TECHNICAL WHITE PAPER · MCCL

mccl: A Measured-Topology Collectives Library for Apple Silicon Clusters

The layer NCCL occupies, for clusters assembled from the machines and cables already in the room: five collectives over a planner that solves ring, tree and hierarchical from probed bandwidth and latency; per-pair addressing that makes a Thunderbolt island and a Wi-Fi bridge one world; vectorised wire compression under a measured crossover rule; and an RDMA-over-Thunderbolt-5 transport built to Apple's TN3205.

NCCL-IDIOM C ABI | MEASURED TOPOLOGY | RING / TREE / HIERARCHICAL | WIRE COMPRESSION | RDMA · TN3205 | 379 TESTS · 87.06%

ABSTRACT

Small clusters of Apple silicon machines are a plausible substrate for distributed inference and fine-tuning, but they have no collectives library. mccl is an NCCL-shaped library for that layer: all five collectives over a planner that measures the fabric rather than assuming it — pairwise bandwidth and minimum latency from a real probe, a closed-form ring/tree crossover, ratio-gap island detection naming no link speed in the code. Because the fabric is whatever cables are in the room, each *pair* of ranks dials its own path, which is what lets a Thunderbolt island and a Wi-Fi bridge rank form one world at all. In-band lossy compression — an axis NCCL has no equivalent of — is governed by a measured rule: a codec pays only when the fabric's uncompressed all-reduce rate is below that codec's own encode/decode ceiling. What was measured at two and three nodes is reported alongside what was not, including where Apple's own JACCL is the better choice.

01 Introduction

The hardware case for a Mac cluster is straightforward: unified memory addressable by both CPU and GPU without a copy, large capacity per dollar, and a Thunderbolt link an order of magnitude faster than Ethernet. The software case is where it falls apart. On NVIDIA hardware an application links NCCL [6]; on Apple silicon there is no equivalent layer — MLX ships a ring all-reduce inside its own framework, and anything else writes its own or does not distribute.

On a DGX box the fabric is NVLink and the library's job is to keep it saturated; on a Mac cluster that is inverted — a Thunderbolt link measures roughly 1 GB/s here, against an M1 Max memory bandwidth two orders of magnitude above it. Any all-reduce large enough to matter is bounded by the wire, so the leverage is in *sending fewer bytes*, which an application-level ring cannot do for the application. mccl rests on three positions: **measure the fabric rather than assume it**; **solve the crossover rather than tune it**; and **refuse sparsification where its correctness argument does not hold** (§3.4).

This is not a claim that mccl is the fastest way to move bytes between two Apple machines. Apple has since shipped RDMA over Thunderbolt 5 [11] and, alongside it, JACCL — a collectives library co-developed with the hardware and validated on it. On a uniform Thunderbolt 5 mesh JACCL is the better choice, and §5.3 says so without hedging. What survives is narrower: mixed fabrics, which JACCL structurally cannot reach; in-band lossy compression, which it does not do; and hardware older than Thunderbolt 5. A C ABI and automatic algorithm selection look like differentiators and are not.

02 Scientific Background

2.1 The Cost Model for a Collective

Collective communication is analysed under Hockney's linear cost model [14]: moving *S* bytes costs α + S/B. It ignores any dependence of B on S — the failure mode observed in §6 — but it yields the standard portfolio [15] and the rule [3] that *different algorithms win at different message sizes*.

2.2 Ring and Tree

The bandwidth-optimal all-reduce is the ring [1], in Rabenseifner's reduce-scatter-then-all-gather form [2]: every link carries (n−1)/n of the payload in each phase — the minimum for the operation — at 2(n−1) dependent hops. The binomial tree inverts the trade: $2\cdot\log_2(n)$ hops, but every level carries the *whole* message. Under the linear model:

$$t_{ring} = 2(n-1)\cdot\alpha + 2\cdot(n-1)/n\cdot S/B \quad \cdot \quad t_{tree} = 2\cdot\log_2(n)\cdot\alpha + 2\cdot\log_2(n)\cdot S/B$$

Equating them and solving for S gives a crossover in closed form:

$$S* = \alpha\cdot B\cdot(\log_2 n - (n-1)) \;/\; ((n-1)/n - \log_2 n)$$

Both factors are ≤ 0 for n ≥ 2, so S* ≥ 0. At n = 2 the numerator vanishes and S* = 0 — correct rather than degenerate, since a two-rank tree *is* a ring. This expression is the whole of mccl's size-based algorithm selection.

