Hewlett Packard
Enterprise

NUMA-Aware TCP and Ethernet Kernel Tuning for Exascale HPC Systems

Optimizing Linux TCP/IP stack for deterministic ultra-high throughput and lowest latency

Ravi Bissa, Ian Ziemba, Duncan Roweth, HPE

May 6, 2025

Latency vs Bandwidth: Significance in HPC TCP

Both latency and bandwidth must be optimized in tandem for exascale TCP performance.

Latency: Time taken for a single packet to travel between endpoints
- Impacts small-message MPI (e.g., MPI_Barrier, control signals).
- High latency stalls job synchronization and collective operations.

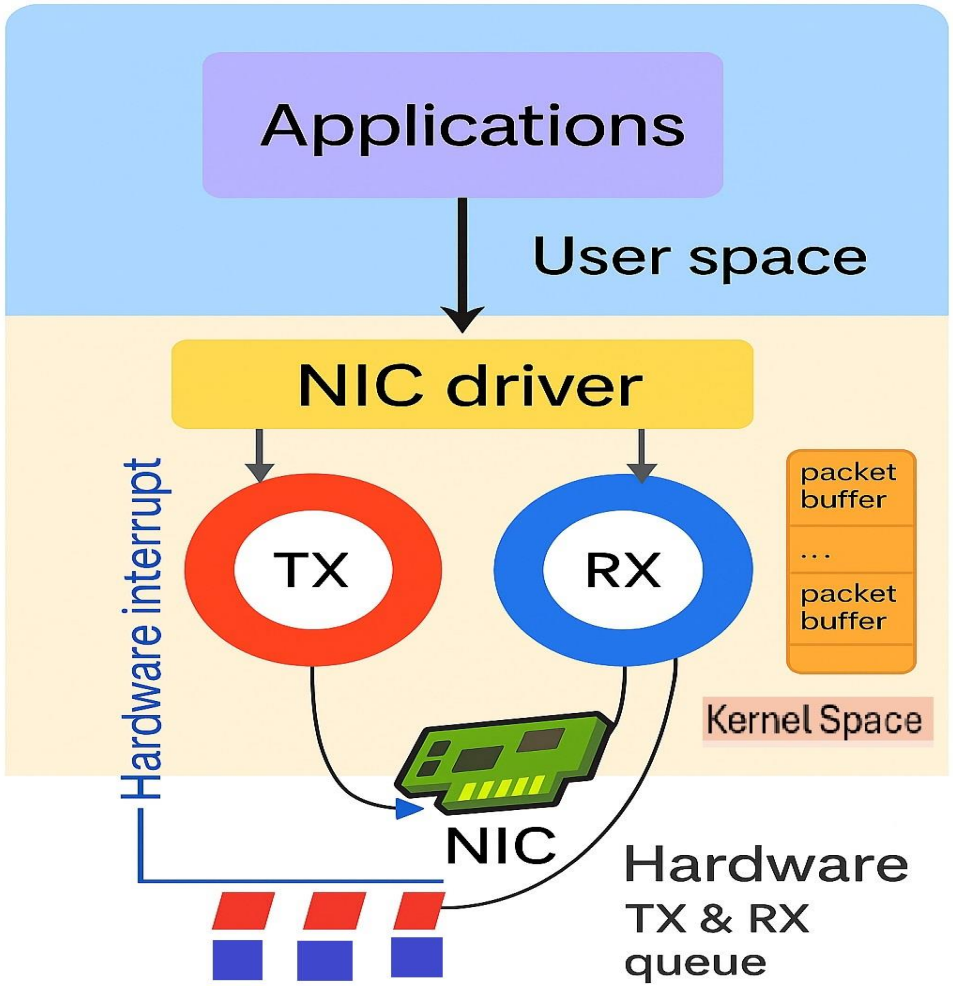
Bandwidth: Maximum rate of data transmission over the network.
- Critical for large data transfers — checkpointing, bulk I/O, MPI_Alltoall.
- Determines how much data can be moved per second.

