Four years of Go at CloudFlare

John Graham-Cumming

CloudFlare

You likely use us without knowing it…

CLOUDFLARE

Server Languages 2011

PHP

www backend

data pipeline

request logic

Server Languages 2011

PHP

- www backend
- data pipeline
- request logic

C

- nginx modules

CLOUDFLARE

Server Languages 2011

PHP

- www backend
- data pipeline
- request logic

C

- nginx modules

C++

- data pipeline

CLOUDFLARE

Server Languages 2015

PHP

www
backend

Server Languages 2015

Lua

request logic

ngx_lua/OpenResty

PHP

www backend

CLOUDFLARE

Server Languages 2015

Lua — request logic

PHP — www backend

C — nginx modules

C++ eliminated

CLOUDFLARE

Server Languages 2015

Go — data pipeline

Lua — request logic

PHP — www backend

C — nginx modules

C++ eliminated

CLOUDFLARE

Server Languages 2015

Go — data pipeline — Lots of new things!

Lua — request logic

PHP — www backend

C — nginx modules

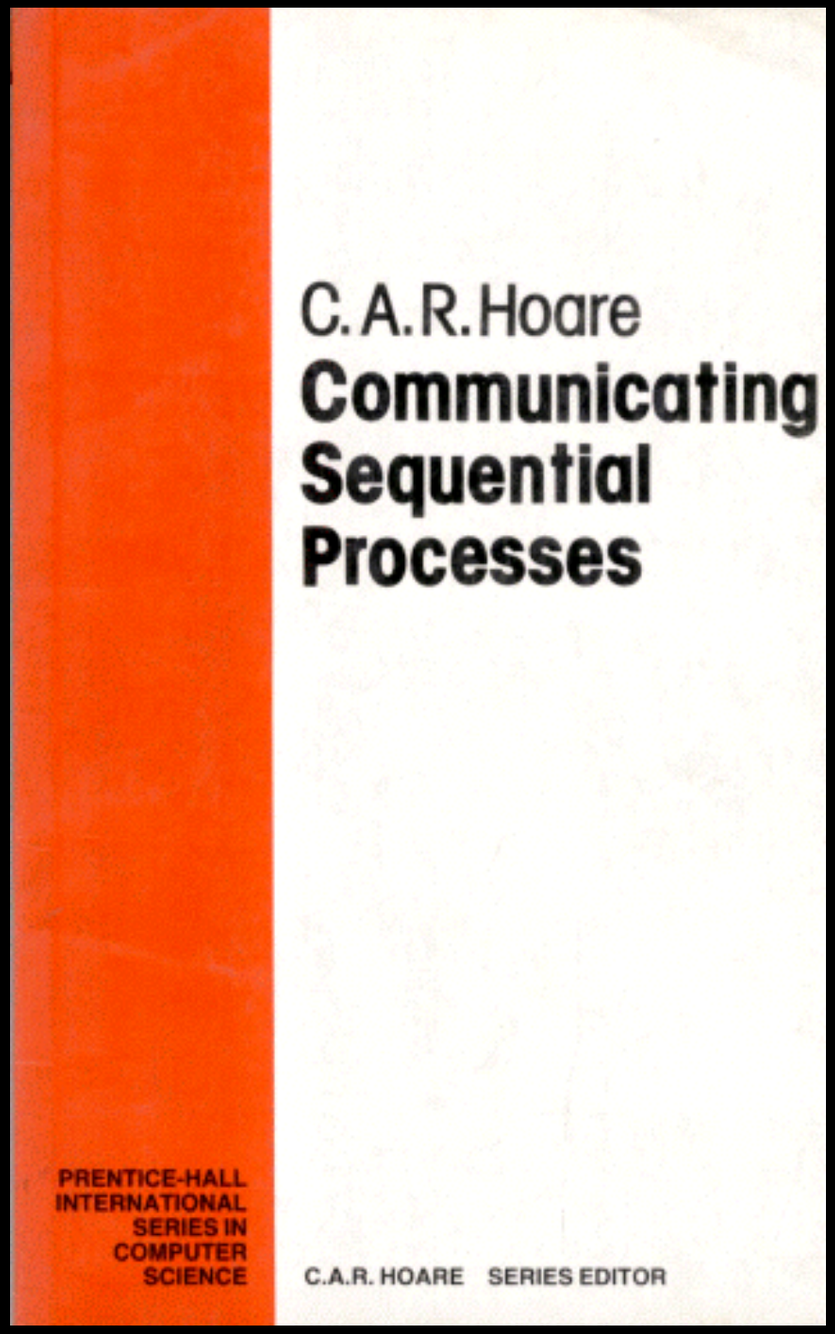
C++ eliminated

CLOUDFLARE

People often ask me:

"How did you persuade CloudFlare to use Go?"

