Tackling Parallelization Challenges of Kernel Network Stack for Container Overlay Networks

Jiaxin Lei*, Kun Suo+, Hui Lu*, Jia Rao+

* SUNY at Binghamton
+ University of Texas at Arlington

Containers Are Widely Adopted by Industry

- OS level virtualization
- Lightweight

Containers Are Widely Adopted by Industry

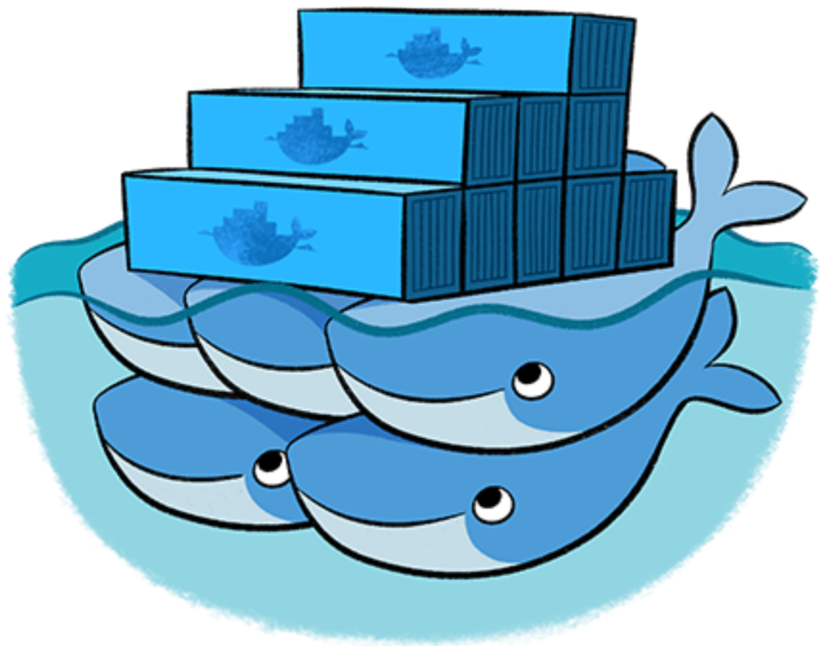
- OS level virtualization
- Lightweight
- **Higher consolidation density**
- **Lower operational cost**

Containers Are Widely Adopted by Industry

- OS level virtualization
- Lightweight
- **Higher consolidation density**
- **Lower operational cost**

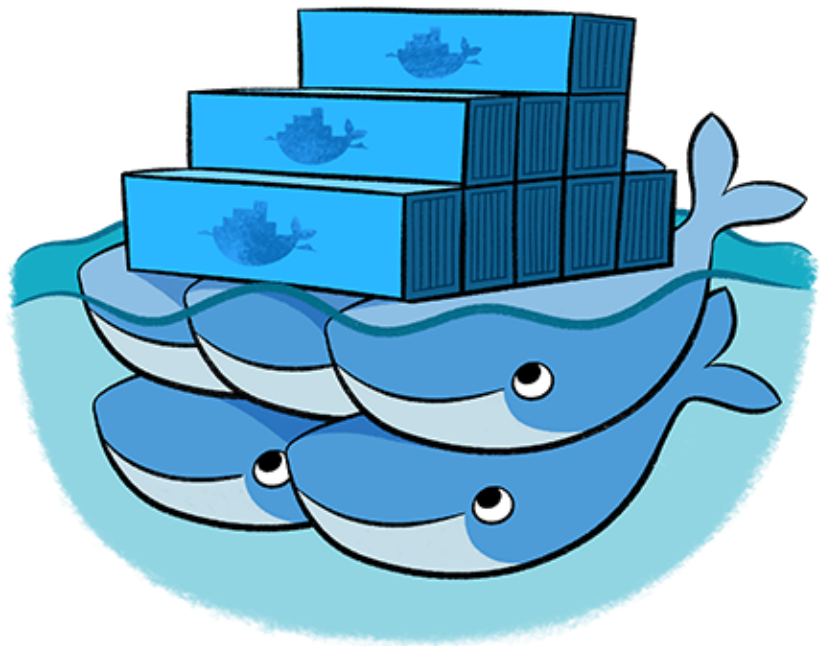
Overlay Networks Are the Technique For Containers Connectivity

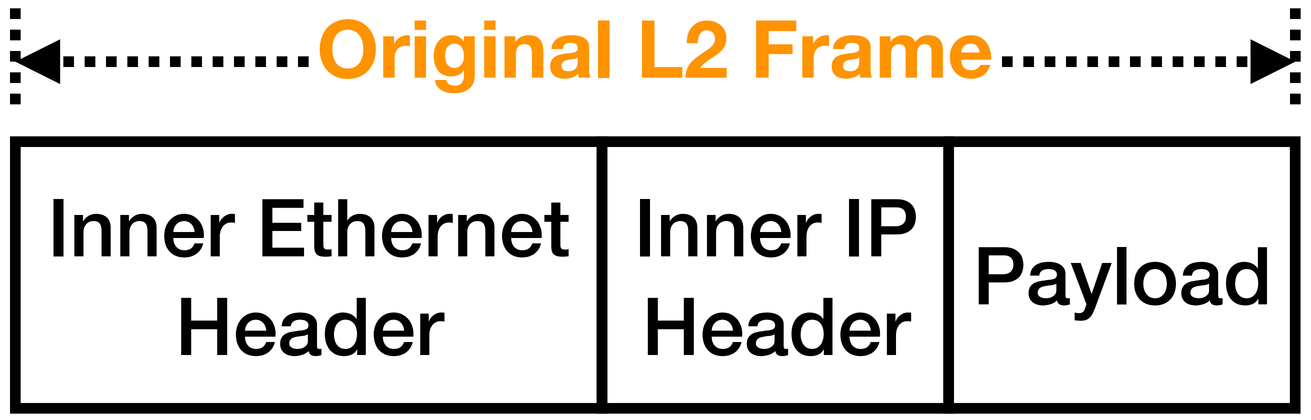
- Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave
- They are generally built upon the tunneling approach like using **VxLAN** protocol.



Overlay Networks Are the Technique For Containers Connectivity

- Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave
- They are generally built upon the tunneling approach like using **VxLAN** protocol.





Overlay Networks Are the Technique For Containers Connectivity

- Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave
- They are generally built upon the tunneling approach like using **VxLAN** protocol.

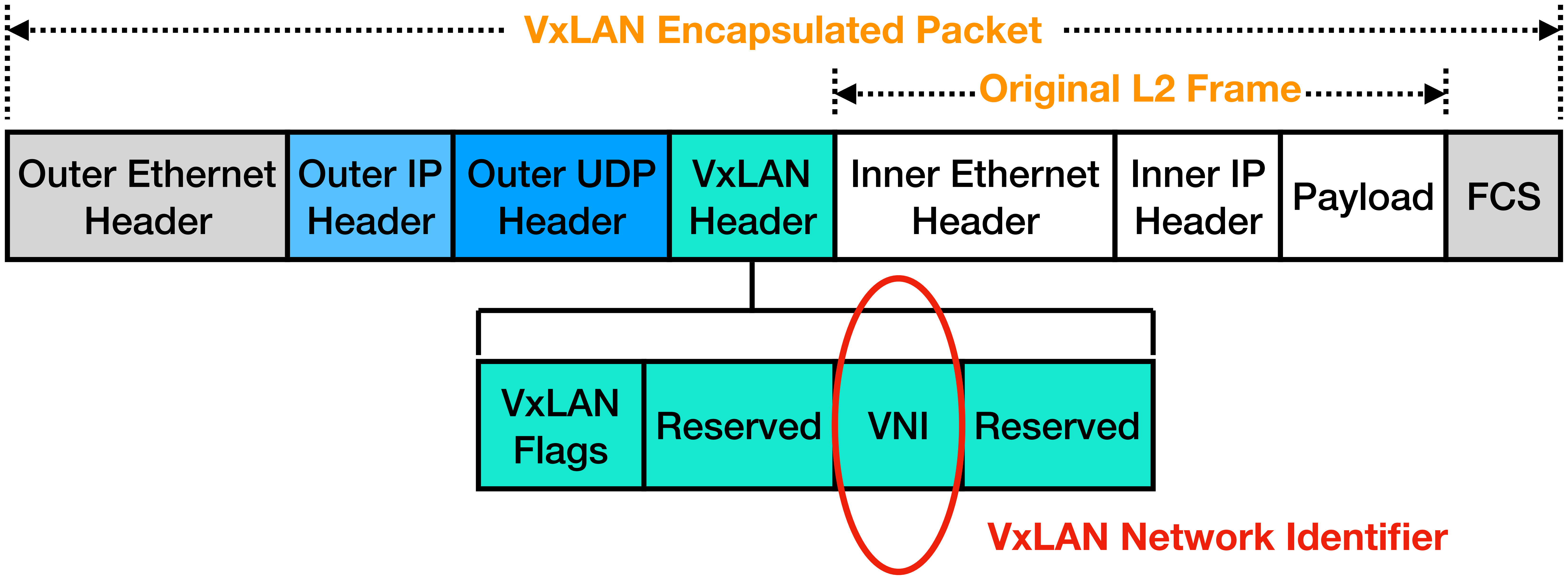
VxLAN Encapsulated Packet

Original L2 Frame

Outer Ethernet Header	Outer IP Header	Outer UDP Header	VxLAN Header	Inner Ethernet Header	Inner IP Header	Payload	FCS

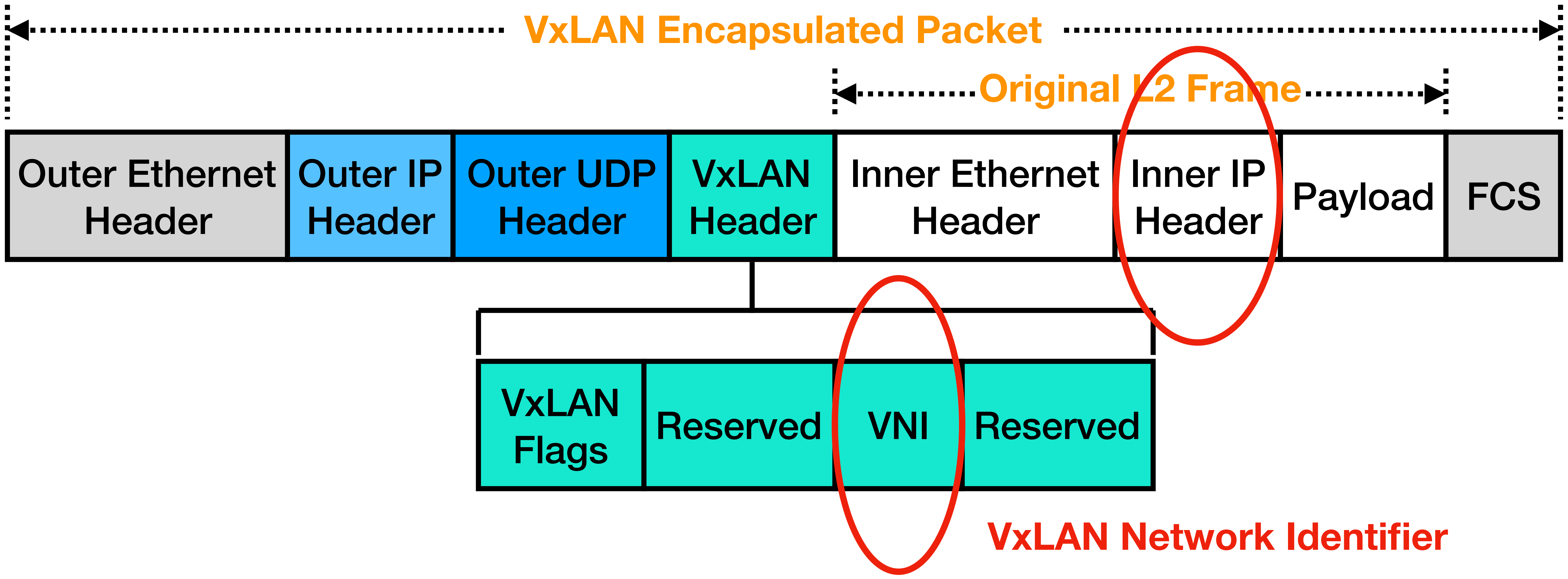
Overlay Networks Are the Technique For Containers Connectivity

- Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave
- They are generally built upon the tunneling approach like using **VxLAN** protocol.



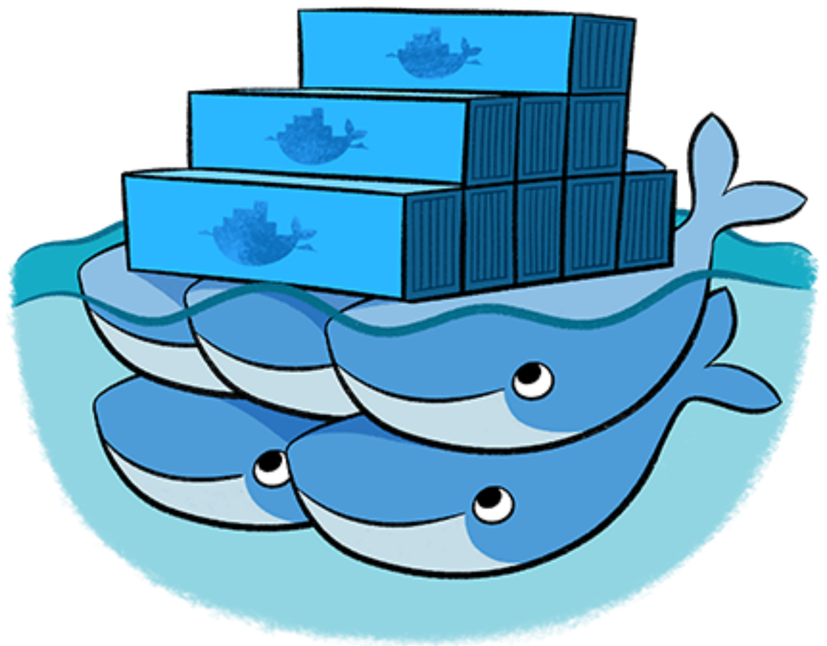
Overlay Networks Are the Technique For Containers Connectivity

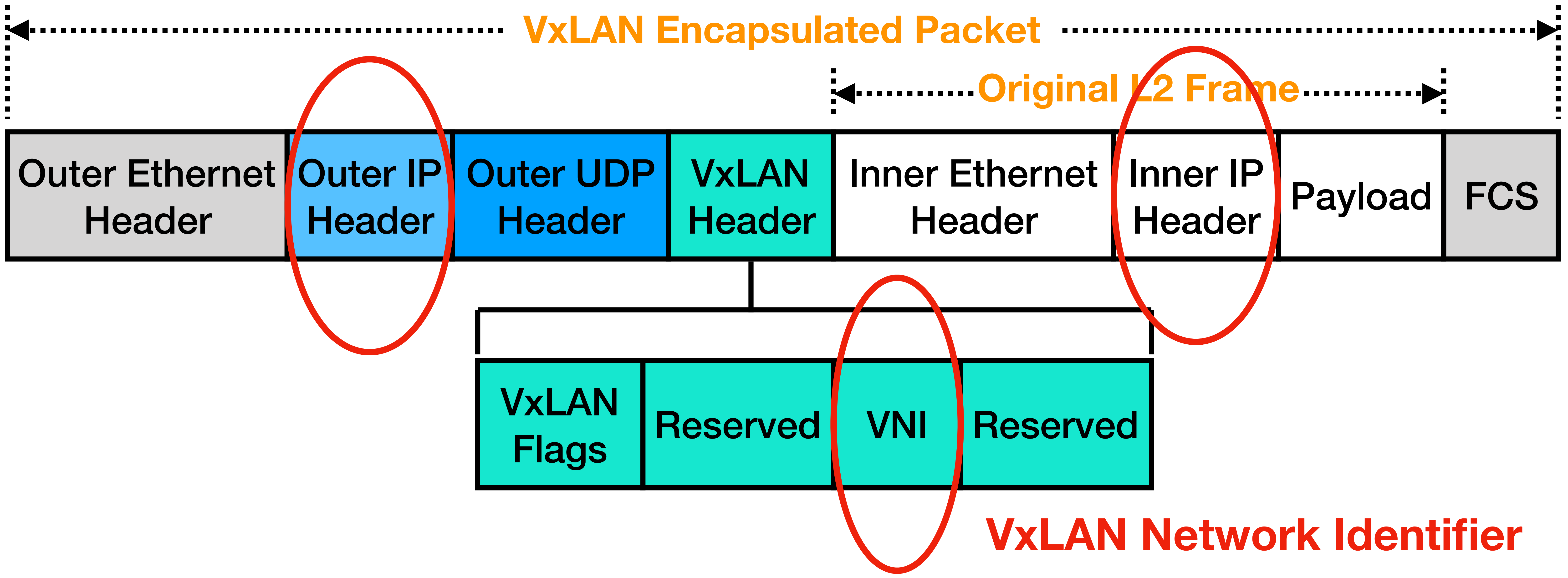
- Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave
- They are generally built upon the tunneling approach like using **VxLAN** protocol.



Overlay Networks Are the Technique For Containers Connectivity

- Typical overlay network solutions: Docker Overlay, Flannel, Calico, Weave
- They are generally built upon the tunneling approach like using **VxLAN** protocol.





Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Network Packet Processing Path

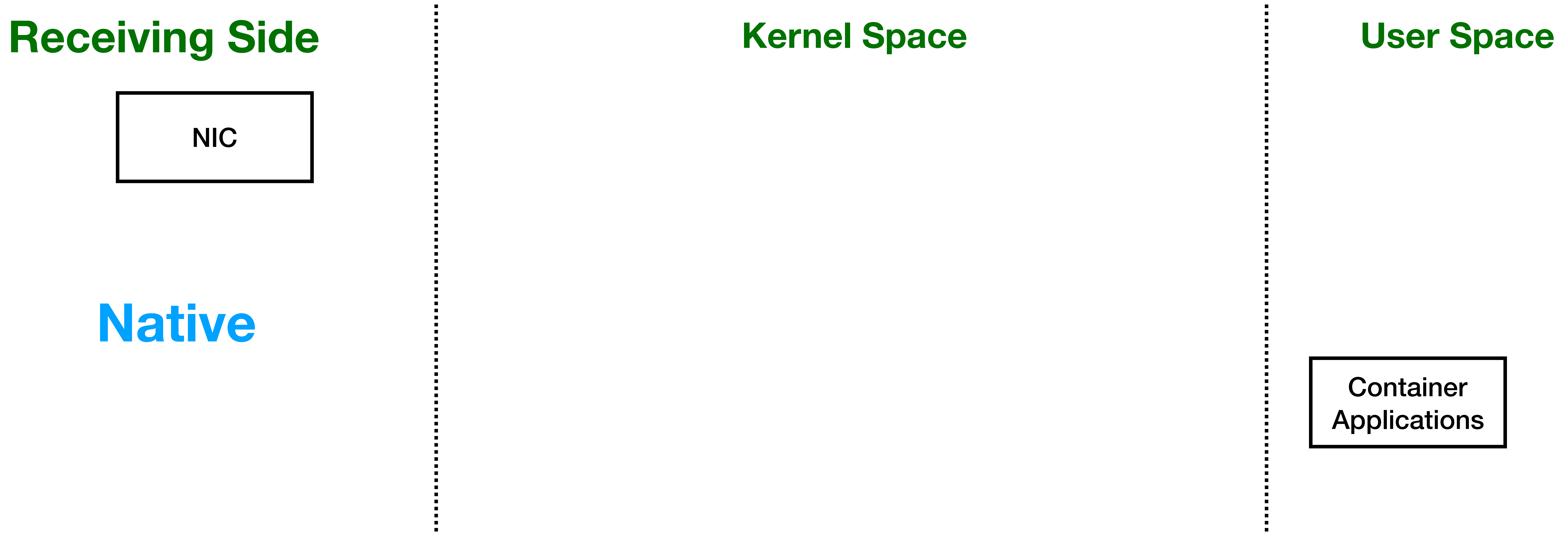
- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

Receiving Side

Native

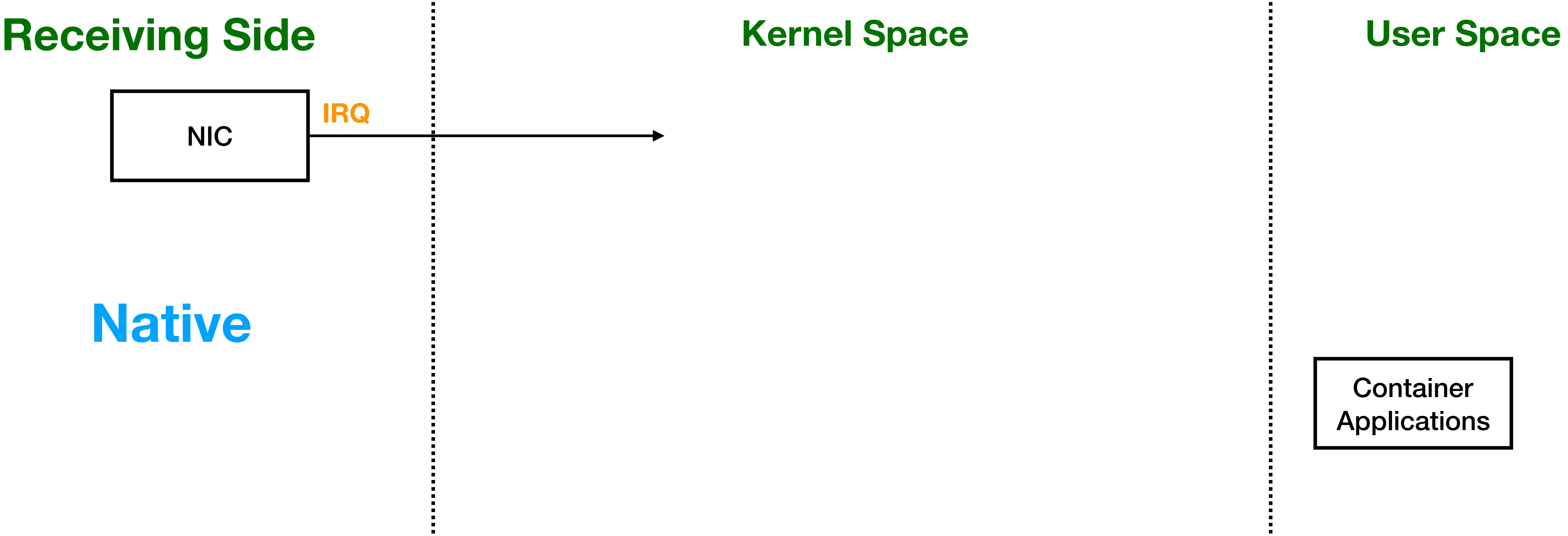
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



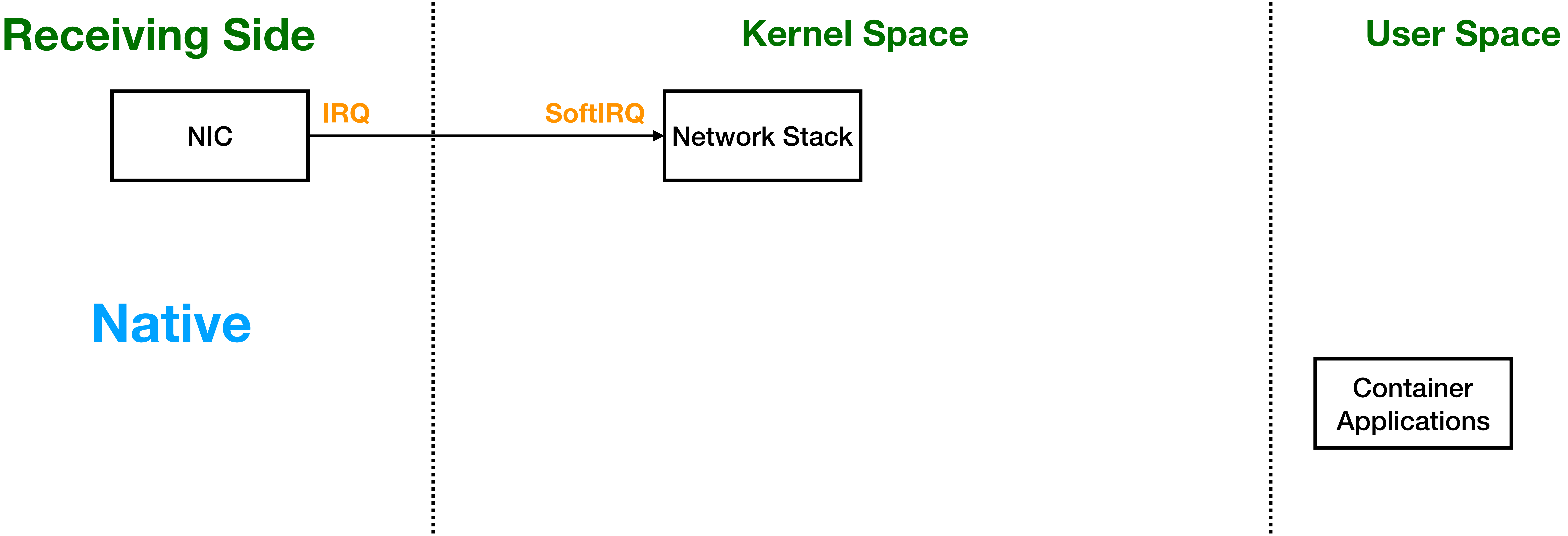
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



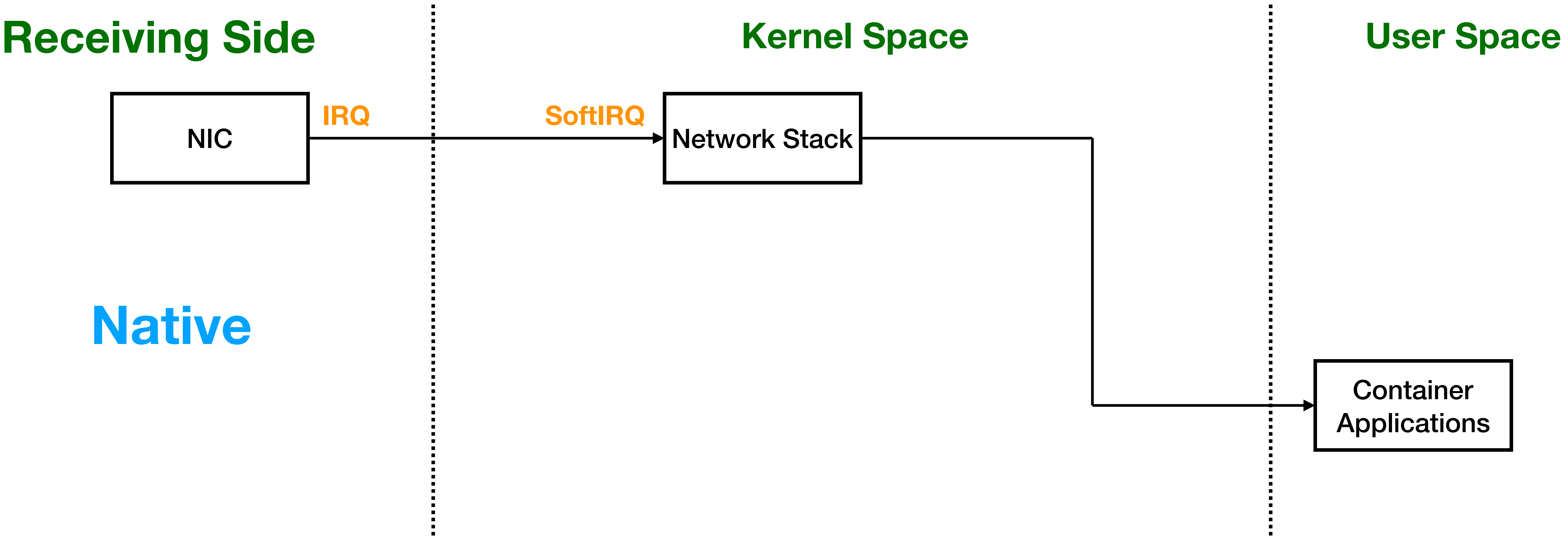
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



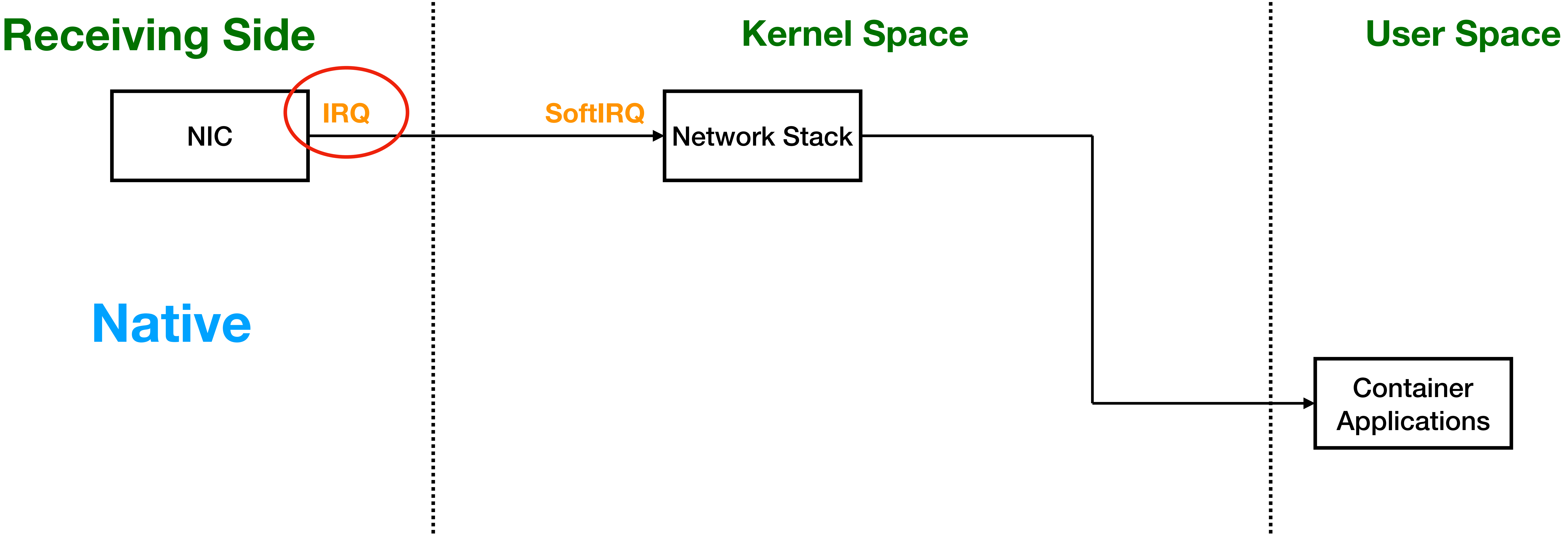
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



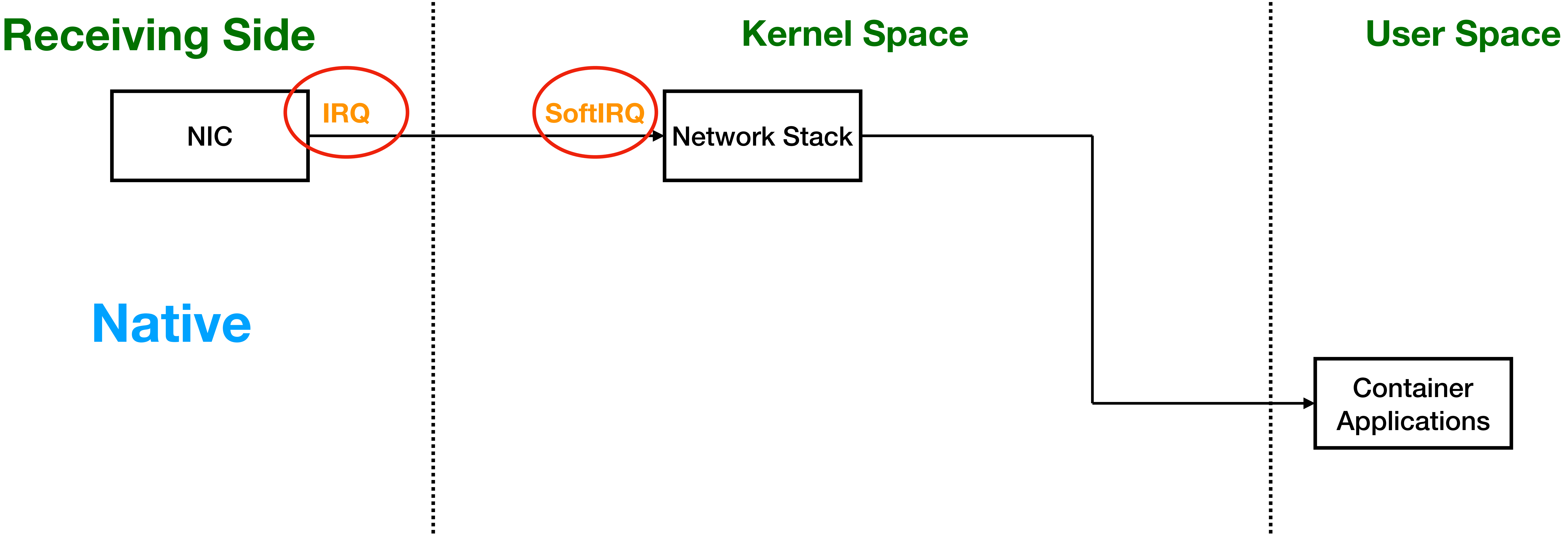
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



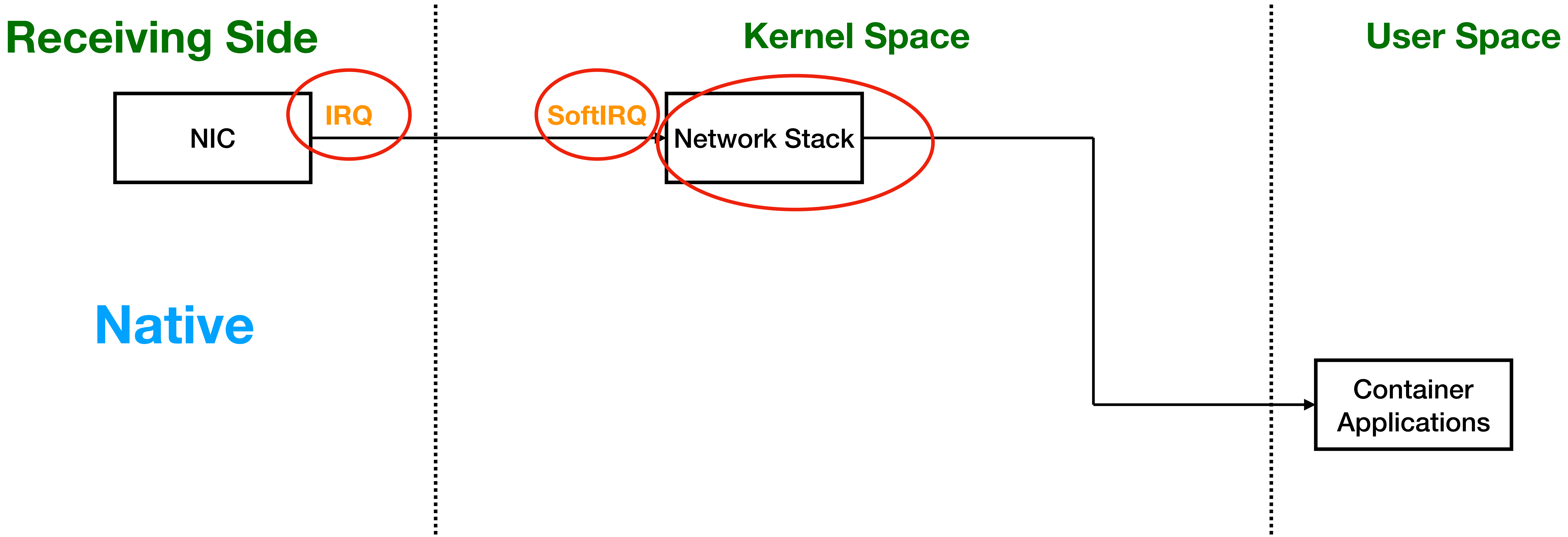
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



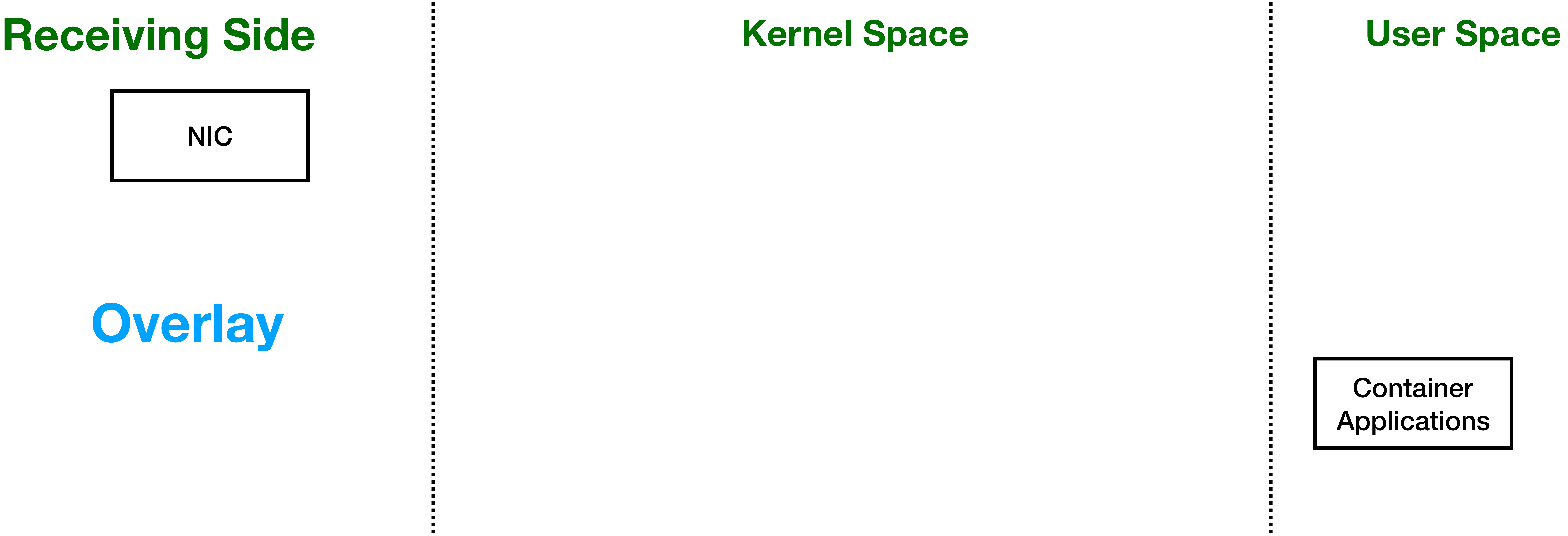
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