Balance: Low latency ensures responsiveness; high bandwidth ensures throughput.
- Optimizing only, one results in partial performance gains.
- HPC systems require both for consistent, scalable performance.

Step-by-Step TCP Packet Flow (Receive Path)

How a TCP packet travels through the system from NIC to application.

1. **NIC Receives Packet from Network**
   - Packet arrives and is placed in NIC's hardware RX queue.
2. **NIC Triggers a Hardware Interrupt**
   - Interrupt is ideally routed to a NUMA-local core for fast handling.
3. **NIC Driver Handles the Interrupt**
   - Reads descriptor from RX queue and accesses packet buffer.
4. **Packet Stored in RX Ring Buffer**
   - Packet is moved or referenced in a circular buffer in kernel memory.
5. **Packet Moves Up the Network Stack**
   - Packet is processed through Ethernet ⟶ IP ⟶ TCP layers.
6. **Socket Buffer Queued for Application**
   - System call (recv) transfers packet from kernel to user space



Problem Statement: Why Default Behavior Fails at Scale

Problem	Why?	Evidence
TCP window sizing caps throughput	Throughput = Window Size / RTT Throughput = 6 MB / 0.001 s = 48 Gbps If you're on a **100 Gbps** link — you're using **less than half** the available bandwidth.	40–60% link use w/ defaults (iperf3)
NUMA-unaware interrupts = 2x latency	When a **NIC** raises an interrupt, it should ideally be handled by a **CPU core local to the NIC's NUMA node**. If the interrupt is handled on a **remote NUMA node**: • Data crosses CPU sockets • Latency jumps	perf stat: remote memory stalls
irqbalance assigns IRQs blindly	irqbalance does **not guarantee IRQs are kept on the same NUMA node** as the device raising the interrupt irqbalance knows **nothing about your HPC workload** or job placement.	/proc/interrupts: random migrations
MTU mismatch = IP fragmentation	TCP performance drops by when large packets are fragmented.	Wireshark traces confirm

Motivation: Why TCP Stack Tuning is Mandatory

Exascale systems magnify TCP inefficiencies.

Minor inefficiencies like interrupt delays scale exponentially across thousands of nodes.

Linux defaults prioritize general cases, not HPC.

Stock TCP/IP settings are optimized for diverse workloads, not sustained large-message transfers.

Example: 1µs interrupt delay at 400G = 400 bits lost, ~50 packets dropped/queue/second.

Even microsecond-level inefficiencies cause major cumulative throughput loss.

Strategic Tuning Pillars

- NUMA-local CPU affinity
  - Minimizes memory access latencies and maximizes L2/L3 cache locality.
  - Enables high Bandwidth Delay Product (BDP) link saturation.
    - **BDP** is the amount of **data that must be “in flight” on the network link** to fully utilize the available bandwidth.
- Deep hardware queues
  - Absorbs traffic microbursts without packet loss.
- Static IRQ pinning
  - Avoids dynamic jitter from random interrupt migrations.
- Jumbo MTU enforcement
  - Reduces packet overhead and CPU load significantly.