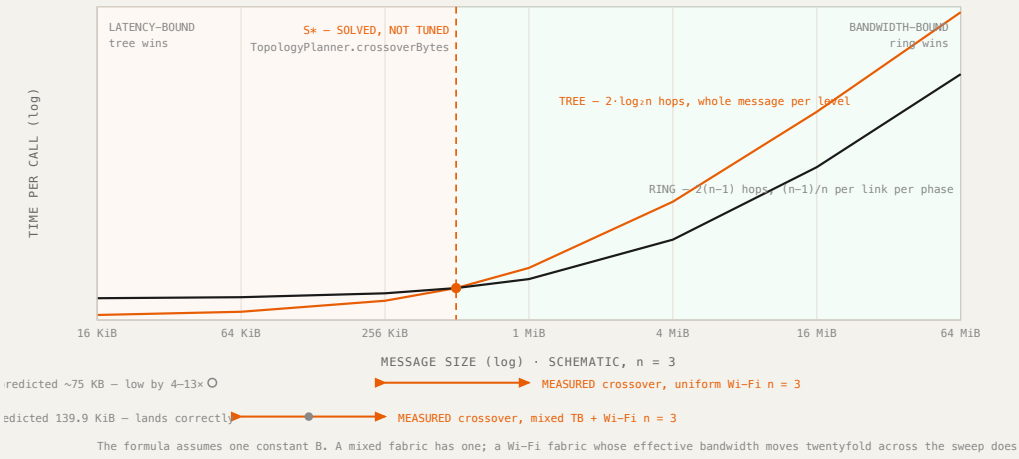

Figure 1. The ring/tree crossover, and where the closed form does and does not locate it; the two strips below the axis are measured. On uniform Wi-Fi at n = 3 it predicts ~75 KB against a measured 256 KiB–1 MiB — low by 4–13×, since it assumes a single B while that fabric's effective bandwidth climbs twentyfold across the sweep. On the mixed fabric of §3.2 it predicts 139.9 KiB against a measured 64–256 KiB.

2.3 Bus Bandwidth as the Reporting Metric

Payload bytes over wall time understates what the fabric did, so mccl reports NCCL's *bus bandwidth*: the algorithmic figure scaled by the traffic each link actually carries, 2(n–1)/n for all-reduce [6].

2.4 Ring All-Reduce in Deep Learning

The ring's migration from HPC into deep learning is well documented [4], [5]; what matters here is the interface it left behind — the `ncclComm_t` / `ncclUniqueId` bring-up idiom, the collective signatures, busbw.

2.5 Gradient Compression and Error Feedback

Because data-parallel communication volume is fixed by parameter count rather than batch size, gradient compression has a substantial literature: blockwise quantisation in the style of QSGD [7], and sparsification [8], which converges only under *error feedback* — what is not sent is retained as a residual and added into the next call's buffer [9], [10]. The argument turns on one invariant, *every unit of mass not sent is retained by some rank's residual*, which §3.4 shows a per-hop top-k destroys.

2.6 What the Fabric Costs Before Any of This Runs

Every measurement here was taken over TCP/IP, Thunderbolt reached as IP over the Thunderbolt bridge, and that stack is not free. Two probes of the same cable between the same two machines read 1.06 GB/s and 1.46 GB/s against a 20 Gb/s (2.5 GB/s) negotiated line rate — 42.4% and 58.4% — the difference being which of a machine's three Thunderbolt ports, all sharing one `169.254/16` prefix, the dialing end selected. The honest statement of the tax is a 42–58% *range* rather than a point, and every efficiency fraction below is against the probe taken alongside its own sweep. A queue pair does not traverse that stack, which is the strongest argument for the RDMA transport of §3.7 and the reason its lack of hardware evidence matters.

03 System Architecture

3.1 Layering and the Transport Seam

mccl is 5,358 lines of Swift in the core library, 955 in the C shim and header, and 1,857 in the benchmark harness and CLIs, with **no external package dependencies**. The layering is `Collectives → Planner → Wire → Transport`; the seam that carries the design is the `Transport` protocol, under which `TCPTransport`, an in-process `LoopbackTransport` and `RDMATransport` (§3.7) sit — the last added with *nothing above it changed*.

`MeshFabric` builds a full rank-addressed mesh rather than only ring neighbours, since tree and hierarchical plans need non-adjacent edges. Each channel has distinct send and receive queues, so a rank sends and receives at once — and at n = 2 that full-duplex overlap lets the ring beat the binomial tree by 1.72× at 16 MiB on identical total bytes.

