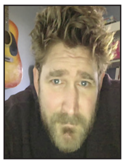
iVoyeur
Distributive Tracing

DAVE JOSEPHSEN



Dave Josephsen is a book author, code developer, and monitoring expert who works for Fastly. His continuing mission: to help engineers worldwide close the feedback loop. dave-usenix@skeptech.org

Last night my niece asked me if I wanted to take the "idiot test." If you're not familiar, the idiot test is a trollish, adolescent joke about zero-indexing, and it goes like this:

Prankster: Do you want to take the idiot test?

Dupe: Sure.

Prankster: What color is the sky?

Dupe: Blue.

Prankster: What do humans breathe?

Dupe: Air.

Prankster: What was the first question I asked you?

Dupe: "What color is the sky."

Prankster: Wrong! It was: "Do you want to take the idiot test!" You failed! <_insert ridicule_> & etc.

Dupe: [*Feels bad about himself*]

As a bonafide grown-up in situations like this, I feel strongly that we have a duty to small children like my niece to consistently fail tests like this. Few things are as formative to a nine year old, I believe, as the sense that you are a contender in the world. And yet, I'd be lying if I said I didn't feel just the slightest pang of childish irritation when I throw one of my niece's harmless little contests of wit.

This may be because I'm self-aware enough to know I'm not immune to making the odd off-by-one error in my day job from time to time, thereby failing the idiot test in earnest. Just the other week I got bit (once again) by Lua, which has odd (read: exasperating) list-indexing behavior. Ah well, few things are as formative to a 40-something as the sense that you are taking yourself too damn seriously.

There's another game, beloved by my niece, who to my chagrin insists on calling it "Chinese Whispers," despite the myriad not-racist names for the same game: whisper down the lane, broken telephone, operator, grapevine, gossip, don't drink the milk, secret message, the messenger game, pass the message, and etc. Between you and me, let's call it: "Telephone."

In this game, players arrange themselves in a circle, and a message passes verbally between them. The first player decides on a message and writes it down. Then, serially, one-by-one, each player whispers the message into the ear of the player to their left, until the message passes all the way back to the first player. Of course, because of the entropic nature of the universe combined with humanity's lack of integrated hash-summing algorithms, the message is corrupted hop-by-hop as it traverses the circle, until it entirely loses its original meaning and becomes an altogether different message (usually somehow now involving poop).

When the original and corrupted messages are openly compared at the end of the game, the group has a good laugh, as if the delta represents something humorous, rather than outright horrifying. It's more fun than I'm probably making it sound.

Tracing

Because I've been spending a lot of my non-existent free time working on the OpenTelemetry project, games like Telephone invariably remind me of Distributed Tracing, where messages, nested within HTTP headers travel between hops like players whispering—albeit with a hopefully more reliable result.

In a previous article (https://www.usenix.org/system/files/login/articles/login_spring18_13_josephsen.pdf), I wrote about another, similar sounding, tracing project, called OpenTracing. In that article I compare HTTP Trace headers to the "Received" header in SMTP, enabling us to apply monitoring to individual hops traversed by a single request. I went on to describe that since HTTP requests can often spawn related non-HTTP requests like database calls, distributed tracing implementations often include mechanisms to enable engineers to embed information about child-requests, metadata, and ad hoc metrics within HTTP headers as a request passes between systems.

For several years now, there have been two quasi-competitive open-source distributed tracing implementations widely used in the wild. The Cloud Native Computing Foundation's OpenTracing (https://opentracing.io) project concerns itself with providing vendor-neutral instrumentation for distributed tracing. OpenTracing provides engineers an API they can use to embed tracing data into their requests, and it leaves the interpretation of that data to pluggable third-party "tracers" like Jaeger (https://www.jaegertracing.io) or Lightstep (https://lightstep.com).

Google's OpenCensus (https://opencensus.io) project, by comparison, provides a more holistic implementation of distributed tracing, complete with performance monitoring and metrics.

The Merger

The last several months have been quite eventful for the distributed tracing community since the announcement in April that the two primary open-source tracing projects, the CNCF's OpenTracing and Google's OpenCensus, are merging to form a single über tracing project called OpenTelemetry.

At the same time (not at all coincidently), the W3C has opened a working group to extend HTTP with a standard header format to propagate context information for distributed tracing scenarios. In other words, HTTP will itself soon have vendor-agnostic distributed tracing built in. The best way to quickly get a sense of what that means and how it will eventually work is to read Alois Reitbauer's write-up (https://medium.com/@AloisReitbauer/trace-context-and-the-road-toward-trace-tool-interoperability-d4d56932369c) on the problems inherent with vendor-specific trace headers and how the W3C plans to work around them.

The group currently has a candidate recommendation (https://www.w3.org/TR/trace-context/), against which many implementations are already coding. You can track this ongoing effort or even help out by joining the team's Slack-channel, available through the group's home page (https://www.w3.org/2018/distributed-tracing/).

Meanwhile OpenTracing and OpenCensus are coming together to form OpenTelemetry at breakneck speed. Driven by an aggressive schedule that includes sunsetting both the OpenTracing and OpenCensus projects by November 2019, a lot of parallel effort is underway to bring myriad language libraries into alpha. You can read the detailed road map and current status in the well-maintained milestones (https://github.com/open-telemetry/opentelemetry-specification/blob/master/milestones.md) document. The spec is available on the project's spec GitHub site (https://github.com/open-telemetry/opentelemetry-specification).

In order to parallelize the workload as much as possible, the project is organized into numerous special-interest groups, including a SIG on cross-language specification, the agent/collector (since the project includes metrics collection as a first-class citizen), and numerous SIG working groups on language-specific SDKs, including those for Java, Golang, Python, .NET, Ruby, and so on.

Most of the SDK SIGs are approaching alpha, and all are in dire need of well-written documentation, GitHub tagging and organization, QA, and, of course, code. If you've ever wanted to dig in to the early stages of an open-source effort that's going to have a huge impact, this is a great time to chip in to an OpenTelemetry Special Interest Group. In a few years, everything from the browser to the database, including the underlying protocol itself, HTTP, is going to support distributed tracing. If you think you might be of assistance, I'd encourage you to take a look at the project's contributing page (https://github.com/open-telemetry/community) and join us on gitter (https://gitter.im/open-telemetry/community).