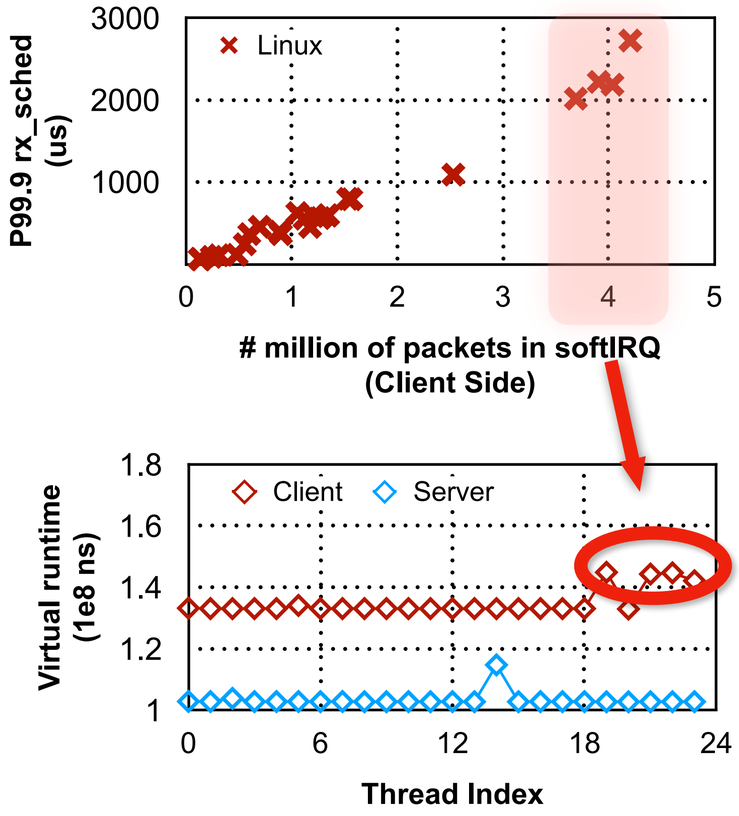


Inaccurate Time Accounting is the Reason for High Rx_sched

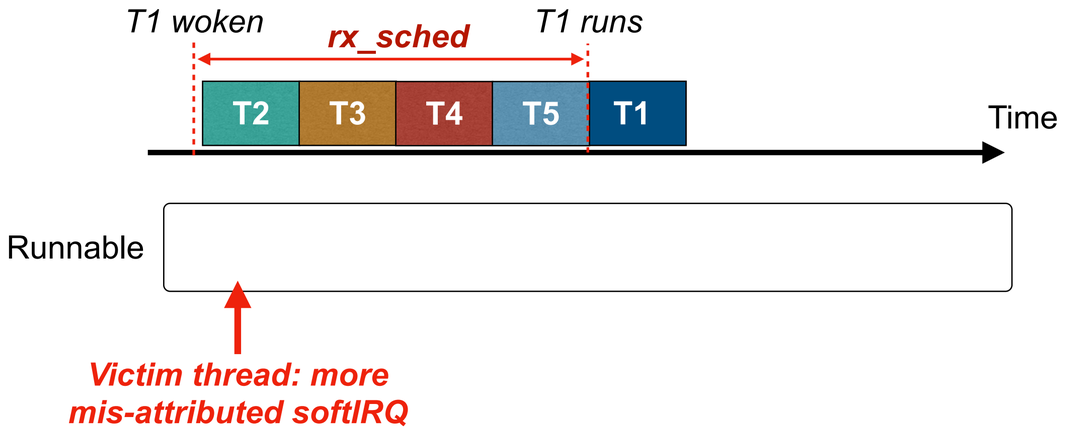


Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

Accumulated Virtual Runtime

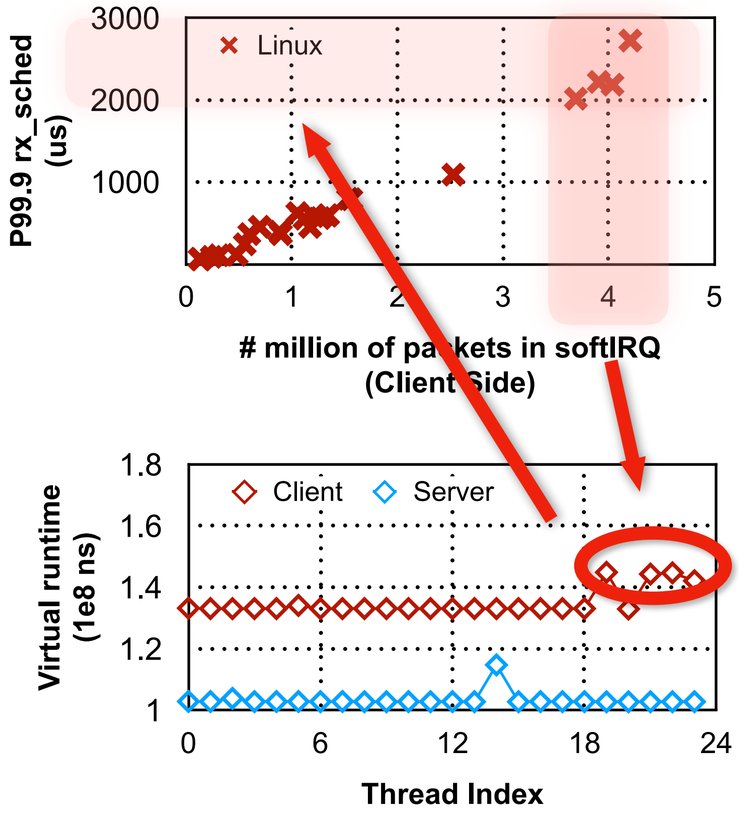


Inaccurate Time Accounting is the Reason for High Rx_sched

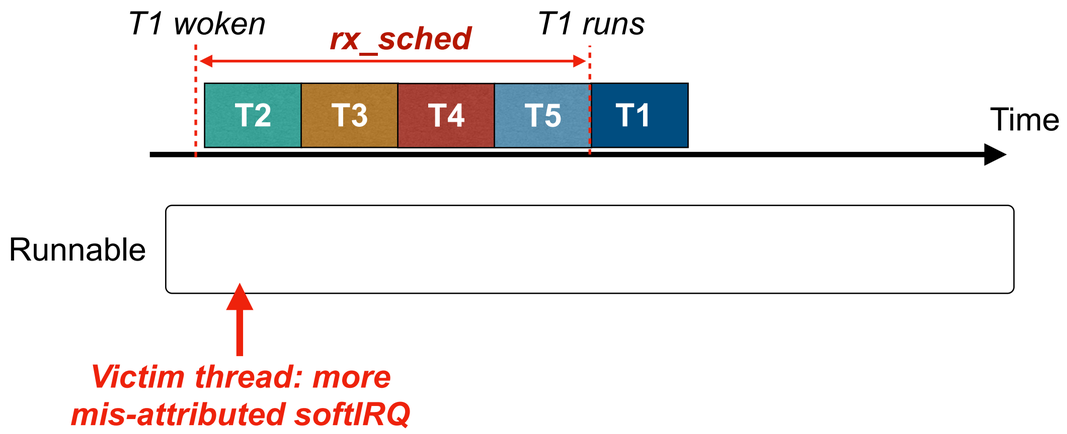


Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

Accumulated Virtual Runtime

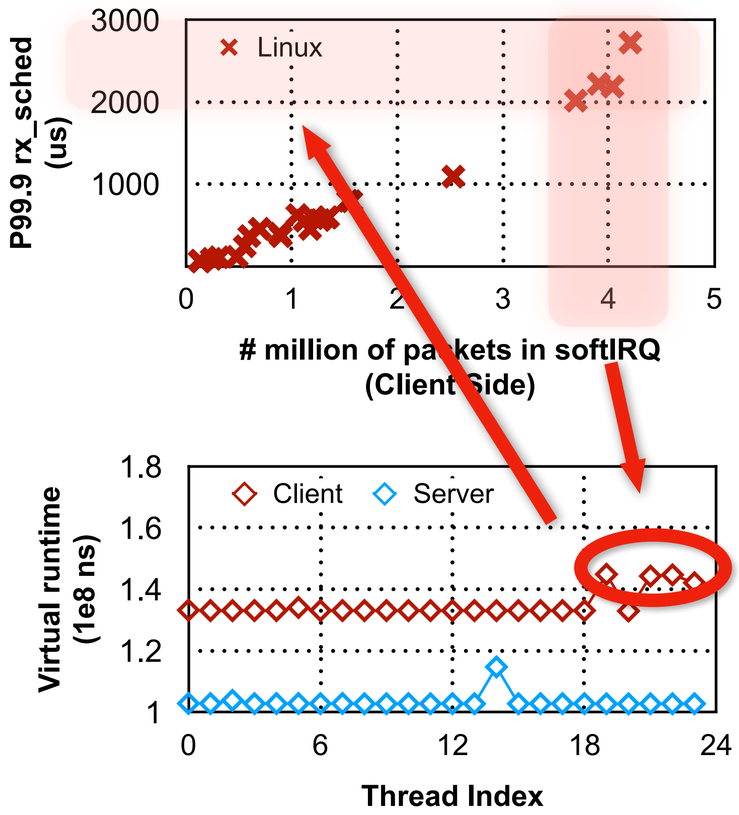


Inaccurate Time Accounting is the Reason for High Rx_sched



Thread	T1	T2	T3	T4	T5
Virtual Runtime	5 (2)	2	2	2	2

Accumulated Virtual Runtime

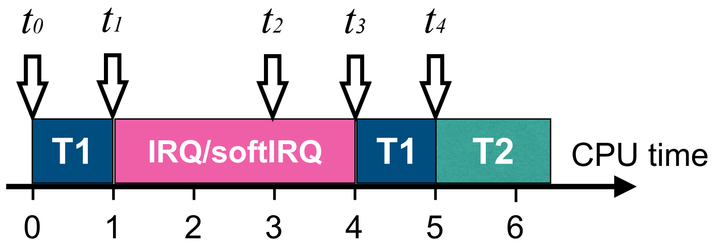


❖ **Inaccurate time accounting leads to high rx_sched, and therefore high tail latency**

Accurate IRQ Time Accounting

❖ **We implemented an accurate IRQ time accounting patch (ACCa) to address the issue** (details in our paper)

- The accumulated virtual runtime accurately reflects threads' CPU occupation



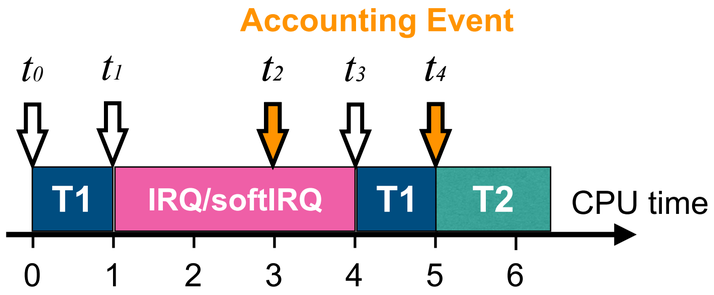
	t_0	t_1	t_2	t_3	t_4
Thread 1	0	1	1	1	2

Accumulated Virtual Runtime

Accurate IRQ Time Accounting