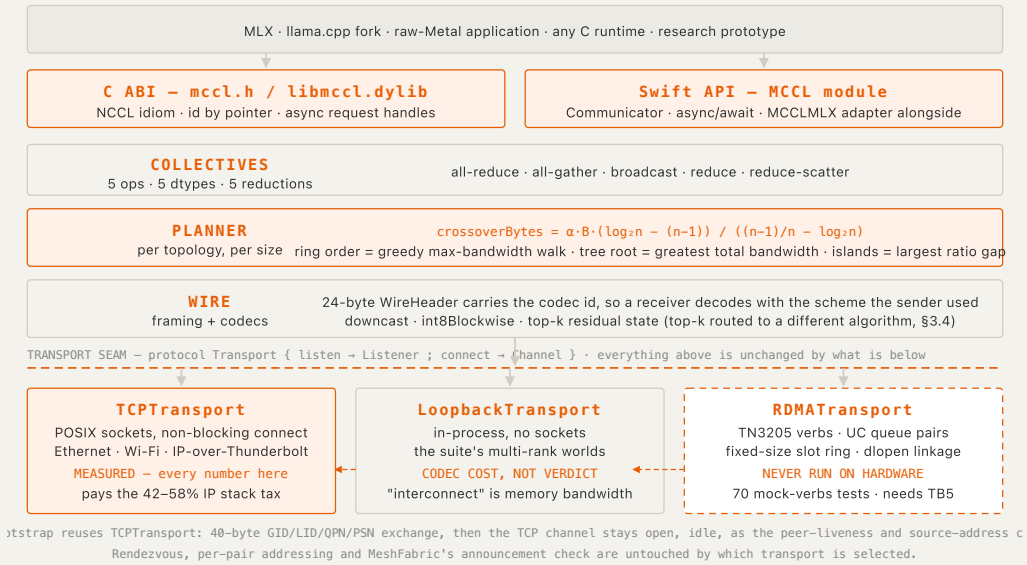

Figure 2. The stack and the transport seam. Three transports with entirely different failure models — a reliable byte stream, a memcpy, and an unreliable-connection queue pair — present one `Channel` to everything above. The RDMA box is dashed because it has never moved a byte over real hardware.

3.2 The Measured-Topology Planner

mccl plans over measurements. `mcclprobe measure` drives real transfers against a server on each node — bandwidth from large streaming transfers after a warm-up chunk, latency as the *minimum* over many 8-byte ping-pongs — and a serving node probes a third node on request, so the map is a genuine pairwise mesh. From it: the ring *order* is a greedy maximum-bandwidth walk, the tree *root* the rank with the greatest total measured bandwidth, and the crossover §2.2's closed form on measured α and B — 0 bytes at n = 2, 139.0 KiB at n = 4, 623.9 KiB at n = 16. No byte constant goes stale when the fabric changes.

Island detection. A flat ring on a mixed-speed fabric routes the whole ring's traffic across the slow bridge; the hierarchical plan rings inside each fast island, rings between island leaders, and broadcasts back down a tree, so the bridge carries one island-sum per island. Finding the islands runs under a deliberate constraint — *no link speed may be named in the code*. mccl sorts the distinct measured bandwidths, cuts at the geometric mean of the pair straddling the largest *ratio* gap, and takes connected components above the cut, after gating on the fastest link being at least 3× the slowest. Gate and gap do different jobs: a four-node loopback map spreads 3.74× and passes the gate, but its largest adjacent gap is only 1.84×, so mccl reports one noisy fabric rather than inventing a hierarchy.

The `Kind` label on a link is cosmetic by design — taken from the interface the peer is reached on, never read by the planner. It buys honesty in operator-facing output: a congested Thunderbolt bridge at 900 MB/s is still Thunderbolt.

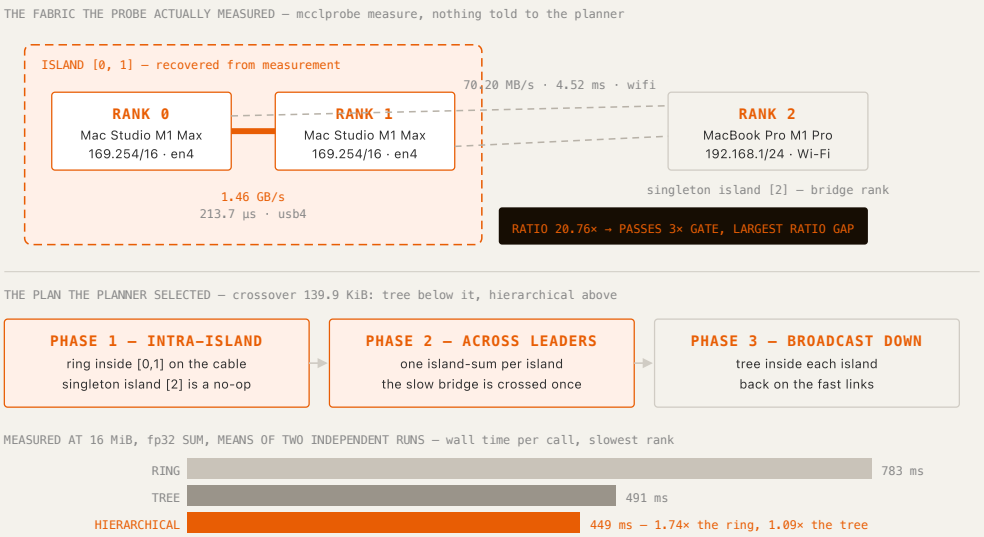

Figure 3. The mixed fabric that was measured, and the plan derived from it. The 20.76× ratio clears the 3× gate and is the largest adjacent ratio gap, so island detection recovers `[0,1]` `[2]` from its own probe with nothing told to it. The hierarchical plan wins in both runs at every size from 256 KiB to 16 MiB; at 64 MiB, on one run, the tree takes it back by 8%.

3.3 Per-Hop Wire Compression

Compression is per hop and never changes the dtype the caller sees; the codec id travels in the frame header, so a receiver decodes with the scheme the sender used.