NUMA-Local CPU Affinity: Anchoring Interrupt and Data Locality

Improves Memory Access Latency

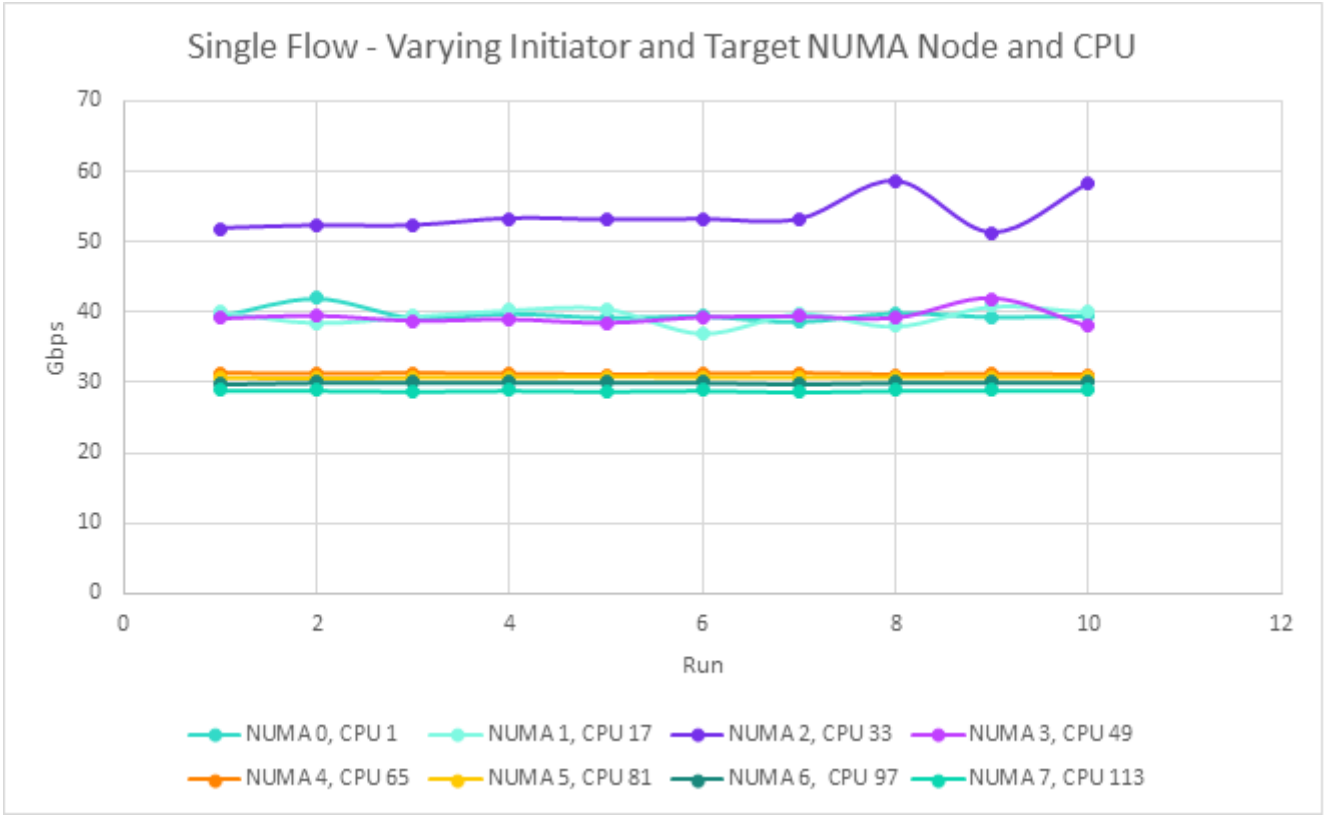
- Local core processing avoids remote memory stalls

Eliminates Cross-Socket Interconnect Penalties

- Avoids traversal delays between NUMA nodes.

