ByteDance

Socket Locality Based Flow Selection in MPTCP

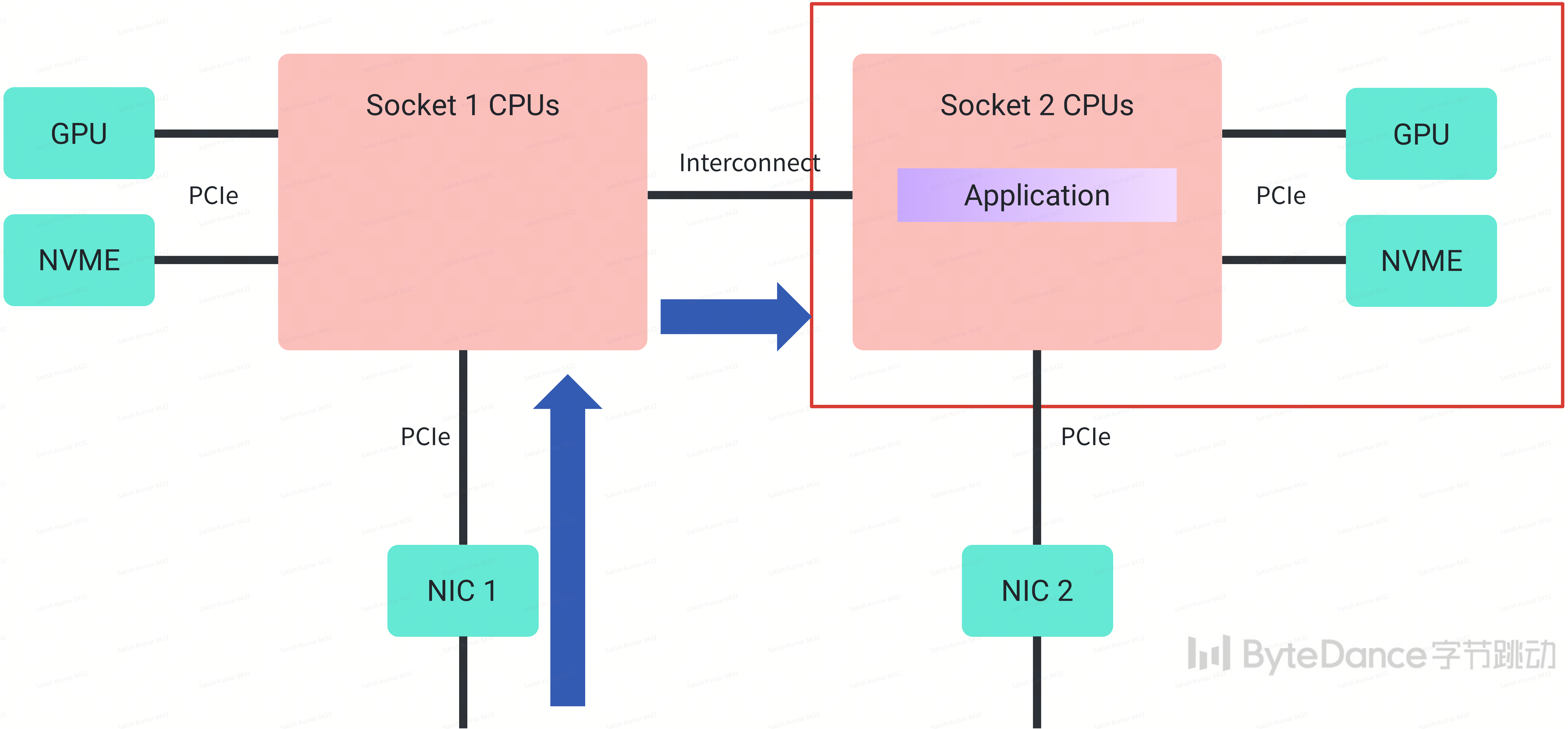
Satish Kumar
System Techonology Engineering (STE)
ByteDance