Scheme	Wire bytes for *n* fp32 elements	Error bound
`downcast`	`2n`	relative ≤ 2^{-11}
`int8Blockwise(b)`	`4⌈n/b⌉ + n`	absolute ≤ blockAbsmax / 254
`topK(f)`	`4 + k(4+4)` per rank, `k = ⌈fn⌉`	unbiased over repeated calls

For a 64 MiB fp32 all-reduce over eight ranks a dense ring puts 112.0 MiB across each link; `downcast` halves that, `int8/256` reduces it to 28.4 MiB, `topk/0.01` to 10.2 MiB per rank. Those reductions are verified at interface byte counters rather than asserted — **2.00×, 3.94× and 49.9×** fewer bytes over five 64 MiB collectives. A scheme that cannot help a dtype degrades to raw rather than erroring.

The rule that decides whether to use it at all came from a failure: on the Thunderbolt cable every scalar codec *lost* above 256 KiB, and the compressed columns were flat in message size — they measured the encoder, not the cable, each plateauing at its own ceiling below what the cable delivered uncompressed. That yields:

THE COMPRESSION CROSSOVER RULE

A codec pays only when the fabric's uncompressed all-reduce rate is below that codec's own encode/decode ceiling.

Both quantities are measurable on the machine in front of you — the fabric's rate with `mcclbench`, the codec's ceiling with `mcclbench --codec-bench`. The rule names no hardware and no particular codec. NCCL has no in-band lossy compression, so this is the one axis with nothing to compare against.

The rule is falsifiable in both directions, and vectorising the encoders tested it: `CodecKernels` reimplements the same wire formats byte-for-byte. Two codecs' ceilings cleared the 0.53–0.66 GB/s the cable delivers uncompressed and both flipped from loss to win; the third's rose by a genuine 1.9×, stayed below that rate, and stayed a loss.

Codec	Ceiling, scalar	Ceiling, vectorised	TB verdict, scalar	TB verdict, vectorised
`downcast`	0.89 GB/s	**10.79 GB/s**	0.79–1.04× (loses)	**1.07–1.53× (wins)**
`int8/256`	0.21 GB/s	**1.69 GB/s**	0.51–0.80× (loses)	**1.27–1.71× (wins)**
`topK/0.01`	0.19 GB/s	0.36 GB/s	0.69–1.25× (loses)	0.72–1.15× (still loses)

Ceilings are round-trip payload GB/s on the rank that gates the collective — a machine shared with an unrelated workload throughout, which is why its figures sit below the same kernels on an idle M1 Max (3.67 → 31.80 GB/s for `downcast`). The rule survived that correction; the constants first attached to it did not. Past the crossover only a codec's *ratio* matters — `int8/256` costs six times the CPU of `downcast` and beats it. On the Wi-Fi path, at 0.052 GB/s, every codec wins.

What made them slow generalises: the encoder took its stride from the dtype at run time and read every element through a per-element dtype switch. *A loop whose body switches on a runtime value cannot be vectorised.* The same pattern hoisted out of the reduction kernels gave 4.4–5.1× on fp32 and 21.3× on int8 sum.

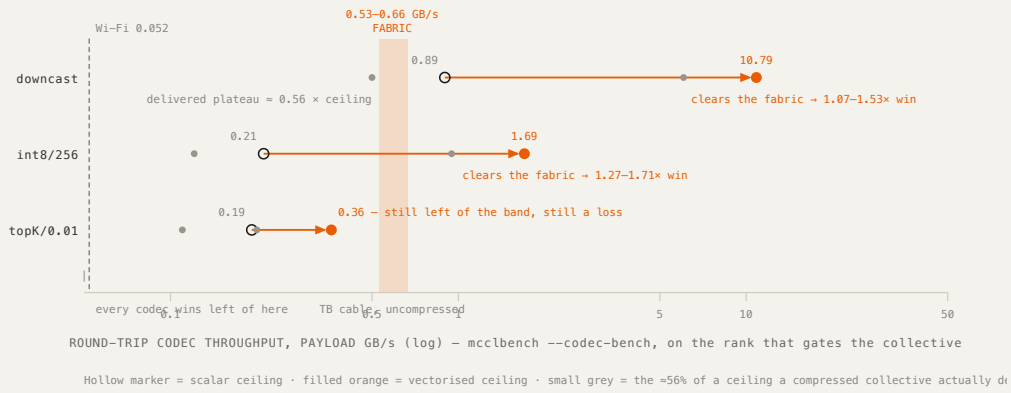

Figure 4. The compression rule's two ceilings on one axis: a codec's position is its own encode/decode throughput, the shaded band what the fabric delivers uncompressed. Scalar `downcast` appears to clear the band at 0.89 GB/s yet still lost — the plateau a compressed collective reaches is roughly 56% of the ceiling.

3.4 Why Top-k Is an Algorithm, Not a Codec

The most consequential position in mccl is that top-k sparsification is not admissible as a per-hop wire codec and must instead replace the collective's algorithm. Applied per hop inside a ring reduce-scatter, each hop must re-sparsify a partially reduced chunk; the mass dropped there belongs to a *partial sum*, so no rank's residual can legitimately hold it. The error is systematic rather than zero-mean, and error feedback cannot repair a bias it does not observe.

mccl therefore routes `.topK` to a different algorithm: sparsify once, at the caller's buffer, then *all-gather* the fixed-size sparse blocks and sum them locally. The reduction is then exact, which is what the convergence argument requires; the format `[nnz][indices][values]` is self-describing and uniform across ranks.