Back in mid-1980s...



A couple of years later...

So far we have developed a simple file–store which services requests along a pair of channels. We now develop a process P_i which takes requests from any user u_j along channel $i.j.in$ and returns replies along $i.j.out$. Requests which are disallowed, because of the restrictions 'no read–up' and 'no write–down', return the message **er**; but valid requests are passed to the local file–store F_i and its replies are forwarded to the user who made the request.

Definition 5.16

$$\begin{aligned}
\alpha\mathsf{P}_i \triangleq\ & \alpha\mathsf{F}_i \\
& \cup \\
& \{i.j.in.(c,f,d) \mid (c,f,d) \in OP \wedge u_j \in USR\} \\
& \cup \\
& \{i.j.out.m \mid m \in DAT \cup MSG \wedge u_j \in USR\};
\end{aligned}$$

$$\mathsf{P}_i \triangleq \mathop{[\!]}_{u_j \in USR} i.j.in?(c,f,d) \rightarrow (\mathsf{R}_i^j(f,d) \ \mathrm{C}\ c = \mathbf{rd} \ \mathrm{B}\ \mathsf{W}_i^j(f,d));$$

$$\mathsf{R}_i^j(f,d) \triangleq (i.j.out!\mathbf{er} \rightarrow \mathsf{P}_i \ \mathrm{C}\ j < i \ \mathrm{B}\ i.j.Fin!(\mathbf{rd},f,d) \rightarrow \mathsf{P}_i^j);$$

$$\mathsf{W}_i^j(f,d) \triangleq (i.j.out!\mathbf{er} \rightarrow \mathsf{P}_i \ \mathrm{C}\ j > i \ \mathrm{B}\ i.j.Fin!(\mathbf{wr},f,d) \rightarrow \mathsf{P}_i^j);$$

CLOUDFLARE

A couple of years later...

So far we have developed a simple file–store which services requests along a pair of channels. We now develop a process P_i which takes requests from any user u_j along channel $i.j.in$ and returns replies along $i.j.out$. Requests which are disallowed, because of the restrictions 'no read–up' and 'no write–down', return the message **er**; but valid requests are passed to the local file–store F_i and its replies are forwarded to the user who made the request.

Definition 5.16

$$
\begin{aligned}
\alpha\mathsf{P}_i \triangleq\ & \alpha\mathsf{F}_i \\
& \cup \\
& \{i.j.in.(c,f,d) \mid (c,f,d) \in OP \wedge u_j \in USR\} \\
& \cup \\
& \{i.j.out.m \mid m \in DAT \cup MSG \wedge u_j \in USR\};
\end{aligned}
$$

$$
\mathsf{P}_i \triangleq \mathop{[\!]}_{u_j \in USR} i.j.in?(c,f,d) \rightarrow (\mathsf{R}_i^j(f,d) \ \mathrm{C}\ c = \mathbf{rd}\ \mathrm{B}\ \mathsf{W}_i^j(f,d));
$$

$$
\mathsf{R}_i^j(f,d) \triangleq i.j.out!\mathbf{er} \rightarrow \mathsf{P}_i \ \mathrm{C}\ j < i\ \mathrm{B}\ i.j.Fin!(\mathbf{rd},f,d) \rightarrow \mathsf{P}_i^j);
$$

$$
\mathsf{W}_i^j(f,d) \triangleq (i.j.out!\mathbf{er} \rightarrow \mathsf{P}_i \ \mathrm{C}\ j > i\ \mathrm{B}\ i.j.Fin!(\mathbf{wr},f,d) \rightarrow \mathsf{P}_i^j);
$$