❖ **We implemented an accurate IRQ time accounting patch (ACCa) to address the issue** (details in our paper)

- The accumulated virtual runtime accurately reflects threads' CPU occupation
- **Linux has similar IRQ time accounting option — it still fails when accounting event occurs during softIRQ.**



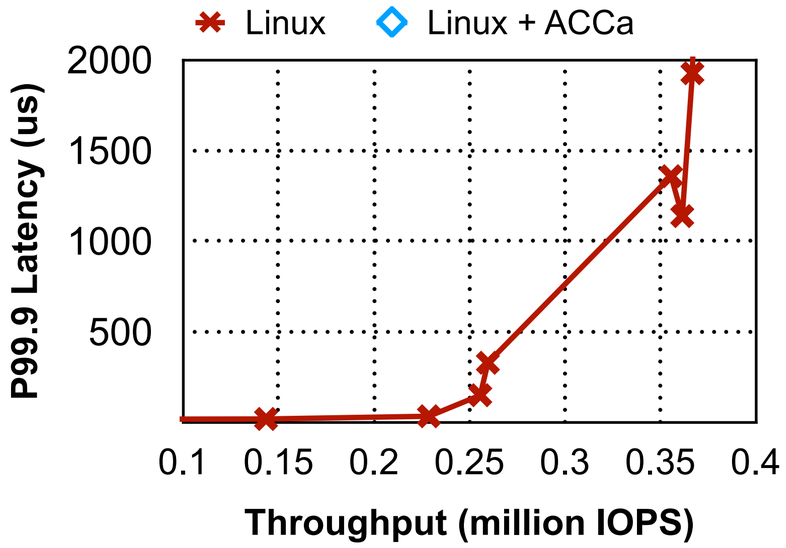
	t_0	t_1	t_2	t_3	t_4
Thread 1	0	1	1	1	2

Accumulated Virtual Runtime

Accurate IRQ Time Accounting Cuts the Tail

❖ **We implemented an accurate IRQ time accounting patch (ACCa) to address the issue** (details in our paper)

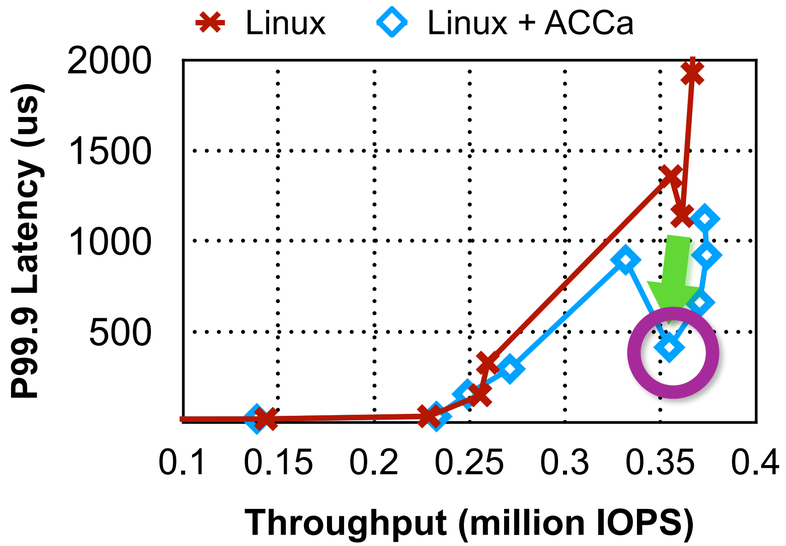
- The accumulated virtual runtime accurately reflects threads' CPU occupation
- **Linux has similar IRQ time accounting option — it still fails when accounting event occurs during softIRQ.**



Accurate IRQ Time Accounting Cuts the Tail

❖ **We implemented an accurate IRQ time accounting patch (ACCa) to address the issue** (details in our paper)

- The accumulated virtual runtime accurately reflects threads' CPU occupation
- **Linux has similar IRQ time accounting option — it still fails when accounting event occurs during softIRQ.**