Best Applied with Static IRQ Pinning + XPS Mapping

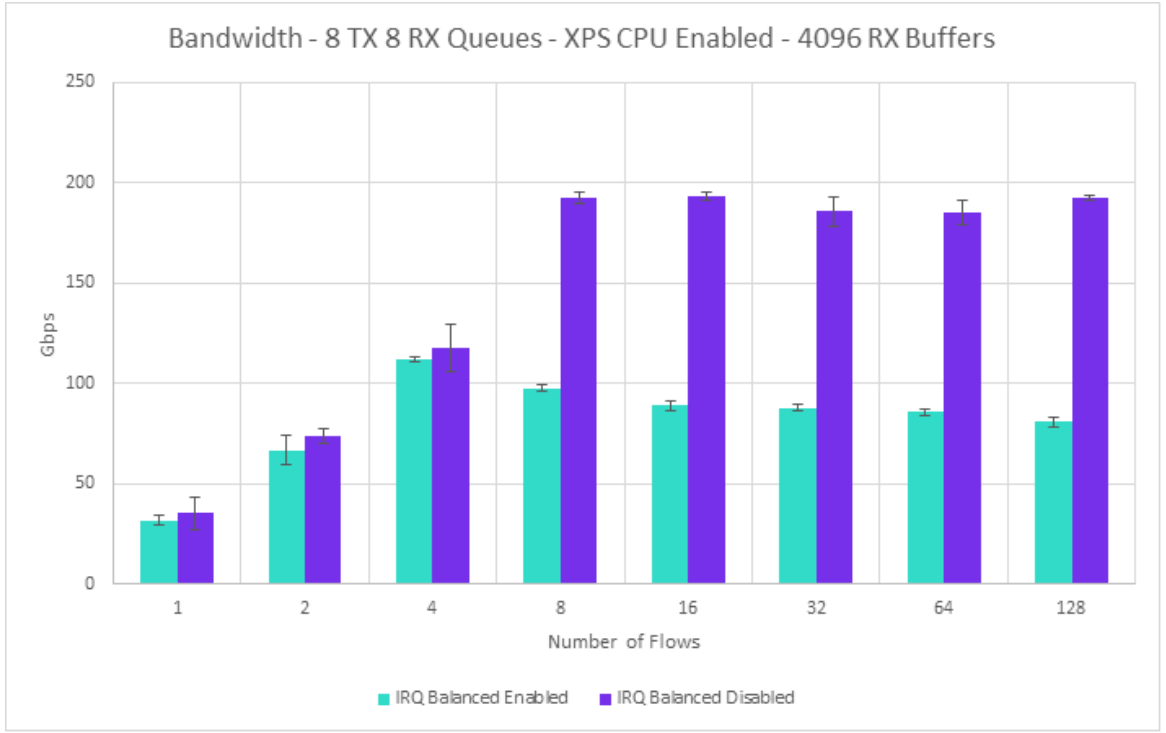
- Guarantees full-stack affinity from NIC to core.



TCP Throughput Before and After Optimization

IRQ Balance Stopped by Design

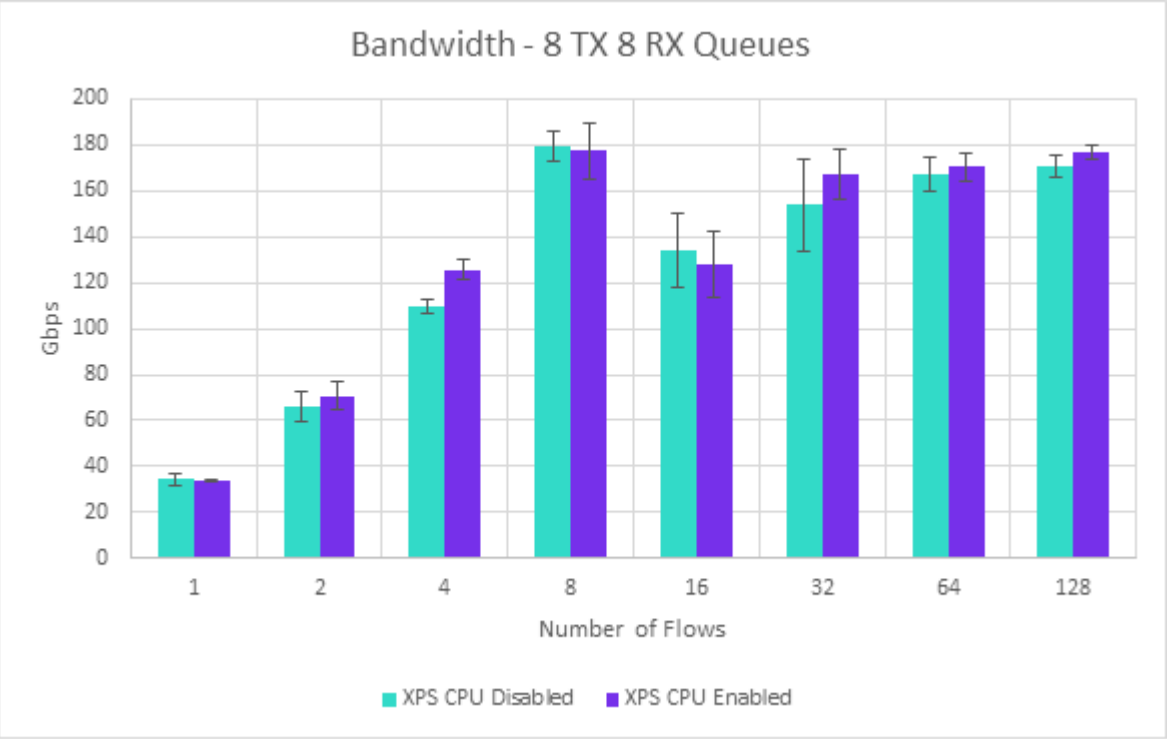
- irqbalance is a Linux daemon that dynamically distributes hardware interrupt requests (IRQs) spreads interrupts across CPUs for general load balancing
  - Ignores NUMA locality, making it unsuitable for high-performance, latency-sensitive environments.
- Static IRQ pinning removes random interrupt migrations.
  - Ensures deterministic interrupt servicing from local CPU to local NIC queues.
- Risks:
  - Interrupt Migration Storms
    - IRQs may **migrate frequently** in response to load, causing performance collapse under loads.
  - Latency Inflation
    - In large-scale TCP workloads, migrating IRQs can inflate interrupt servicing latency, directly impacting flow completion time