CLOUDFLARE

occam

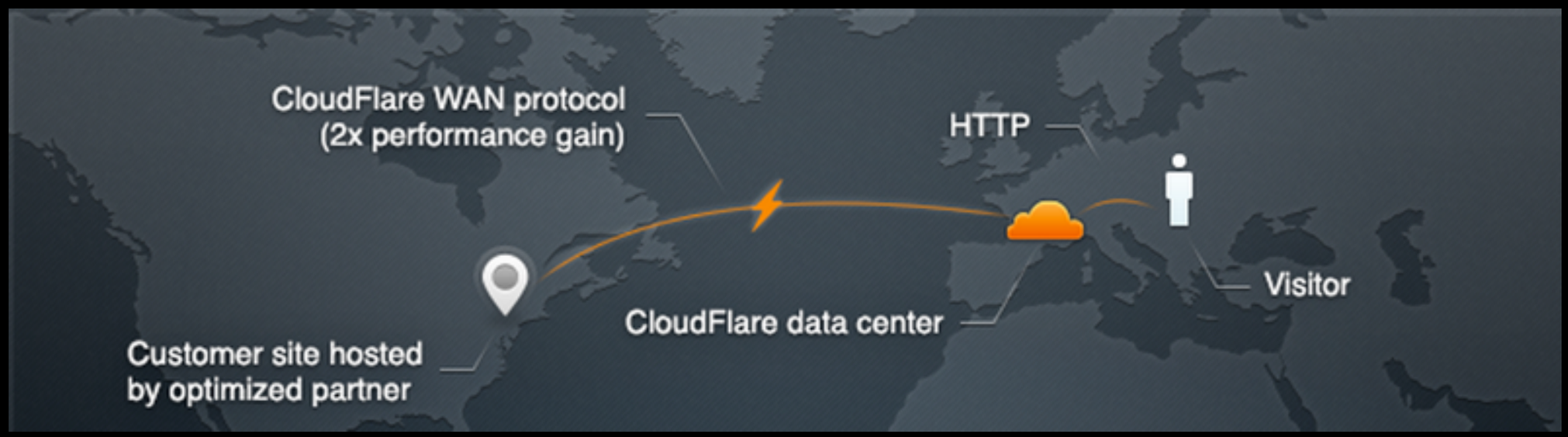
```
PROC procD(CHAN rep[], req[], VALUE i) =

  CHAN F.in[nu], F.out[nu] :

  PROC procF(CHAN in[], out[]) =

    VAR s[nf],
        cmd[3] :

    SEQ
      SEQ f = [0 FOR nf]
        s[f] := -1
      WHILE TRUE
        ALT j = [0 FOR nu]
          in[j] ? cmd[0 FOR 2]
            IF
              cmd[0] = Rd
                IF
                  s[cmd[1]] < 0
                    out[j] ! Er
                  TRUE
                    out[j] ! s[cmd[1]]
              cmd[0] = Wr
                SEQ
                  s[cmd[1]] := cmd[2]
                  out[j] ! Ok :

PROC procP(CHAN rep[], req[], in[], out[], VALUE i) =

  VAR cmd[3],
      msg,
```

OK

So, I'm a concurrent-processes-communicating-via-channels hipster

Railgun (2012)

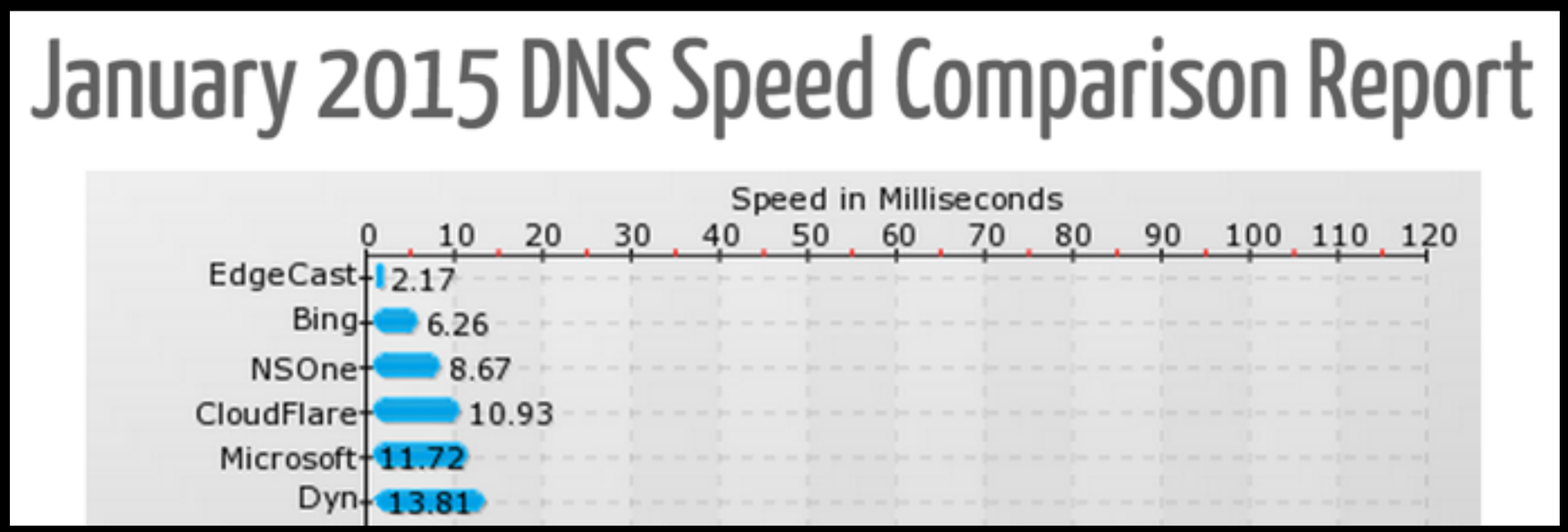
- First major Go project at CloudFlare
  - Wrote prototype in Perl
- Delta-compression based acceleration for origin connections
- 16,626 lines of Go; 2,518 lines of Go tests
- Single executable deployment == happiness