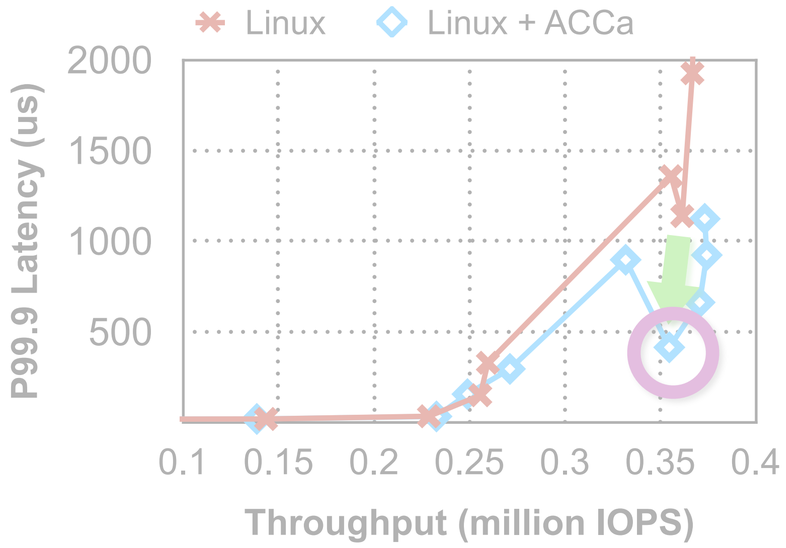


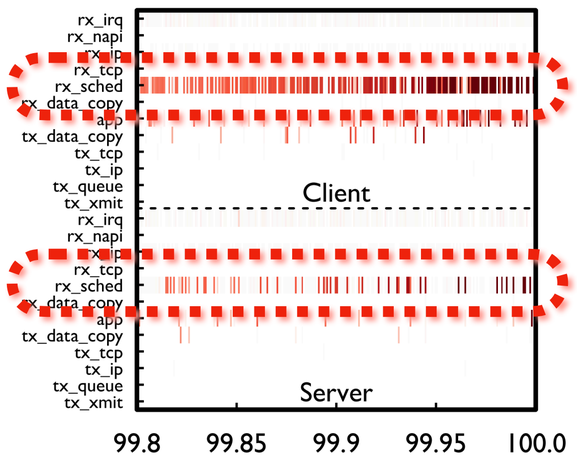
❖ **Accurate IRQ time accounting improves P99.9 latency by 2.7x**

Accurate IRQ Time Accounting Cuts the Tail

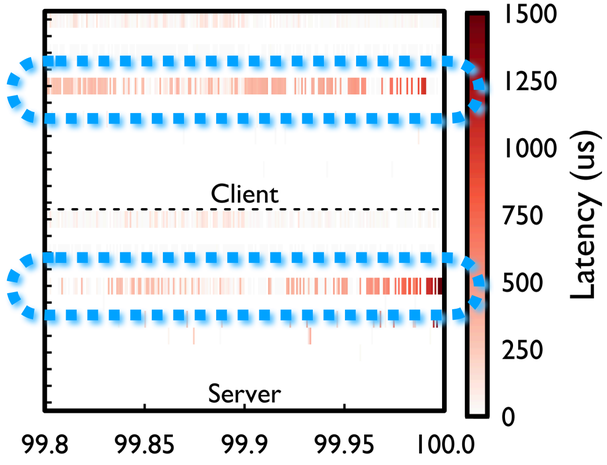
❖ **We implemented an accurate IRQ time accounting patch (ACCa) to address the issue** (details in our paper)

- The accumulated virtual runtime accurately reflects threads' CPU occupation
- **Linux has similar IRQ time accounting option — it still fails when accounting event occurs during softIRQ.**





Linux



Linux + ACCa

❖ **Accurate IRQ time accounting improves P99.9 latency by 2.7x**

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

✦ Choose your parallelism carefully

- More in-flight requests can be much more latency-efficient than more connections

See our paper

Four Main Lessons from Our Study

✦ Scheduling dominates the high tail latency

- Misattributed softIRQ time inflates virtual runtime and scheduling latency
- Accurate CPU time accounting reduces P99.9 tail latency by **2.7x**

✦ Runtime may not be the ideal scheduling abstraction for low latency

- Virtual runtime reflects not only computation, but also hardware overhead
- Request/response-based scheduling further reduces P99.9 tail latency by **1.5x**

✦ Traffic predictability can make interrupt tuning more effective

- DIM struggles to find the right interrupt setting under bursty traffic
- Predictable traffic enables another **1.28x** latency reduction

✦ Choose your parallelism carefully

- More in-flight requests can be much more latency-efficient than more connections

See our paper

The Runtime Gap across Threads Persists

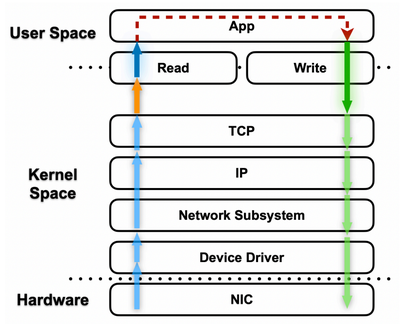
To verify our patch, we compare scheduler's **Virtual Runtime** to our measured **Processing Time**:

The Runtime Gap across Threads Persists

To verify our patch, we compare scheduler's **Virtual Runtime** to our measured **Processing Time**:

	t_0	t_1	t_2	t_3
Thread 1	0	1	1	2
Thread 2	3	3	3	3
Thread 3	4	4	4	4

Scheduler's Virtual Runtime



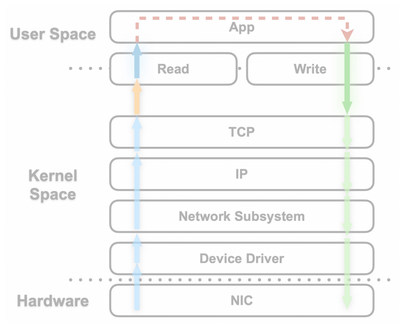
Measured Processing Time

The Runtime Gap across Threads Persists

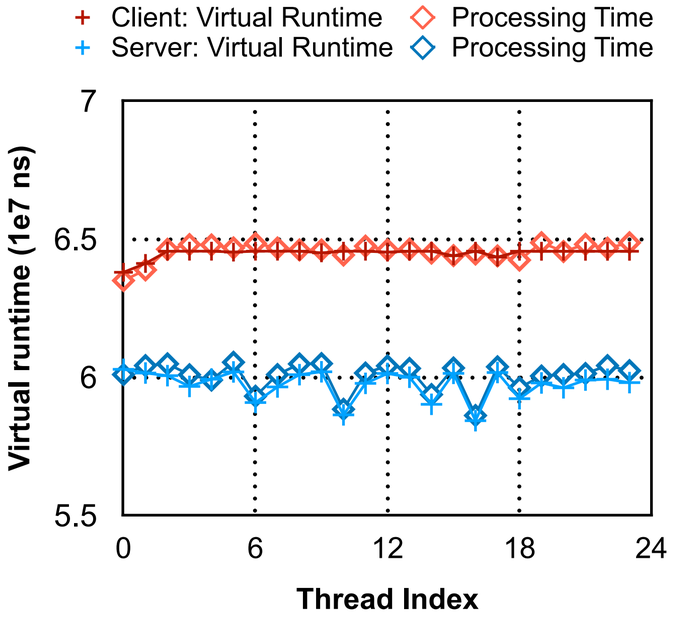
To verify our patch, we compare scheduler's **Virtual Runtime** to our measured **Processing Time**:

	t_0	t_1	t_2	t_3
Thread 1	0	1	1	2
Thread 2	3	3	3	3
Thread 3	4	4	4	4

Scheduler's Virtual Runtime



Measured Processing Time



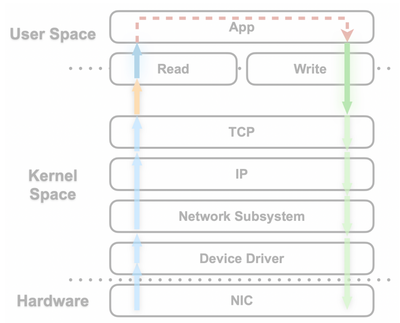
❖ **Our patch resolves the time accounting issue: Virtual Runtime = Processing Time**

