WP-2026-004 — Router Pools: What Pooling Costs, What It Buys, and Where It Stops (Whitepaper)

Version 3 of DOC-2026-493. Supersedes v2.

Document number: DOC-2026-493 **Whitepaper:** WP-2026-004 **Version:** 3 **Category:** Marketing Material **Product family:** fleet (XyDromatics Router, SynthIQ) **Owner:** Synthology Healthcare Solutions Group, LLC (SHSG) **Audience:** Radiology IT decision-makers, CIOs, biomedical engineering, and PACS administrators sizing a routed DICOM tier — deciding how many routers to run, how large a load balancer needs to be, and how long a router can run before its own bookkeeping becomes the limit. **Regulatory posture:** Non-device software under **FD&C Act §520(o)(1)(D)**. No clinical or diagnostic function; nothing in this whitepaper describes a medical device. **Companion:** Figures render on the marketing site at `/benchmarks`. Method is DOC-2026-490 v4.

> **Measurement note:** every number here is a measured result from dedicated performance rigs on two independent cloud providers, at shipped defaults with no tuning. 251 measured runs were produced; 86 are used and **165 were discarded**, each with the reason recorded. §7 lists what was thrown away and why, because a table of only the surviving runs says nothing about how they were chosen. Those counts cover the original two-cloud campaign. The pooled **delivery** measurement added to §2 in this revision is a separate, later campaign of its own — three runs per condition, all of them used, none discarded.

Executive summary

Three questions decide the shape of a routed DICOM tier: what does putting a load balancer in front of your routers cost you, does a bigger load balancer help, and how long can a router run before it stops accepting images. We measured all three, on two clouds, and publish the conditions alongside the results. This revision answers the first question twice — once for the rate at which a pool **accepts** images and once for the rate at which it **delivers** them. They are not the same number, and they do not lead to the same advice.

- **At the front door, pooling costs 55–64% of the raw sum of its parts.** Two routers that individually accept 875 instances/sec each do not accept 1,750 through a pool — they accept 626 to 785 depending on the load balancer. Those figures, and every other throughput figure in this paper's earlier versions, are **ingest acceptance** rates: how fast the tier takes images in, not how fast it moves them on. §1 draws the distinction; §6 explains why we are only labelling it correctly now.
- **On delivery, pooling costs about a third — and a pool out-delivers a single router.** Measured for the first time in this revision, at the 8 vCPU balancer tier: the pool delivers **432** instances/sec against **667** for the two routers running direct, a ratio of **0.65**; and against **350** for one router alone, which is **1.24x**. The ingest figures from the very same runs reproduce the published 0.45, which is what makes the delivery figure worth acting on. On the ingest framing a pool reads as a cost paid for availability; on delivery it adds throughput as well. **Delivery is measured at one balancer tier only** — do not read a delivery curve off the ingest curve in §3.
- **A bigger load balancer helps, then abruptly stops helping — on ingest.** Quadrupling it from 2 to 8 vCPU buys **25%**, and most of that arrives by 4 vCPU. Past there the constraint moves off the balancer and onto the routers — measured directly, not inferred: at 2 vCPU the balancer runs at **97%** CPU while the routers sit at 73–76%; at 8 vCPU the balancer drops to **28%** and the routers become the busiest thing in the path.
- **Every router has a volume at which it refuses images, and ours is at 438–583 MB of operational database.** We measured it rather than leaving you to discover it.
- **That limit travels; throughput does not.** The two clouds' volume limits differ by **3%** while runs on the same cloud vary by up to **33%** — so the limit is a property of the software. Throughput between the same two clouds differs by **16–35%**, because that is a property of your hardware. Both are worth knowing, for opposite reasons.
- **There is no gradual warning before the limit.** Throughput held between **66% and 103%** of baseline right up to the burst in which refusals began. One run was measurably *faster* than its own baseline as it hit the wall. No threshold you could watch would have warned you.
- **We corrected our own published figures upward during this work, and then withdrew the percentage we had put on that correction.** A defect in our own software had been suppressing the figures. But the same curation rule we apply everywhere else voids every pre-fix run, so the old baseline cannot support a percentage. §6 and §7 describe both, because the correction and its retraction are equally part of the record.