10th March 2025
NetDev 0x19

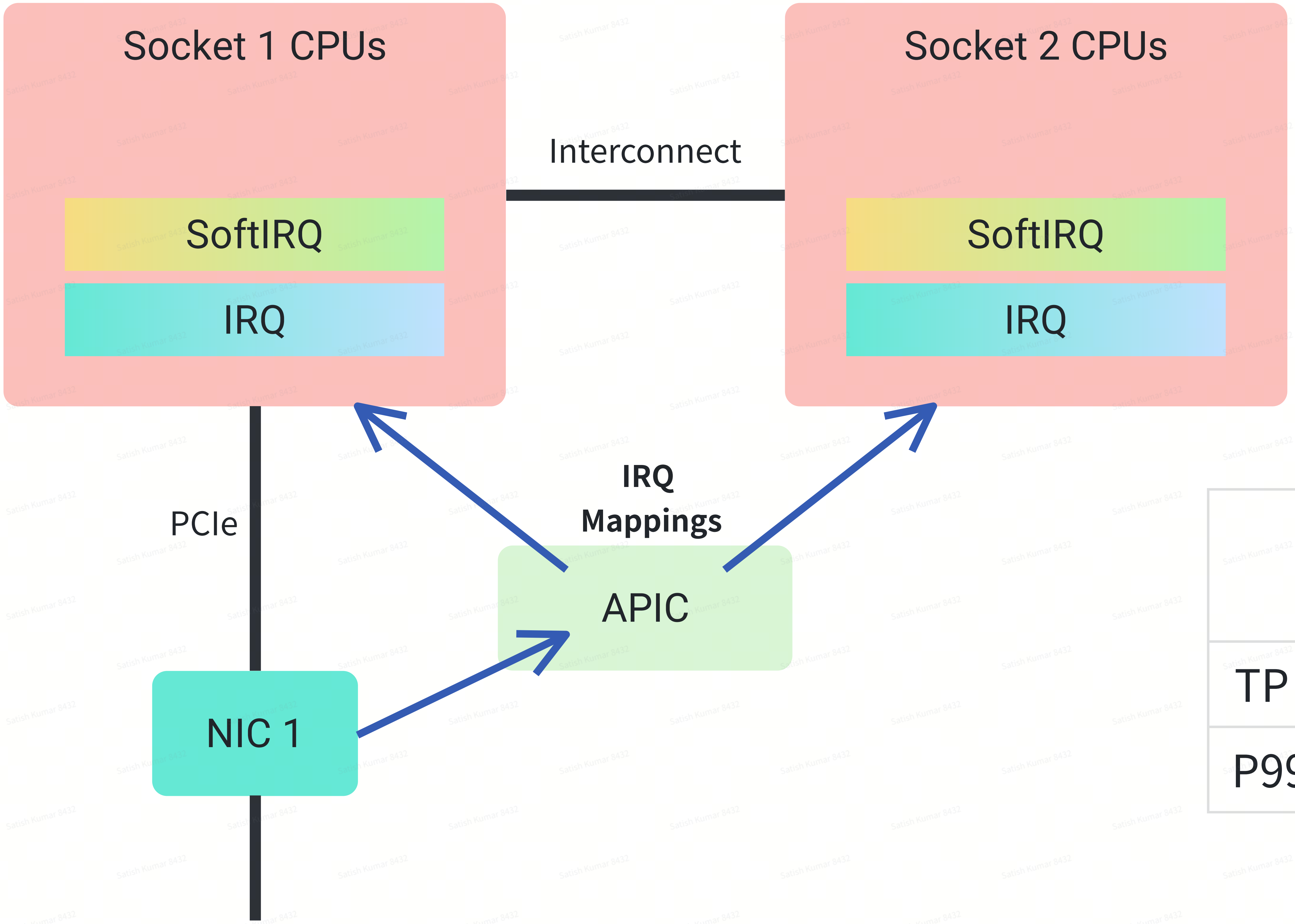
satish.kumar@bytedance.com

Agenda

The agenda of this topic is to minimize communication across sockets for network received data.



Case 1: RSS is distributed across sockets

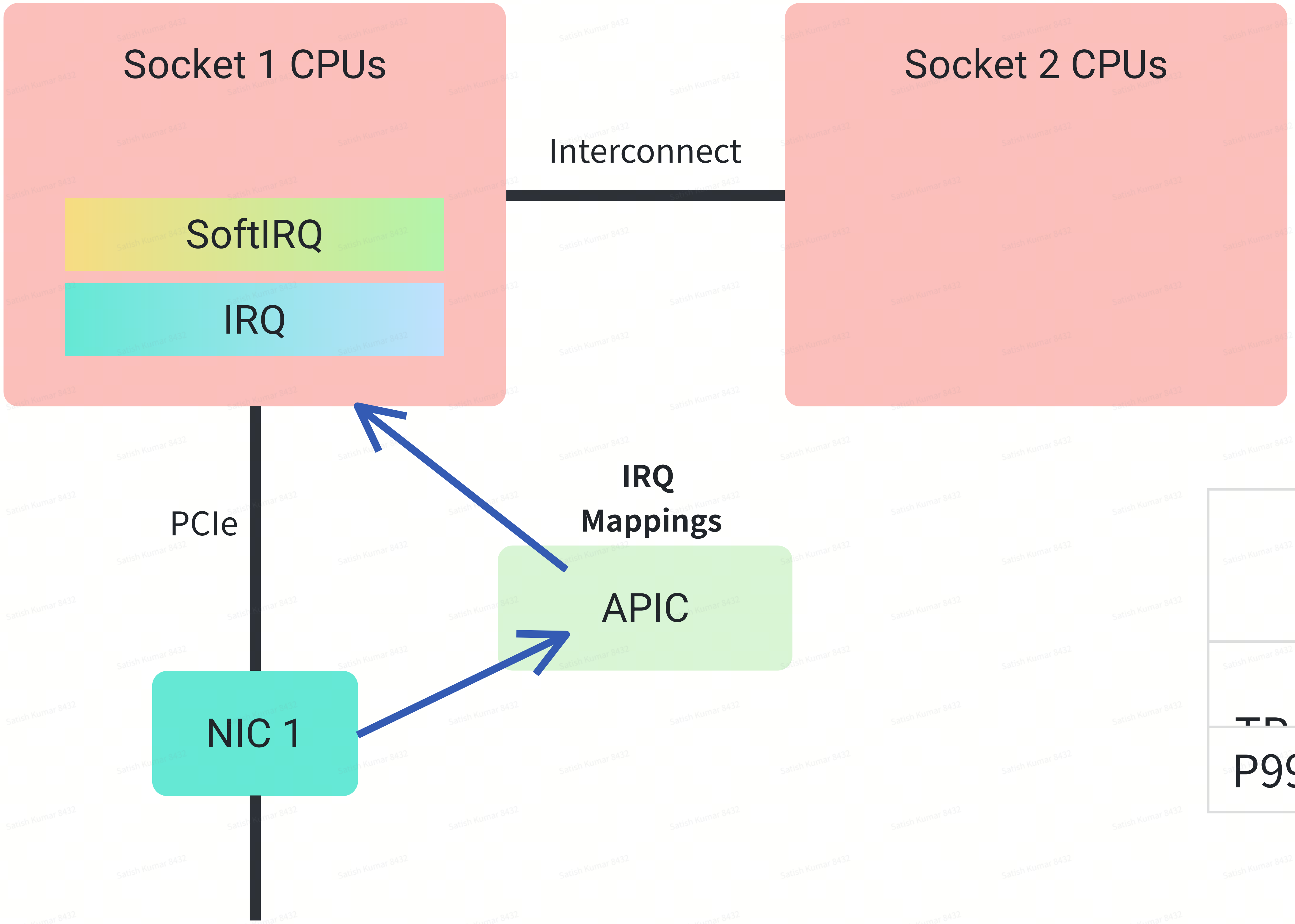


- RSS: Receive Side Scalling, Multi Q NIC
- Interrupts are evenly distributed between both sockets.
- Redis-memtier benchmark
  - Connected by TOR switch

	Redis Socket-1	Redis Socket-2	
TP (MB/s)	39.88	28.03	29.71% drop
P999 (ms)	2.00	2.37	18.5% incr

Ref: https://github.com/RedisLabs/memtier_benchmark

Case 2: RSS to local socket



- RSS: **Receive Side Scalling**, Multi Q NIC
- Interrupts are mapped to the local socket where the NIC is attached.
- Redis-memtier benchmark
    - Connected by TOR switch

	Redis Socket-1	Redis Socket-2	
TP (MB/s)	[illegible]	[illegible]	[illegible]
P999 (ms)	2.00	2.52	26 % incr

