MVAPICH USER GROUP CONFERENCE - 2026

From Transport to Collectives

Building an End-to-End Communication Stack for AI Supercomputers with MRC and MSCCL++

Networking transport and collective-communication software for next-generation AI training and inference at extreme scale.

Dr. Jithin Jose

Partner Software Engg. Manager, Microsoft

AGENDA

Two technologies, one goal: extreme-scale AI communication

PART 1 · TRANSPORT

MRC — Multipath Reliable Connection

A resilient, load-balanced RDMA transport for multi-plane Ethernet AI clusters.

PART 2 · COLLECTIVES

MSCCL++ — Programmable Collective Communication

Layered abstractions for building high-performance, portable collective kernels.

BRINGING IT TOGETHER

An End-to-End Communication Stack

How transport and collectives co-design to serve modern AI training & inference.

LOOKING AHEAD

Open Problems & Where This Is Headed

Frontiers in transport-aware collective communication at extreme scale.

THE PROBLEM

Communication is one of the key bottlenecks in AI supercomputers

Synchronous pretraining runs in lock-step across up to 100,000+ GPUs. Every communication round is only as fast as the **slowest transfer** — and every collective op competes for GPU communication bandwidth that is increasingly the bottleneck in LLM training and inference.

100K+

GPUs in a single synchronous training job

10–40%

of LLM inference time spent in GPU communication kernels

180K

switches studied — 17% of failures caused by software bugs

Any solution must do three things at once:

Load-balance evenly to prevent flow-collision congestion

Absorb incast without creating latency outliers

Handle link & fabric failures gracefully — without stopping the job

PART ONE

MRC

Multipath Reliable Connection

A resilient RDMA transport that sprays, load-balances, and self-heals across multi-plane AI training networks — at 100,000+ GPU scale.

A cross-industry collaboration with OpenAI, NVIDIA, Broadcom, AMD, and Intel

Why conventional transport breaks down at scale

As networks scale past 100,000 GPUs, communication becomes **increasingly outlier-dominated** — and network failures grow more frequent with scale. Small operations teams must manage many thousands of switches, without relying on human intervention for every failure.

- **Flow collisions**

  Single-path transports hash entire flows onto one path — collisions stall whole collectives.

- **Incast congestion**

  Lossless Ethernet's PFC creates head-of-line blocking between unrelated collectives.

- **Silent + noisy failures**

  Link flaps and switch software bugs (17% of failures) must be tolerated automatically, not escalated.

MRC's approach

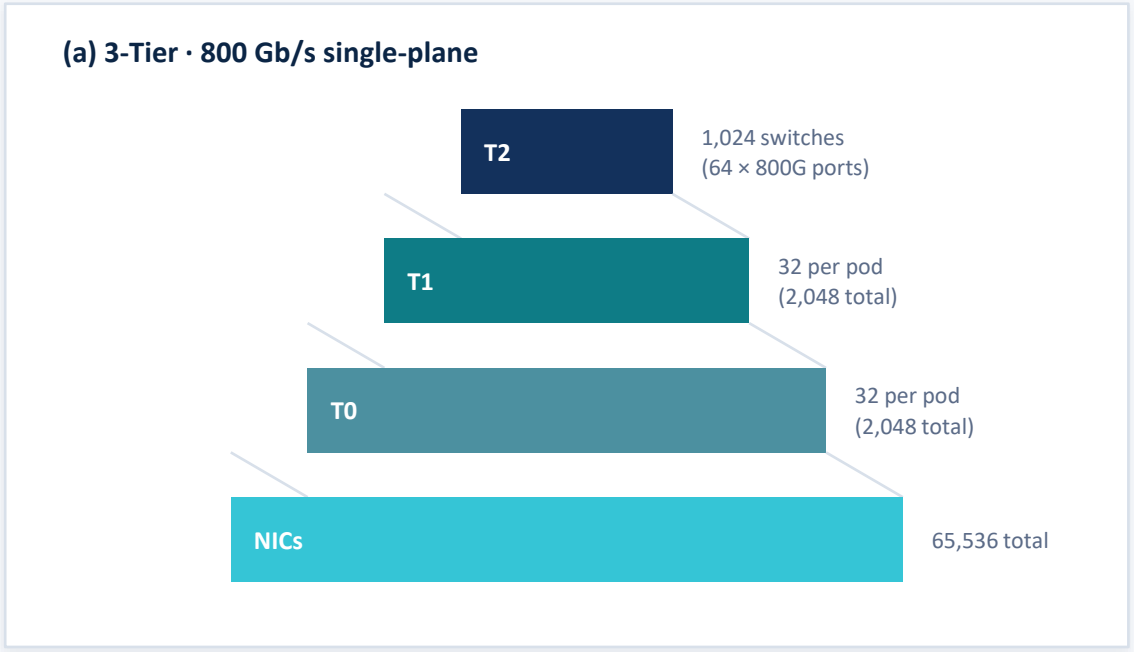
Extend RoCE Reliable Connection with multipath, self-healing behavior:

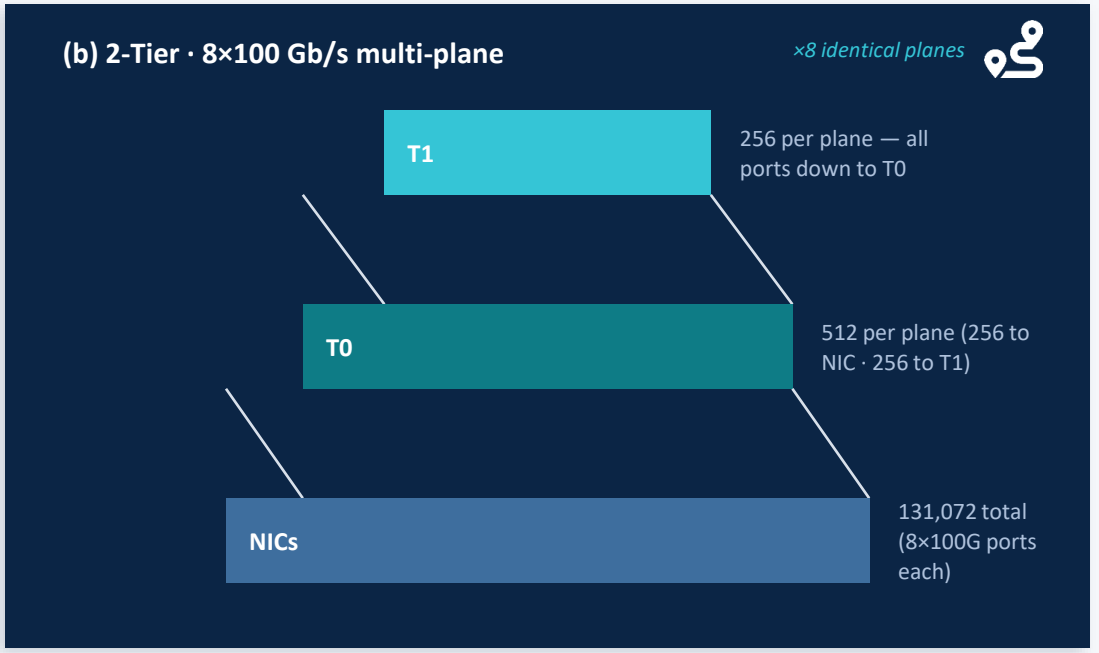
- **Packet spraying** across every path, on every plane
- **Adaptive load balancing** using ECN as a live congestion signal
- **Selective retransmission** with SACK/NACK — no head-of-line stalls
- **Packet trimming** to distinguish congestion loss from failure

"Like UET, MRC employs packet spraying, adaptive load balancing based on ECN, out-of-order placement, and packet trimming."

Co-designing the network: multi-plane over 3-tier Clos

Splitting an 800 Gb/s NIC into 8×100 Gb/s ports builds 8 parallel Clos planes on the same switch silicon — reaching 131K GPUs with only two switch tiers.





64 pods · up to 5 switch hops · needs a 4th tier (or oversubscription) for 100K GPUs