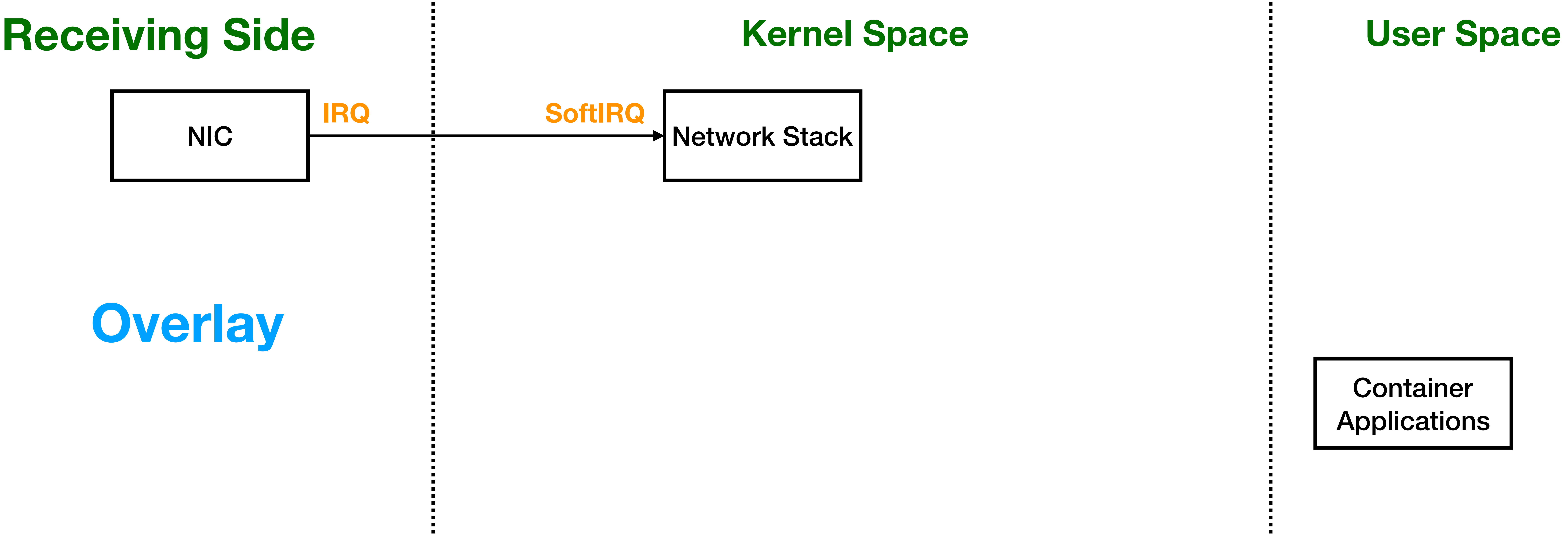
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



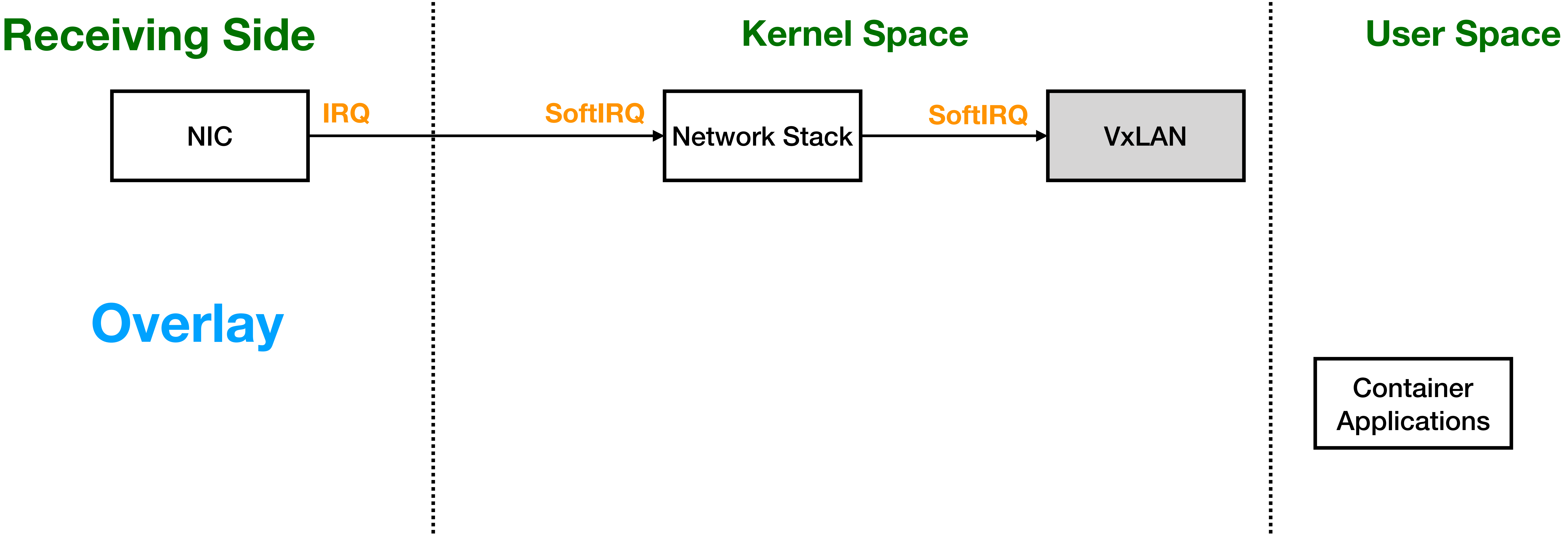
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



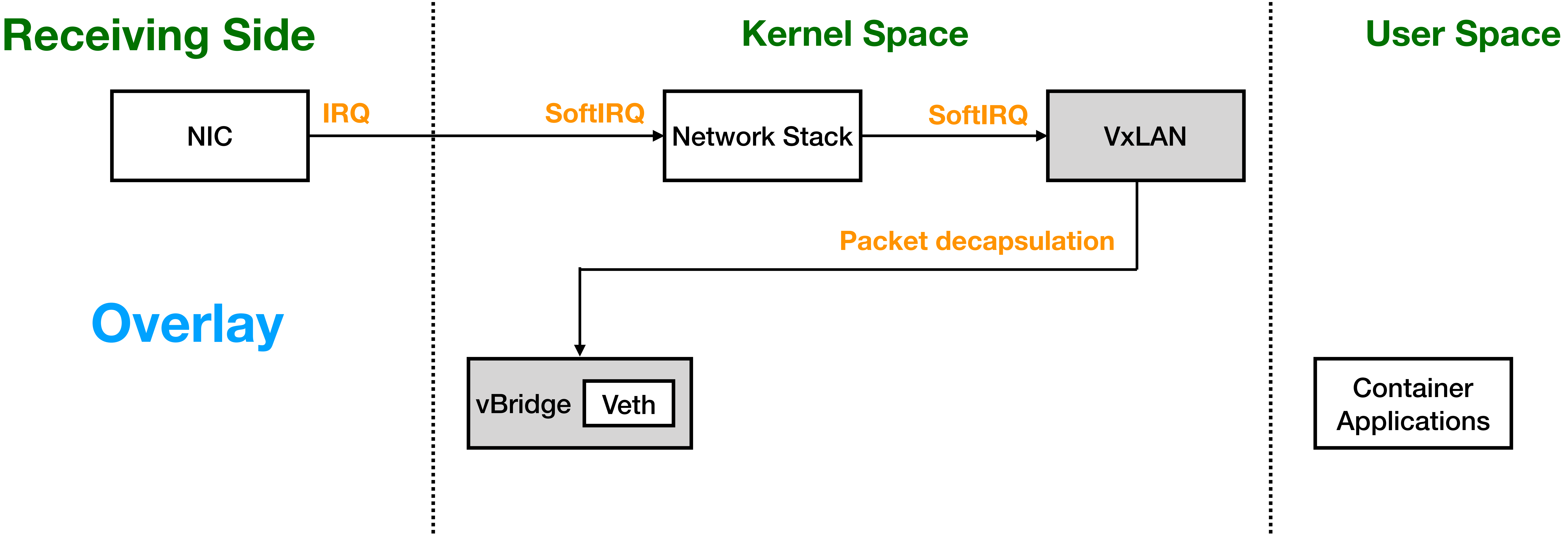
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



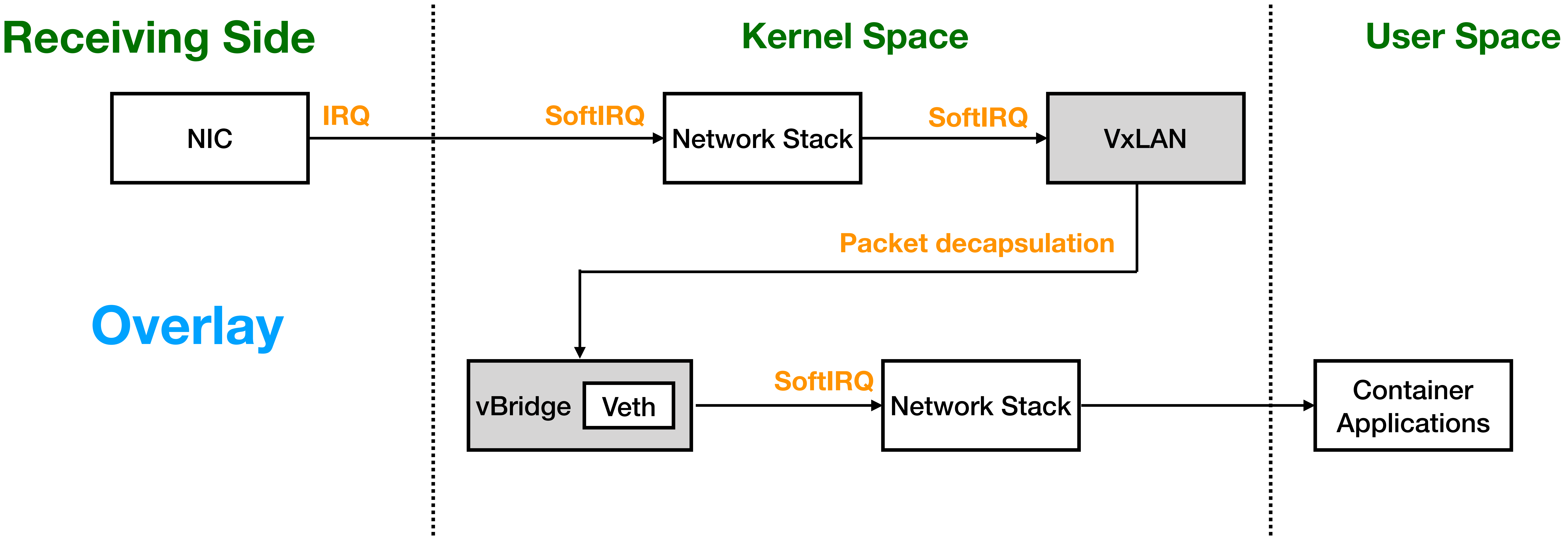
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



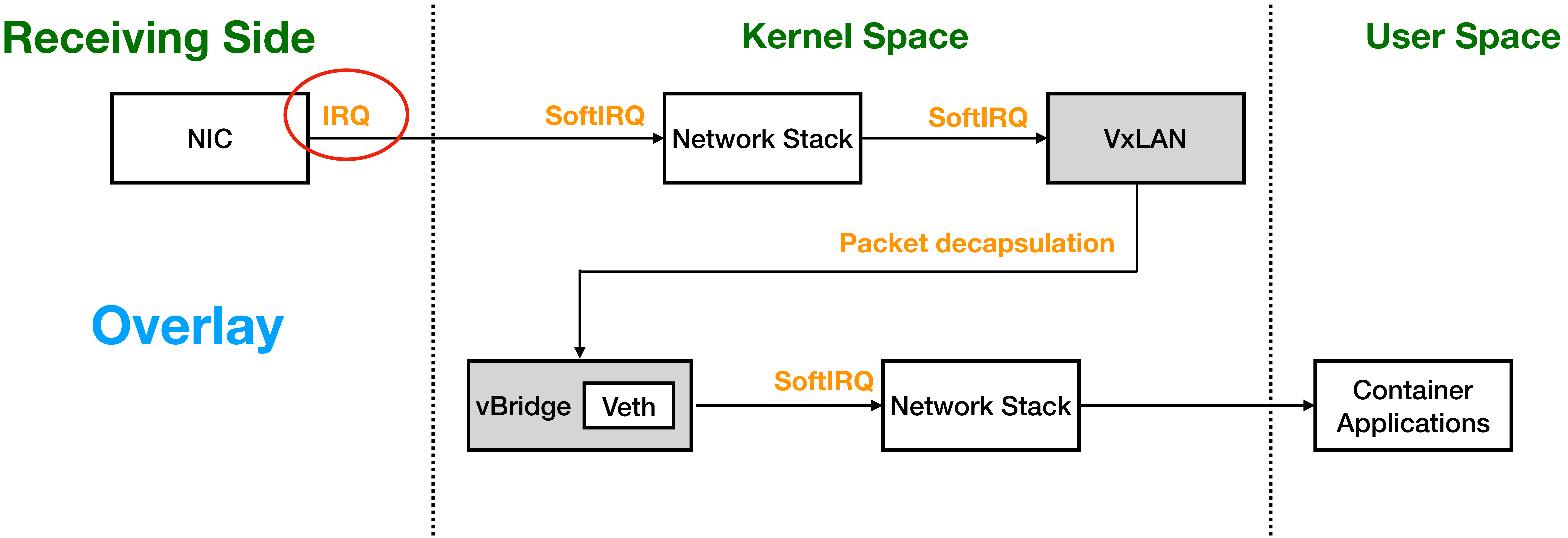
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



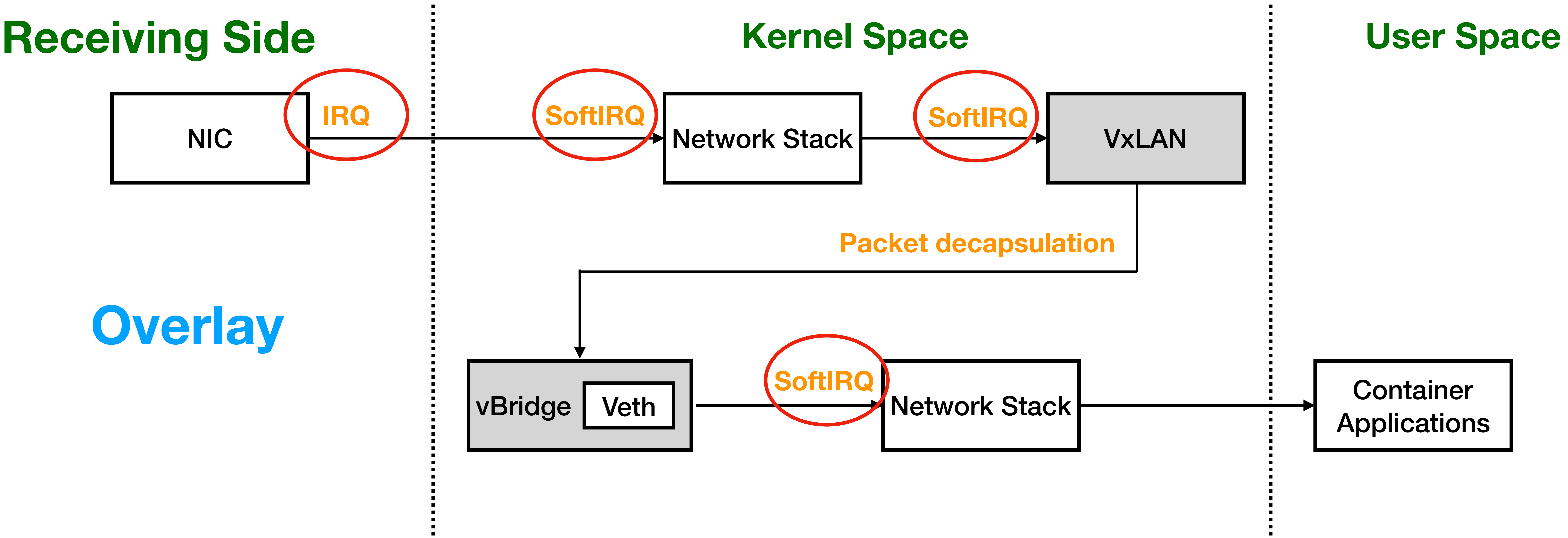
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