The practical headline: **size the pool for the pooled number — the delivery one if you are sizing for work moved, the ingest one if you are sizing the front door; stop growing the load balancer at 4 vCPU on ingest, and ask for a delivery measurement at the tier you are actually buying; and bound your retention before volume bounds it for you.**

§1 — The three legs, and why the ratio is the answer

A pooled measurement is only meaningful against the alternative. Every campaign therefore takes three legs under identical conditions, back to back:

Leg	Path	What it establishes
X′	Generator → Router A → Archive A	What one router does alone
Y′	Generator → Router B → Archive B	What the other does alone
Z	Generator → **load balancer** → both routers → both archives	What the pool does

Every leg is 20,000 synthetic CT instances at 32 concurrent senders, with a live receiving archive that drains — never a discard rule and never a dead destination. Completeness is checked per SOP instance and a run is only reported if **every** instance was accounted for: in the ingest legs that is the sender-side C-STORE record, and in the delivery runs of §2 it is the count of instances the receiving sink acknowledged.

The ratio **Z / (X′ + Y′)** is the finding, on either rate. If pooling were free it would be 1.00. It is not.

Two rates, and which one each figure is

A router that has accepted an image and a router that has delivered it are not at the same point in the pipeline, and the two are not the same rate.

Router answers a C-STORE as soon as it has taken the object onto its processing queue; forwarding to the destination runs asynchronously behind that queue. The load harness computes its rate as sender-side C-STORE successes over send-loop elapsed. That is a real and useful number — it is how fast the tier accepts work at the front door — but it is an **ingest acceptance** rate, not a delivery rate. Every throughput figure published in v1 and v2 of this paper is an ingest acceptance rate, as is every figure in §3, §4 and §5 below. The harness has no second timing mode.

Delivery is counted differently, and separately: one record per instance that Router forwarded to the sink and the sink acknowledged, clocked from first arrival to last delivery. §2 reports both rates for the pool, from the same runs.

§2 — What pooling costs

At the front door: ingest acceptance

Load balancer	X′	Y′	Pooled (Z)	Z / (X′+Y′)
2 vCPU	870.7	874.0	**626.1**	0.36
4 vCPU	861.1	872.6	**741.2**	0.43
8 vCPU	875.4	876.7	**785.1**	0.45

Ingest acceptance, instances/sec, mean of 4 runs each. Azure, AMD EPYC 9V74 on every participating node — identical silicon throughout, which is what makes this table a scaling measurement rather than a hardware comparison.

Two routers that accept ~875 each do not accept 1,750 pooled. They accept 626 at a small load balancer and 785 at a large one — **36% to 45%** of the arithmetic sum.