2 tiers only · max 3 switch hops · same 65,536 physical NICs, 8 ports each

2/3
the optics needed

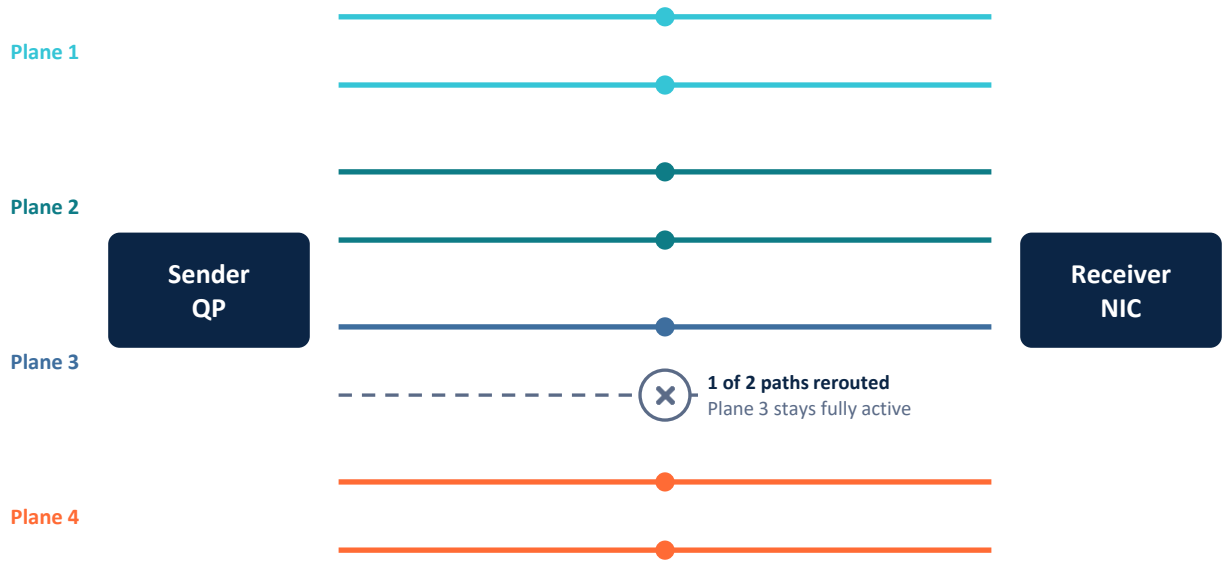
3/5
the switch count

0.4%
capacity lost per T0-T1 link (vs 3%)

256
nodes reachable in one hop (vs 32)

Entropy values: one QP, hundreds of live paths

At QP startup, MRC generates an **entropy value (EV) set** of ~128–256 entries. Each packet rotates to a new EV, spraying traffic evenly across every path on every plane, with no application awareness needed.



EV set: ~128–256 paths sprayed across all planes

Reliability innovations

No PFC — lossy Ethernet

Best-effort mode avoids head-of-line blocking between collectives.

Selective retransmission

SACK packets pinpoint exactly which packets arrived; only losses are resent.

Packet trimming

Congested packets are trimmed & priority-forwarded, triggering fast NACK-based recovery.

ECN as load signal

Congestion echoes steer senders away from busier paths in real time.

Static SRv6 source routing — on purpose

Dynamic routing (e.g., BGP) fighting MRC's own path avoidance makes network behavior harder to reason about. So MRC disables dynamic routing entirely and instead uses **IPv6 segment routing (SRv6)** with static routes — letting MRC alone own all failure handling.

uN micro-segment (uSID) forwarding

switch left-shifts uSID by 16 bits at each hop, then forwards on the static table

SRv6 prober: walks every static path end-to-end detecting faulty paths.

Why static beats dynamic here

No control-plane ambiguity

MRC removes a path the instant packets stop flowing — no waiting on BGP convergence.

Ground-truth telemetry

MRC/EV Telemetry gives instant health signals — pinpointing NIC–T0 vs T0–T1 faults.

Dedicated SRv6 prober

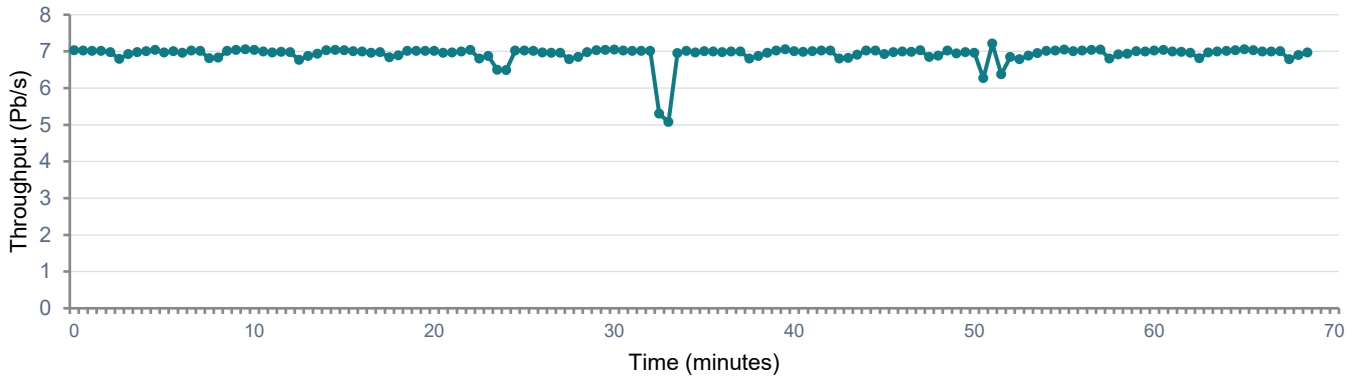
A data-plane prober walks every static path end-to-end for millisecond-frequency health checks.