Four restrictions follow, each a typed error: **all-reduce only**; **`.sum` and `.avg` only**, since dropped elements are identities of a sum but not of `min`; **floating-point dtypes only**; and **residuals in `Float`**, since an fp16 residual would flush back to zero the values it preserves. Residuals are keyed per stream and exposed through the public API, because they are model state.

3.5 Per-Pair Addressing

A collectives library for a homogeneous fabric can carry one address per rank; NCCL does. On the fabric island detection exists for, that cannot work: the studio next to you is on `169.254/16`, the laptop across the room on `192.168.1/24`, and a rank advertising either is unreachable from the other side.

So each rank advertises every usable address it owns and each **pair** selects its own path, ranked by the media of the *dialer's own egress interface* — what sets the cost of a hop is the cable the bytes leave on, not the hardware the peer claims. Ordering alone is insufficient, since three Thunderbolt ports on one machine all carry `169.254/16` addresses, so mccl resolves the order by dialing it and taking the first connection that completes.

An inbound handshake is then checkable against the one fact its sender does not choose, the source address it arrived from; anything else is refused rather than allowed to half-form a world that then

deadlocks.

3.6 The C ABI and Asynchrony

`mccl.h` is hand-written rather than generated, because it is the published contract; it mirrors `ncclComm_t`, `ncclUniqueId` and the collective signatures argument for argument. Two decisions are worth recording.

Blocking semantics. mccl has no device queue, so every call runs the collective to completion and returns when the buffers are safe to touch — honest v0 behaviour rather than fake asynchrony, and it makes the raw-pointer bridge safe by construction. The one departure from NCCL's spelling is **the by-value struct**: a 128-byte `ncclUniqueId` passed by value has different calling conventions on arm64 and x86_64, so mccl exports a variant taking the token by pointer and restores the idiomatic call site as a `static inline` wrapper.

Asynchrony without a device queue. With no CUDA stream to borrow, mccl takes MPI's shape: an async all-reduce returns a request handle with wait and test beside it. Collectives still run on one serial queue, so requests execute in issue order; coverage is all-reduce only.

3.7 An RDMA Transport, Built to a Specification Rather Than to a Measurement

macOS 26.2 added an InfiniBand Verbs API for the Thunderbolt 5 controller, documented in Apple's Technical Note TN3205 [11]. `RDMATransport` implements the `Transport` protocol over it with nothing above the seam changed. **This subsection describes code that has never run on hardware** — every machine in this lab is M1-generation with Thunderbolt 4. What follows is design, not evaluation.

The verbs subset is small and its constraints are load-bearing: two-sided send and post-receive only, unreliable-connection queue pairs, ten of them, 4 KiB frames, a 16,773,120-byte message ceiling, and page-aligned buffers behind an IOMMU.

The same-size rule determines the framing. TN3205 requires sender and receiver to post messages the same number of frames long, and a receive buffer must be posted before its message arrives — so the receiver must know a size it has not been told, and announcing each size first does not close the regress. mccl fixes one size for the life of the connection, negotiated during the metadata exchange; the defaults (64 KiB slots, 32 per direction, 2 MiB registered) come from the specification, there being no hardware to tune against.

The absence of hardware acknowledgement determines the failure model. A UC queue pair does not retransmit [17], and TN3205 says plainly that send completion does not mean the data arrived or arrived intact. A Thunderbolt cable is reliable in practice, which is why Apple considers UC sufficient, but a collectives library may not quietly depend on that: a dropped slot becomes a plausible tensor with a hole in it, and then it trains a model. So every slot carries a sequence number, a gap or a repeat is a hard error, and mccl attempts no recovery. **What this does not catch is corruption within a delivered slot**: a sequence number detects a missing slot, not a wrong one, and there is no end-to-end payload checksum — a known gap, since a checksum is a full pass over every byte and should be decided against hardware.

Linkage was resolved from two measured facts: Apple's header and stub library are present in the macOS 26.x SDK, including on Macs with no Thunderbolt 5, while passing `-lrdma` against any pre-26.2 SDK is a hard link error. Compile time therefore uses Apple's own header behind `__has_include`, taking struct

layouts and enum values from the SDK; link time uses `dlopen` and `dlsym` with no `-lrdma`, so the binary carries zero load commands referencing librdma and runs on macOS 14.

04 Verification Strategy

4.1 The Shape of the Suite

The suite is 379 tests across 30 suites, all passing, run in CI on macOS arm64 on every push, split across two targets so the core library's tests keep running on a machine that has never fetched mlx-swift. Three carry disproportionate weight.