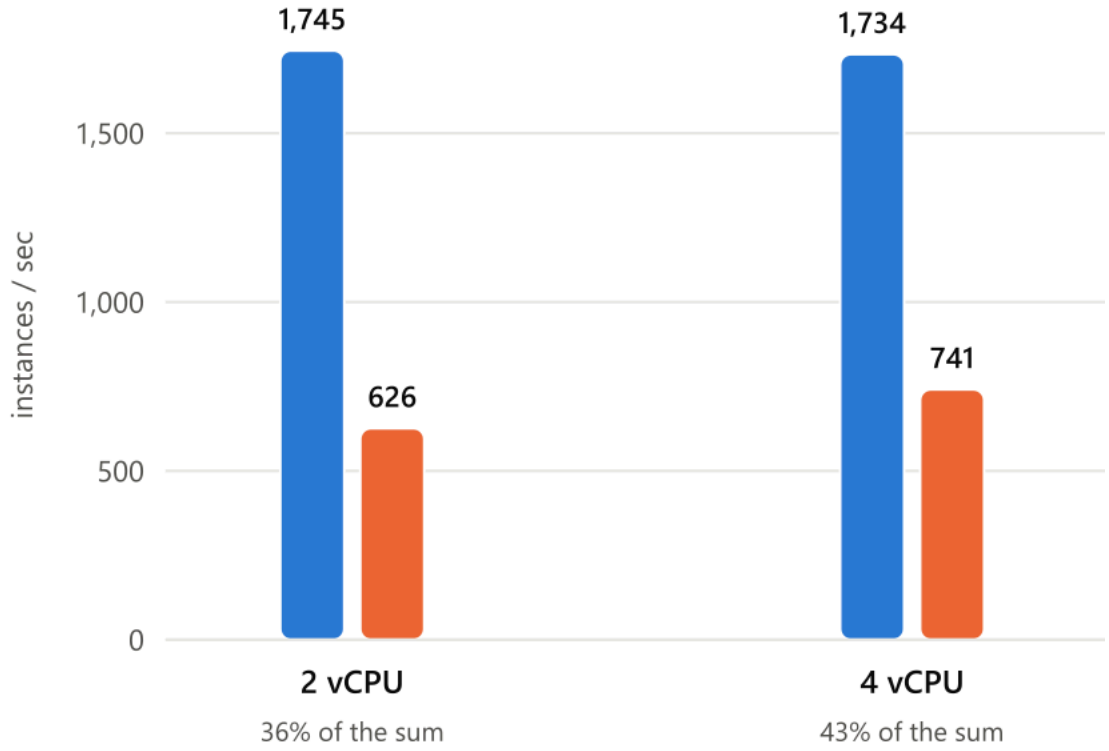

Figure 2 — the gap is the cost of pooling at the front door. It narrows as the load balancer grows, and stops narrowing once the routers behind it become the constraint.

This is not a defect and it is not hidden overhead. A load balancer that accepts an image, chooses a destination, forwards it, and waits for the receiving router's acknowledgement before answering the sender is doing real work on the critical path, and that work is serialised per image.

On delivery

The ingest ratio was the only pooled ratio anyone had measured. On 2026-08-24 we measured the delivery ratio, at the 8 vCPU balancer tier, with all four Router-class hosts — **including both sinks** — reset to a clean database first, the published workload run verbatim, and three runs per condition. Every run settled at exactly 20,000 delivered, with zero failures.

Rate	X′ (Router A)	Y′ (Router B)	Sum	Pooled (Z)	Z / (X′+Y′)
Ingest acceptance	858	864	1,722	770	0.45
Delivery	350	317	667	**432**	**0.65**

Instances/sec, mean of 3 runs per condition. Azure, 8 vCPU AMD EPYC 9V74 on every participating node. Load split evenly to within 1% — 10,100 / 9,900 and 10,000 / 10,000.

Four things follow, and the middle two change this paper's advice.

The ingest row reproduces the published 0.45. Same tier, same workload, same ratio as the table above — and it came from the very same runs that produced the delivery row. That is what makes the delivery figure worth acting on: it is not a different campaign returning a different answer, it is a second reading of the same pool.

Pooling costs about a third of delivery throughput, not more than half. The published 0.45 understates the pool's real efficiency, because it is the ratio of two front doors, and the front door is not where the work is. On the leg that actually moves images to their destination, the pool retains **0.65** of the sum of its parts.

A pool of two out-delivers a single router: 432 against 350, or 1.24x. On the ingest framing, pooling was a tax you paid for one ingress address and a node that can fail. On delivery it is not only a tax — the pool moves more work than either router in it. If you are choosing between one router and a pool of two, ingest made the pool look like a compromise. Delivery does not.

This is one tier. The delivery figures above are the 8 vCPU balancer and nothing else. The 2 and 4 vCPU tiers have been measured on ingest only. The delivery curve across balancer sizes **must not be read off the ingest curve in §3**: at the one tier where both exist, the ratio moved from 0.45 to 0.65, so the shape is not preserved and there is nothing to interpolate with. If the delivery number at a smaller balancer matters to your decision, ask us to measure it rather than scaling ours.

What you are buying

What a pool does buy: one address for every modality to target, a node that can fail without the ingress disappearing, and capacity you can grow without touching modality configuration. At the front door you pay 55–64% of the arithmetic sum for that. On delivery, at the one tier measured, you pay about a third — and you get more throughput than a single router while you are at it.