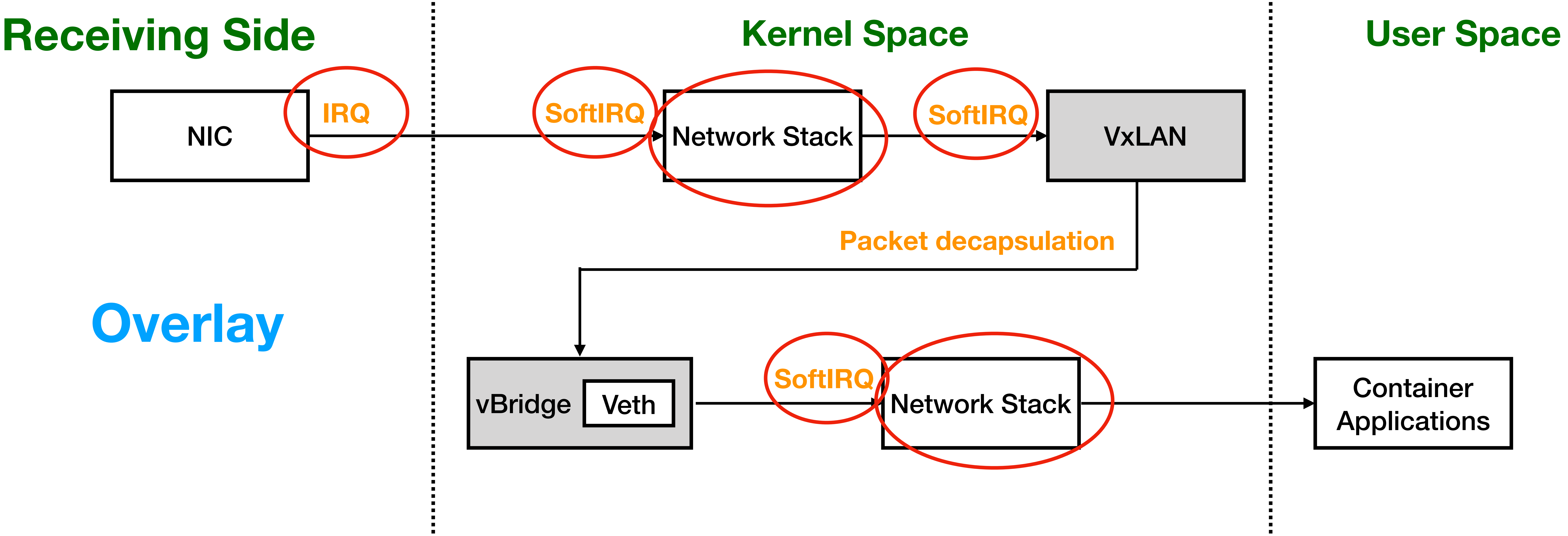
Network Packet Processing Path

- **Prolonged** network packet processing path
- **Additional** virtual devices overhead

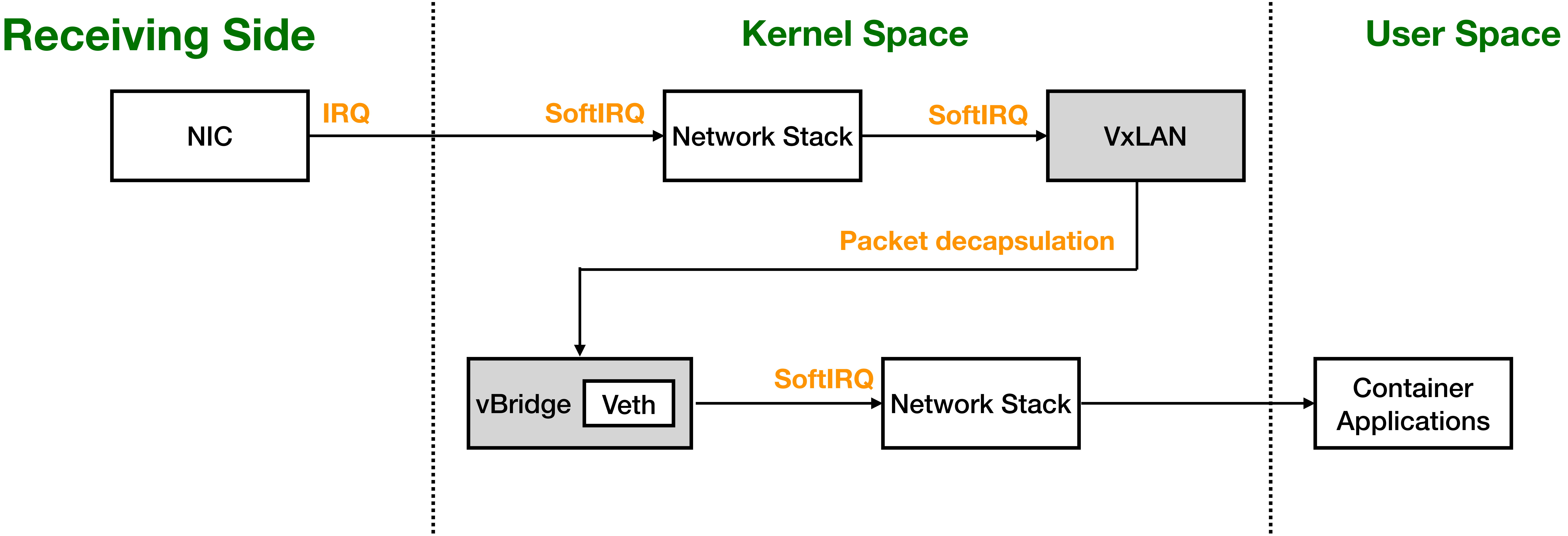


Network Packet Processing Path

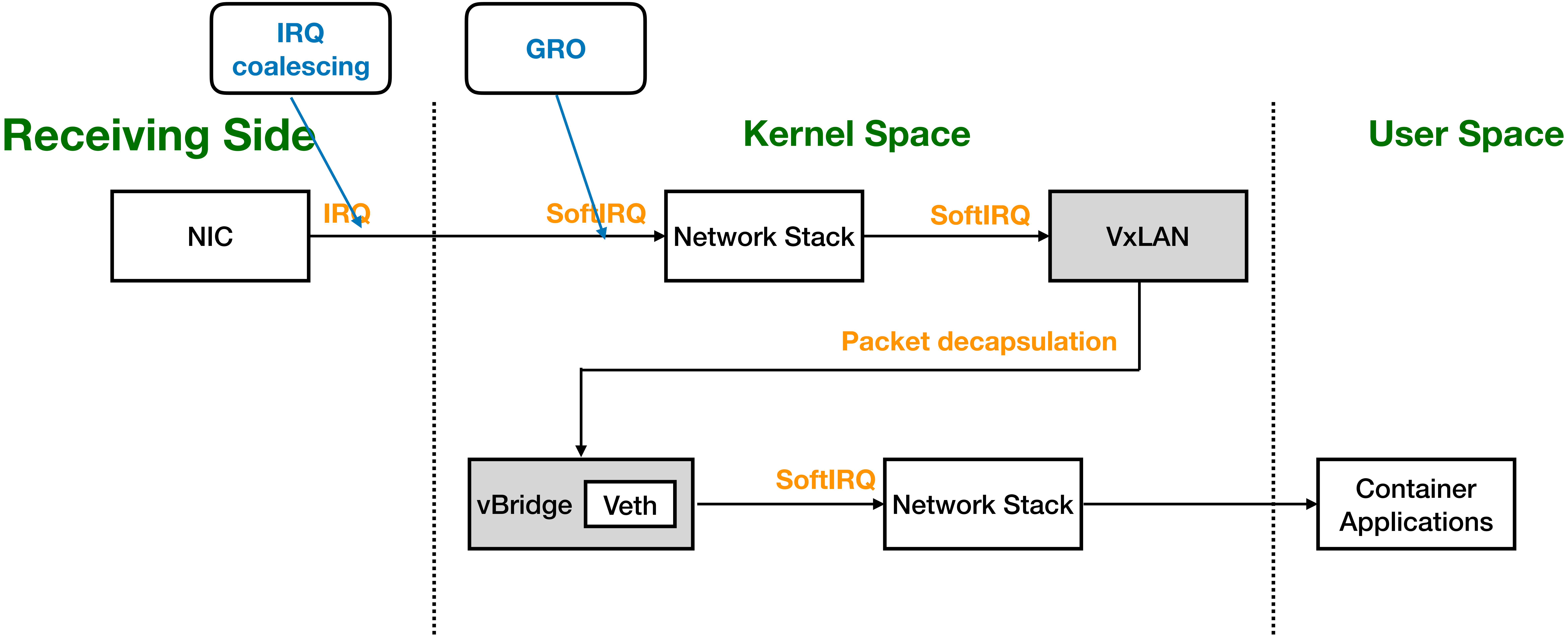
- **Prolonged** network packet processing path
- **Additional** virtual devices overhead



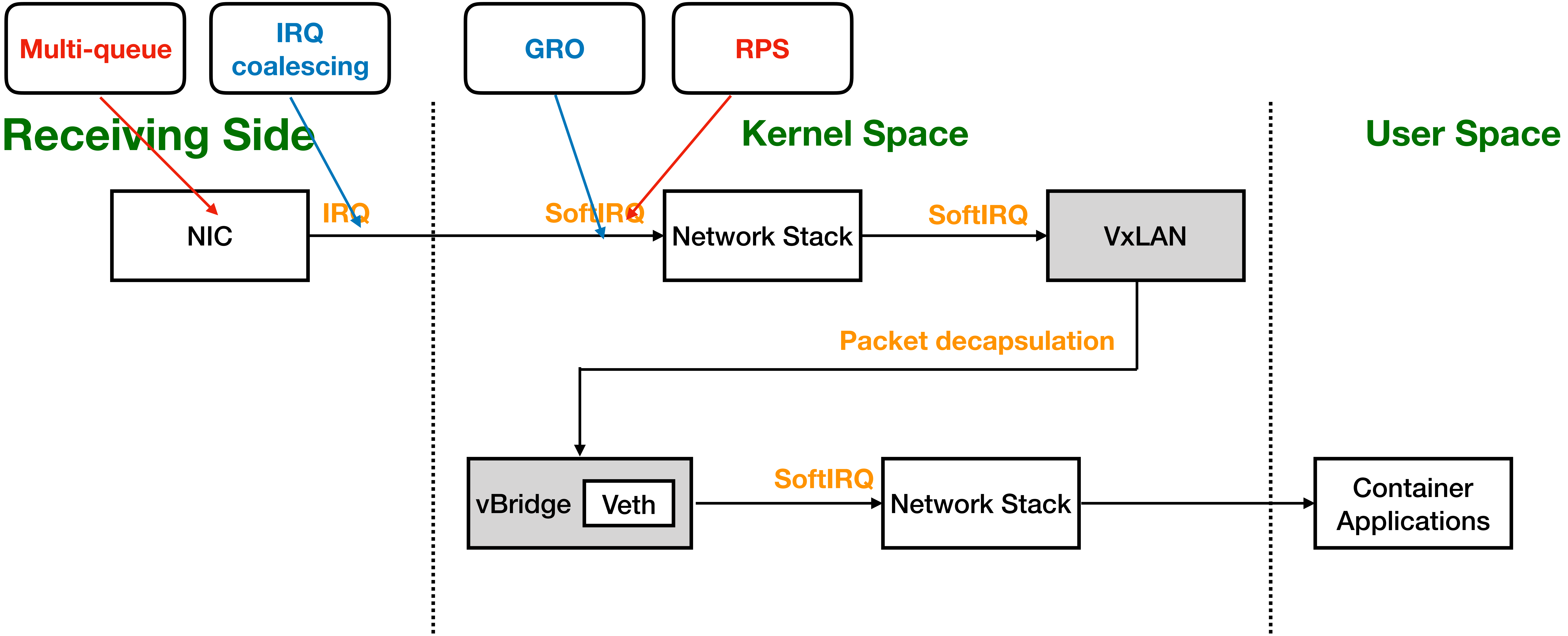
Existing Optimizations for Packet Processing



Existing Optimizations for Packet Processing



Existing Optimizations for Packet Processing



Experimental Settings

Hardware			Software	
CPU	**Memory**	**NIC**	**Throughput**	**CPU Usage**
2.2 GHz	64GB	**40Gbps**	iPerf3	mpstat
10 cores		Multi-queue		

Experimental Settings

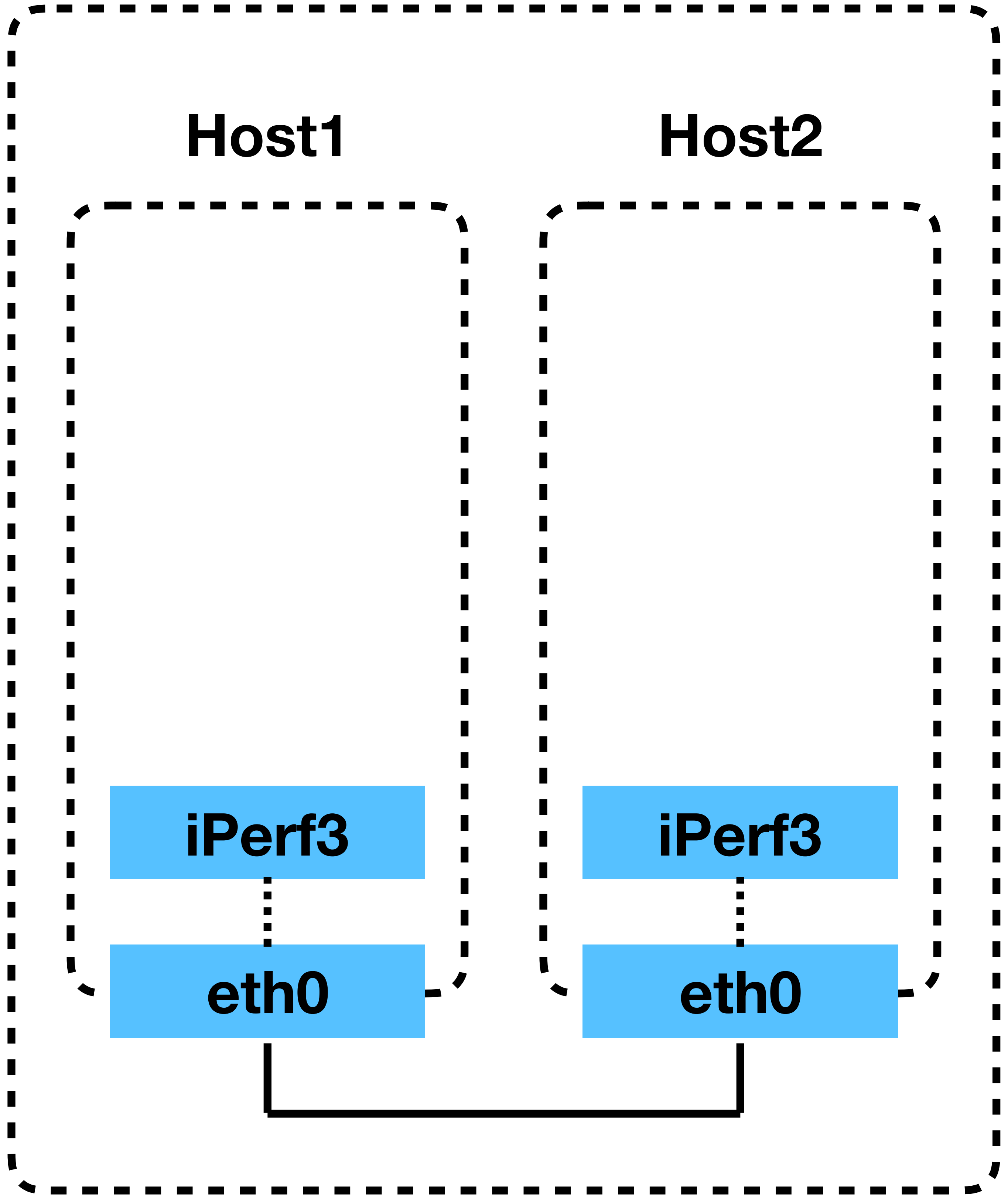
Hardware			Software	
CPU	**Memory**	**NIC**	**Throughput**	**CPU Usage**
2.2 GHz	64GB	**40Gbps**	iPerf3	mpstat
10 cores		Multi-queue		

S1 - Native **S2 - Linux Overlay** **S3 - Docker Overlay**

Experimental Settings

Hardware			Software	
CPU	**Memory**	**NIC**	**Throughput**	**CPU Usage**
2.2 GHz	64GB	**40Gbps**	iPerf3	mpstat
10 cores		Multi-queue		