TCP Throughput Before and After Optimization

XPS (Transmit Packet Steering) Configuration:
Ensuring TX Queues Affinity to Local NUMA Cores

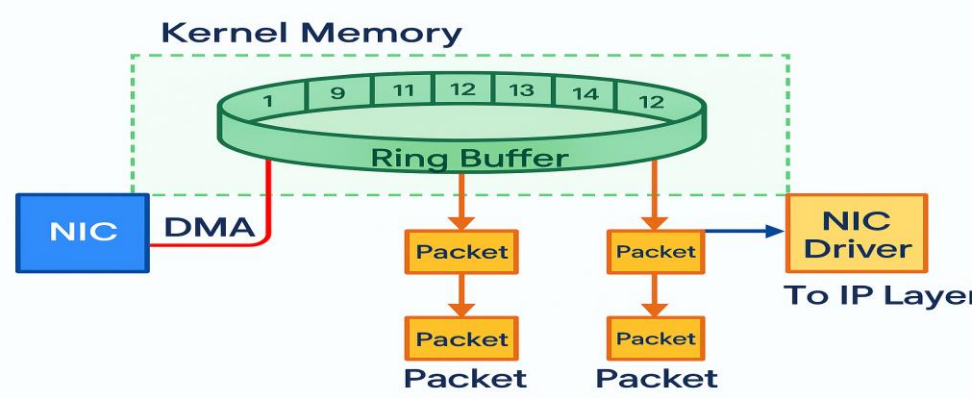
- **XPS (Transmit Packet Steering)** maps NIC TX queues to specific CPU cores, ideally **NUMA-local**, to improve **cache locality** and **reduce TX-side latency**.
- Risks:
  - Cross-NUMA mapping
    - Causes remote memory access which leads to **microsecond-scale latency spikes**.
  - Interference with IRQ pinning
    - If TX affinity (XPS) and IRQ (RX) aren't coordinated, you lose **full-stack locality**, hurting determinism.