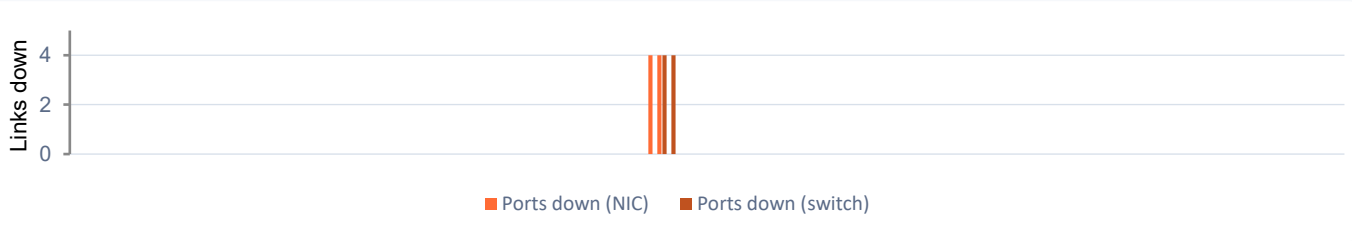
"When we see a T1 switch that is not happy, we simply reboot it — without concern for routing convergence."

Failures happen constantly. Jobs keep training.

Running on OpenAI and Microsoft's largest training clusters — used to train frontier LLMs for ChatGPT and Codex.

Job throughput during a flapping NIC–T0 switch transceiver





Real production telemetry from Cluster A (50K-GPU pretraining job, Nvidia CX-8 NICs). An optical transceiver on a T0 switch glitched, flapping all four of its links in rapid succession across three active nodes. Throughput dipped ~25% for about a minute (min. 32–33), then recovered immediately. (Courtesy: OAI)

25%

throughput dip during the NIC-T0 transceiver glitch above — full recovery within a minute, no QP failure

~2 min

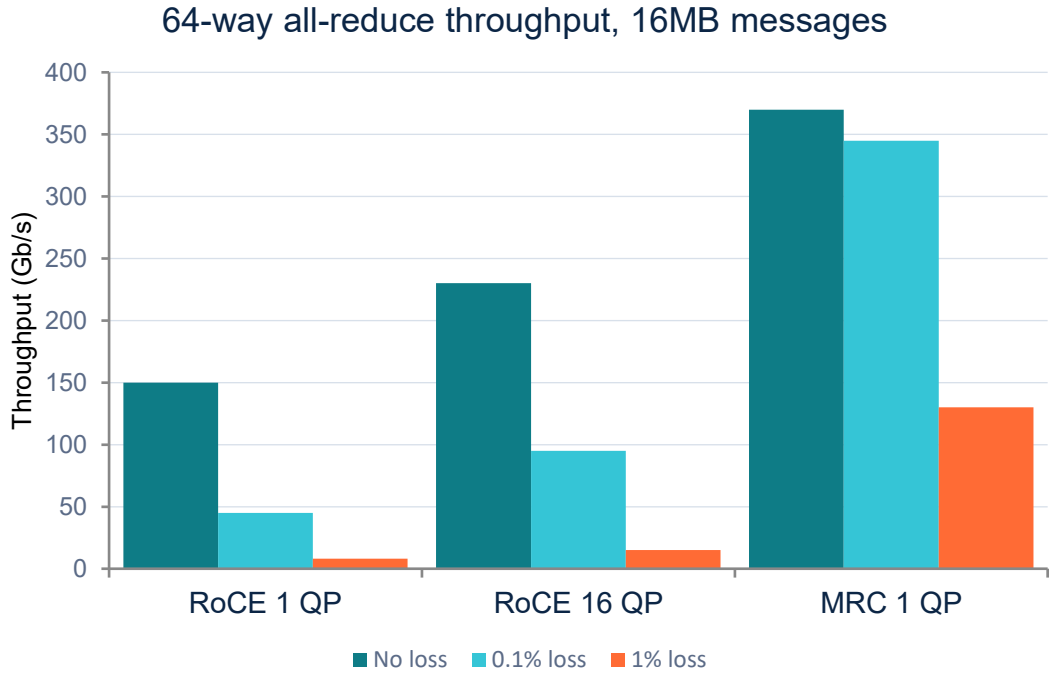
for a T1 switch to fully resume forwarding after an automated reboot — job barely dipped

