How and why the leap second affected Cloudflare DNS

2017-01-01

John Graham-Cumming

3 min read

At midnight UTC on New Year's Day, deep inside Cloudflare's custom RRDNS software, a number went negative when it should always have been, at worst, zero. A little later this negative value caused RRDNS to panic. This panic was caught using the recover feature of the Go language. The net effect was that some DNS resolutions to some Cloudflare managed web properties failed.

The problem only affected customers who use CNAME DNS records with Cloudflare, and only affected a small number of machines across Cloudflare's 102 data centers. At peak approximately 0.2% of DNS queries to Cloudflare were affected and less than 1% of all HTTP requests to Cloudflare encountered an error.

This problem was quickly identified. The most affected machines were patched in 90 minutes and the fix was rolled out worldwide by 0645 UTC. We are sorry that our customers were affected, but we thought it was worth writing up the root cause for others to understand.

A little bit about Cloudflare DNS

Cloudflare customers use our DNS service to serve the authoritative answers for DNS queries for their domains. They need to tell us the IP address of their origin web servers so we can contact the servers to handle non-cached requests. They

do this in two ways: either they enter the IP addresses associated with the names (e.g. the IP address of example.com is 192.0.2.123 and is entered as an A record) or they enter a CNAME (e.g. example.com is origin-server.example-hosting.biz).

This image shows a test site with an A record for `theburritobot.com` and a CNAME for `www.theburritobot.com` pointing directly to Heroku.

When a customer uses the CNAME option, Cloudflare has occasionally to do a lookup, using DNS, for the actual IP address of the origin server. It does this automatically using standard recursive DNS. It was this CNAME lookup code that contained the bug that caused the outage.

Internally, Cloudflare operates DNS resolvers to lookup DNS records from the Internet and RRDNS talks to these resolvers to get IP addresses when doing CNAME lookups. RRDNS keeps track of how well the internal resolvers are performing and does a weighted selection of possible resolvers (we operate multiple per data center for redundancy) and chooses the most performant. Some of these resolutions ended up recording in a data structure a negative value during the leap second.

The weighted selection code, at a later point, was being fed the negative number which caused it to panic. The negative number got there through a combination of the leap second and smoothing.

A falsehood programmers believe about time

The root cause of the bug that affected our DNS service was the belief that *time cannot go backwards*. In our case, some code assumed that the *difference* between two times would always be, at worst, zero.

RRDNS is written in Go and uses Go's time.Now() function to get the time. Unfortunately, this function does not guarantee monotonicity. Go currently doesn't offer a monotonic time source (see issue 12914 for discussion).

In measuring the performance of the upstream DNS resolvers used for CNAME lookups RRDNS contains the following code:

```
// Update upstream sRTT on UDP queries, penalize it if it fails
if !start.IsZero() {
    rtt := time.Now().Sub(start)
    if success && rcode != dns.RcodeServerFailure {
        s.updateRTT(rtt)
    } else {
        // The penalty should be a multiple of actual timeout
        // as we don't know when the good message was supposed to arrive,
        // but it should not put server to backoff instantly
        s.updateRTT(TimeoutPenalty * s.timeout)
    }
}
```

In the code above `rtt` could be negative if time.Now() was earlier than `start` (which was set by a call to `time.Now()` earlier).

That code works well if time moves forward. Unfortunately, we've tuned our resolvers to be very fast which means that it's normal for them to answer in a few milliseconds. If, right when a resolution is happening, time goes back a second the perceived resolution time will be *negative*.

RRDNS doesn't just keep a single measurement for each resolver, it takes many measurements and smoothes them. So, the single measurement wouldn't cause RRDNS to think the resolver was working in negative time, but after a few measurements the smoothed value would eventually become negative.

When RRDNS selects an upstream to resolve a CNAME it uses a weighted selection algorithm. The code takes the upstream time values and feeds them to Go's rand.Int63n() function. `rand.Int63n` promptly panics if its argument is negative. That's where the RRDNS panics were coming from.

(Aside: there are many other falsehoods programmers believe about time)

The one character fix

One precaution when using a non-monotonic clock source is to always check whether the difference between two timestamps is negative. Should this happen, it's not possible to accurately determine the time difference until the clock stops rewinding.

In this patch we allowed RRDNS to forget about current upstream performance, and let it normalize again if time skipped backwards. This prevents leaking of negative numbers to the server selection code, which would result in throwing errors before attempting to contact the upstream server.

The fix we applied prevents the recording of negative values in server selection. Restarting all the RRDNS servers then fixed any recurrence of the problem.

Timeline

The following is the complete timeline of the events around the leap second bug.

2017-01-01 00:00 UTC Impact starts2017-01-01 00:10 UTC Escalated to engineers2017-01-01 00:34 UTC Issue confirmed2017-01-01 00:55 UTC Mitigation deployed to one canary node and confirmed2017-01-01 01:03 UTC Mitigation deployed to canary data center and confirmed2017-01-01 01:23 UTC Fix deployed in most impacted data center2017-01-01 01:45 UTC Fix being deployed to major data centers2017-01-01 01:48 UTC Fix being deployed

everywhere2017-01-01 02:50 UTC Fix rolled out to most of the affected data centers2017-01-01 06:45 UTC Impact ends

This chart shows error rates for each Cloudflare data center (some data centers were more affected than others) and the rapid drop in errors as the fix was deployed. We deployed the fix prioritizing those locations with the most errors first.

Conclusion

We are sorry that our customers were affected by this bug and are inspecting all our code to ensure that there are no other leap second sensitive uses of time intervals.

Discuss on Hacker News

DNS Post Mortem RRDNS Bugs Reliability

Follow on X

Cloudflare | @cloudflare

RELATED POSTS

May 27, 2026

Iran's Internet is partially restored, Cloudflare Radar data shows

Cloudflare Radar data confirms early indications of a partial Internet restoration in Iran, nearly three months after the shutdown began. Traffic spikes and DNS queries have risen, but network activity is currently just 40% of pre-shutdown levels....

By Lai Yi Ohlsen, Sabina Zejnilovic

Internet Traffic, Internet Trends, Internet Quality, Internet Shutdown, Radar, DNS

May 06, 2026

When DNSSEC goes wrong: how we responded to the .de TLD outage

On May 5, 2026, DENIC published broken DNSSEC signatures for the .de TLD, making millions of domains unreachable. Here's what 1.1.1.1 saw, how serve stale cushioned the impact, and how we restored resolution....

By Sebastiaan Neuteboom, Christian Elmerot, Max Worsley

DNS, DNSSEC, 1.1.1.1, Reliability, Outage

May 01, 2026

Code Orange: Fail Small is complete. The result is a stronger Cloudflare network

We have completed a massive engineering effort to make our infrastructure more resilient. Through new tools like Snapstone and the Engineering Codex, we've implemented safer configuration changes and automated best practices to prevent future incidents....

By Jeremy Hartman

Outage, Post Mortem, Code Orange

April 22, 2026

Making Rust Workers reliable: panic and abort recovery in wasm-bindgen

Panics in Rust Workers were historically fatal, poisoning the entire instance. By collaborating upstream on the wasm-bindgen project, Rust Workers now support resilient critical error recovery, including panic unwinding using WebAssembly Exception Handling....

By Guy Bedford, Hood Chatham, Logan Gatlin

Cloudflare Workers, Rust, Rust Workers, WebAssembly, WASM, Reliability, Engineering, Open Source, Developer Platform, Developers