The Runtime Gap across Threads Persists

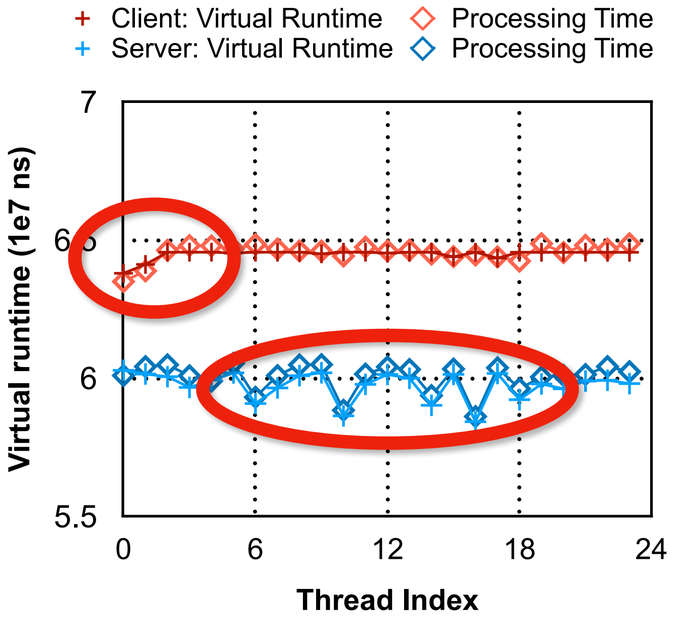
To verify our patch, we compare scheduler's **Virtual Runtime** to our measured **Processing Time**:

	t_0	t_1	t_2	t_3
Thread 1	0	1	1	2
Thread 2	3	3	3	3
Thread 3	4	4	4	4

Scheduler's Virtual Runtime



Measured Processing Time



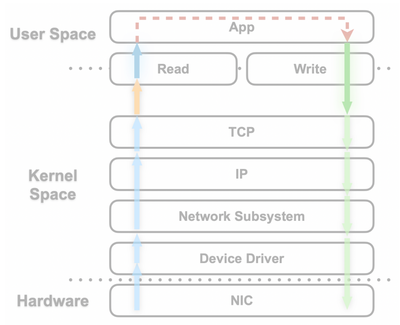
- **Our patch resolves the time accounting issue: Virtual Runtime = Processing Time**
- **Virtual Runtime is still different across threads**

The Runtime Gap across Threads Persists

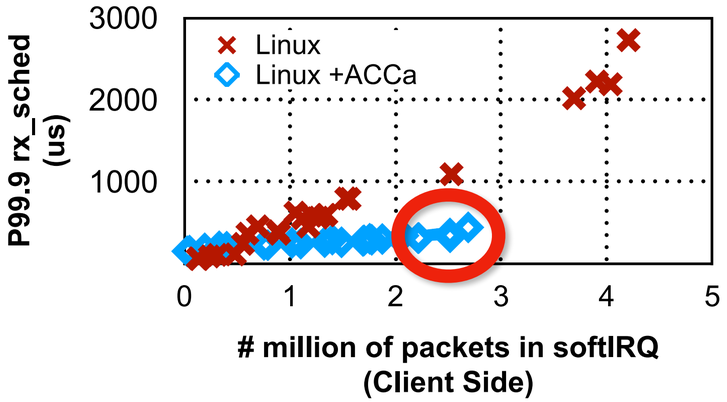
To verify our patch, we compare scheduler's **Virtual Runtime** to our measured **Processing Time**:

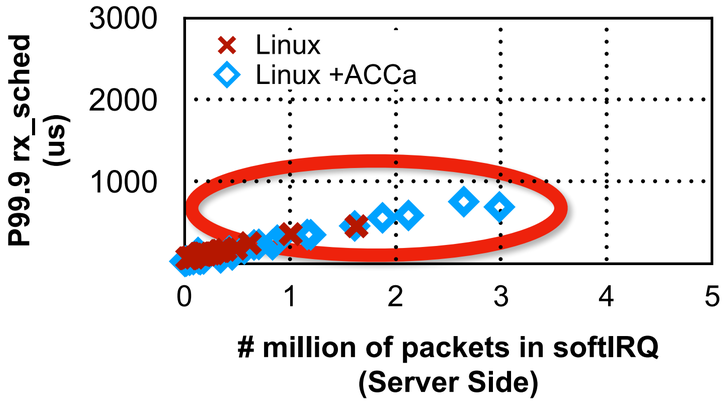
	t_0	t_1	t_2	t_3
Thread 1	0	1	1	2
Thread 2	3	3	3	3
Thread 3	4	4	4	4

Scheduler's Virtual Runtime



Measured Processing Time





- **Our patch resolves the time accounting issue: Virtual Runtime = Processing Time**
- **Virtual Runtime is still different across threads, leading to unfair rx_sched**

Virtual Runtime Can Lie about Fairness

Counterintuitive with the same workload:

- **Virtual Runtime = Processing Time**

Virtual Runtime Can Lie about Fairness

Counterintuitive with the same workload:

- **Virtual Runtime = Processing Time = Code/Instructions** **(Same)**

Virtual Runtime Can Lie about Fairness

Counterintuitive with the same workload:

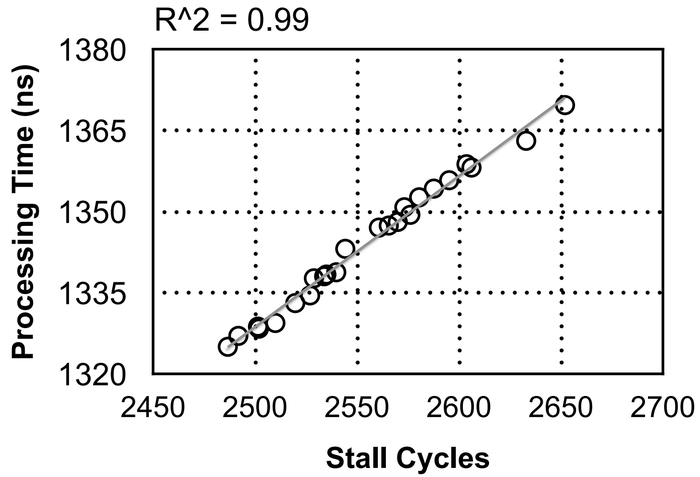
- **Virtual Runtime = Processing Time = Code/Instructions (Same) + Stall Cycles (Different)**

Virtual Runtime Can Lie about Fairness

Counterintuitive with the same workload:

❖ **Virtual Runtime = Processing Time = Code/Instructions (Same) + Stall Cycles (Different)**

- Stall Cycles vary across threads, and have linear relationship with Processing Time

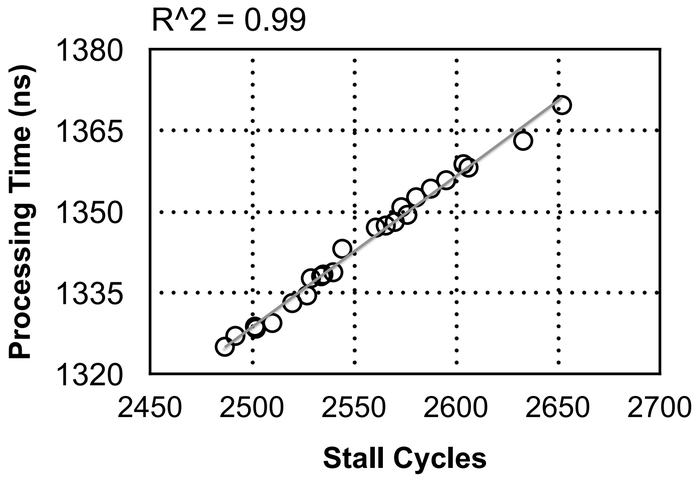


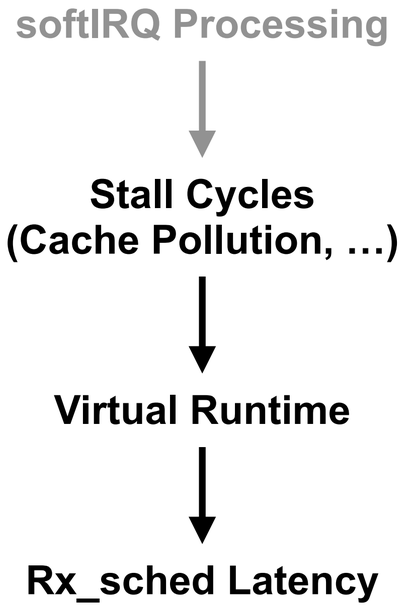
Virtual Runtime Can Lie about Fairness

Counterintuitive with the same workload:

❖ **Virtual Runtime = Processing Time = Code/Instructions (Same) + Stall Cycles (Different)**

- Stall Cycles vary across threads, and have linear relationship with Processing Time



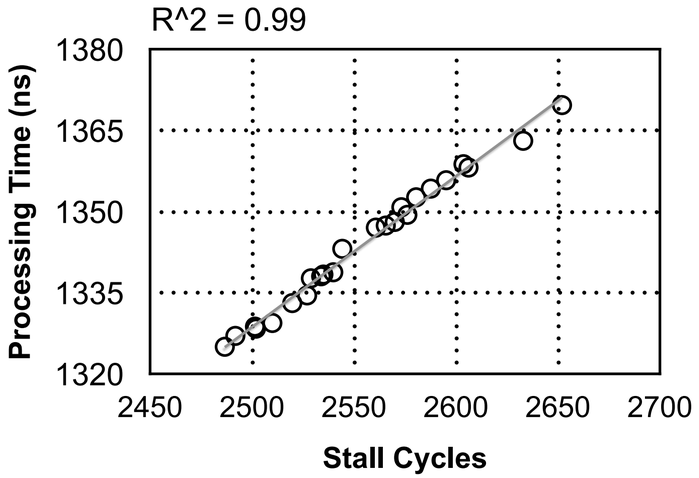


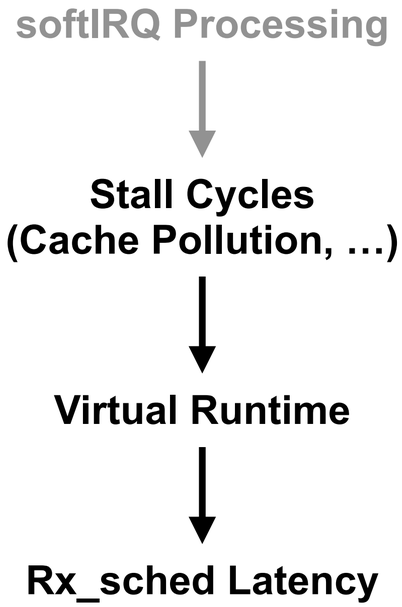
Virtual Runtime Can Lie about Fairness

Counterintuitive with the same workload:

❖ **Virtual Runtime = Processing Time = Code/Instructions (Same) + Stall Cycles (Different)**

- Stall Cycles vary across threads, and have linear relationship with Processing Time





❖ **Identical work does not essentially take identical CPU time — hardware states matter**

Accounts for Application Progress

Accounts for Application Progress

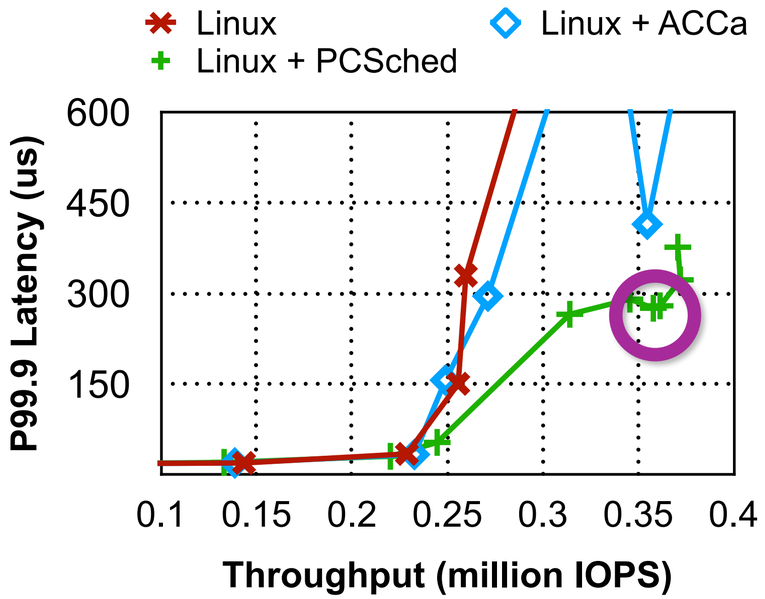
As a proof of concept:

- **Scheduling threads based on number of requests (+PCSched) further improves latency by 1.5x**

Accounts for Application Progress

As a proof of concept:

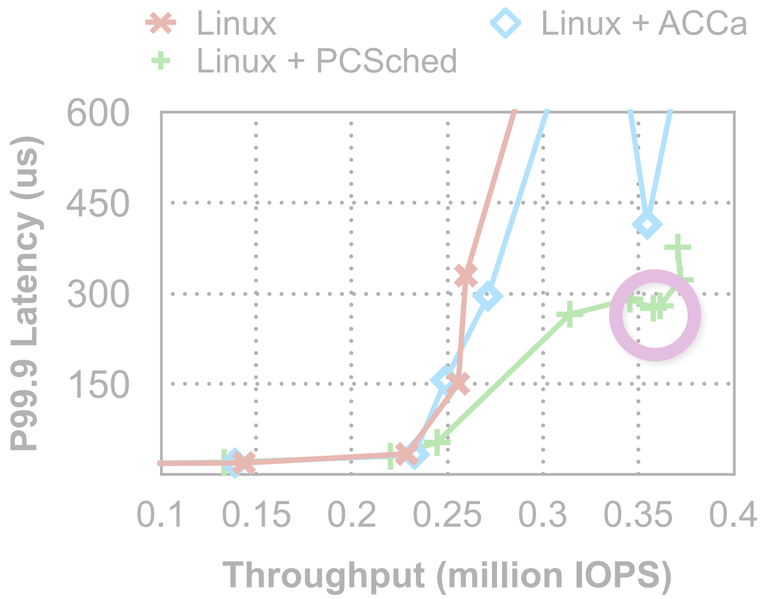
- **Scheduling threads based on number of requests (+PCSched) further improves latency by 1.5x**



Accounts for Application Progress

As a proof of concept:

- **Scheduling threads based on number of requests (+PCSched) further improves latency by 1.5x**

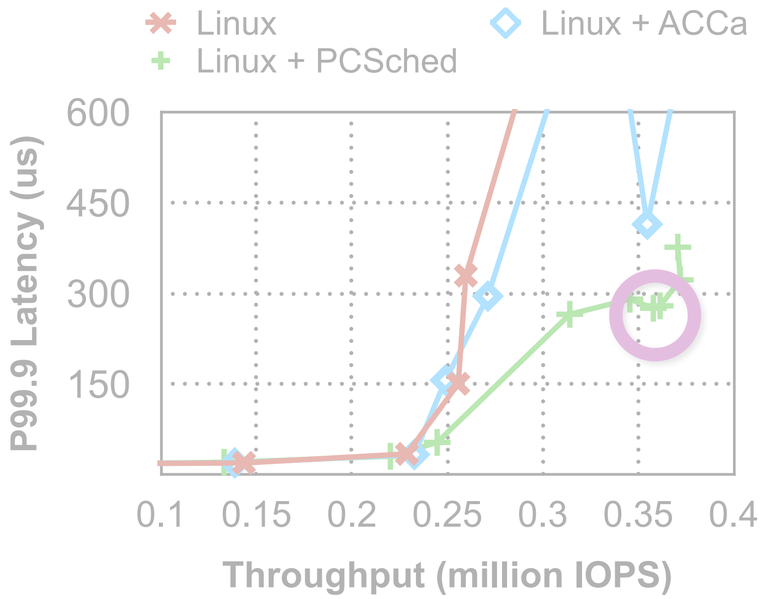


Two possible and interesting research directions:

Accounts for Application Progress

As a proof of concept:

- **Scheduling threads based on number of requests (+PCSched) further improves latency by 1.5x**



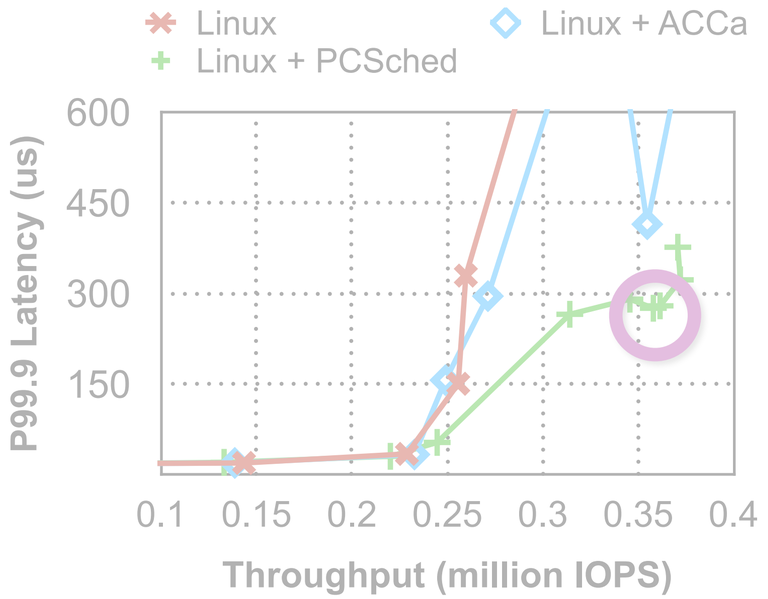
Two possible and interesting research directions:

- **Identifying the appropriate scheduling abstraction for different classes of applications**

Accounts for Application Progress

As a proof of concept:

- **Scheduling threads based on number of requests (+PCSched) further improves latency by 1.5x**



Two possible and interesting research directions:

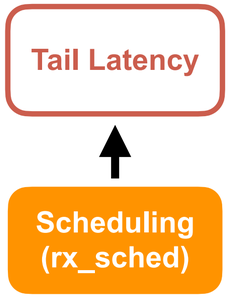
- **Identifying the appropriate scheduling abstraction for different classes of applications**
- **Otherwise, make CPU runtime less sensitive to hardware noise, potentially by hardware counters**

Summary

Tail Latency

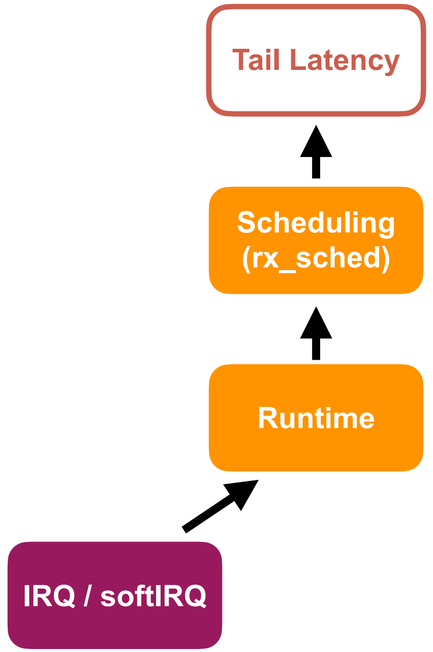
Summary

- **Packet processing itself is not the reason for tail latency**
  - An isolated thread is able to achieve ~19us P99.9 latency



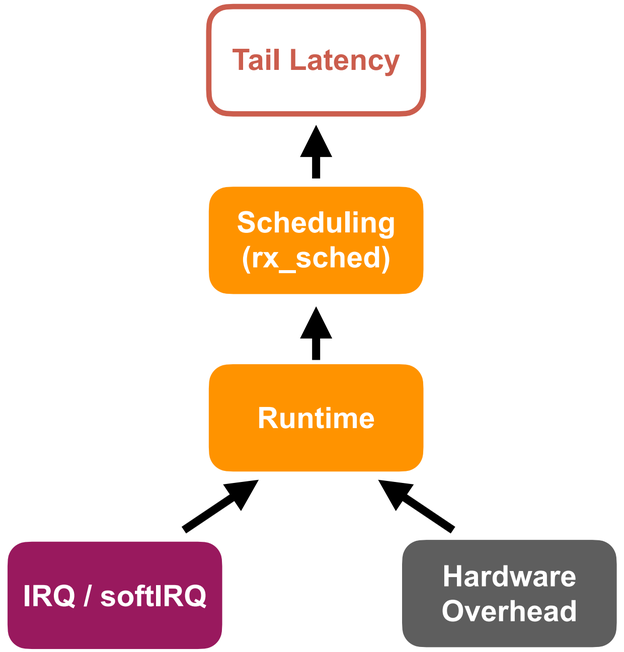
Summary

- ✦ **Packet processing itself is not the reason for tail latency**
  - An isolated thread is able to achieve ~19us P99.9 latency
- ✦ **Inaccurate IRQ time accounting leads to high scheduling latency**
  - ~2.7x improvement in P99.9 latency with accurate time accounting (ACCa)
  - ➡ Achieve **accurate** time accounting with our ACCa patch



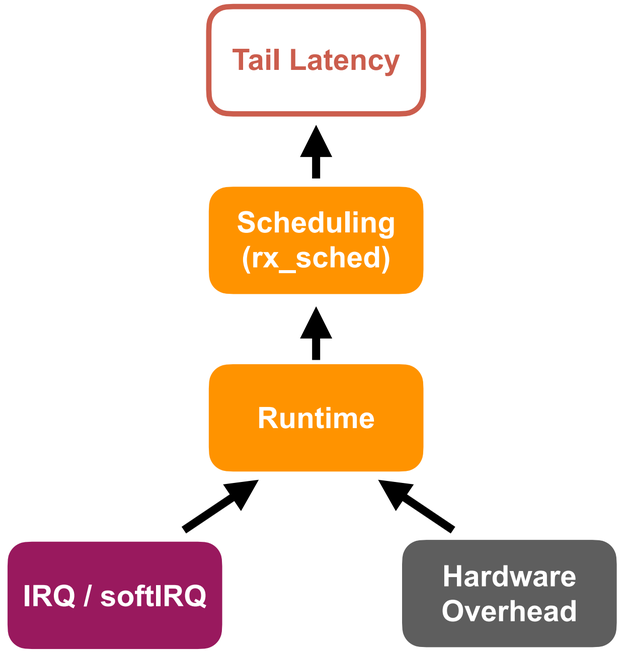
Summary

- **Packet processing itself is not the reason for tail latency**
  - An isolated thread is able to achieve ~19us P99.9 latency
- **Inaccurate IRQ time accounting leads to high scheduling latency**
  - ~2.7x improvement in P99.9 latency with accurate time accounting (ACCa)
  - ➡ Achieve **accurate** time accounting with our ACCa patch
- **Runtime may not be the ideal fairness unit for low latency**
  - ~1.5x improvement with request/response-based scheduling (PCSched)
  - ➡ Identify appropriate **scheduling abstraction** for different applications
  - ➡ Filter out the **hardware states** from the existing runtime abstraction