S1 - Native



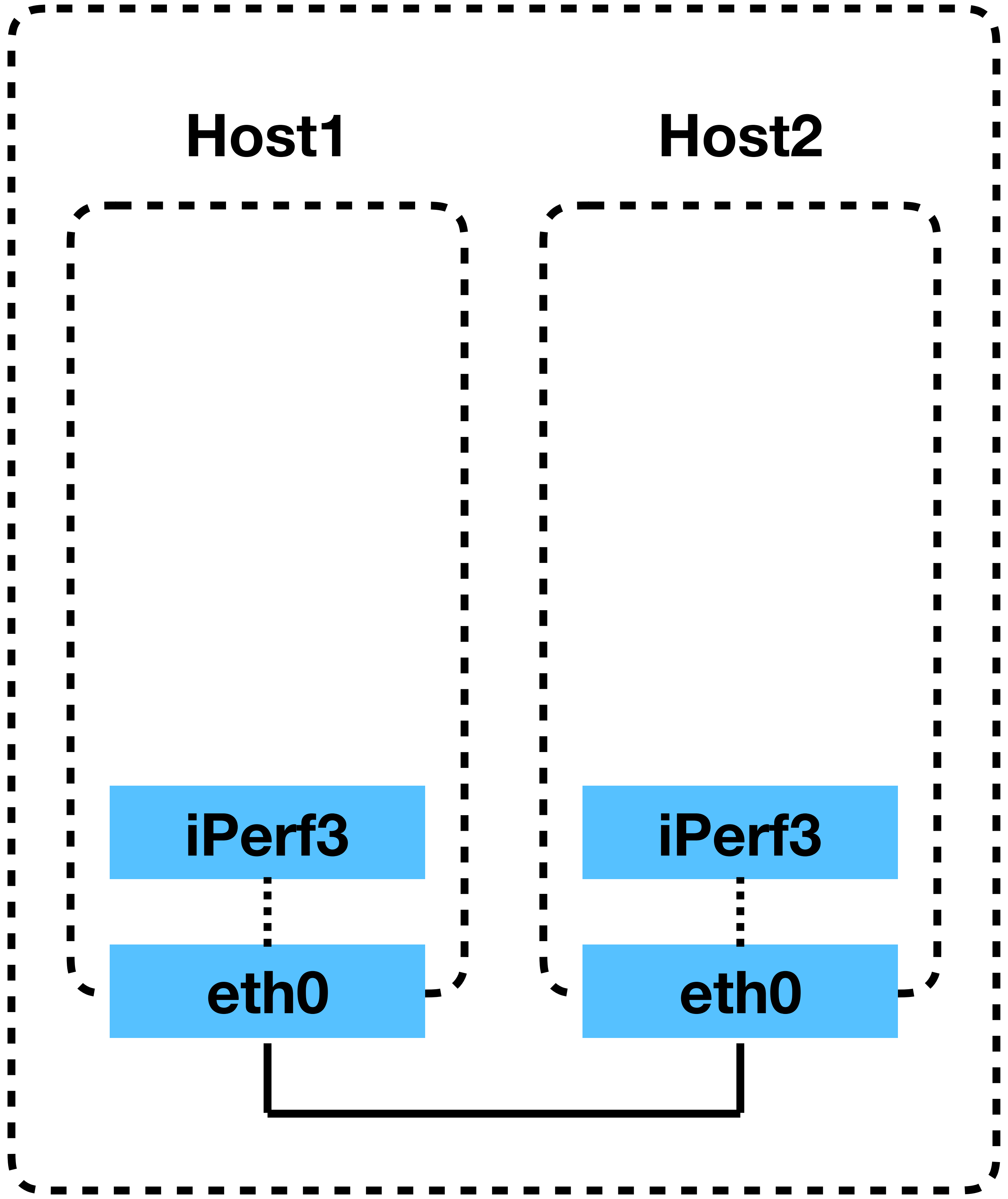
S2 - Linux Overlay

S3 - Docker Overlay

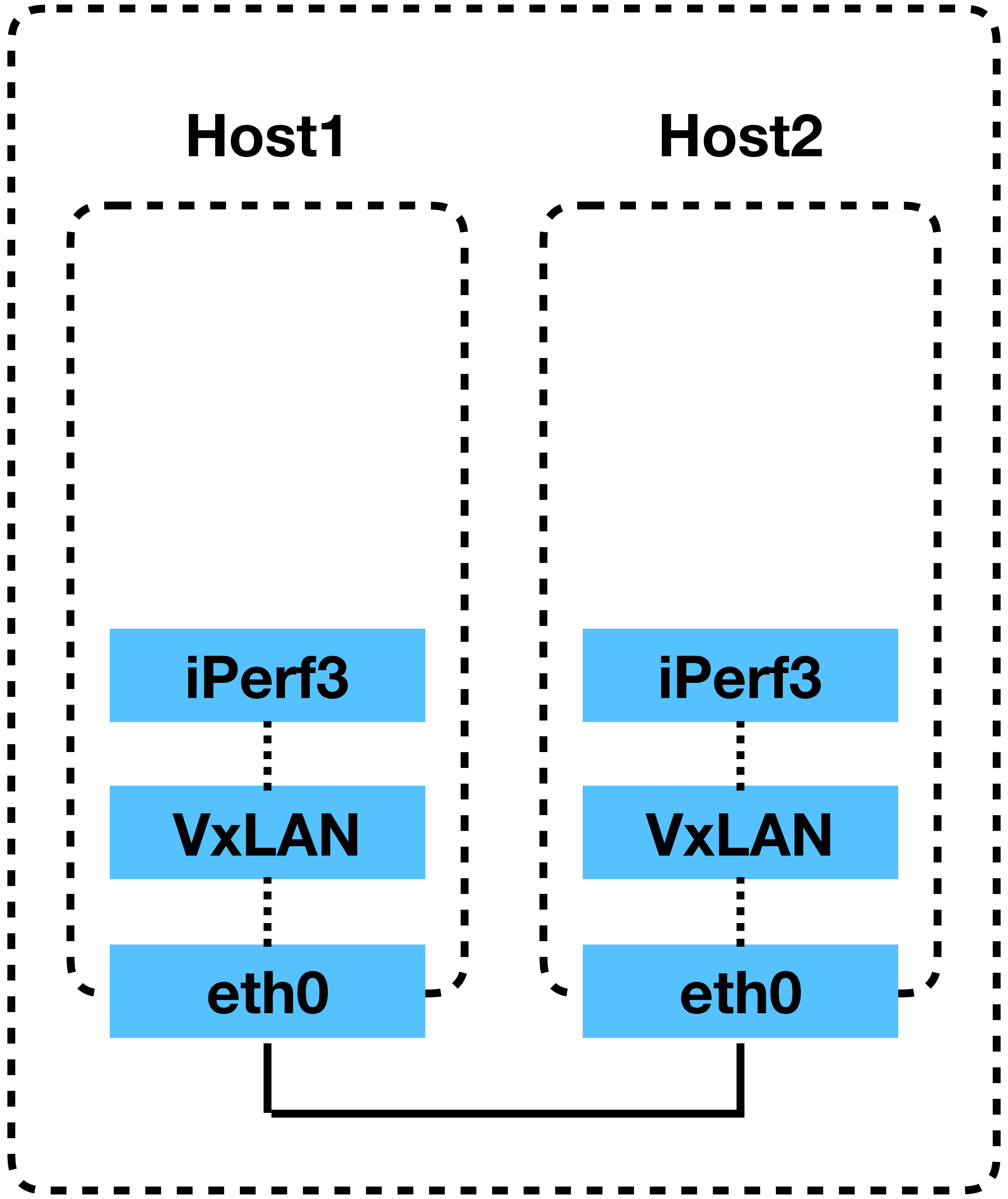
Experimental Settings

Hardware			Software	
CPU	Memory	NIC	Throughput	CPU Usage
2.2 GHz	64GB	**40Gbps**	iPerf3	mpstat
10 cores		Multi-queue		

S1 - Native



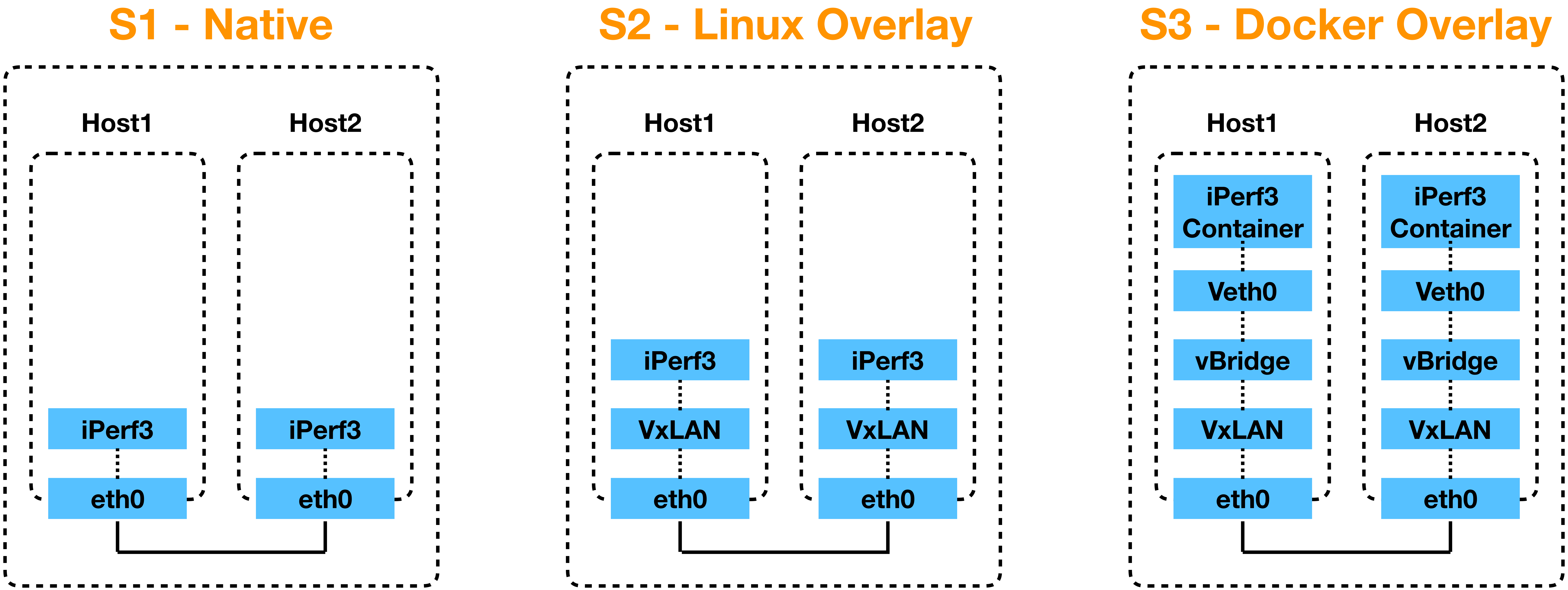
S2 - Linux Overlay



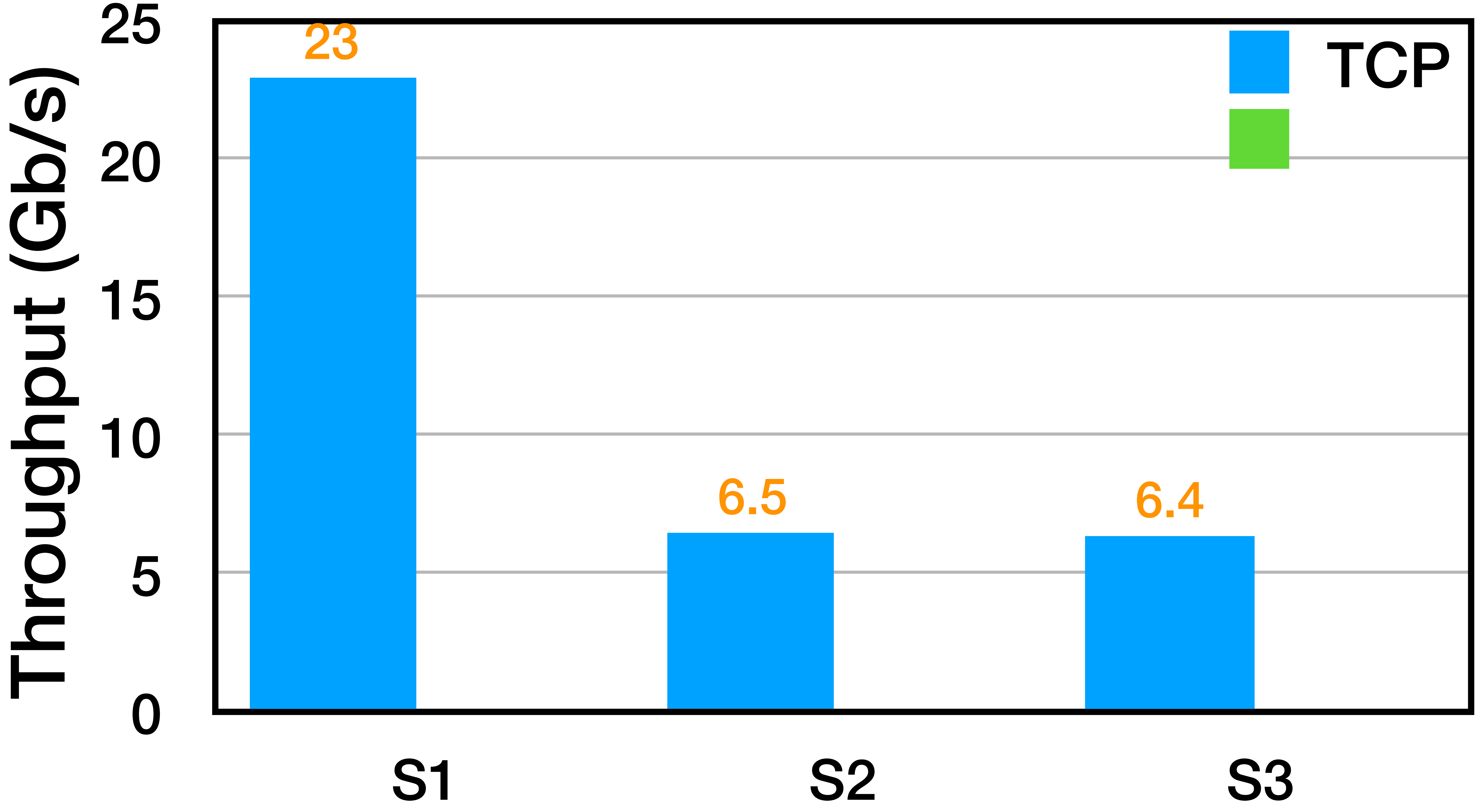
S3 - Docker Overlay

Experimental Settings

Hardware			Software	
CPU	**Memory**	**NIC**	**Throughput**	**CPU Usage**
2.2 GHz	64GB	**40Gbps**	iPerf3	mpstat
10 cores		Multi-queue		

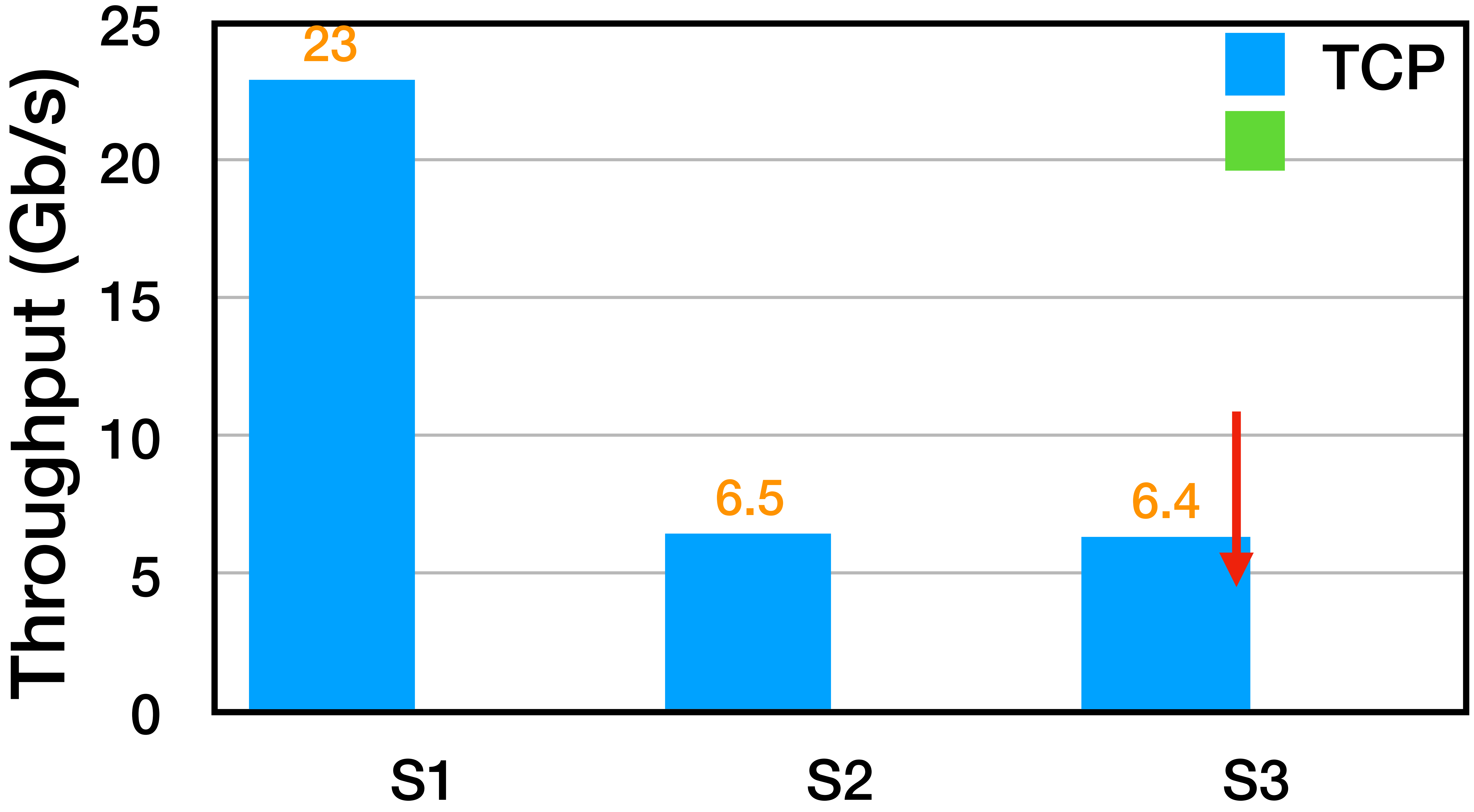


Single Flow Performance



TCP and UDP Throughputs under Three Different Cases

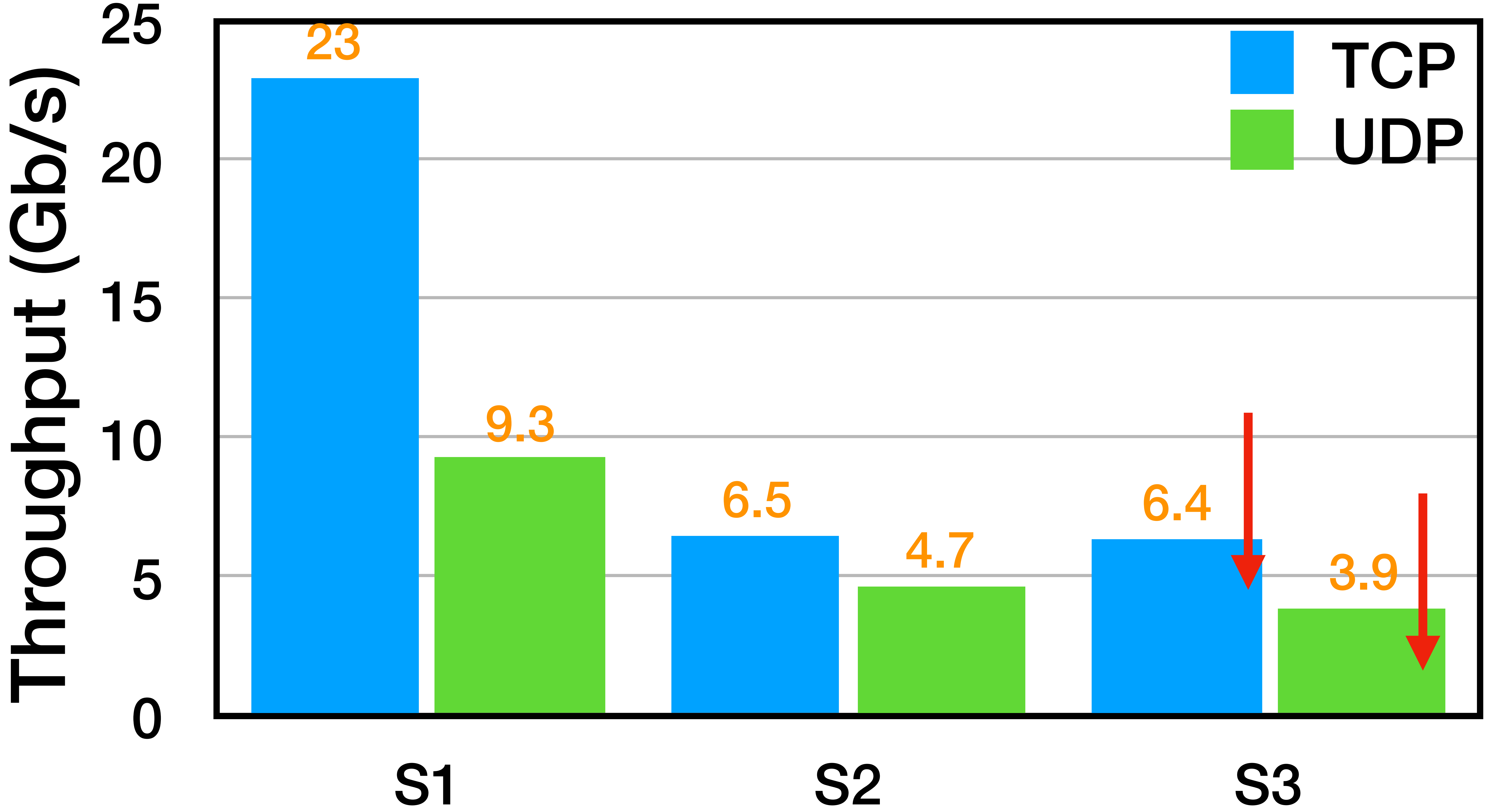
Single Flow Performance



TCP and UDP Throughputs under Three Different Cases

- **TCP Throughput** of Docker Overlay case drops **72%** compared with native case.