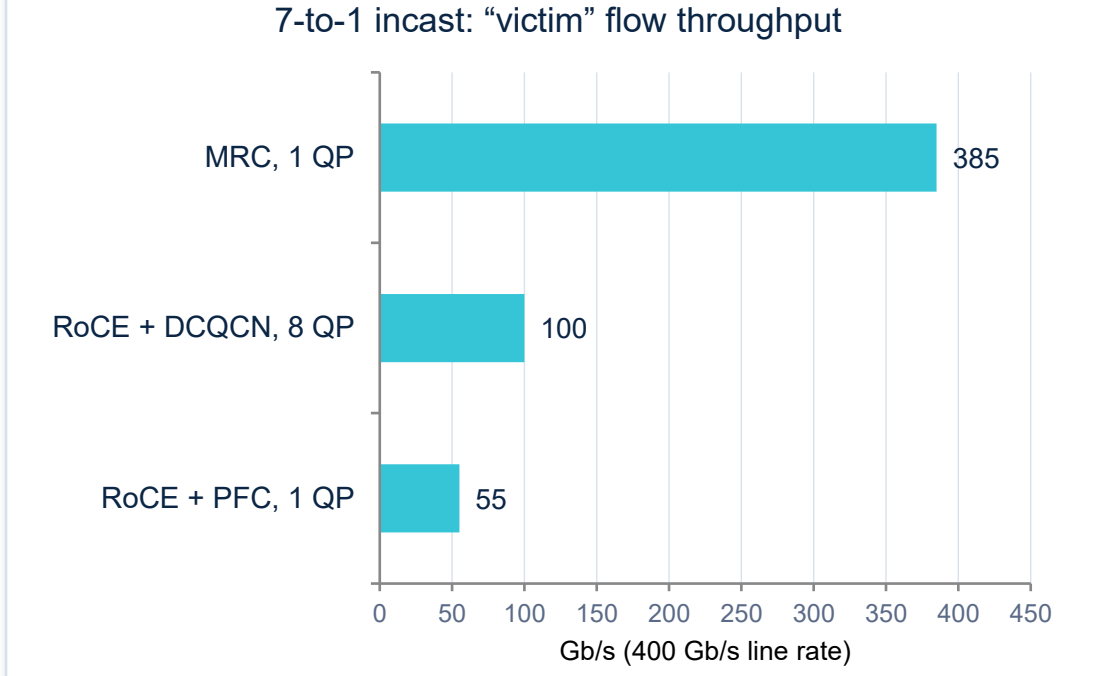
5.09 / 6.54 µs

T0-local vs cross-T1 latency — both reach ~770 Gb/s (96% of peak) bandwidth

MRC vs. RoCEv2: load balancing that actually holds up

64-way ring all-reduce, large messages, on identical GPUs/NICs/switches — RoCE tuned with QP scaling, MRC using a single QP.





One MRC QP spraying across ~256 paths outperforms 16 scaled RoCE QPs — and stays usable under loss where RoCE collapses.(Courtesy: AMD)

Lossless PFC lets incast congestion spread to unrelated flows. MRC's spraying nearly eliminates this “collateral damage.” (Courtesy: Broadcom)

PART TWO

MSCCL++

Rethinking GPU Communication Abstractions

A layered communication stack — Primitive, DSL, and Collective APIs — giving experts and application developers the same portable foundation.

ASPLOS '26 paper: arxiv.org/html/2504.09014

General-purpose libraries can't keep up with hardware

GPU communication kernels account for **10–40%** of many real-world LLM inference workloads. Practitioners keep writing custom communication code from scratch — e.g. TensorRT-LLM's own AllReduce — rather than waiting for NCCL to catch up.

Workload-specific tuning

Peak performance needs latency/bandwidth trade-offs tailored per message size — general libraries can't optimize for all of them.

Compute-comm coupling

Collectives like AllReduce include reductions; performance depends on overlapping that compute with transfer.

Rapid hardware evolution

New chip & interconnect features arrive faster than general libraries like NCCL can adopt them.

Result: writing custom communication code is powerful but error-prone, non-portable, and expensive to maintain.