Summary

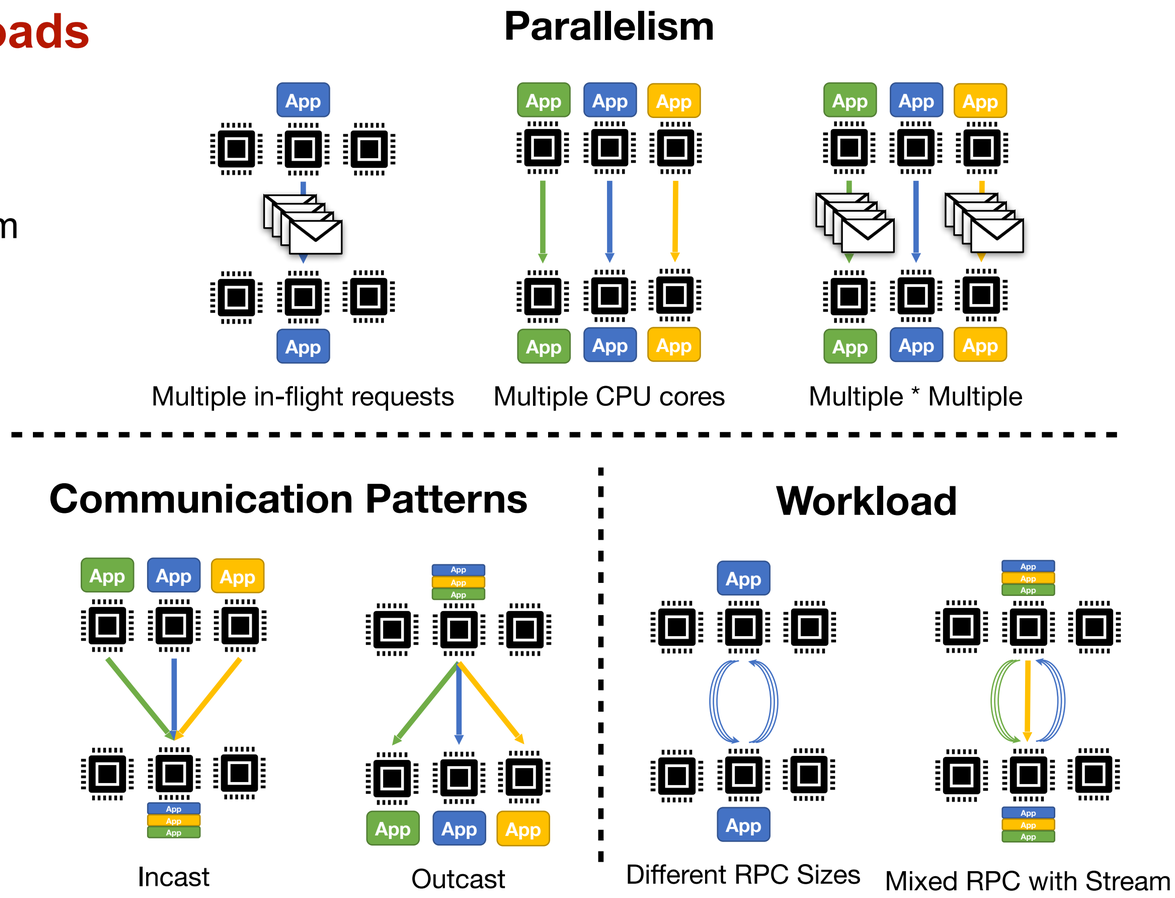
- **Packet processing itself is not the reason for tail latency**
  - An isolated thread is able to achieve ~19us P99.9 latency
- **Inaccurate IRQ time accounting leads to high scheduling latency**
  - ~2.7x improvement in P99.9 latency with accurate time accounting (ACCa)
  - ➡ Achieve **accurate** time accounting with our ACCa patch
- **Runtime may not be the ideal fairness unit for low latency**
  - ~1.5x improvement with request/response-based scheduling (PCSched)
  - ➡ Identify appropriate **scheduling abstraction** for different applications
  - ➡ Filter out the **hardware states** from the existing runtime abstraction
- **Traffic predictability can make interrupt tuning more effective**
  - ~1.3x improvement by tuning interrupt based on traffic characters
- **Choose your parallelism carefully**
  - More in-flight requests per thread beat more connections
  - ➡ Use **receiver-driven** transport protocols for better predictability and parallelism



More results

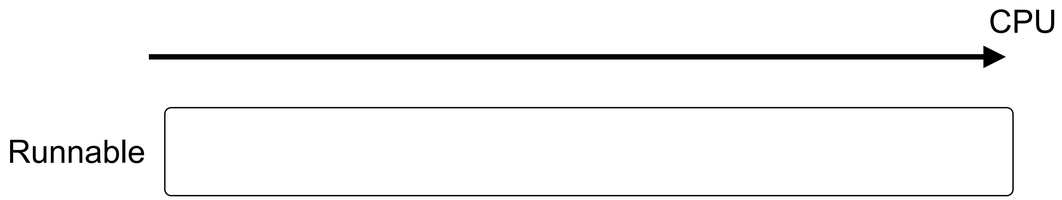
We include more results in our paper and Github:

- **Parallelism, communication patterns, and workloads**
  - Multiple in-flight requests × threads × CPU cores
  - Incast and outcast
  - Different RPC sizes (64B to 1024B) and mixed RPC with stream
- **Generality on different scheduler**
  - Recent EEVDF scheduler
- **Validation with real-world application**
  - Redis YCSB benchmark
  - Apache HTTPD (in our future technical report)
- **Github:** https://github.com/Terabit-Ethernet/understand_latency

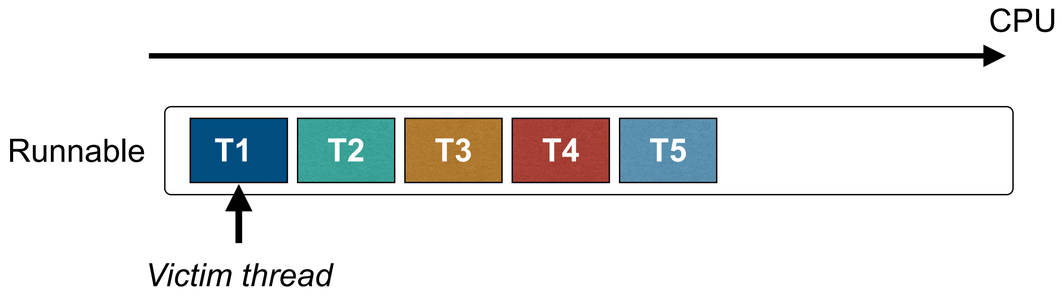


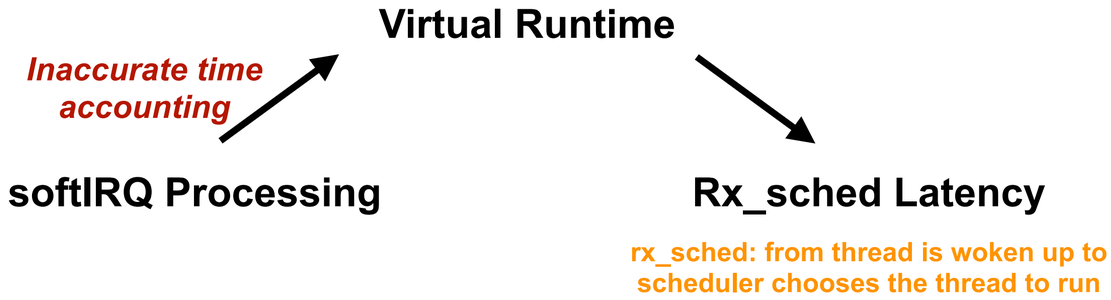
Questions?

