Throughput and Capacity

Early Access · MessageFoundry v0.3.2 · July 2026

How much traffic MessageFoundry carries, what we measured to establish it, and how to size your own deployment.

Summary

MessageFoundry sustains **40 million message events per day**. That figure deliberately holds back **more than 20% of measured capacity as reserve**, rather than quoting the ceiling itself.

That figure is not a projection. It comes from a measured sustainable ceiling of approximately **603 message events per second** — about 52 million events per day — from which we publish roughly three quarters. The reserve is deliberate: a published number should be one you can run at every day, not a record set once under ideal conditions.

Three things are worth knowing before the numbers mean anything:

- **A "message event" is one message in or one message out.** An interface that receives a message and delivers it to one destination produces two events.
- **At peak, nothing in the system was running out of capacity.** The database and the processors both had substantial headroom. The limit is a sequencing constraint, not exhausted hardware.
- **In real deployments the engine spends most of its time waiting for the systems it talks to.** Our measurements assume a partner that replies instantly. Yours will not.

1. What "message events per day" counts

Interface engine capacity is conventionally counted in **total message events** — every message that enters the engine plus every message it delivers. This is the unit used across the industry for daily-volume figures, and it is the unit used here.

Shape	Events per message
One message in, one delivery out	2
One message in, four deliveries out	5

So 40 million events per day is roughly **20 million inbound messages** in a simple one-in-one-out feed, or fewer inbound messages where feeds fan out to several destinations.

Quoting a bare "messages per day" without saying which unit is meant is the most common way capacity figures mislead. When comparing any two engines, confirm you are comparing the same one.

---

2. The measurement

Conditions

Topology	Four engine processes sharing one database
Store	Microsoft SQL Server on its own host, local NVMe storage
Fan-out tested	Both one-in-one-out and one-in-four-out
Duration	Sustained holds, not short bursts
Counted	**Completed** deliveries, confirmed drained — not messages offered
Correctness	Zero loss, nothing stranded, per-lane order preserved, pipeline fully drained

Result

Approximately 600 to 605 message events per second, sustained.

The most useful property of this result is that it **did not change with fan-out**. Feeds that deliver to one destination and feeds that deliver to four converged on the same total-event ceiling, within measurement noise. Capacity is therefore governed by total event volume, not by how many destinations a feed serves — which makes sizing considerably simpler. One caveat we would rather state: the two fan-out cases also differed in destination count and offered rate, so this is an observed equality on the configurations tested rather than a clean single-variable isolation.

From ceiling to published figure

	Events/second	Per hour	Per day
Measured sustainable ceiling	~603	~2.17 M	~52.1 M
Published capacity (>20% reserve)	**~463**	**~1.67 M**	**40 M**

Publishing 40 million against a measured 52 million holds back about 23%.

Why reserve at all. A ceiling is the point at which the system stops keeping up. Running a production interface engine at its ceiling means any variation — a busier hour, a slower partner, a maintenance window — turns into a growing backlog. Reserve is what absorbs that, and a capacity claim without one is a claim you cannot actually operate at.

---

3. Nothing was saturated at the ceiling

We checked specifically what ran out first at peak load. **Nothing did.**

Resource	At the ceiling	Exhausted?
Database CPU	~69% average, measured while deliberately overloaded 25% past the ceiling	No
Engine host CPU	Roughly half idle	No
Database commit capacity	~23,600–27,200 commits/second available against ~600 demanded	No — roughly 40× headroom
Connection pool	Never fully in use; waits to acquire a connection averaged ~0.015 ms	No
Worker thread pool	Queue essentially empty — zero for 84% of samples	No
Outbound delivery lanes	Around 90% idle	No

The limit is that certain durable writes must happen **in sequence** — one after another to preserve the ordering and at-least-once delivery guarantees the engine makes. Sequencing is not something more hardware relieves.

That has a directly practical consequence: **adding processor cores to the database does not raise this ceiling.** We measured that rather than assuming it. Nor do the two changes people usually reach for next. Cutting the number of database transactions per message was measured directly: a 28% reduction in committed transactions moved sustained throughput by less than one percent — inside measurement noise — so transaction reduction is a measured dead end rather than a lever. Faster transaction-log storage is on the same footing: the store commits far below its measured ceiling, so the log is not what the pipeline is waiting on. We have no identified throughput lever, and we would rather say so than name one we have already falsified.