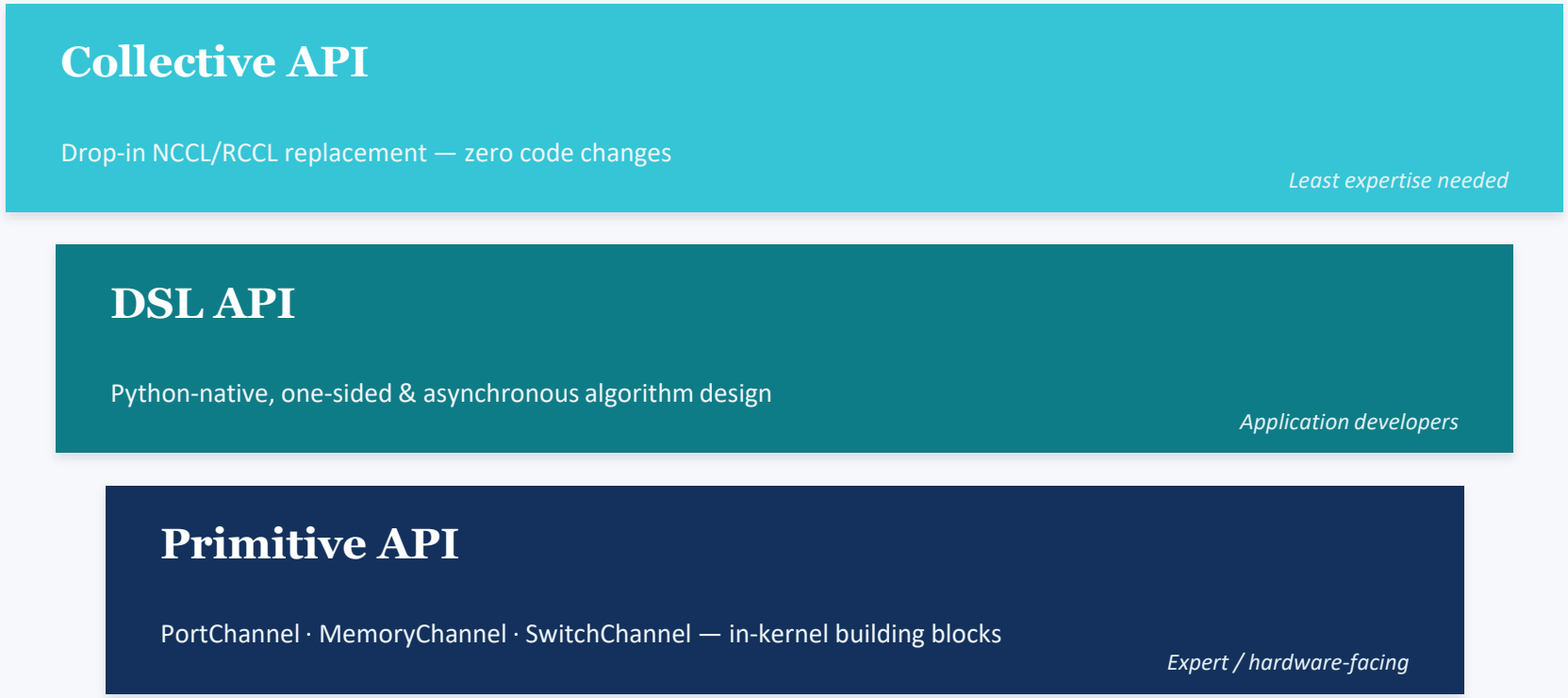
Where NCCL's primitives fall short

- Synchronous send/recv wastes GPU cycles in busy-wait loops
- Local memory copies through internal buffers add overhead
- Hard-codes one transfer mode per link (no fine-grained choice)
- Limited control over reduction/copy parallelism

MSCCL++'s answer: separate performance-preserving primitives from productivity — without giving up either.

Three layers, one separation of concerns

The closer to hardware, the more control — and the more expertise required. MSCCL++ lets every user pick their layer.



Why layering wins

- Zero register spills, no extra copies
- Co-optimize compute + communication
- Portable across GPU vendors & generations
- Progressive customization, layer by layer

Three channels for three transfer modes

Zero-copy, one-sided, asynchronous primitives — close to hardware, portable across vendors.

PortChannel

Port-mapped I/O over dedicated DMA/RDMA hardware (e.g. ibv_post_send). Best for cross-node & CPU-offloaded transfers.

put · signal · wait · flush

MemoryChannel

Direct peer-to-peer GPU memory access. HB protocol for bandwidth, LL protocol for latency-sensitive small transfers.

put · read · write · signal

SwitchChannel

Switch-based multicast/aggregation (e.g. NVLink SHARP). Hardware performs reduce & broadcast in the network.

reduce · broadcast

Fused primitives (e.g. put_with_signal, reduce_put) cut API-call overhead and avoid unnecessary memory round-trips.

Write the algorithm. Let overlap happen automatically.

The Python-native DSL exposes a **global view** across all GPUs, retains one-sided/asynchronous semantics, and lowers to an execution plan — with automatic synchronization and operation fusion.

ring_reduce_scatter.py

```
def ringRS(portChannels, bufs, sz, N, tb=0):
    for rank in range(N):
        putChan = portChannels[(rank+1)%N]
        for step in range(N):
            beg, mid, end = chunk(step)
            # (a) send 1st half — overlaps with (b)
            putChan.put(dst[beg:mid], src[beg:mid], tb)
            putChan.signal(tb)
            if step != 0:
                src[pmid:pend].reduce(recv[pmid:pend], tb)
            recvChan.wait(tb)   # 1st half arrived
            # (b) send 2nd half — overlaps with reduce
            putChan.put(dst[mid:end], src[mid:end], tb)
            src[beg:mid].reduce(recv[beg:mid], tb)
```