Direct delivery to both routers is still the faster arrangement on both rates: a ratio below 1.00 is a ratio below 1.00. If you need the raw sum and can live with modality configuration pointing at individual routers, point it at individual routers. The pool is the better answer against **one** router, not against two.

§3 — Does a bigger load balancer help?

Up to a point, sharply, and then it stops. Every figure in this section is an ingest acceptance rate.

Load balancer	Pooled ingest acceptance	vs 2 vCPU
2 vCPU	626.1	baseline
4 vCPU	741.2	**+18%**
8 vCPU	785.1	**+25%**

Quadrupling the balancer buys 25%, and roughly three-quarters of that gain has already arrived at 4 vCPU. The reason is directly observable in the CPU record rather than inferred:

Load balancer	Balancer CPU	Router CPU	What is actually limiting
2 vCPU	**97%**	73–76%	the balancer
8 vCPU	28%	**64–66%**	the routers

At 2 vCPU the balancer is saturated and is the constraint. At 8 vCPU it is loafing at 28% and the routers behind it have become the busiest component. You have not removed the wall; you have moved it. **Growing the balancer past 4 vCPU buys progressively less ingest, and past 8 vCPU it buys none at all until you add routers.**

Receiving archives never exceeded 42% CPU in any run, which is how we know they were not the constraint in any of these figures.

This is an ingest curve

We have one delivery measurement of a pool (§2), at 8 vCPU, and it does not sit on this curve: the ratio at that tier is 0.65 on delivery against 0.45 on ingest. Nothing in this section licenses an inference about how delivery behaves at 2 or 4 vCPU. The sizing advice here — stop at 4 vCPU — is advice about the rate at which the tier accepts work. Where the constraint sits for delivery at smaller balancers is not established, and we are not going to estimate it.

A note on which rig this table comes from

These figures are from one rig only, and deliberately so. Our Vultr rig mixes AMD EPYC-Rome and EPYC-Milan across its load-balancer tiers, so its scaling curve varies core count *and* silicon at the same time. Its numbers are real and reproduce, but they cannot answer a scaling question, so they are not used for one. The Azure rig runs an identical processor on every node, so its curve isolates core count and nothing else.

§4 — Where a router stops accepting images

Every store-and-forward router keeps an operational record of what it has handled. That record grows. At some volume it becomes the limiting factor and the router begins refusing new images — correctly, because refusing is safer than accepting what it cannot account for.

We pushed eight sustained soaks, four per cloud, until that happened.

Rig	Database size at first refusal	Individual soaks
Azure, 4 soaks	**486 MB** mean	453 / 458 / 462 / 570 MB
Vultr, 4 soaks	**499 MB** mean	438 / 463 / 514 / 583 MB

The two clouds differ by 3%. Runs on the same cloud vary by up to 33%.

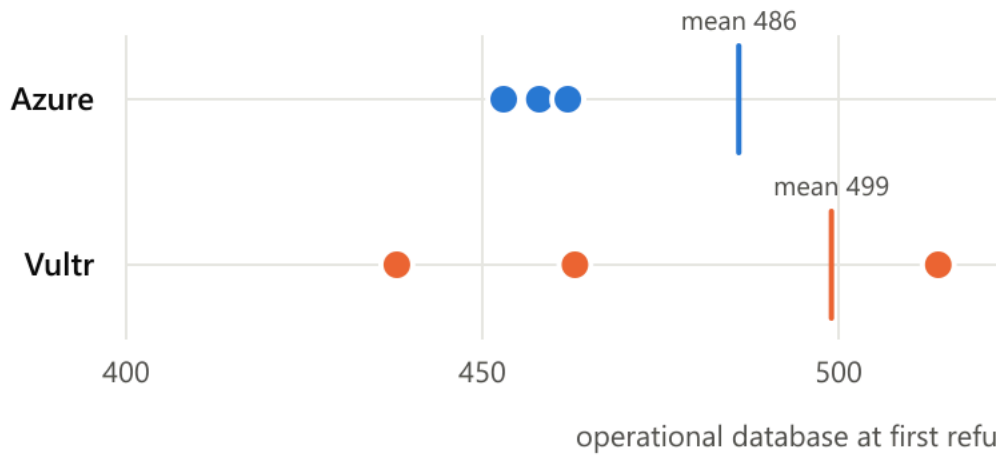

Figure 1 — the finding is the SPREAD, not the means. Runs on one cloud vary by up to 33% while the two cloud means sit 3% apart, on hosts whose raw throughput differs by 60%.