A compiled C client — a four-rank C program built with `cc` against `libmccl.dylib` and run — is the only check that simultaneously proves `mccl.h` is valid C, that its declarations match the exported symbols, and that a non-Swift process can link the library. **Real spawned benchmark processes**: the distributed tests spawn actual `mcclbench` processes at two and four ranks and assert that the bus figures carry the right 2(n−1)/n factor and that a mismatched sweep fails rather than deadlocking. **Byte-level codec interoperation**: the vectorised encoders are checked both ways against a retained copy of the scalar loops, so §3.3's experiment measured speed and not format.

4.2 Coverage, and Where It Deliberately Fell

Metric	Covered	Missed
Region	**87.06%**	420 of 3,245 regions
Function	**88.50%**	113 of 983
Line	**92.71%**	492 of 6,745

Region coverage fell from 89.98% when the RDMA transport landed: the verbs binding sits at 33.33% and the transport at 39.51%, the uncovered half of each being the code calling Apple's librdma and the paths that need a queue pair to exist. The mock covers the logic above them — framing 97.22%, connection layer 85.96% — but nothing executes a real post-send without a Thunderbolt 5 cable.

4.3 Verifying a Transport With No Hardware

The RDMA transport is verified three ways. *By compilation:* the shim compiles against the macOS 26.5 SDK with the verbs header, every static assertion cross-checking mccl's constants against Apple's enums passes, and it also compiles as a stub against an older SDK; the built binaries contain zero load-time references to librdma. *At runtime without TB5:* the dynamic open succeeds, all seventeen symbols resolve, the device list returns valid and empty, and the availability check reports the no-devices case with enablement instructions. *Against a mock enforcing TN3205's published rules:* 70 tests covering the queue-pair state transitions in order with their attribute masks, page-aligned registration, byte-stream round trips at every boundary, the same-size rule on every posted message, a dropped slot surfacing as a sequence-gap error rather than corrupt data, back-pressure blocking rather than failing, teardown, and refusal of the eleventh queue pair.

What none of that verifies is that Apple's hardware behaves as its technote says. The repository carries an ordered validation checklist for the first user with Thunderbolt 5: enable RDMA in Recovery on every node, confirm the devices and their GIDs, cable the cluster, establish a TCP baseline on the same cables in the same session *before* the RDMA sweep, sweep the slot geometry, re-derive both crossovers from the new numbers, run the training equivalence over RDMA — and, last, the head-to-head against JACCL, the comparison every adopter will ask for and the one this lab cannot run.

70

MOCK-VERBS TESTS — AND ZERO BYTES OVER A QUEUE PAIR

The transport's state machine, framing, chunking, sequence-gap detection, back-pressure and teardown are enforced against a mock implementing TN3205's published rules. That is the strongest verification available without the hardware, and it is not evidence that the transport works. Whether it recovers the 42–58% of line rate lost to the IP stack is the largest open question in the system.

05 Application Domains and Economic Analysis

5.1 Efficiency Fractions, With Both Denominators

Absolute cross-platform throughput numbers are meaningless. What means something is the *efficiency fraction*: the share of each interconnect's ceiling the stack extracts.

Setting	Achieved (busbw)	Ceiling	Fraction
NCCL, 8× A100, NVLink3, all-reduce [16]	~200 GB/s	430–520 GB/s measured P2P	~40–50%
NCCL, 100 Gbps Ethernet, TCP sockets, no RDMA [16]	4–6 GB/s	12.5 GB/s line rate	~32–48%
mccl, 2-node Thunderbolt, 16 MiB	**0.746 GB/s**	1.06 GB/s, same-session probe	**70.4%**
mccl, 2-node Thunderbolt, 1–64 MiB	0.577–0.746 GB/s	2.5 GB/s raw line rate	23–30%

Two denominators, both honest. Against the raw line, mccl's 23–30% sits *below* NCCL-over-TCP's 32–48%; the difference is the IP-over-Thunderbolt tax the probe itself pays before mccl runs, which the RDMA transport is built to avoid though it has never run on hardware. Against what the link demonstrably carried in the same session the collective extracts 70.4% — and since the probe measures a one-directional stream while an all-reduce turns the link around twice, that is evidence neither the framing nor the ring schedule is where the time goes. The denominator must be labelled with its probe: the same cable

read 1.06 GB/s under single-address probing and 1.46 GB/s under the per-pair probing a mixed fabric requires, and every fraction above is against the probe taken alongside its own sweep.

5.2 The Axis With No Competitor

NCCL has no in-band lossy compression, because on an NVLink fabric the wire is not the constraint — which is why a Mac cluster's advantage is largest where a CUDA cluster is weakest: without RDMA, NCCL collapses by 5–20× onto TCP sockets [16]. On the lab's Wi-Fi path — 0.052 GB/s uncompressed — `topk/0.01` returns 5.4× at 16 MiB; on the Thunderbolt cable `downcast` returns 1.07–1.53× and `int8/256` 1.27–1.71×.