Ref: https://github.com/RedisLabs/memtier_benchmark

Problem Statement

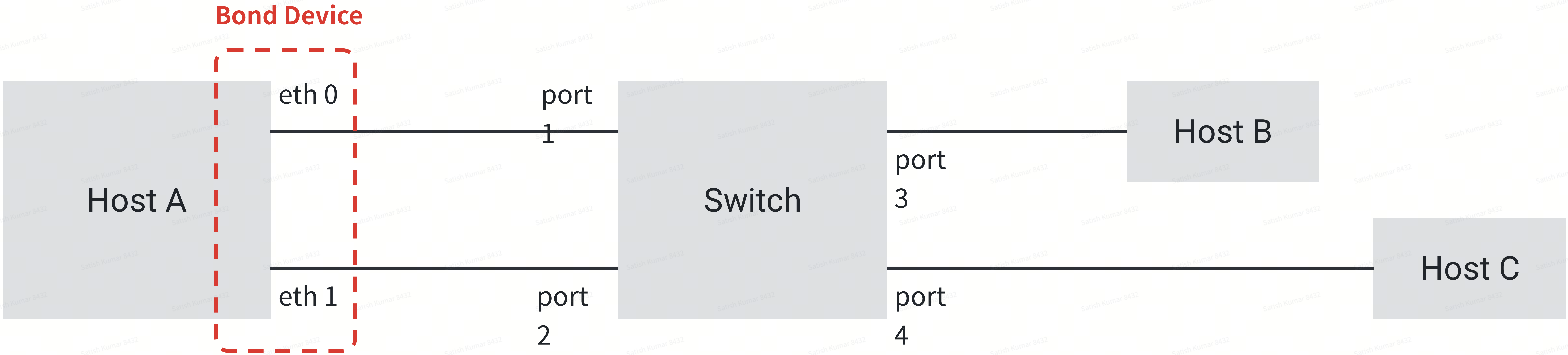
- For each connection, receive network data on the NIC that is closest in NUMA distance to the socket's read system call.

- While statically defining such a relationship is straightforward, the challenge lies in achieving it dynamically.

- Why socket read system calls?
  - Thats where application consumes network data.

- Requires connection level steering of traffic.

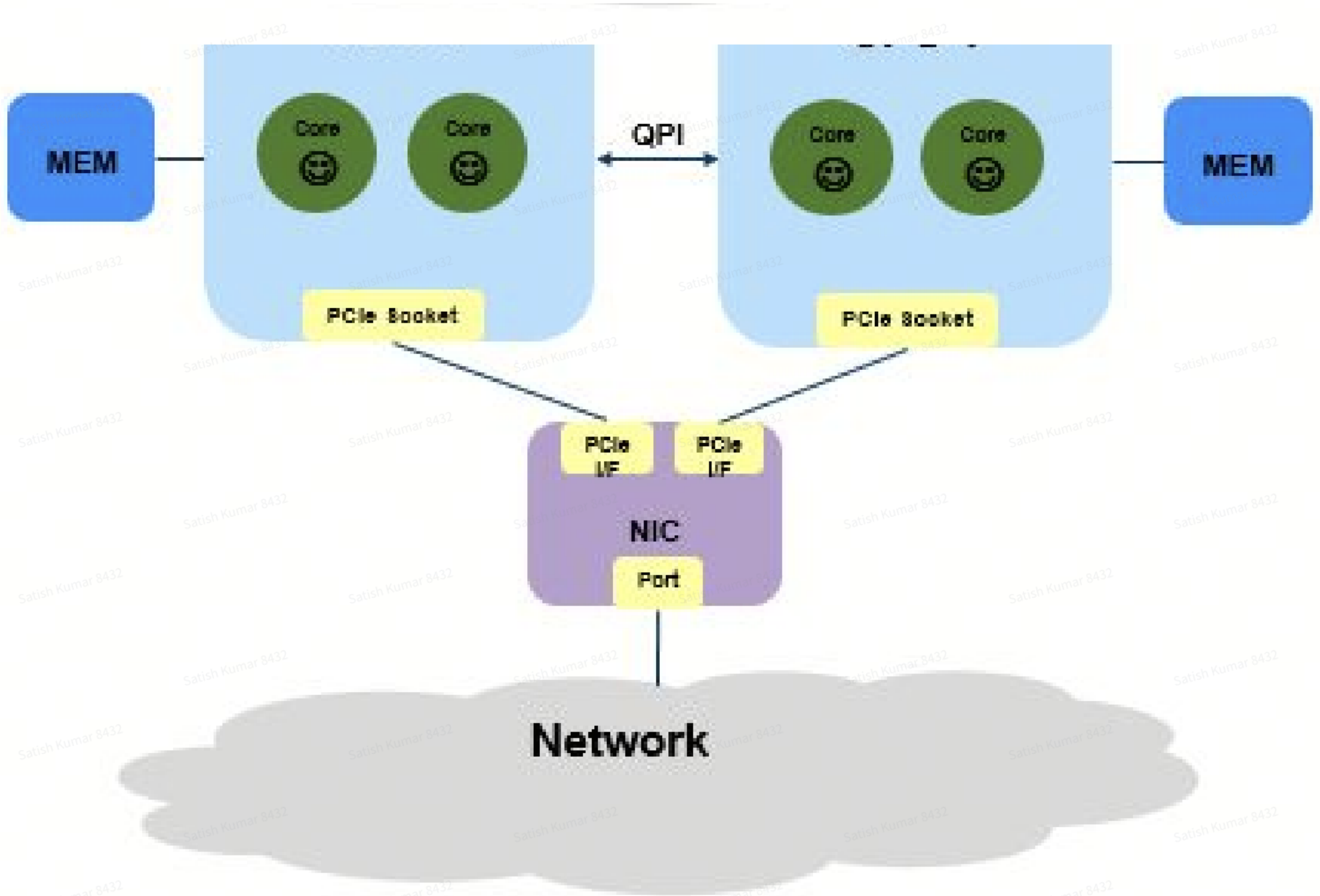
Linux bond device does not solve the problem.



- Bond devices are based on MAC addresses rather than connections <ip, port>.
- Need L3 bonding, not L2
  - MPTCP provides L3 bonding.