That is the whole finding. The variation *between* providers is an order of magnitude smaller than the variation *within* one, on hosts whose raw throughput differs by 60%. So this limit is a property of the software and it travels with it — you can plan against it on hardware we have never tested. Compare that with the ingest throughput figures in §2 and §3, which differ by 16–35% between the same two clouds precisely because they are a property of the hardware.

A supporting constant: across all eight soaks and both clouds, the operational record consumed **896–902 bytes per handled instance**. A budget in records converts to a budget in bytes the same way on any host.

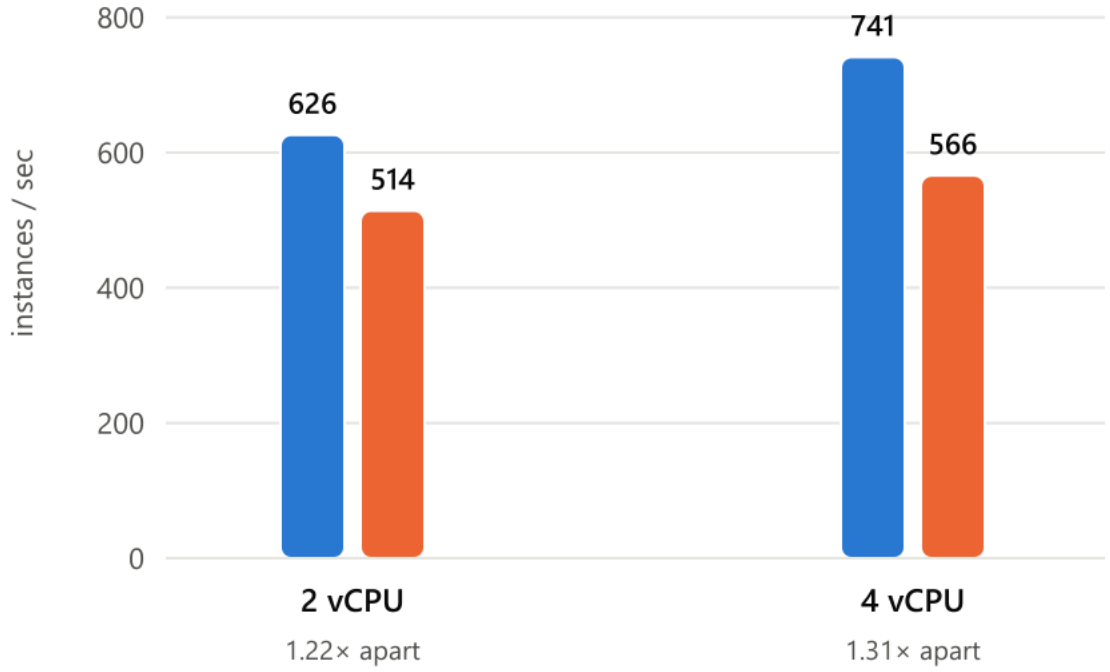

Figure 3 - the counterpoint to Figure 1. Identical build, workload and topology; only the hardware differs. Throughput moves 1.16-1.35x between providers while the volume limit in Figure 1 moves 1.03x. One follows the software, the other follows your hardware.

§5 — There is no warning

The result that should change how you operate a router is not the limit itself. It is that nothing tells you that you are approaching it.

Ingest throughput across the eight soaks held between **66% and 103% of baseline right up to the burst in which refusals began**. One run was measurably *faster* than its own baseline in the same burst that it started refusing images. There is no slow degradation to alarm on, no rising latency to trend, no threshold that would have given warning. Note which rate that is: the front-door acceptance rate is the one that eventually goes to zero when the router starts refusing, so it is the most favourable early-warning signal available — and it still gave none.