5.3 Where Apple's Own Stack Is the Right Answer

Apple co-developed RDMA over Thunderbolt with JACCL, the `jaccl` backend of MLX Distributed [12], which TN3205 links to directly. **For a uniform Thunderbolt 5 mesh, use JACCL.** It is Apple's, co-developed with the silicon, in-tree, validated on the hardware; mccl's RDMA transport is written to the same technote and has never been run. Nor is JACCL MLX-only: it builds standalone and exposes a C++ API — WWDC26 session 233 says it "can be built without MLX" and "is not limited to machine learning" [13].

Two things that look like differentiators therefore are not. **A C ABI is parity:** mccl's C boundary is useful for a runtime that cannot link C++, but "works outside MLX" describes both libraries. **Automatic topology selection is not unique:** JACCL already chooses between mesh and ring by message size, so mccl's planner claim narrows to *measured* per-link bandwidth and latency across *mixed-speed* fabrics, with the crossover solved rather than thresholded.

	JACCL	mccl
Fabric	RDMA over TB5 only; no TCP data path	TCP anywhere, RDMA where available, **mixed in one world**
Hardware floor	Apple silicon with TB5, macOS 26.2+	Any Mac; RDMA is an optional accelerant
Heterogeneous fabrics	Not addressed	Per-pair path selection, ratio-gap islands, hierarchical plans
Wire compression	None	downcast / int8 / top-k, under the measured crossover rule
Collectives	all-sum, all-max, all-min, all-gather, send, recv, barrier	Plus broadcast, reduce, reduce-scatter
Algorithm choice	Size-based, mesh versus ring	Solved from measured bandwidth and latency
RDMA validated on hardware	**Yes, by Apple**	**No**

Three claims survive, and they are narrow. **Mixed fabrics** — JACCL hard-requires RDMA over Thunderbolt 5, down to a Recovery-mode enablement step, and has no data path to a node not on the mesh; there is no degrading onto Ethernet or Wi-Fi for one awkward rank. That is a works-versus-does-not-work difference in coverage, not a speed win, and §3.2's run is that shape on hardware. **Compression where the fabric is slow** — the rule pays exactly where RDMA is unavailable. **Hardware older than Thunderbolt 5** — the requirement excludes every M1- and M2-generation Mac.

5.4 The Economic Frame: Cables in the Room

The economic argument follows from the hardware rather than from a benchmark. NVIDIA's cluster story prices in the fabric: an NVLink domain or an InfiniBand network is capital expenditure incurred before the first collective runs, and it buys a fabric fast enough that the library's job is to keep it saturated. On a Mac cluster the interconnect is whatever is already there — so where the fabric is bought the library keeps it full, and where it is whatever cables are in the room the leverage is in moving fewer bytes and crossing the slow link as rarely as possible.

5.5 Data-Parallel Training, Including Where It Loses

The MLX adapter adds a gradient-averaging call that flattens a model's gradient tree into one contiguous buffer with a rank-independent layout and all-reduces it once; ten separate all-reduces would spend 1.7 ms on round trips alone at MLP scale.

Correctness is an equivalence, not a tolerance. With equal shards and an optimiser that is a function of the gradient, N ranks averaging their shard gradients and one process on the combined batch are the same arithmetic, so the loss trajectories must agree to floating-point noise and any real defect diverges within a few steps. Over 200 steps they agree to **4.768e-07** in one process and **3.624e-05** (9.694e-06 relative) across the two lab machines over the Thunderbolt cable.

On throughput the result is negative, and the negative result is the informative one. Two nodes lose at every model size at batch 256 — communication is O(P) and compute O(P·B), so only scaling the *batch* moves the ratio. That prediction was tested:

Global batch	1 node (steps/s)	2 nodes (steps/s)	Speedup
256	**186.2**	49.4	0.27×
1,024	**83.4**	39.4	0.47×
4,096	27.4	**29.6**	**1.08×**
16,384	7.5	**10.9**	**1.45×** of a 2.0× ceiling

A Thunderbolt cable pays for a second machine when the global batch is in the thousands, and not before. The codec rule holds here too — latency-bound at an 83 KiB gradient so `downcast` buys 1.02×, bandwidth-bound above it so it buys 1.25–1.26× — and does not rescue the small-batch case.

06 Limitations and Roadmap

Evaluation reaches three machines, not four. The planner's qualitative claim is tested and holds — tree below the crossover, ring above — and island detection and the hierarchical plan have hardware evidence on a mixed fabric. But that plan has run only in its smallest form, one island of two plus one bridge rank, and nothing measured here speaks to sixteen nodes.

The crossover constant is wrong on a fabric whose bandwidth varies with message size. The closed form is 4–13× low on uniform Wi-Fi, because it assumes a single B while that fabric's effective bandwidth moves twentyfold across the sweep. "Solved, not tuned" survives for *which* algorithm and fails for *where* to switch there. On the mixed fabric, whose bottleneck really is one constant, the same expression lands correctly.

The RDMA transport has never run on hardware, and the expectation for it is a prediction rather than a result. A caution belongs here: a "~3 µs versus ~300 µs" latency pairing has circulated in secondary coverage of TN3205 and *does not come from Apple*. TN3205 contains no latency figures of any kind; the circulating numbers are unattributed and mutually inconsistent, and should not be used as a target. The strongest sourced claim is the MLX documentation's, that its JACCL backend achieves latency "an order of magnitude lower than the ring backend" [12] — a ratio, with no absolute figure. Separately, **UC corruption within a delivered slot is not caught**.