We describe the precise mechanism as well-corroborated rather than proven — it is consistent with everything we measured, but we have not isolated it to the exclusion of all alternatives, and we would rather say so than overclaim.

4. Why your throughput will differ: partner systems

This is the single most important factor in real deployments, and it is not the engine.

Ordered delivery is inherently serial. When a feed must arrive in the order it was sent — the default, and a hard requirement for most clinical interfaces — the engine can have only one message in flight per destination:

1. Send message *N* to the partner.
2. **Wait** while the partner receives it, processes it, usually writes it to its own database, and acknowledges it.
3. Only then send message *N+1*.

Step 2 is not the engine's time. It belongs to the network round-trip and to the partner's own processing, and because the stream is serial it lands squarely in the critical path.

Approximately:

> **messages/second ≈ 1000 ÷ (engine's fixed per-message cost + partner round-trip)**, in milliseconds

The engine's fixed cost is small and roughly constant. The partner's round-trip is yours, and it dominates as soon as it exceeds a few milliseconds. A partner acknowledging in 50 ms holds a single ordered interface to roughly 15 messages per second; at 250 ms, to about 4. **No amount of engine or database speed changes that.**

Every figure in this document was measured against an **instantly acknowledging partner**. That is a deliberate choice — it isolates what the engine contributes — but it means these are ceilings under ideal conditions, not forecasts for your environment.

What you can do about a slow partner

- **Relax ordering where it is safe.** Feeds that tolerate out-of-order delivery let the engine keep many messages in flight at once, hiding the partner's round-trip behind concurrency.
- **Open multiple connections.** Each destination connection is its own ordered lane.
- **Split the feed at the source** — by facility, service line, or region — so aggregate volume is not gated by one serial lane.
- **Ask your partner about their acknowledgement time.** It is usually the cheapest thing to improve and the term that matters most.

---

5. Sizing a deployment

1. **Start from total events, not messages.** Multiply expected inbound volume by (1 + destinations per message).
2. **Size to the busy hour, not the daily average.** Healthcare traffic is not flat: volume is light overnight, climbs through the morning, and peaks during clinical hours. Use your own feed's busiest hour divided by its daily average — your integration team can measure this from existing interface logs, and it is worth doing rather than assuming.
3. **Apply your partners' acknowledgement times** (§4). For ordered feeds this is usually the largest single reduction.
4. **Scale by adding interfaces, not by making one interface faster.** A strictly ordered interface is a single serial lane with a bounded rate — that is the physics of guaranteed ordering, not a product limitation.

One caution on adding interfaces. Aggregate capacity is *not* simply the sum of each interface's individual ceiling. Interfaces share a database and an engine host, and we have measured concurrent aggregate throughput landing well below the naive sum. Size against a measured concurrent figure where you have one, and treat the sum as an upper bound only.

---

6. Scope and limits

We would rather you know what these numbers do and do not cover.

- **Synthetic data, laboratory conditions.** All measurements use generated HL7 messages on dedicated test infrastructure. Nothing here was measured against a live clinical system.
- **Instant-acknowledging partner.** See §4. This is the largest single reason a production number will be lower.
- **Message size.** Measurements used compact messages typical of ADT traffic. Feeds carrying large embedded documents cost more per message.
- **Pass-through processing.** Heavy transformation, strict validation, and live enrichment lookups all reduce throughput, and are the largest hardware-independent factors.
- **Database state.** The measurement ran against a store already holding several million rows, which is representative of a system in service rather than a freshly initialised one.
- **Not externally audited.** These are our own measurements, reported with their conditions so you can judge how they map onto your environment.
- **No head-to-head comparisons.** We publish no competitive benchmark. Throughput figures are hardware- and workload-dependent, and a comparison run by one vendor against another is not evidence we would ask you to trust from anyone else.

Treat every throughput figure — ours included — as a starting point for measurement in your own environment, not as a guarantee.

Four questions to ask of any throughput claim

Ours or anyone's:

1. **What unit?** Total events, inbound messages, or deliveries?
2. **Ordered or unordered delivery?**
3. **How fast did the receiving system acknowledge?**
4. **What is the peak-hour rate behind that daily average?**

Without those four, a messages-per-second or messages-per-day figure is marketing rather than engineering.