Ref:

Multi PCIe Socket Network Device

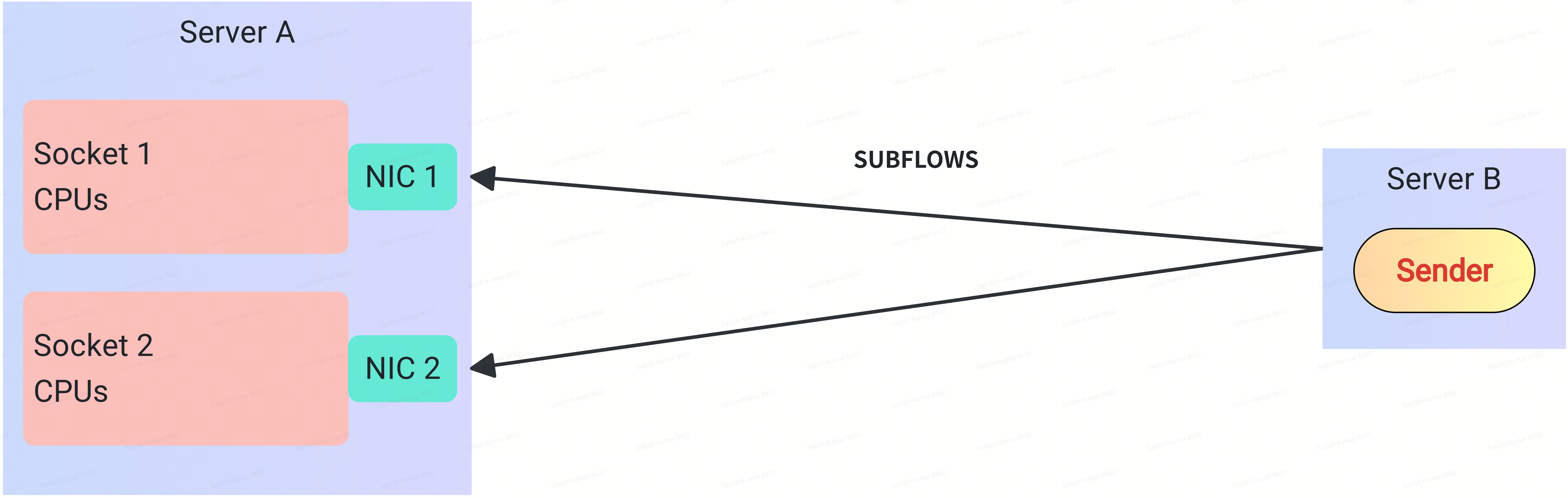


- Multi PCIe socket device along with aRFS (flow steering inside the device) can solve the problem.
- But..
  - Need a different hardware.
  - RFS typically associates the flow with a single CPU.
    - 'I think' combining RSS with RFS for scalability is not possible.

Ref: https://netdevconf.org/2.2/papers/shochat-devicemgmt-talk.pdf

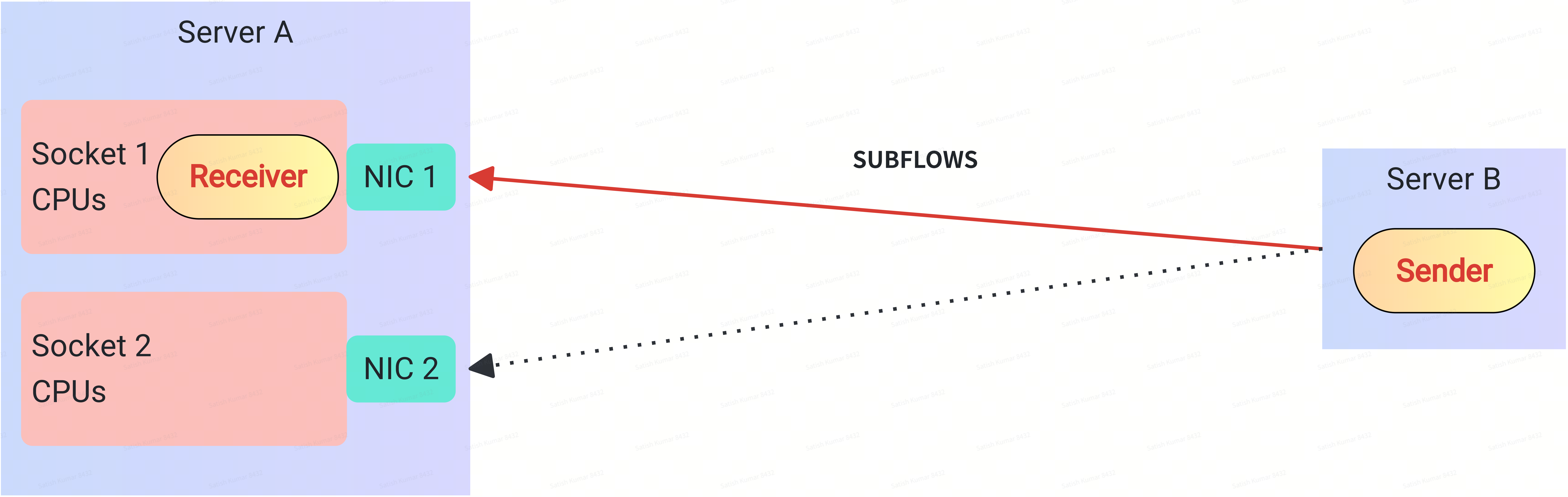
MPTCP Subflows

- MPTCP can create multiple subflows per connection / socket.
  - i.e per { port, ip } socket interface.



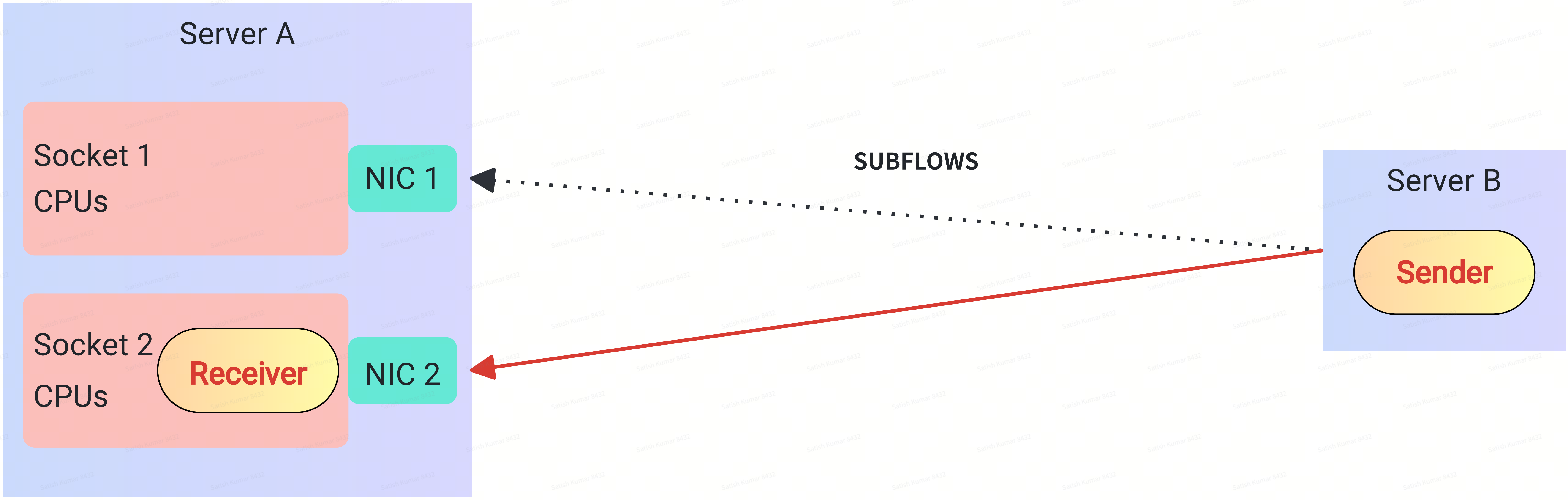
MPTCP Subflows

- MPTCP can create multiple subflows per connection / socket.
  - i.e per { port, ip } socket interface.