The probe does not report which route a link-local address reached; printing the resolved egress interface per path is what would collapse the 42–58% tax from a range to a number. **There is no same-language head-to-head against MLX's ring**, and the obstacle is upstream: mlx-swift does not build MLX's distributed backends at all, and a cross-language measurement against Python MLX compares two language runtimes rather than two schedulers.

Compression is still a caller's flag, and should not be — the rule connecting the probe and the codec benchmark is one comparison the planner has the data to make. **Asynchrony stops at all-reduce**, and the rendezvous is not a service: one round trip through rank 0, with no discovery, no membership change, no recovery from a rank restarting.

07 Conclusion

Mac clusters lack a collectives library, and the absence matters more than it would elsewhere, because what limits them — the interconnect — is exactly what an application-level ring cannot optimise on the application's behalf. mccl is an attempt at that layer, at 87.06% region coverage over 379 tests.

Three results are worth carrying away. The first is a *rule* rather than a speed-up, and is the most portable thing here: a codec pays only when the fabric's uncompressed all-reduce rate is below that codec's own encode/decode ceiling. It names no hardware, both terms are measurable on the machine in front of you, and it was confirmed from both sides in one experiment — vectorising the encoders lifted two ceilings above the cable's rate and both codecs flipped from losing to winning, while the third's rose 1.9×, stayed below, and stayed a loss. The second is that a measured-topology planner beats a fixed ring on a real mixed-speed cluster: two Mac Studios on a Thunderbolt cable plus a laptop reachable only over Wi-Fi is a 20.76× ratio inside one world, the planner finds the islands from its own probe, and the hierarchical plan runs 1.74× the ring at 16 MiB and wins at every size from 256 KiB up across two runs. The third was not

anticipated: forming that world at all required per-pair addressing — a mixed fabric is one where a single address per rank does not describe the cluster.

What should not be carried away is equally specific. This is the *smallest* mixed fabric, two islands of two is still only tested in the suite, and nothing here speaks to sixteen nodes. Every number was taken over TCP/IP, paying a 42–58% stack tax before mccl's scheduling began, which is why the raw-line efficiency sits below NCCL-over-TCP's published band even where the same-session fraction is 70%. The RDMA transport that exists to recover that tax has never moved a byte over a queue pair, and on the fabric it targets JACCL is the better choice. What remains unserved is the rest of the space: the mixed, older, slower clusters assembled from machines someone already owned.

// References

[1] Barnett, M., et al., "Interprocessor Collective Communication Library (InterCom)," Proc. Scalable High Performance Computing Conference, pp. 357–364, 1994.

[2] Rabenseifner, R., "Optimization of Collective Reduction Operations," ICCS 2004, LNCS vol. 3036, pp. 1–9, Springer, 2004.

[3] Thakur, R., Rabenseifner, R., Gropp, W., "Optimization of Collective Communication Operations in MPICH," Int. J. High Performance Computing Applications, vol. 19, no. 1, pp. 49–66, 2005.

[4] Gibiansky, A., "Bringing HPC Techniques to Deep Learning," Baidu Research technical report, 2017.

[5] Sergeev, A., Del Balso, M., "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," arXiv:1802.05799, 2018.

[6] NVIDIA Corporation, "NCCL — NVIDIA Collective Communications Library," Developer Documentation, 2024.

[7] Alistarh, D., et al., "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding," NeurIPS 30, 2017.

[8] Lin, Y., et al., "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training," ICLR, 2018.

[9] Karimireddy, S.P., et al., "Error Feedback Fixes SignSGD and other Gradient Compression Schemes," ICML 36, pp. 3252–3261, 2019.

[10] Stich, S.U., Cordonnier, J.-B., Jaggi, M., "Sparsified SGD with Memory," NeurIPS 31, 2018.

[11] Apple Inc., "TN3205: Low-Latency Communication with RDMA over Thunderbolt," Apple Developer Technical Note, 19 March 2026, rev. 13 April 2026.

[12] ml-explore, "MLX Distributed — JACCL backend," github.com/ml-explore/mlx, MIT License, 2026.

[13] Apple Inc., "WWDC26 Session 233," Apple Worldwide Developers Conference, 2026.

[14] Hockney, R.W., "The Communication Challenge for MPP: Intel Paragon and Meiko CS-2," Parallel Computing, vol. 20, no. 3, pp. 389–398, 1994.

[15] Chan, E., et al., "Collective Communication: Theory, Practice, and Experience," Concurrency and Computation, vol. 19, no. 13, pp. 1749–1783, 2007.

[16] NVIDIA Corporation, nccl-tests issue #149; nccl issues #450 and #209 — the published all-reduce efficiency figures cited in §5.1.

[17] InfiniBand Trade Association, *InfiniBand Architecture Specification*, vol. 1, rel. 1.7, 2020 — unreliable-connection queue-pair semantics.