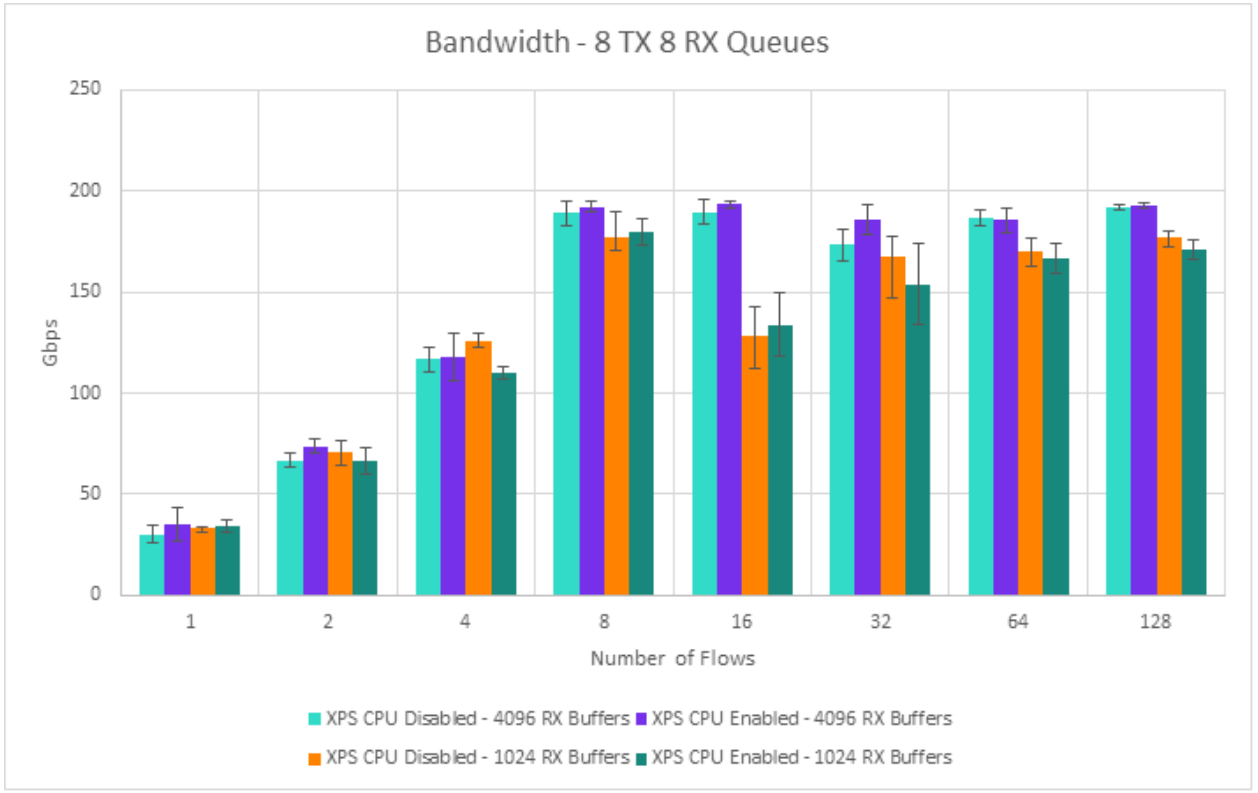


TCP Throughput Before and After Optimization

Increase Ring Buffer size to reduce a high packet drop

- Network ring buffer is a memory region where NIC have direct access (over DMA) to store incoming data.
  - The kernel uses the RX ring buffer to store incoming packets until the device driver can process them.
  - The kernel uses the TX ring buffer to hold outgoing packets which should be sent to the network.
- Deep ring buffers ⟶ absorb congestion microbursts.
  - Allows temporary overloading without immediate packet loss.
- Risks: Too small ⟶ drops; too big ⟶ memory bloat.
  - Buffer sizes must balance latency and loss-resilience.





TCP Throughput Before and After Optimization

Jumbo Frames Enforcement (MTU 9000)

- MTU 9000 is supported on almost all HPC fabrics
- Reduces packet rate 5–6x ⟶ lower CPU IRQ load.
  - Larger frames mean fewer interrupts and system calls per second.
- 400G link: 1500 MTU = 330K PPS vs 9000 MTU = 55K PPS.
  - Directly decreases context switch and interrupt load.
- Risks: MTU mismatch ⟶ blackhole drops, fragmentation.
  - Requires strict network-wide MTU consistency to avoid hidden failures.