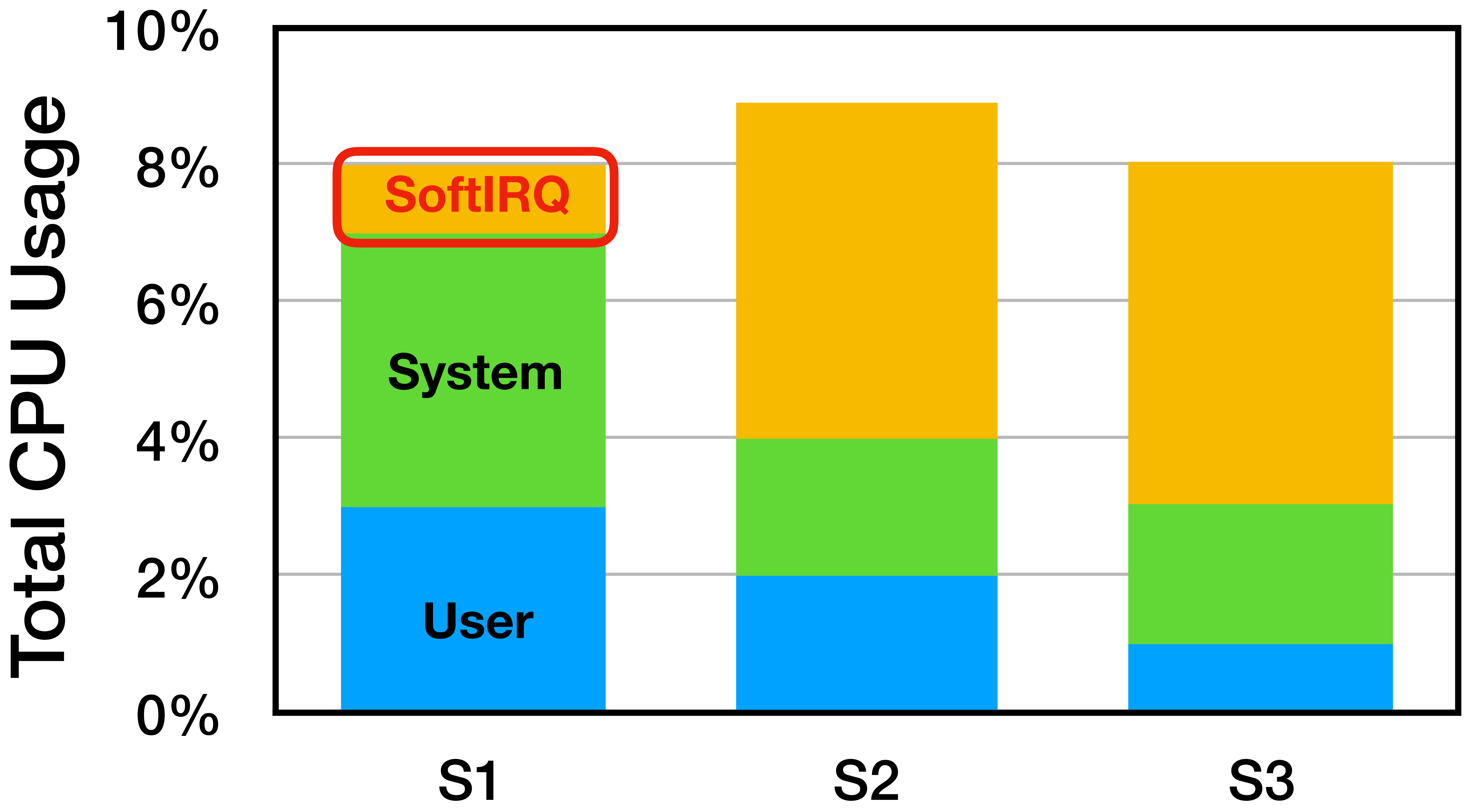
Single Flow Performance



TCP and UDP Throughputs under Three Different Cases

- **TCP Throughput** of Docker Overlay case drops **72%** compared with native case.
- **UDP Throughput** drops **58%**.

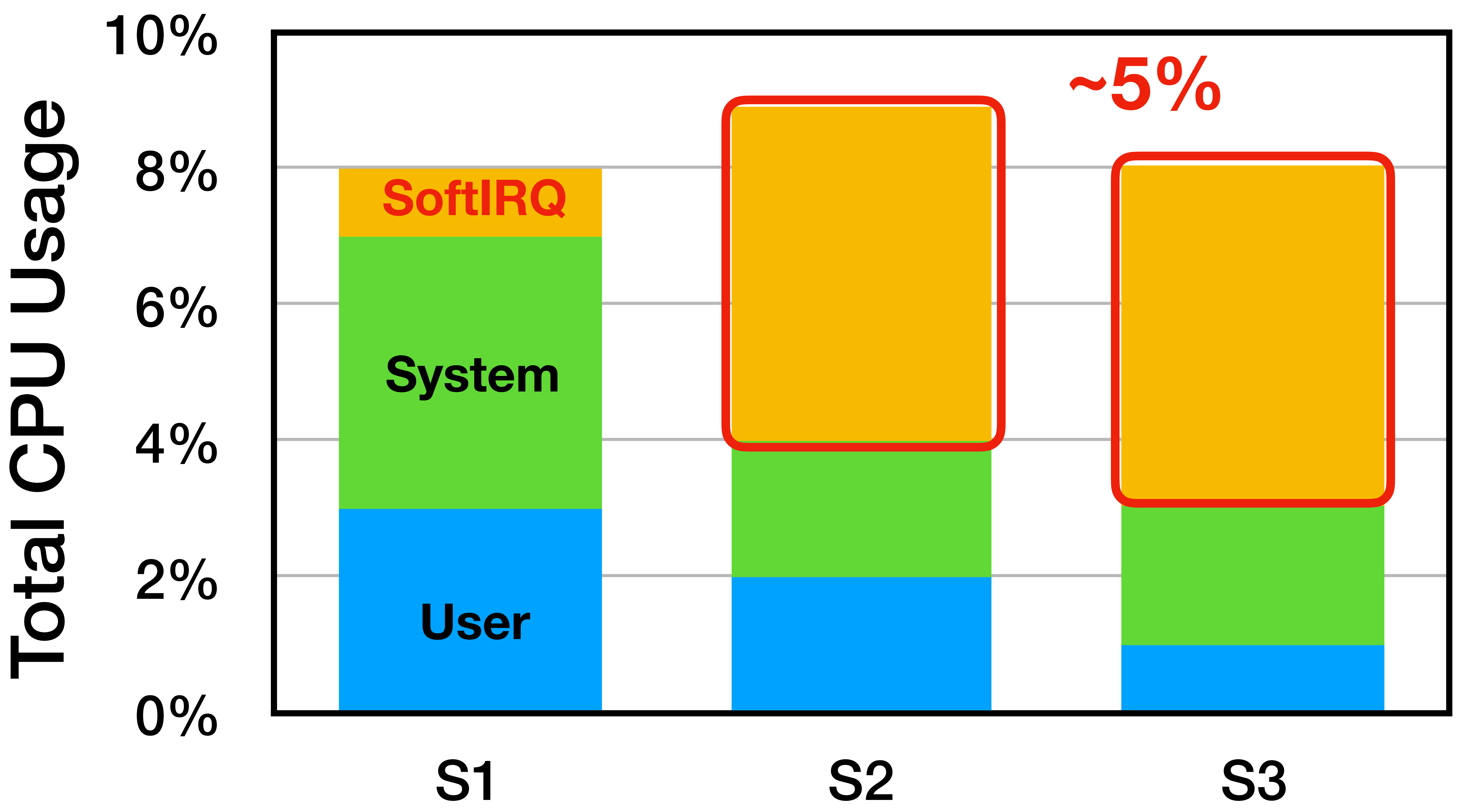
Single Flow Performance



CPU Usage under Three Different Cases for TCP

* 5% indicates one cpu core is fully saturated.

Single Flow Performance

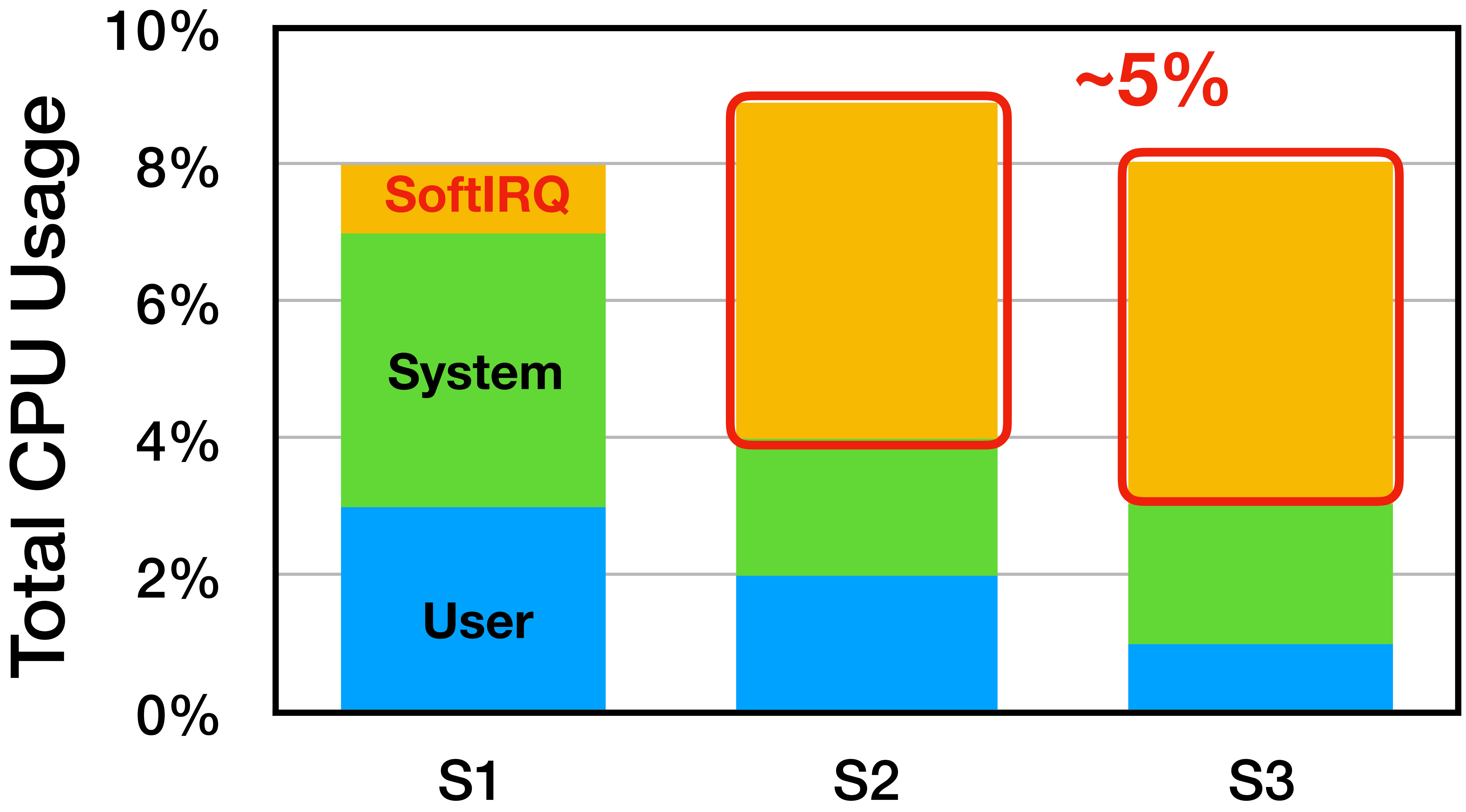


CPU Usage under Three Different Cases for TCP

- Packet processing overhead **fully saturates** one cpu core in two overlay cases.

* 5% indicates one cpu core is fully saturated.

Single Flow Performance

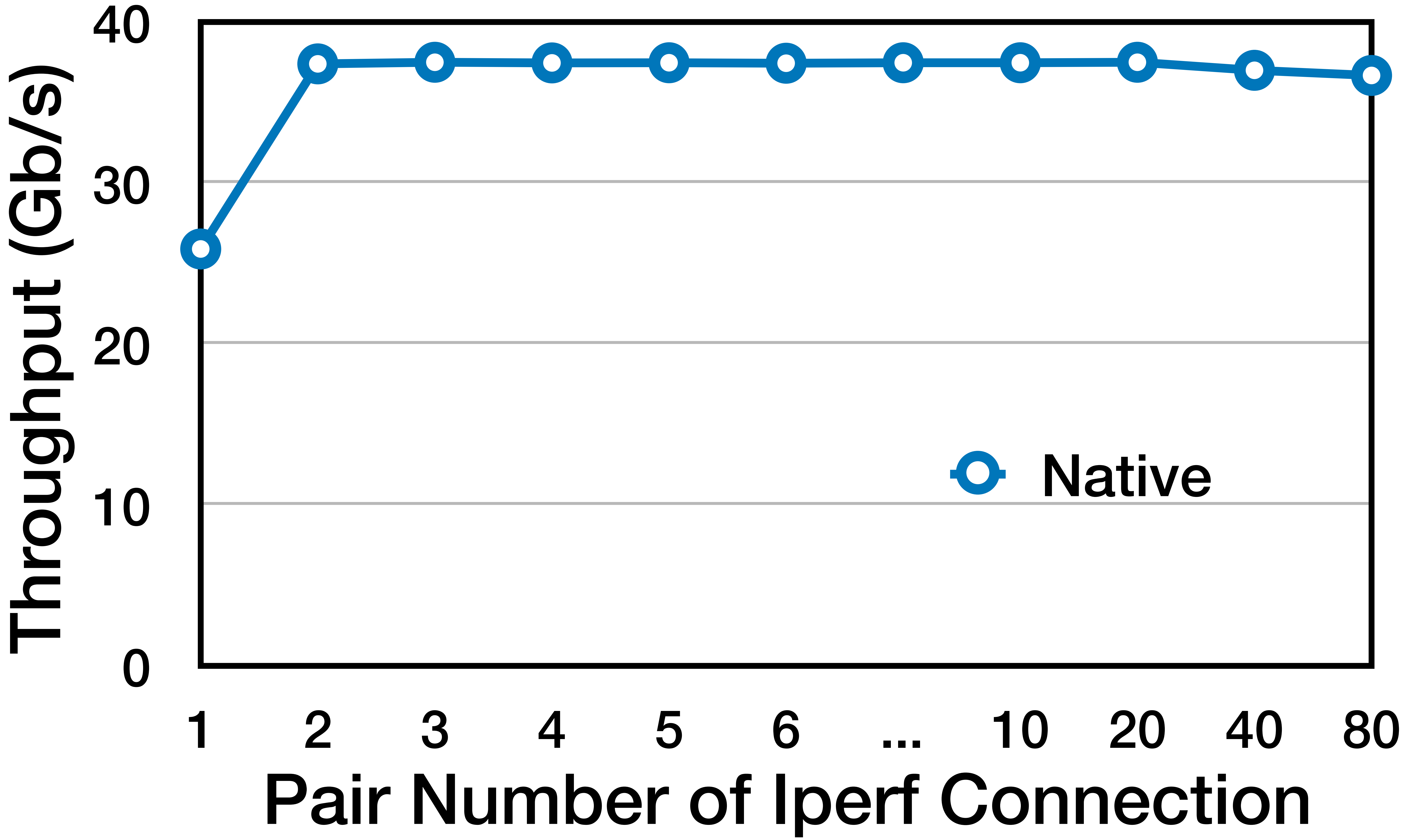


CPU Usage under Three Different Cases for TCP

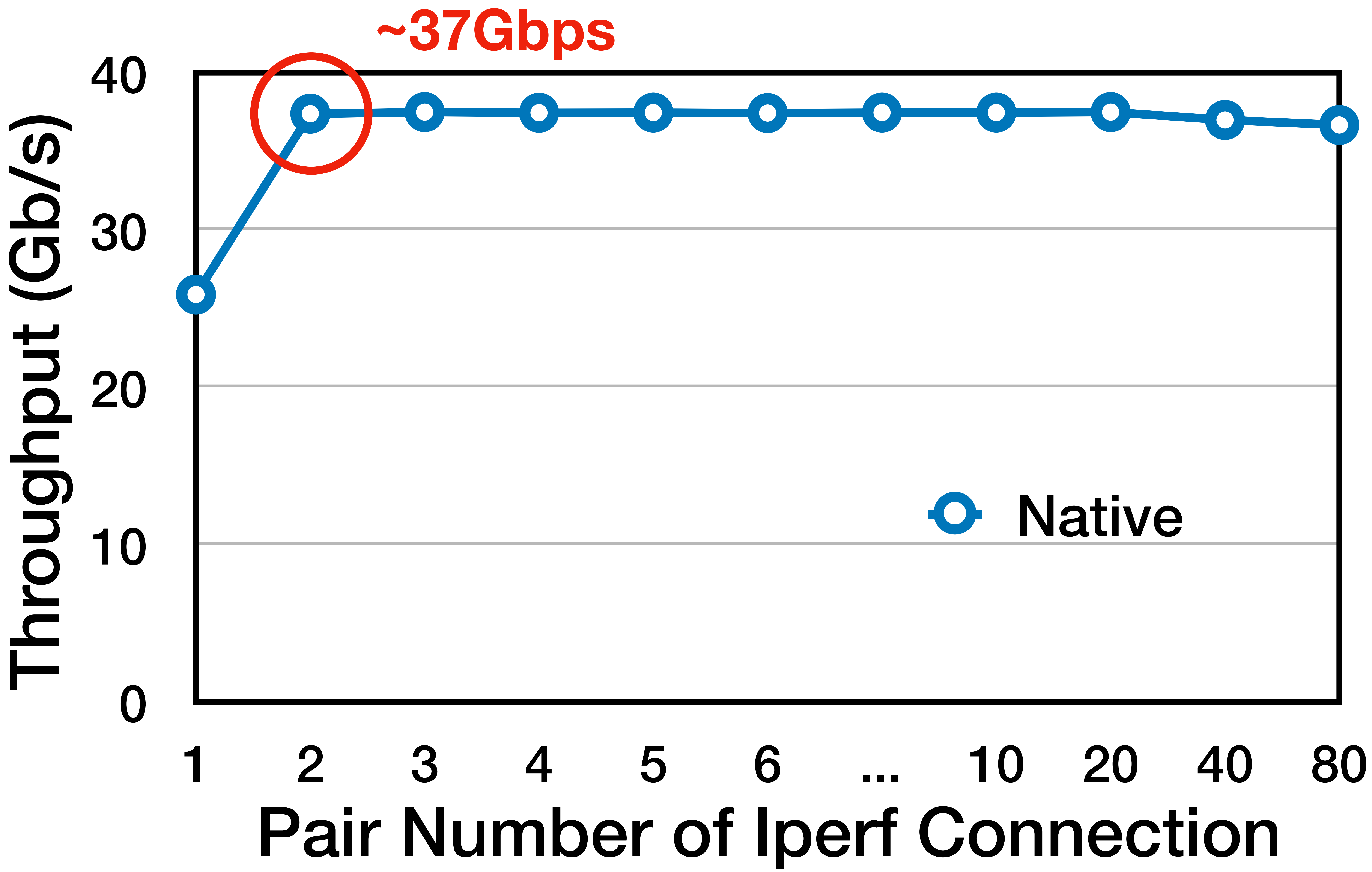
- Packet processing overhead **fully saturates** one cpu core in two overlay cases.
- Current solutions **can't scale** single flow performance.