Two things follow.

Monitoring cannot substitute for retention. An alert on throughput would not have fired. An alert on latency would not have fired. The only signal that precedes the cliff is the size of the operational record itself, which is why that figure is now reported in our own product telemetry.

A bigger disk does not help, and neither does a different database. The limit is reached by accumulating records, not by filling a volume. A retention policy that deletes nothing under sustained load fails identically on any database engine — just later, and at a larger number. Bound the record count.

§6 — We corrected our own numbers upward, and here is why that matters

The pooled figures previously published on our own site averaged **347.4 instances/sec**. They are now **537.9** — the same rig, the same hardware tier, the same 20,000-instance workload.

We first published that gain as **+48%**. We are withdrawing the percentage rather than restating it. The rule this document applies to every other run — that a leg routed through a node on the older build is void — disqualifies every pre-fix run, including all of the ones that produced the 347.4 baseline. A percentage measured against a void baseline is not a measurement; it only looks like one. The absolute figures stand on their own, and they are the only thing we claim.

We did not find this because a customer challenged it. We found it re-checking our own rig, and two separate faults were responsible:

Our own software had a defect. The router's operational database used a write-ahead log with no bound on its growth. Under sustained load that log grew without limit, and throughput decayed as it grew. Every earlier figure was measured on a router already carrying that penalty. Fixed, the log now peaks at **8.2 MB** against a pre-fix peak of **242.3 MB** on the same rig and workload.

And our test rig had a fault we had not detected. One of the two receiving archives had been left on the previous software build for an entire measurement series — on both rigs. Because a pooled run routes through both archives, every pooled number we had published passed through a node that degraded as it received.

We are stating this plainly rather than quietly replacing a table, for a reason that matters to anyone reading a vendor benchmark: **the failure mode of a benchmark is not usually a wrong number, it is a plausible one.** Both faults produced results that looked entirely reasonable. Neither was visible in the output. They were found by verifying the rig itself rather than trusting that the tooling had done what it reported doing — which is now a written requirement of our test method rather than a habit.

And in this revision: what our rate had been measuring

v3 carries a second correction, of a different kind. The first was a wrong number. This one is a mislabelled one.

Every throughput figure this paper has published — v1, v2, and the ingest tables that remain in v3 — was computed by the load harness as sender-side C-STORE successes over send-loop elapsed. Router returns success as soon as it has taken the object onto its processing queue, and forwarding runs asynchronously behind that queue. So the harness has been measuring how fast the tier **accepts** images, while we described it as how fast the tier **moves** them. The harness has no second timing mode, so there was never a delivery figure sitting alongside these to contradict them.

Nothing in the ingest tables changes. 0.36, 0.43 and 0.45 are correct: both sides of every one of those ratios were measured the same way, and the re-measurement in §2 reproduces 0.45 at the 8 vCPU tier. What changes is the label on them, and the conclusion we drew from them — see §2 for the delivery ratio, which is materially better than the ingest ratio and which we did not have when v2 was written.

§7 — What we discarded, and why

The campaign produced **251 measured runs. 86 are used. 165 were discarded.** Each discarded run is retained with the reason it was dropped:

- **Runs measured before the write-ahead-log defect was found.** Throughput on those decayed with cumulative volume, so any rate is a function of how much had already passed through that node rather than of the node itself.
- **Runs where a receiving archive was on the previous build.** A pooled run routes through both archives, so these passed through a degrading node — including every pooled figure we had previously published.
- **Runs on the rig whose processors differ between tiers, where a scaling question was being asked.** The numbers are real; they simply cannot answer that particular question.

Five runs were **kept that should not have been**, and finding them is the most useful thing in this section.

Of five pooled runs at the smallest tier, four cluster between 529 and 550 instances/sec and one sat at 418.7. We checked whether that one could be attributed to a fault: every host showed 0.00% hypervisor steal and the balancer was at 99% CPU — the same profile as a normal run. With no fault to blame, we kept it.