MPTCP Subflows

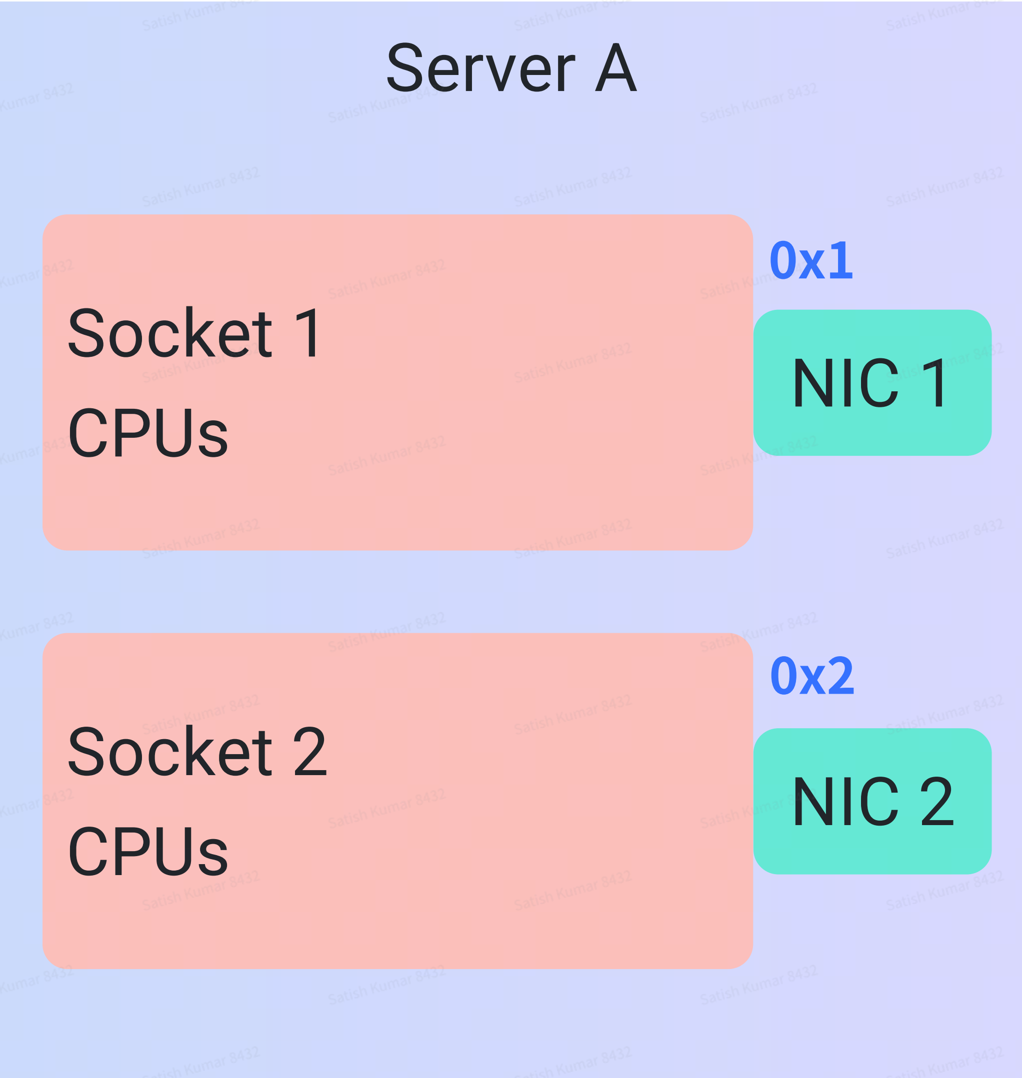
- MPTCP can create multiple subflows per connection / socket.
  - i.e per { port, ip } socket interface.



Implementaion Details

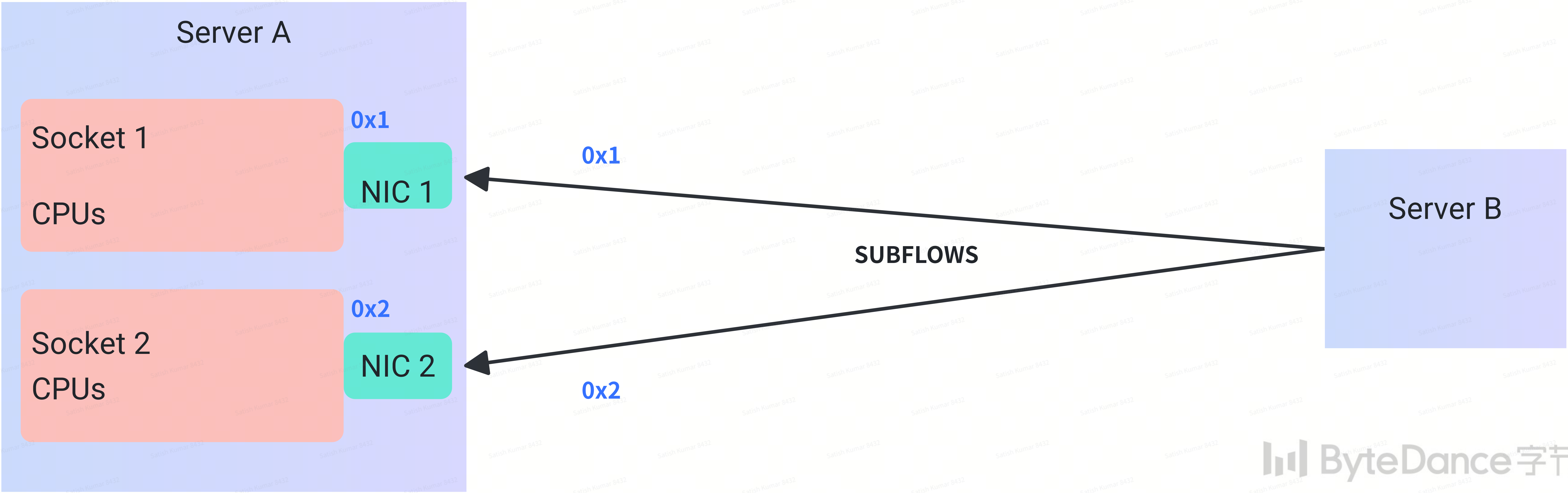
1. Add endpoints with information about the NUMA nodes to which they are locally attached.
    - `struct mptcp_addr_info.numa_affinity;`

```
root@debian:~# ip mptcp help
Usage:  ip mptcp endpoint add ADDRESS [ dev NAME ] [ id ID ]
                                      [ port NR ] [ numa_nodes HEX_MASK ] [ FLAG-LIST ]
```