Right-Sizing TCP Socket Buffers (rmem/wmem) for Throughput Efficiency

- These TCP socket buffer size parameters used to control how much data Linux can receive(rmem) or send(wmem) over a TCP connection.
- High BDP links require large window sizes.
  - Large windows are necessary to fully utilize high-speed links with even moderate RTTs.
  - For a **100 Gbps** link with a **1 ms** round-trip time (RTT):
    - BDP (bytes) = (100×10^9bits/sec)/8 × 0.001sec = **12.5MB**
- Risks:
  - Small buffers ⟶ window starvation
    - The TCP sender runs out of buffer space before ACKs return (especially on long RTT paths), leading to **idle links**, **lower throughput**, and **increased latency**.
  - Large buffers
    - Large buffer allocations waste system memory and degrade performance in multi-flow environments.

Scalable TCP Performance: Why Determinism Wins at Exascale

- **HPC networks require predictability — not just speed.**
  - At scale, variability in IRQ routing or CPU mapping amplifies jitter and latency.
- **Non-deterministic behavior collapses collective performance.**
  - Random IRQ/XPS placement introduces up to **20–30% goodput loss**.
- **Static IRQ pinning and NUMA-aware XPS restore flow stability.**
  - Results in **consistent per-flow throughput**, and lower latencies.
- **Exascale readiness demands end-to-end path consistency.**
  - NIC ↔ CPU affinity, TX queue steering, and NUMA topology must be coordinated.

Implementation Strategy

- DRY-RUN safe mode available.
  - Provides change preview before applying sensitive kernel and NIC parameters.
- Dynamic NUMA detection per interface.
  - Ensures tuning dynamically adapts to hardware topology.
- Per-device fine-tuning: MTU, queues, buffers, IRQ affinity.
  - Allows granular customization according to device and topology requirements.

NVIDIA Grace Hopper IOMMU Configuration

Balancing GPU compatibility with NIC performance under IOMMU constraints.

1. **Problem: IOMMU default is 'DMA' for all devices**
   - Causes performance penalty for NICs that benefit from 'identity' mapping.
2. **Preferred Fix (Fails on Grace): Kernel Boot Parameter iommu.passthrough=1**
   - Works on Intel/AMD but breaks CUDA/GPU functionality on NVIDIA Grace.
3. **Discovery: NVIDIA discourages SMMU passthrough for GPUs**
   - Official NVIDIA tuning guide warns against passthrough mode.
4. **Conflict: NICs need 'identity' mapping; GPUs need 'DMA'**
   - Can't apply a single setting cluster-wide.
5. **Workaround: Use modprobe wrapper to defer NIC driver load**
   - Wrapper script sets IOMMU type to 'identity' before binding the ethernet driver.

Conclusion

- Default settings fail at exascale.
  - Small defaults are unsustainable at 100G+ fabric scales.
- Our approach delivers >20% better TCP stability, >40% lower CPU cost.
  - Backed by measurements across multiple HPC deployments.
- **Recommended next step:**
  - Grab eth_tuning.sh from https://github.hpe.com/hpe/hpc-shs-utils
    - Controlled pilot → Measure → Rollout cluster-wide.

Glossary

- BDP – Bandwidth-Delay Product
- XPS – Transmit Packet Steering
- IRQ – Interrupt Request
- NUMA – Non-Uniform Memory Access
- IOMMU – Input-Output Memory Management Unit
- DMA – Direct Memory Access
- MTU – Maximum Transmission Unit

References

- RFC 7323: *TCP High Performance Extensions*
  https://datatracker.ietf.org/doc/html/rfc7323
- RFC 1191: *Path MTU Discovery*
  https://datatracker.ietf.org/doc/html/rfc1191
- Red Hat Performance Tuning Guidelines
  https://access.redhat.com/documentation/enus/red_hat_enterprise_linux/8/html/performance_tuning/
- Linux Kernel Documentation on IRQ Affinity & XPS
  https://www.kernel.org/doc/Documentation/networking/scaling.txt
- IEEE 802.3x: Ethernet Flow Control
  https://standards.ieee.org/standard/802_3x-1997.html

Thank you

ravi.bissa@hpe.com, ian.ziemba@hpe.com, duncan.roweth@hpe.com