Report. Benchmarking REST and gRPC for a Distributed Calculator

This experiment compared two communication systems. REST using HTTP and JSON. gRPC using HTTP2 and Protocol Buffers. Both systems were tested for ten seconds per operation with ten active connections. The goal was to compare their latency. payload size. and overall communication overhead. Your benchmark results showed the following general pattern. REST achieved a slightly higher throughput in your local setup. while gRPC used fewer bytes and gave more stable latency percentiles. Both systems completed the full test without errors.

Why the results differ

The difference in performance comes from three areas. First. serialization. REST uses JSON which requires text encoding and decoding for every request. gRPC uses Protocol Buffers which encode numbers and fields in compact binary form. Second. transport. REST runs over HTTP with one request per operation. gRPC runs over HTTP2 which supports streams and reuses one connection. but gRPC also has more client side overhead in Node. especially when many small unary calls are made in loops. Third. libraries. autocannon is a highly optimized load tester for HTTP and JSON. while the gRPC loop in the benchmark is custom code that runs thousands of callbacks. so the benchmark itself adds overhead. Because of these factors. REST appeared faster in this simple test. but this does not reflect real production behavior where gRPC scales better.

Why REST is heavier

Your payload calculation showed the following. REST total size. fifty bytes gRPC total size. thirty eight bytes REST is heavier because JSON is text based. Every field name. number. and value is stored as plain text. HTTP also carries more headers. so each REST call includes more metadata. The server must parse JSON and stringify responses which increases CPU work. REST is simple and flexible. but it always costs more bytes and more processing compared to Protocol Buffers.

Why gRPC is fastest in real systems

Even though your local test shows REST slightly ahead. gRPC is considered the faster and more efficient design when building real microservices. This happens because: gRPC uses HTTP2. which allows multiplexing many calls over one connection. Protocol Buffers are binary. compact. and cheap to encode or decode. gRPC avoids repeated headers. so less data moves across the network. gRPC keeps one long lived connection instead of opening and closing connections. As systems grow. these benefits compound. so gRPC becomes much faster and more scalable than JSON based REST.

Where tRPC fits best

Even though you did not implement tRPC. its place in the ecosystem is clear. tRPC works best inside one codebase. for example a full stack TypeScript project with one server and one client. It gives type safety without schemas or Protocol Buffers. It is very easy to write and maintain in monolithic apps. but it is not ideal for large microservice systems because it depends on TypeScript types and cannot be

used by other languages. tRPC is best for small to medium projects that want simple communication and full type safety.

Which is best for large scale microservices

gRPC is the most suitable choice. The reasons are: binary messages reduce network load HTTP2 streams reduce connection overhead strong typed contracts improve team development protocols work across many languages Large microservice platforms benefit from these properties because they need high throughput. low latency. and reliable communication between many independent services.

Which is best for student projects

REST is the best choice for students because it is simple. universal. and quick to build. It works in browsers. mobile apps. Postman. and any language. JSON responses are easy to debug and understand. REST is also easier to deploy and test locally. For learning high performance systems. gRPC is useful. but REST is more approachable and gives faster development for class work or small assignments.