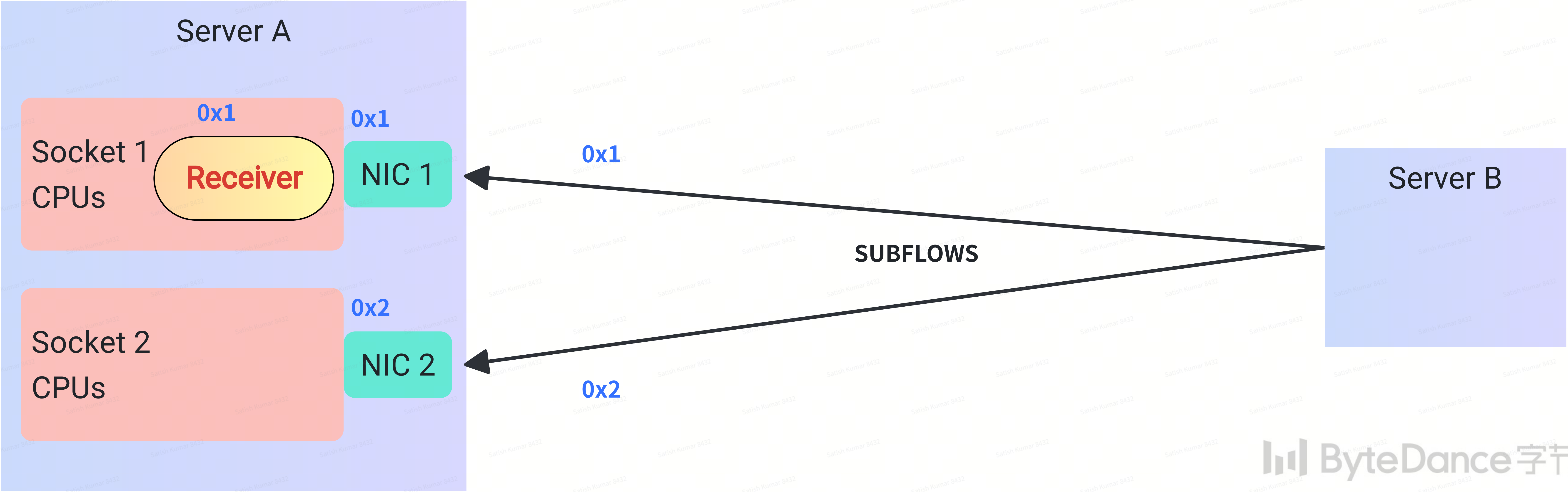
Implementaion Details

1. Add endpoints with information about the NUMA nodes to which they are locally attached.
    - `struct mptcp_addr_info.numa_affinity;`
2. Subflows inherits the NUMA affinity of the endpoints.
    - `struct mptcp_subflow_context.numa_affinity;`



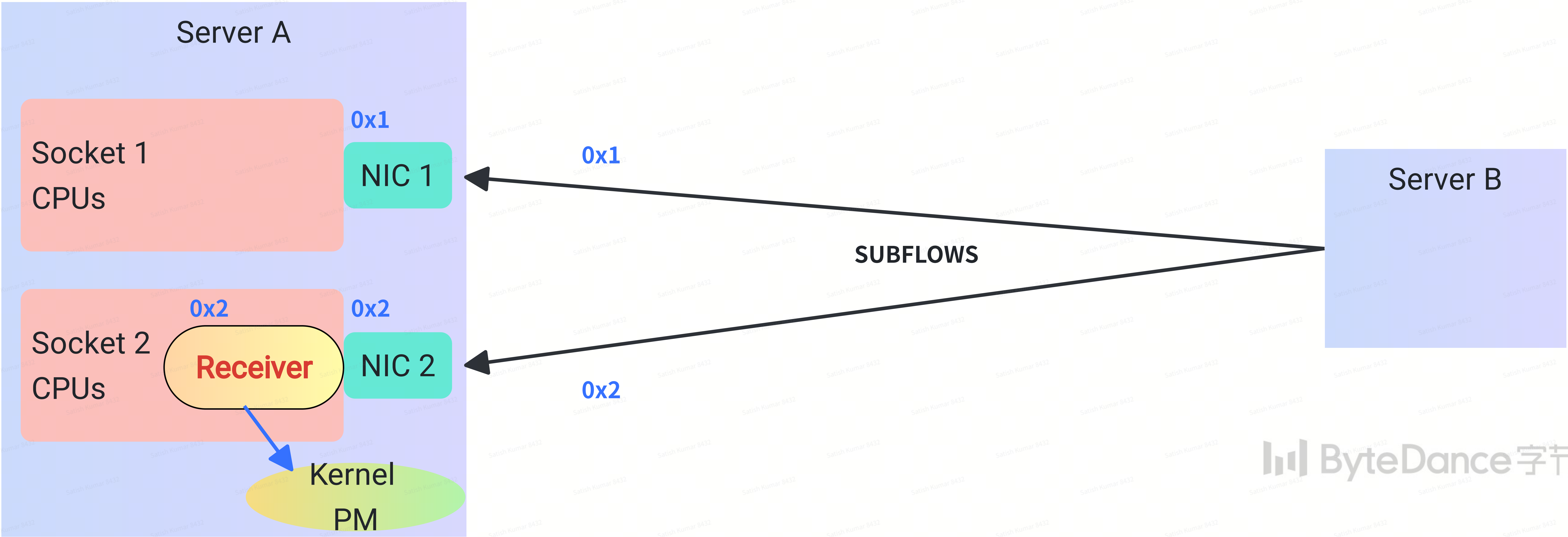
Implementaion Details

1. Add endpoints with information about the NUMA nodes to which they are locally attached.
   - `struct mptcp_addr_info.numa_affinity;`
2. Subflows inherits the NUMA affinity of the endpoints.
   - `struct mptcp_subflow_context.numa_affinity;`
3. Recvmsg thread state is stored inside:
   - struct mptcp_sock.numa_state;