* 5% indicates one cpu core is fully saturated.

Multiple Flows Performance

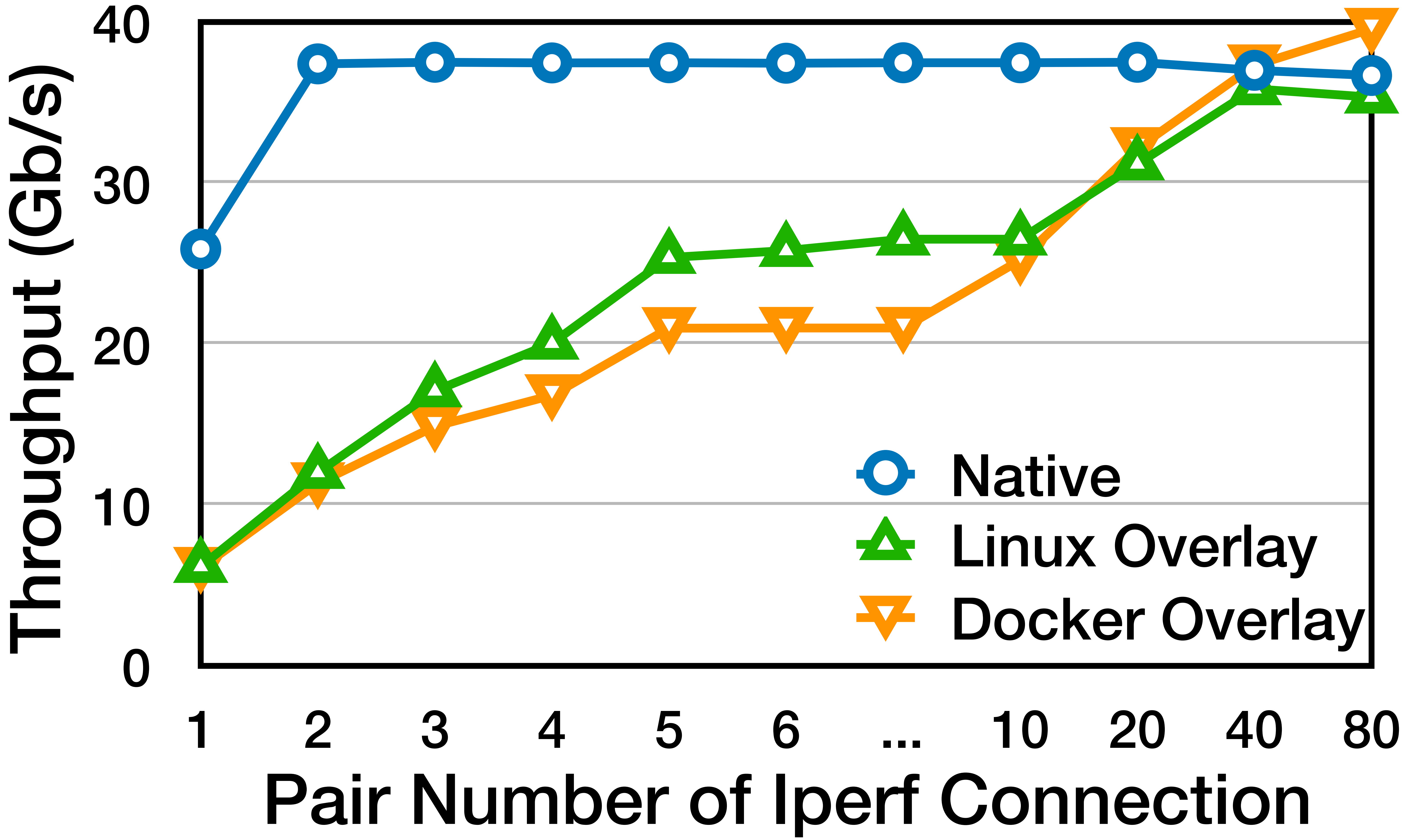


Multiple Flows Performance



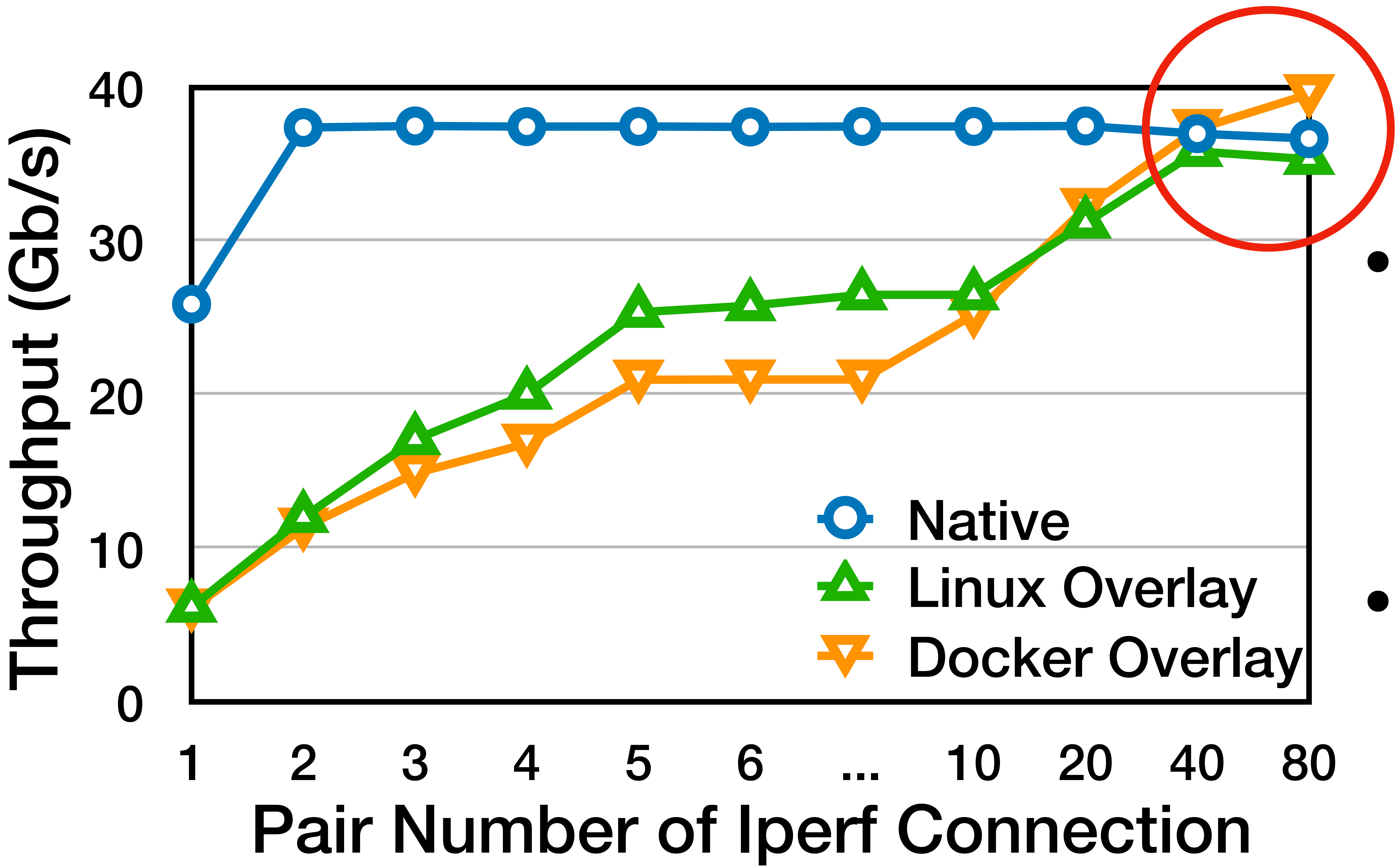
- Native case quickly reaches ~**37 Gbps** under TCP with only **2 pairs**.

Multiple Flows Performance



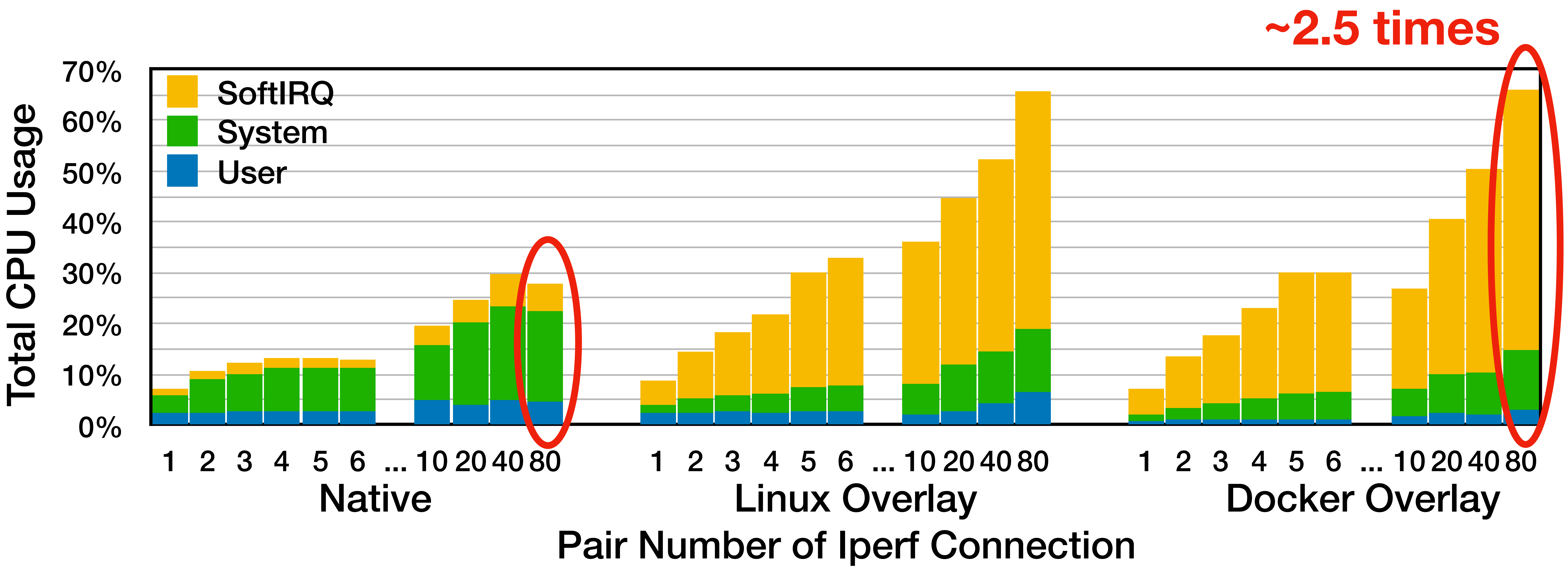
- Native case quickly reaches ~**37 Gbps** under TCP with only **2 pairs**.
- In two overlay cases, TCP throughput **grows slowly.**

Multiple Flows Performance



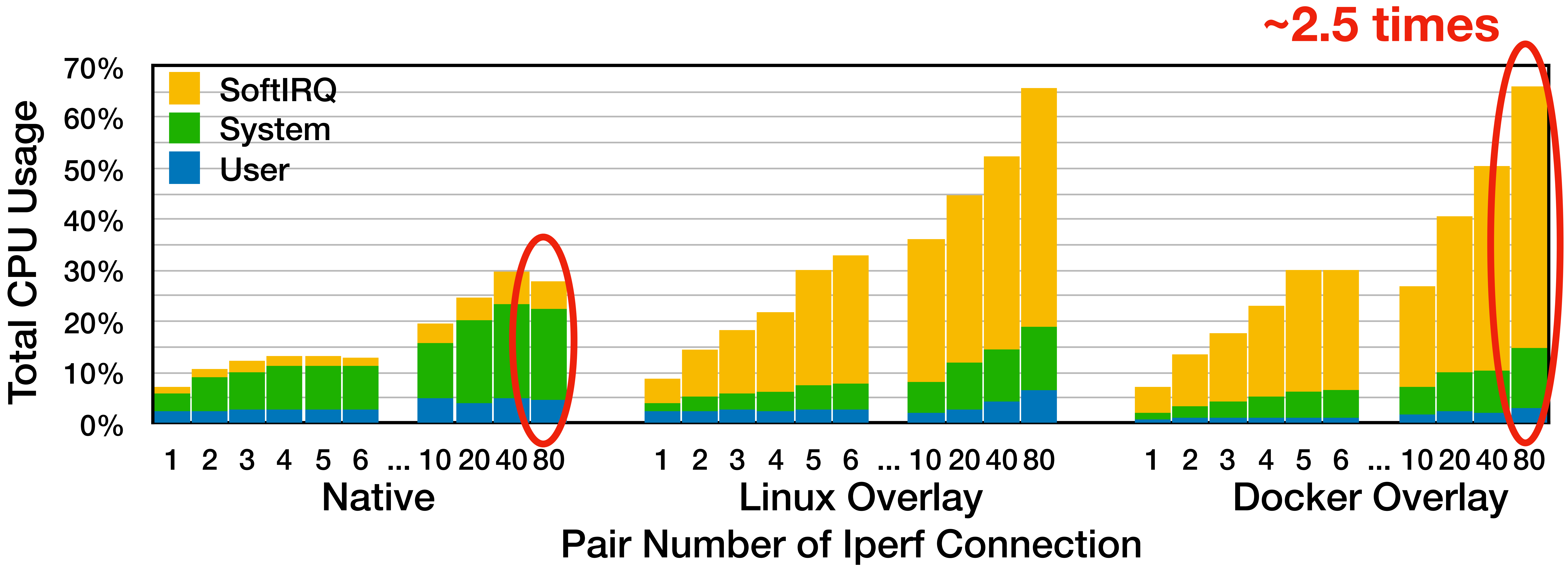
- Native case quickly reaches ~**37 Gbps** under TCP with only **2 pairs**.
- In two overlay cases, TCP throughput **grows slowly.**

Multiple Flows Performance



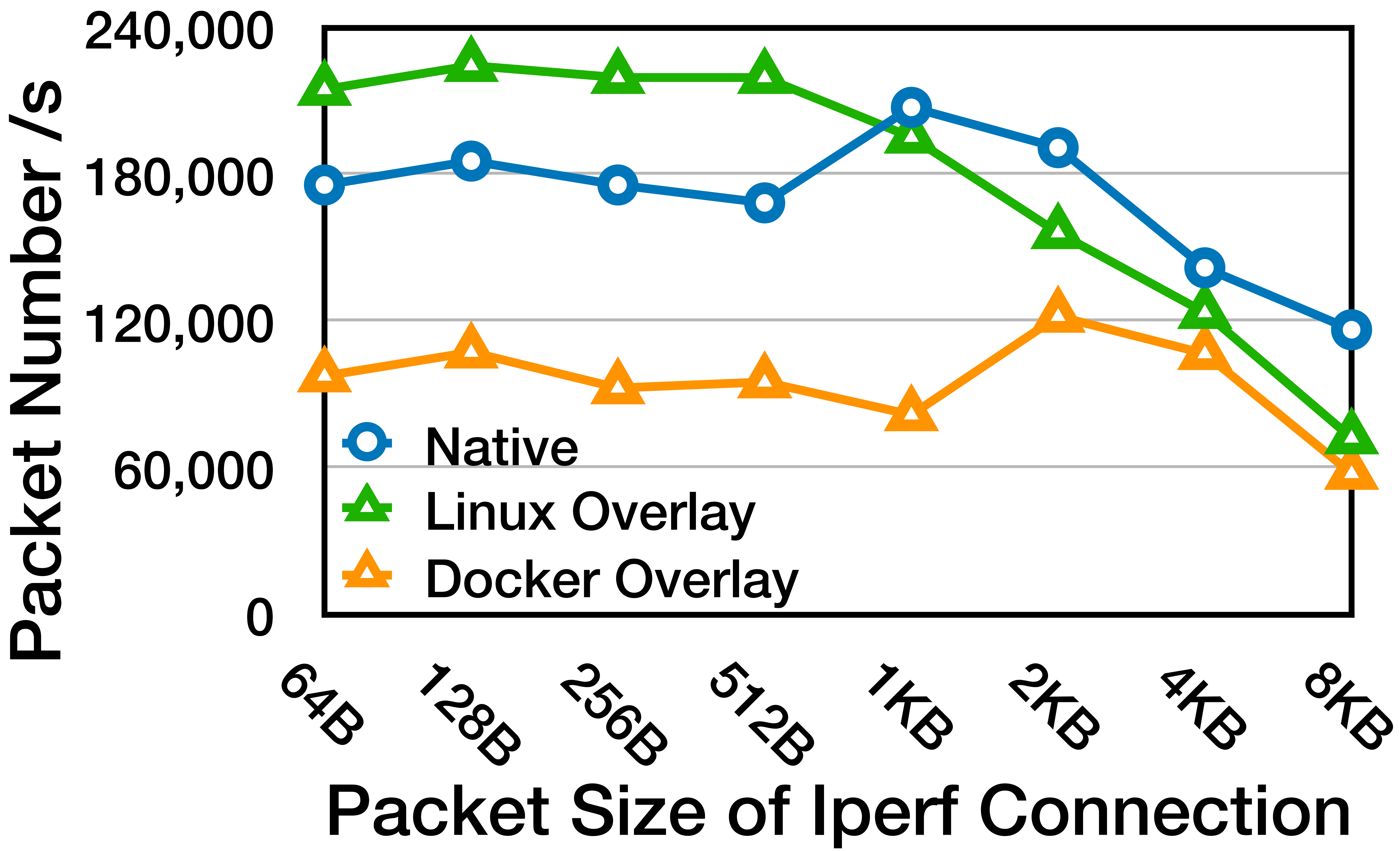
- Under the same throughput (e.g., 40 Gbps), overlay networks consume **much more** CPU resources (e.g., around **2.5 times**) than the native case.