What the DSL Executor handles for you

Data-dependence analysis

tracks last-writer & readers per chunk, inserts sync only where needed

Operation fusion

merges contiguous ops (e.g. reduce + put) into one, cutting memory round-trips

Compute/comm overlap

splits chunks so reduction of one half overlaps transfer of the other

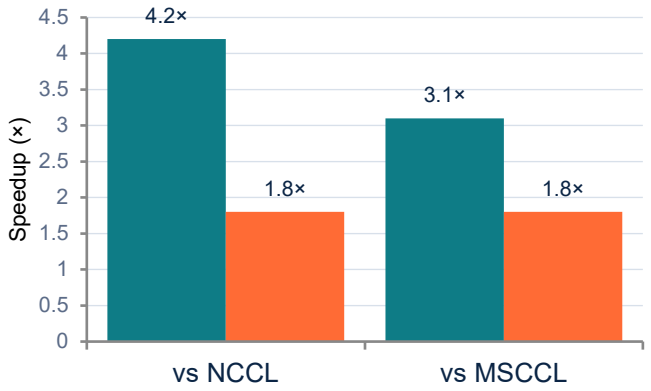
Portability

same DSL program compiles to any hardware the Primitive API supports

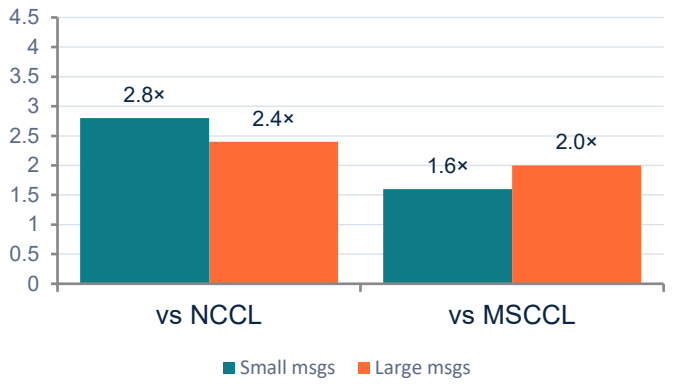
Faster collectives, across every hardware generation

Geomean speedup: 1.7× (up to 5.4×) for collective communication vs. state-of-the-art baselines. *Bars show MSCCL++'s speedup over each baseline library, on the platform named above each chart (higher = MSCCL++ faster).*

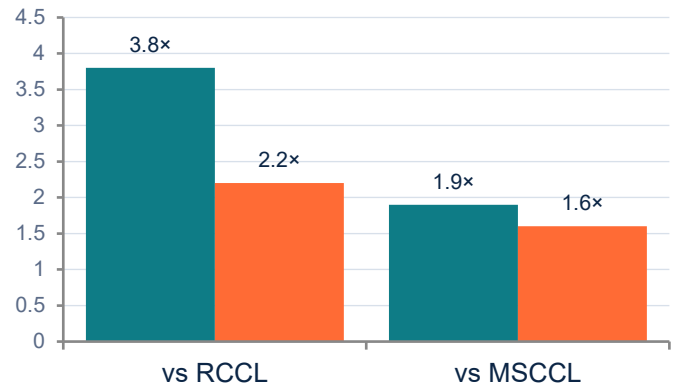
NVIDIA A100



NVIDIA H100



AMD MI300x



47% latency cut for 1KB single-node messages (9.5µs → 5.0µs) vs MSCCL

15 lines of DSL code to drive NVLink SHARP hardware reduction via SwitchChannel

1.99× / 2.08× average AllReduce/AllGather speedup vs NCCL / RCCL respectively

Speedups that reach production LLM inference

Geomean 1.2× (up to 1.38×) end-to-end AI inference speedup — already adopted by SGLang and RCCL.

1.11×

avg decode latency speedup

Llama3-70B on vLLM, tensor-parallel-8, single A100-80G node, replacing NCCL AllReduce with MSCCL++.

1.31×

avg decode throughput speedup

DeepSeek-V3 on SGLang across two H100 nodes (16 GPUs) with tensor parallelism = 16.

≈ same perf

as IBGDA/NVSHMEM, fully portable

DeepEP expert-parallel dispatch/combine re-implemented on PortChannel — no NVIDIA-only stack required.

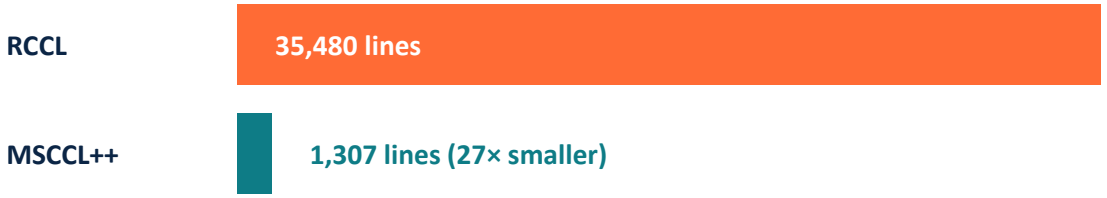
Also handles the hard cases:

Mixture-of-Experts dispatch (DeepEP) • communication-computation overlap • large-scale inference serving • cross-vendor NVIDIA + AMD deployments

New hardware, weeks not quarters

The layered design pays off in engineering velocity — not just runtime performance.

Lines changed to add AMD MI300x support



AMD-specific code (excluding algorithms) is fewer than 10 lines — because MSCCL++'s Primitive API is a shallow, portable abstraction over CUDA/HIP.