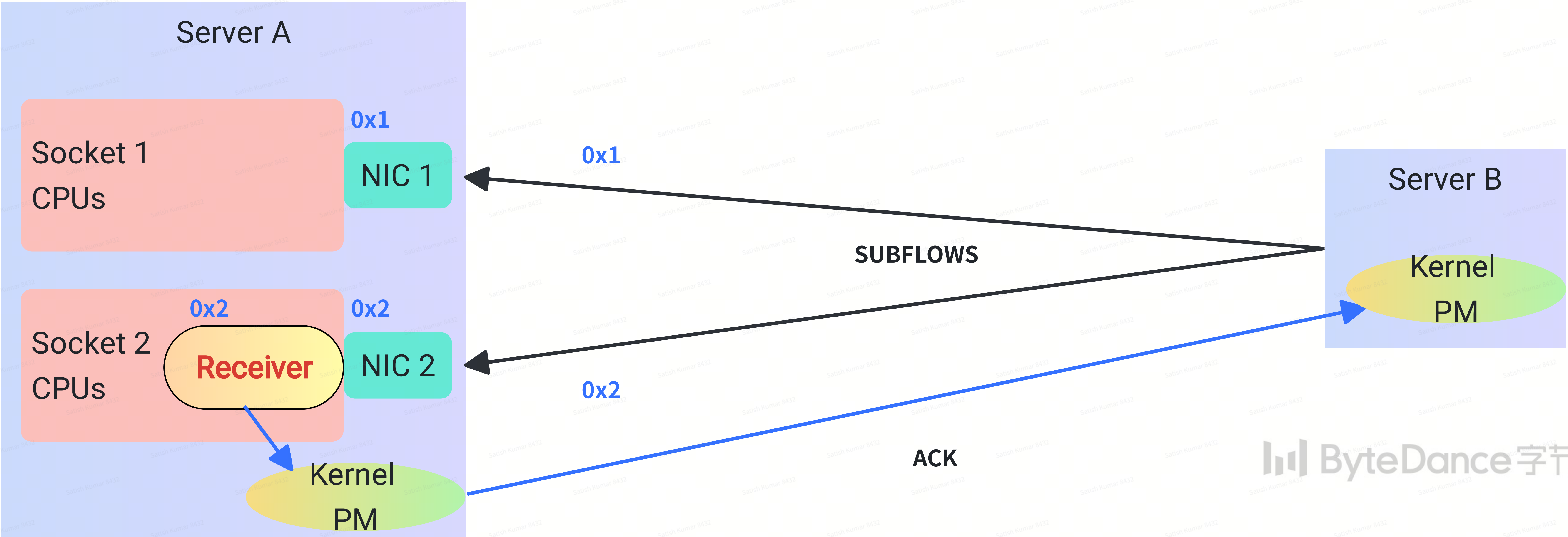
Implementaion Details

1. Add endpoints with information about the NUMA nodes to which they are locally attached.
   - `struct mptcp_addr_info.numa_affinity;`
2. Subflows inherits the NUMA affinity of the endpoints.
   - `struct mptcp_subflow_context.numa_affinity;`
3. Recvmsg thread state is stored inside:
   - struct mptcp_sock.numa_state;
4. Notify Kernel Path Manager
   - `mptcp_schedule_work(MPTCP_PM_SUBFLOW_NUMA_AFFINITY)`



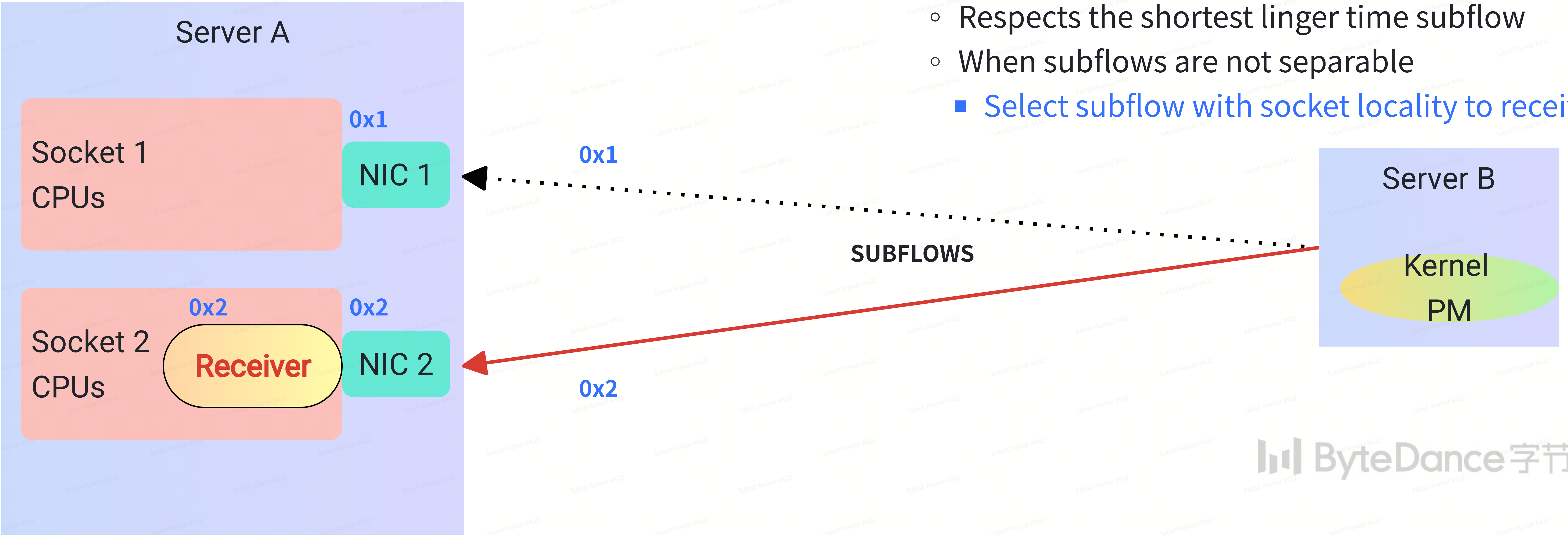
Implementaion Details

1. Add endpoints with information about the NUMA nodes to which they are locally attached.
   - `struct mptcp_addr_info.numa_affinity;`
2. Subflows inherits the NUMA affinity of the endpoints.
   - `struct mptcp_subflow_context.numa_affinity;`
3. Recvmsg thread state is stored inside:
   - struct mptcp_sock.numa_state;
4. Notify Kernel Path Manager
   - `mptcp_schedule_work(MPTCP_PM_SUBFLOW_NUMA_AFFINITY)`
5. Send ACK message to peer about change in backup flags
   - __mptcp_pm_send_ack