CLOUDFLARE

RRDNS (2013)

- Authoritative DNS server — has to be fast!
- Also does DNS proxying and DNSSEC
- 15,143 lines of Go; 9,377 lines of Go tests



CLOUDFLARE

CFSSL (2014)

- TLS/SSL/PKI Toolkit
  - Can be used as a CA
  - Does optimal cert bundling
- https://github.com/cloudflare/cfssl

CLOUDFLARE

Polish (2014)

- Automatic image recompression
- Recompresses images in CloudFlare cache in background
- Uses cgo to talk to FreeImage and shells out to pngcrush, etc.
- 5,977 lines of Go; 3,635 lines of Go tests

Data (2014)

- CloudFlare's event log aggregation and analysis
  - Event log forwarding
  - Attack analysis
- Use Cap'n Proto instead of gob
- 103,986 lines of Go; 36,585 lines of Go tests

Started using Go pre-v1.0

- First Go project shipped using go1.0.2 (mid-2012)

Started using Go pre-v1.0

- First Go project shipped using go1.0.2 (mid-2012)
- crypto and compression were *slow*
  - used cgo to call C/assembler code

Started using Go pre-v1.0

- First Go project shipped using go1.0.2 (mid-2012)
- crypto and compression were *slow*
  - used cgo to call C/assembler code
- log/syslog was “odd”

Started using Go pre-v1.0

- First Go project shipped using go1.0.2 (mid-2012)
- crypto and compression were *slow*
  - used cgo to call C/assembler code
- log/syslog was “odd”
- Worst problem… on BSD

```
void runtime·SysUnused(void *v, uintptr n)
{
        USED(v);
        USED(n);
        // TODO(rsc): call madvise MADV_DONTNEED
}
```

What we write in Go

- “Production Lines”
  - Small programs that do small jobs and need to scale up and down
  - Polish, data pipeline

What we write in Go

- “Production Lines”
  - Small programs that do small jobs and need to scale up and down
  - Polish, data pipeline
- “Crannies”
  - Small programs that run periodically but sometimes need a kick

What we write in Go

- “Production Lines”
  - Small programs that do small jobs and need to scale up and down
  - Polish, data pipeline
- “Cronnies”
  - Small programs that run periodically but sometimes need a kick
- “Network I/O”
  - Programs that do a lot of network I/O
  - RRDNS, Railgun

What we write in Go

- “Production Lines”
  - Small programs that do small jobs and need to scale up and down
  - Polish, data pipeline
- “Cronnies”
  - Small programs that run periodically but sometimes need a kick
- “Network I/O”
  - Programs that do a lot of network I/O
  - RRDNS, Railgun
- “Crypto Things”
  - Internal cryptographic tools

What we didn't write in Go

- Keyless SSL
  - Wanted to link against OpenSSL
  - 4,175 lines of C; 2,044 lines of tests

Seemed like a good idea at the time!

What we didn’t write in Go

- Keyless SSL
  - Wanted to link against OpenSSL
  - 4,175 lines of C; 2,044 lines of tests

Uses libuv for concurrency

CLOUDFLARE

Aside: Shakespeare as OpenSSL programmer

MACDUFF
O horror, horror, horror!
Tongue nor heart cannot conceive nor name thee!

MACBETH & LENNOX
What's the matter?

MACDUFF
Confusion now hath made his masterpiece.

What we didn't write in Go

- Keyless SSL
  - Wanted to link against OpenSSL
  - 4,175 lines of C; 2,044 lines of tests
- "Request Logic"
  - nginx integration with LuaJIT excellent and fast
- www backend
  - Lots of legacy PHP code

What we didn't write in Go

- Keyless SSL
  - Wanted to link against OpenSSL
  - 4,175 lines of C; 2,044 lines of tests
- "Request Logic"
  - nginx integration with LuaJIT excellent and fast
- www backend
  - Lots of legacy PHP code

Today's Pain Points

- Garbage Collection
  - Stop the world is a world of hurt
  - Spend significant effort on eliminating garbage creation
- Third-party import management

Garbage Elimination

- Top tips (thanks Daniel Morsing!)
  - Use `io.Reader`/`io.Writer`
  - Don’t allocate small structs; pass them by value
  - Use arrays for buffers instead of slices when you can
- Profile!

Lots of little programs

- `torhoney`: get list of Tor exit nodes, score against Project Honeypot

- https://github.com/cloudflare/golibs

  - bytepool, circularbuffer, ewma, lrucace, pool, spacesaving

- http://cloudflare.github.io/#cat-Go