Multiple Flows Performance

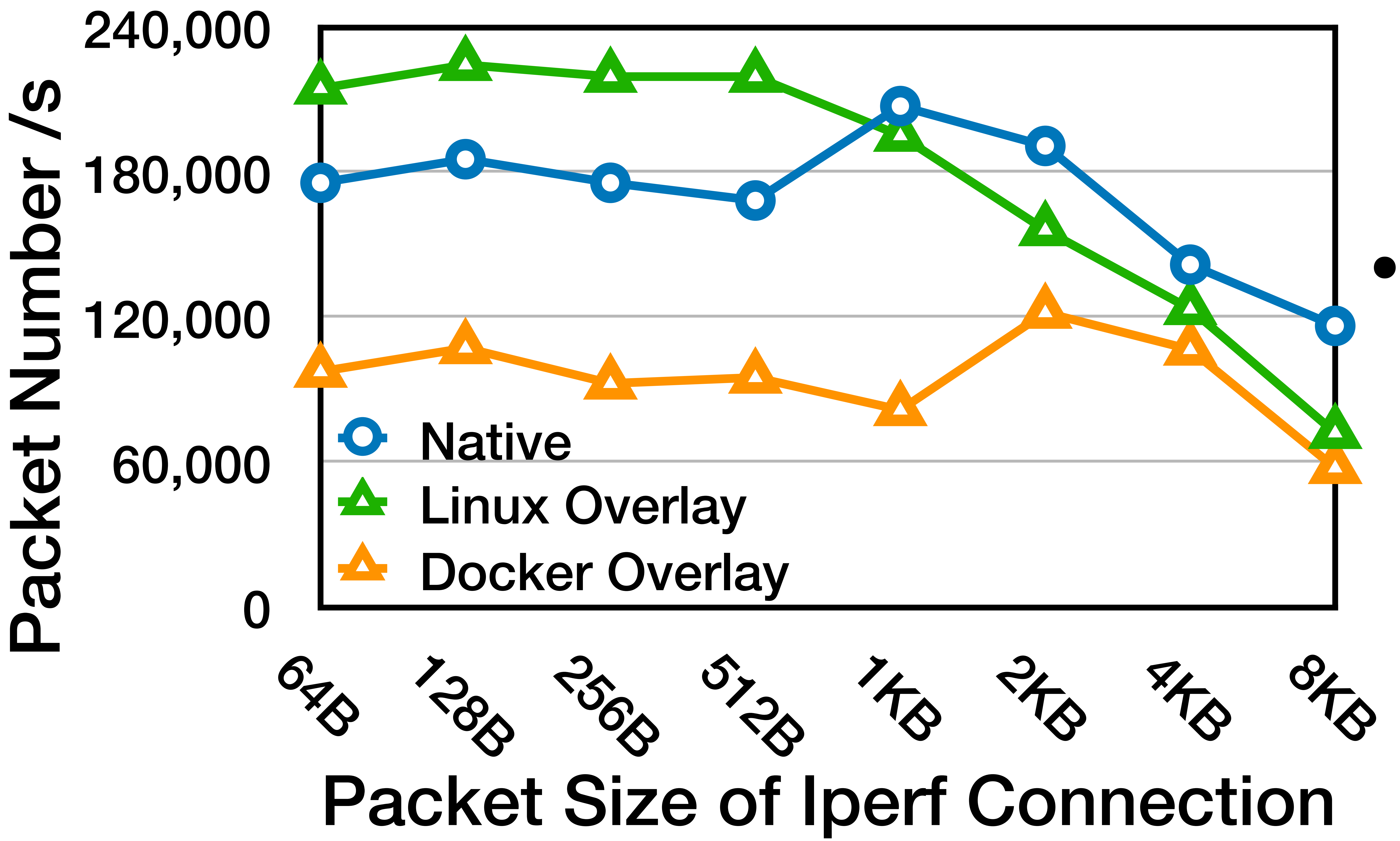


- Bad scalability is largely due to the **inefficient interplay** of many tasks.

Small Packet Performance



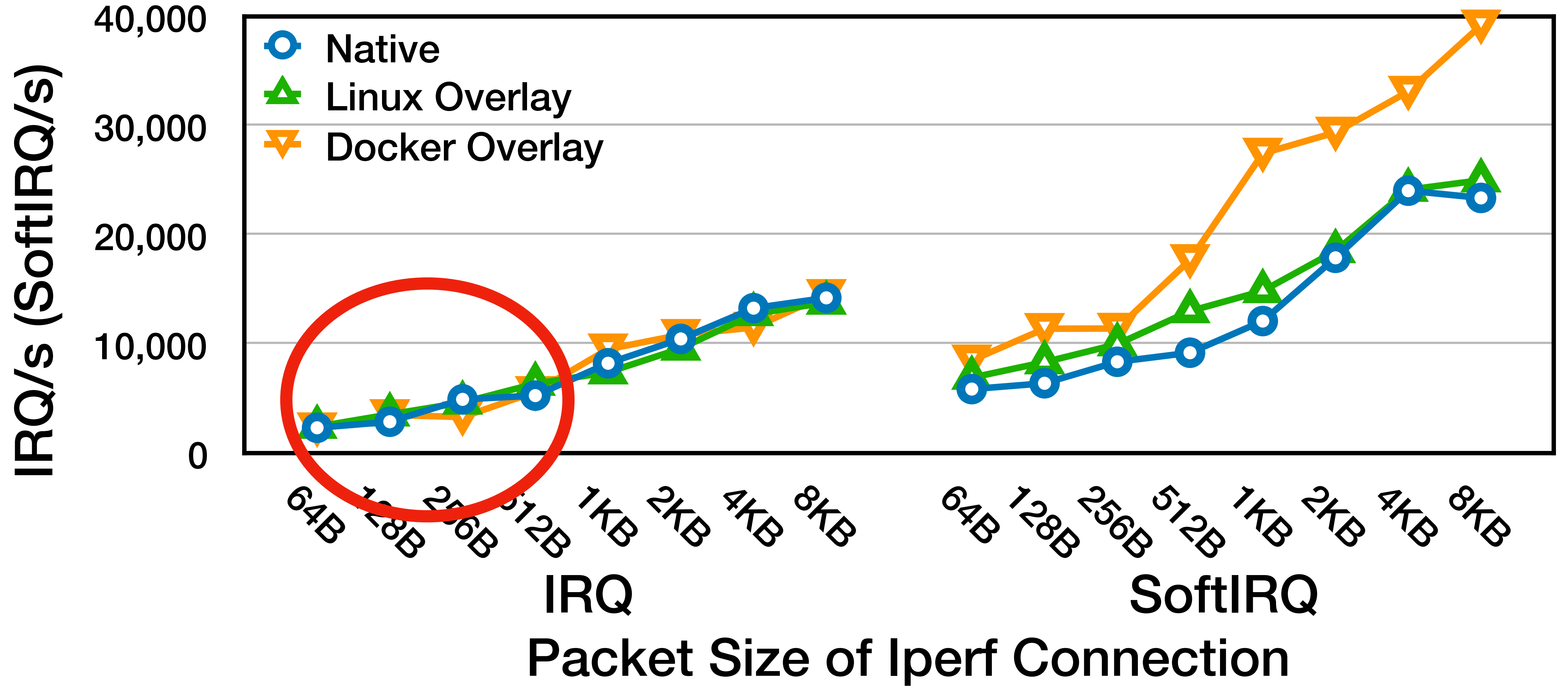
Small Packet Performance



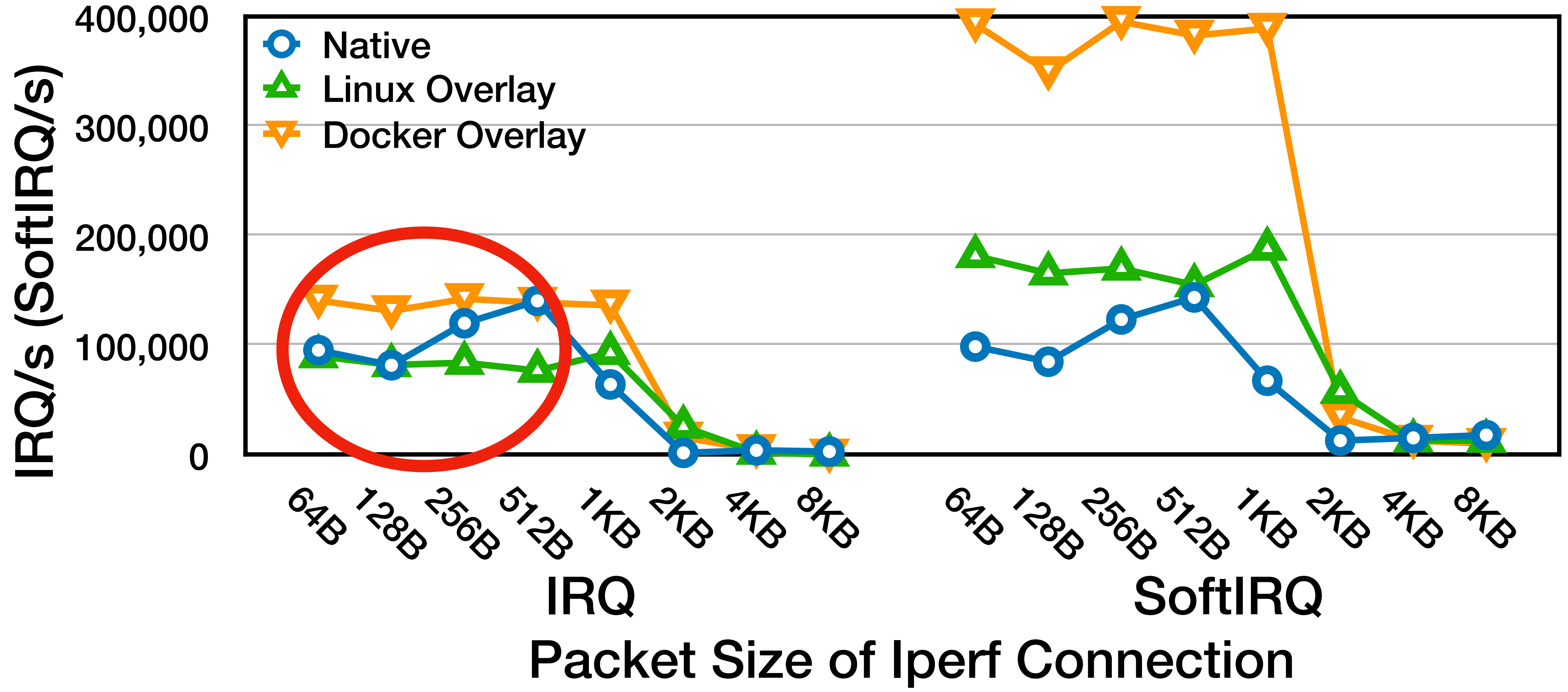
- Docker overlay achieves as low as **50%** packet processing rate of that in the native case.

Interrupt Number with Varying Packet Sizes

Interrupt number for TCP

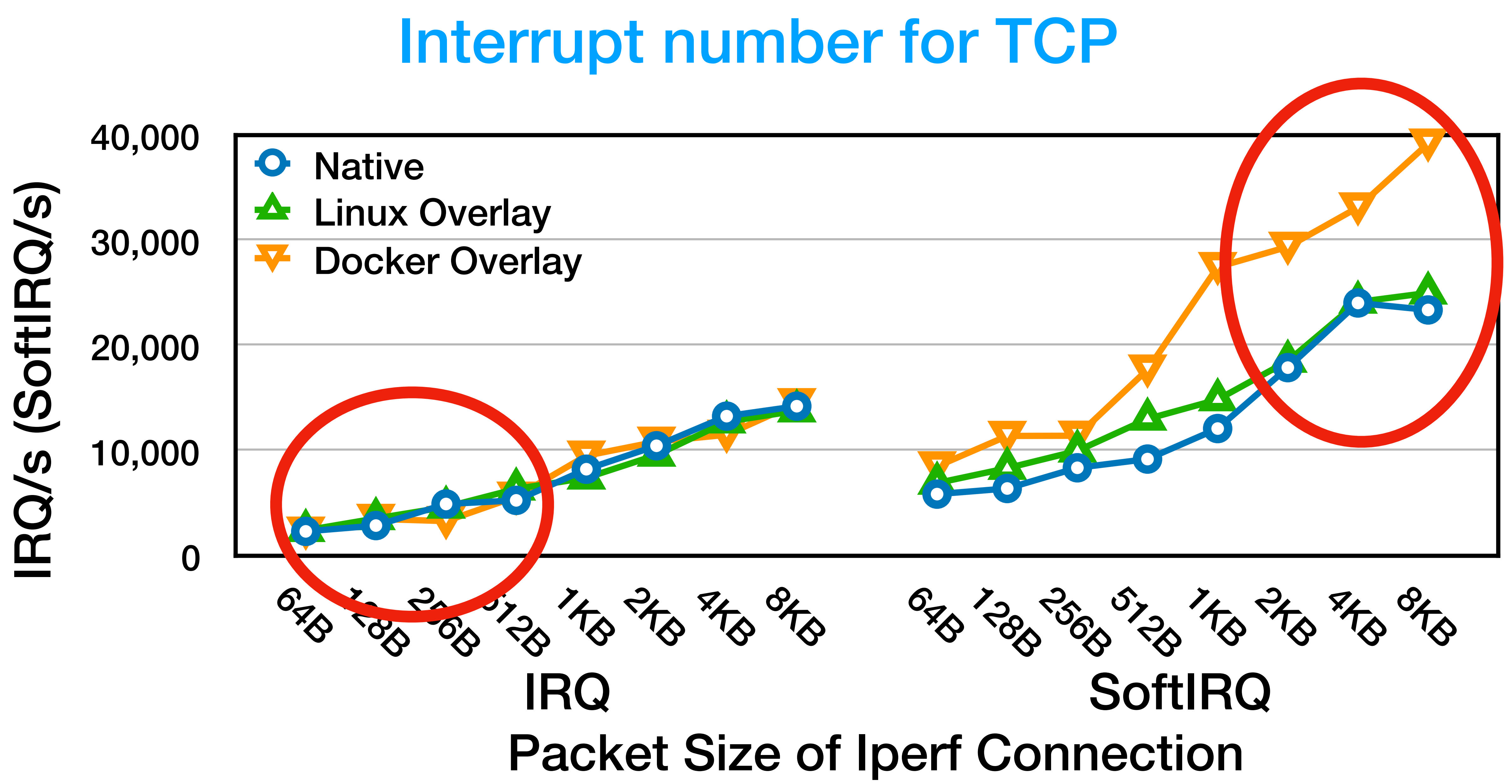


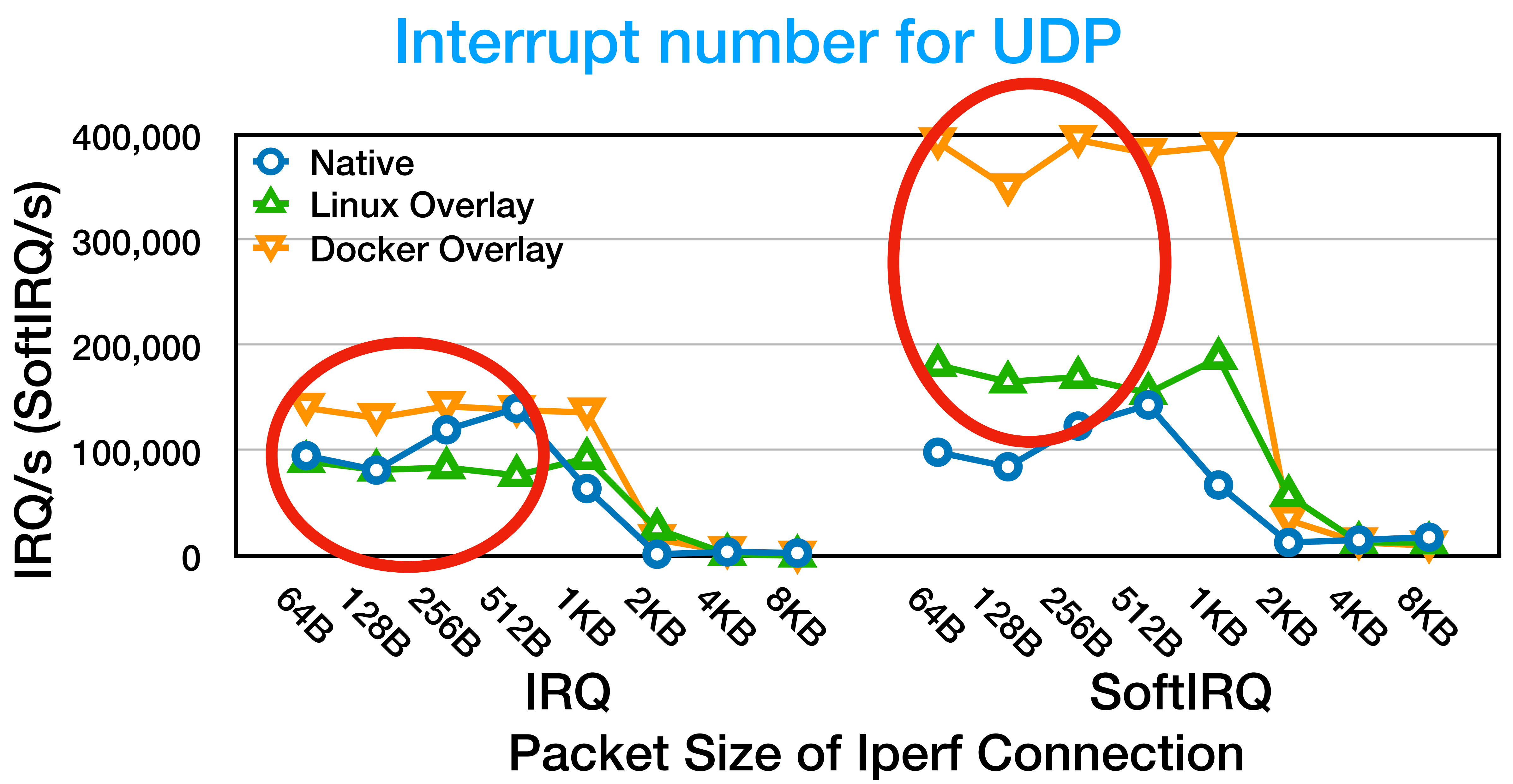
Interrupt number for UDP



- IRQ number increases dramatically in the Docker overlay UDP case — **10x** of that in the TCP case.

Interrupt Number with Varying Packet Sizes





- IRQ number increases dramatically in the Docker overlay UDP case — **10x** of that in the TCP case.
- **3x** softIRQ numbers are observed in Docker Overlay case compared with the IRQ numbers.

Insights and Conclusions

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
- Bottlenecks become more severe for **small packets**.

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
- Bottlenecks become more severe for **small packets**.

Thinking about future works:

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
- Bottlenecks become more severe for **small packets**.

Thinking about future works:

- Is it feasible to provide packet-level parallelization for a **single network flow**?

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
- Bottlenecks become more severe for **small packets**.

Thinking about future works:

- Is it feasible to provide packet-level parallelization for a **single network flow**?
- How can the kernel perform a **better isolation among multiple flows** especially for efficiently utilizing shared hardware resources?

Insights and Conclusions

- Kernel does not provide **per-packet level parallelization**.
- Kernel does not efficiently handle **various packet processing tasks**.
- Bottlenecks become more severe for **small packets**.

Thinking about future works:

- Is it feasible to provide packet-level parallelization for a **single network flow**?
- How can the kernel perform a **better isolation among multiple flows** especially for efficiently utilizing shared hardware resources?
- Can the packets be **further coalesced** with optimized network path for reduced interrupts and context switches?

Thank you!