We had checked the wrong thing. The fault was not in the host telemetry, it was in the instance record: that run had 19,657 of 20,007 instances accepted. It shed 350. Auditing the rest of the campaign on that basis found four more legs in the same state — all on the same rig — shedding 50, 215, 3,317 and, in the worst case, 10,485 of 20,000 instances. That last one lost more than half its workload and still carried no void reason.

The rule stated in §1 — a run counts only if every instance was accounted for — already disqualified all five. We had applied it to 160 other runs without noticing it applied here too. The tally above is the corrected one: **86 used, 165 discarded**, not the 91 / 160 we first reported.

The correction runs **against our own interest in the direction that matters**: removing the 418.7 leg moves the smallest-tier figure *up*, from 514.0 to 537.9. We had been under-reporting ourselves by keeping a run we should have dropped. A curation rule that is only applied when it flatters you is not a curation rule, which is why all five are named here rather than quietly removed.

§8 — The conditions every figure was measured under

A rate without its conditions is not much use. Published alongside these results:

- **The processor model, core count and memory of all fourteen participating nodes.** Equal vCPU counts do not imply equal silicon.
- **The operational-database size on every router at the time of measurement** — 477 MB to 1.9 GB across the campaign, against the 438–583 MB limit in §4. Those ranges overlap, and the top of the first sits well above the second. **This variable was not controlled, and v2 of this paper presented that as a deliberate choice. It was not one.** The soak script that produced §4 resets the operational database before every measurement, and states in its own comments why: the sink is itself a Router and accumulates as it receives, so leaving it loaded would mean the single-router leg degrading for a reason that is not the variable under test. The two campaign scripts that produced every throughput figure in §2 and §3 do neither — they never reset the database and they never record its size. When we went back and looked, every host on both rigs was already past the knee: 1,467 MB on Router A, 1,901 MB on one of the sinks. Sizes were captured at end-of-series rather than per run, so we cannot say what volume any individual figure was taken at, and we do not. Those figures were measured at unknown and drifting database volume. Because throughput declines as that record grows, the direction of the effect is to understate rather than to flatter — but the amount was not measured and we do not put a number on it. The delivery re-measurement in §2 was run from a clean database on all four Router-class hosts, sinks included, for exactly this reason.
- **CPU utilisation captured on every host for every run**, which is how §3 can state where the constraint actually sat rather than infer it.

Conclusion

Three findings should shape how you size a routed DICOM tier.

Size for the pooled figure, and use the right pooled figure. At the front door a pool retains 36–45% of the arithmetic sum of its routers. On delivery, at the 8 vCPU tier where we measured it, it retains 65% — and out-delivers a single router by 1.24x. Size the ingress against the ingest ratio; size for the work the tier will actually move against the delivery ratio. Because delivery has been measured at one balancer size only, ask us for the tier you are actually buying rather than scaling the number you have.

Stop growing the load balancer at 4 vCPU — for ingest. Most of the available front-door gain has arrived by then, and past 8 vCPU the constraint has moved to the routers, where more balancer cores buy nothing. What the delivery rate does across balancer sizes is not measured and cannot be read off the ingest curve: at the one tier where we have both, the ratio is 0.45 on ingest and 0.65 on delivery.

Bound your retention before volume bounds it for you. The limit is real, it is measurable, it travels across hardware — and nothing warns you before you reach it.

Every figure here is reproducible on request, including by vendors who wish to dispute one. The method is published as DOC-2026-490. The raw results, the per-instance records, the CPU samples and the scripts that produced them are retained and individually hashed — and the records say which of the two rates each figure is. **If a number here matters to your decision, ask us for the run behind it.**

Synthology Healthcare Solutions Group, LLC. All software referenced is general-purpose, non-device software under FD&C Act §520(o)(1)(D) — not a medical device, and no clinical diagnostic function.

> **Regulatory classification.** All Synthology and XyDromatics software described here is non-device software under the FD&C Act §520(o)(1)(D) software exclusion — not a medical device, and no clinical diagnostic function. Published by Synthology Healthcare

Solutions Group, LLC as a controlled document; the authoritative version is the approved record in the Synthology quality-management system.