Inaccurate Time Accounting is the Reason for High Rx_sched

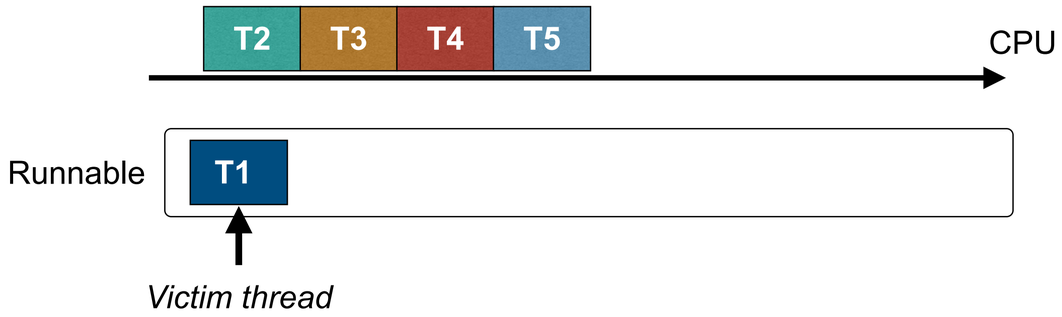


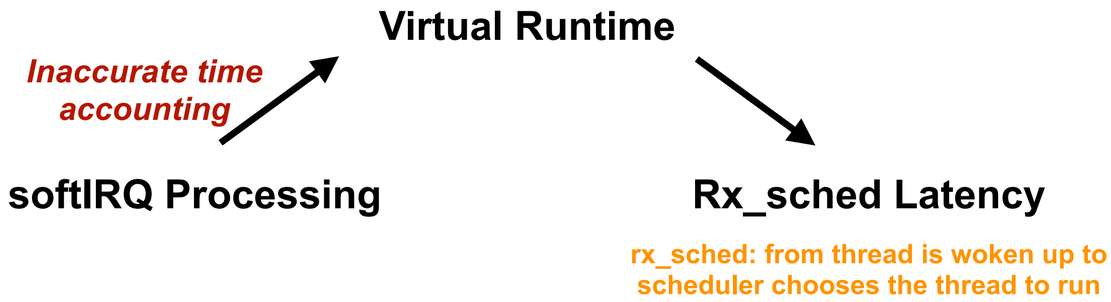
Inaccurate Time Accounting is the Reason for High Rx_sched



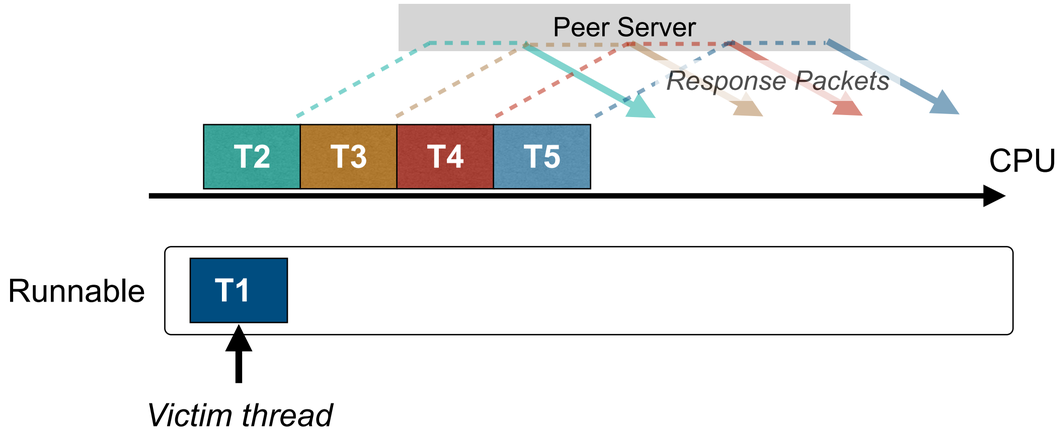


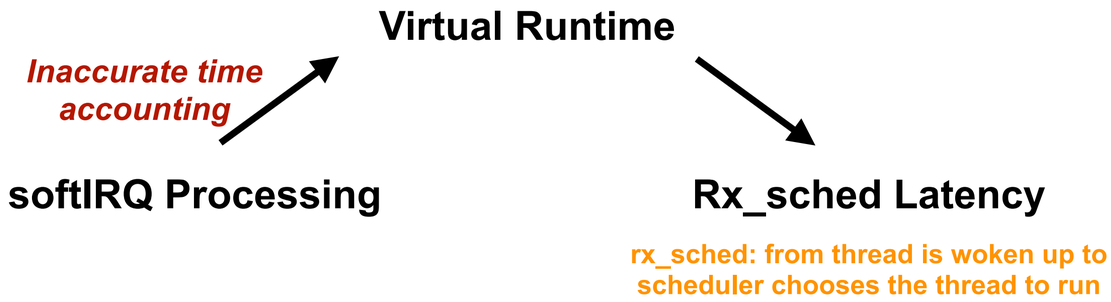
Inaccurate Time Accounting is the Reason for High Rx_sched



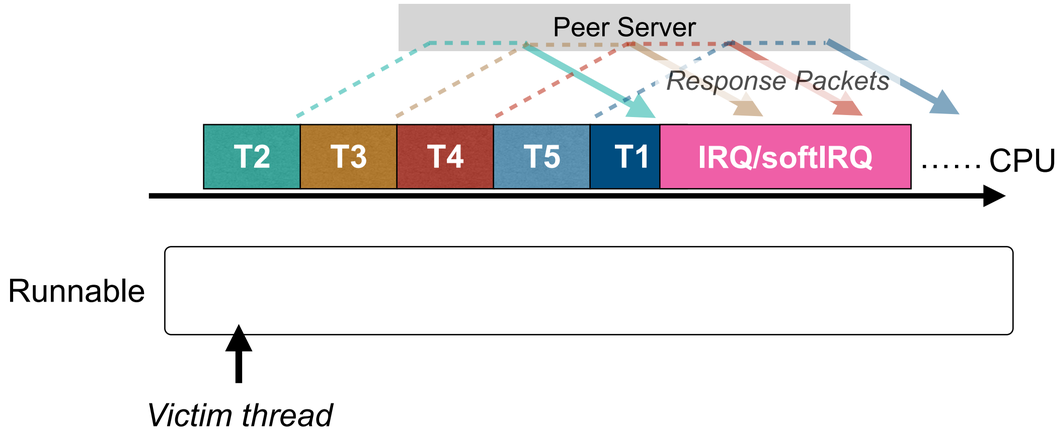


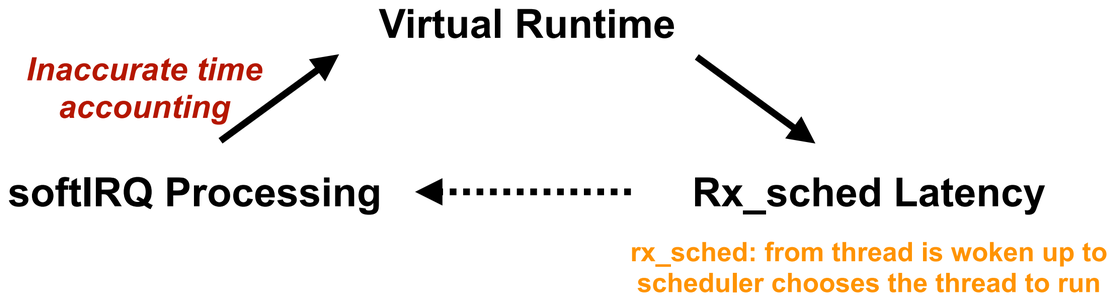
Inaccurate Time Accounting is the Reason for High Rx_sched





Inaccurate Time Accounting is the Reason for High Rx_sched





Positive Feedback Loop