Implementaion Details

1. Add endpoints with information about the NUMA nodes to which they are locally attached.
   - `struct mptcp_addr_info.numa_affinity;`
2. Subflows inherits the NUMA affinity of the endpoints.
   - `struct mptcp_subflow_context.numa_affinity;`
3. Recvmsg thread state is stored inside:
   - struct mptcp_sock.numa_state;
4. Notify Kernel Path Manager
   - `mptcp_schedule_work(MPTCP_PM_SUBFLOW_NUMA_AFFINITY)`
5. Send ACK message to peer about change in backup flags
   - __mptcp_pm_send_ack
6. Flow selection algorithm
   - Respects the shortest linger time subflow
   - When subflows are not separable
     - Select subflow with socket locality to receiver



Performance Numbers

- Intel Xeon 128 CPU dual core servers connected via TOR switch.
- Comparison of MPTCP configurations:
  - Config 1: all endpoints are configured as "subflow signal"
  - Config 2: all endpoints are configured with respective NUMA affinity "subflow signal numa_affiniity 0x.."

Redis server count		Config 1	Config 2 NUMA	
8	TP (MB/s)	178	236	**32%**
	P999 (msec)	26	23	**11%**
16	TP (MB/s)	344	447	**29%**
	P999 (msec)	68	65	**4.5%**
32	TP (MB/s)	569	745	**30%**

Against TCP

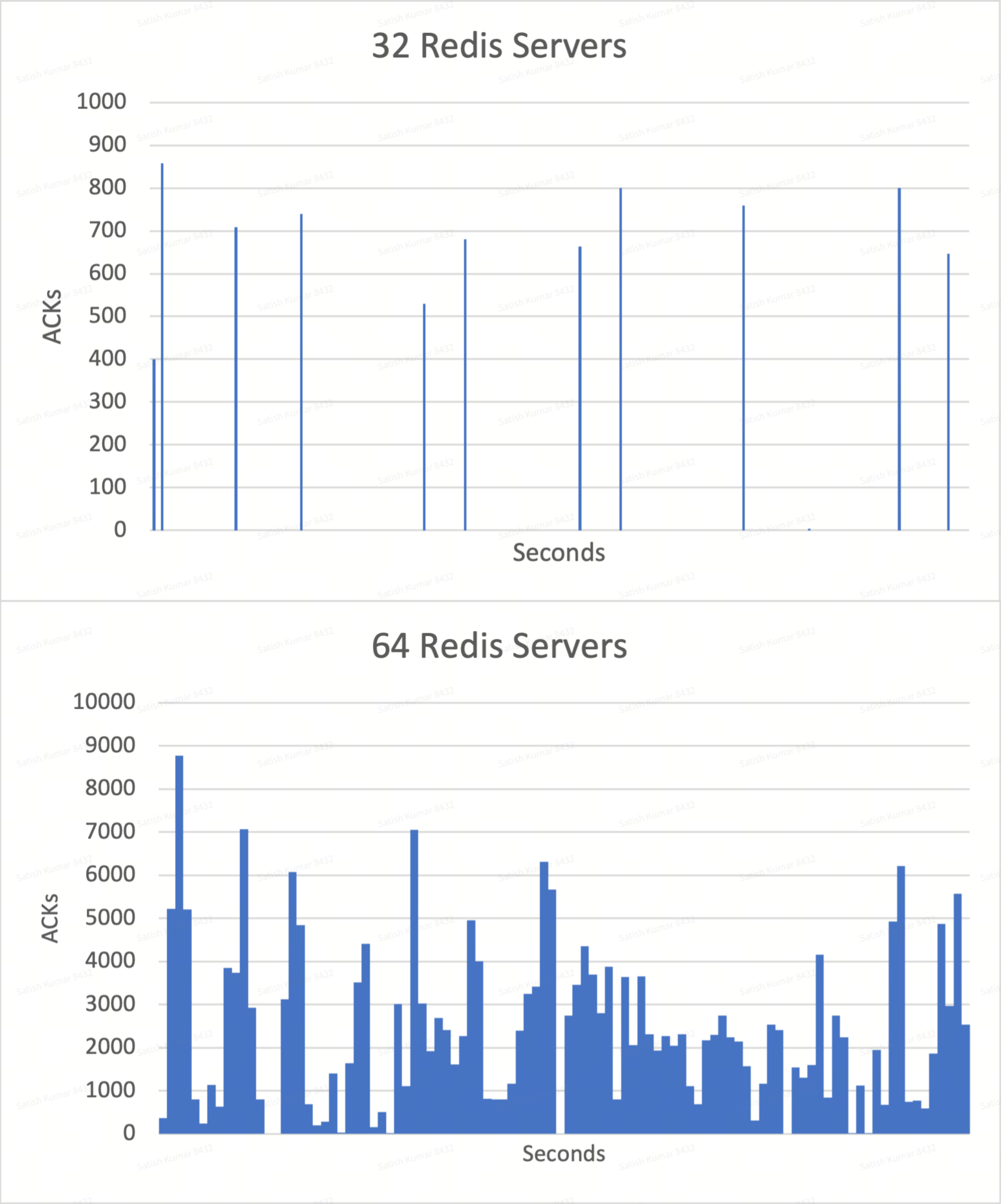
- Intel Xeon 128 CPU dual core servers connected via TOR switch.
- Comparison of MPTCP against TCP:
  - Mptcp: only one endpoint added with "signal subflow" flags.
  - Tcp: default settings of latest kernel

Redis server count		**Mptcp**	**Tcp**	
32	TP (MB/s)	547	744	**-26%**
	P999 (msec)	223	216	**-3.2%**

- What causes such a significant performance gap, even in a single flow scenario within a controlled lab environment without external traffic?
  - Purely mptcp stack overhead?
- Reducing the gap is crucial for all datacenter scenarios.

Fundamental Assumptions

- Single receiver thread per socket:
  - Common datacenter applications reads data within event loop.
- The scheduler rarely assigns the reader thread to CPUs outside the current socket zone.
  - Otherwise too many ACKs can reduce the performance.
- Steering the sender network data to the nearest NUMA NIC can be accomplished using eBPF hooks, or alternatively, it is part of our implementation within MPTCP that is not discussed in the presentation.



Thank you
Q&A

ByteDance字节跳动