Engineering velocity

7 weeks

full AMD MI300x support (1 developer): 3 weeks bring-up + 4 weeks new AllReduce algorithms

2 person-weeks

to add NVIDIA multi-node NVLink support

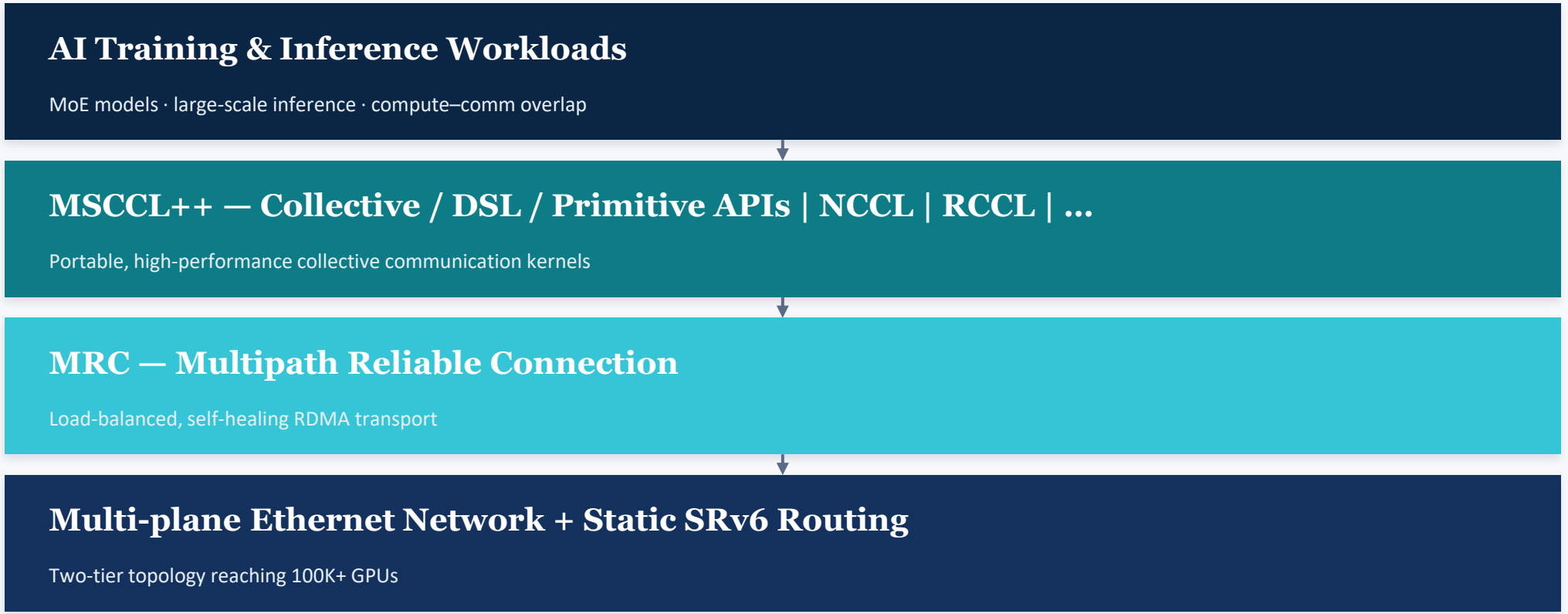
8 dev-weeks

for two developers to build SwitchChannel for NVLink SHARP multimem — 2.2×+ speedup result

BRINGING IT TOGETHER

Transport and collectives, co-designed end to end

MRC guarantees the network stays saturated and resilient; MSCCL++ turns that raw bandwidth into fast, overlap-friendly collectives — together they carry training and inference traffic at extreme scale.



LOOKING AHEAD

The open problems are what make this exciting

Both stacks are open source and still evolving — plenty of hard, high-impact systems problems remain.

Training at MRC scale

MSCCL++ has focused on inference; extending its overlap-friendly algorithms to large-scale synchronous training is open.

Smarter path synthesis

Automating EV-set and algorithm selection (in the spirit of TACCL / TE-CCL) instead of offline profiling.

Beyond GPUs

Generalizing MSCCL++'s I/O abstractions to non-GPU accelerators and storage transports.

New silicon, fast

Keeping pace as switch radix, NIC speeds, and multimem-style features keep evolving hardware every year.

If building the communication backbone for the next generation of AI supercomputers sounds like your kind of problem — let's talk.

Thank You

Questions & discussion welcome

PAPERS & SPECS

ASPLOS '26 — MSCCL++ paper
arxiv.org/html/2504.09014

Hot Interconnects '26 — MRC paper
hoti.org/2026/program.html

MRC Specification v1.0
Open Compute Project

OPEN SOURCE

MSCCL++
github.com/microsoft/mscclpp

MRC NCCL plugin
github.com/microsoft/mrc-nccl-plugin

MRC verbs shim library
github.com/microsoft/mrc-verbs-shim-lib

jijos@microsoft.com
Jithin Jose · Microsoft