Cap'n Proto

Cap’n Proto is a data serialization format and Remote Procedure Call (RPC) framework for exchanging data between computer programs. The high-level design focuses on speed and security, making it suitable for network as well as inter-process communication. Cap'n Proto was created by the former maintainer of Google's popular Protocol Buffers framework (Kenton Varda) and was designed to avoid some of its perceived shortcomings.

Cap'n Proto

Original author(s)	Kenton Varda
Stable release	0.8 / April 23, 2020
Repository	github.com/capnproto/capnproto (https://github.com/capnproto/capnproto)
Written in	C++
Type	Remote procedure call framework, serialization format and library, IDL compiler
License	MIT License
Website	capnproto.org (https://capnproto.org)

Contents

Technical overview

IDL Schema

Like most RPC frameworks dating as far back as Sun RPC and OSF DCE RPC (and their object-based descendants CORBA and DCOM), Cap'n Proto uses an Interface Description Language (IDL) to generate RPC libraries in a variety of programming languages - automating many low level details such as handling network requests, converting between data types, etc. The Cap'n Proto interface schema uses a C-like syntax and supports common primitives data types (booleans, integers, floats, etc.), compound types (structs, lists, enums), as well as generics and dynamic types.[1] Cap'n Proto also supports Object Oriented features such as multiple inheritance, which has been criticized for its complexity.[2]

```
@0xa558ef006c0c123; #Unique identifiers are manually or automatically assigned to files and
compound types

struct Date @0x5c5a558ef006c0c1 {
  year @0 :Int16; #@n marks order values were added to the schema
  month @1 :UInt8;
  day @2 :UInt8;
}

struct Contact @0xf032a54bcb3667e0 {
  name @0 :Text;
  birthday @2 :Date; #fields can be added anywhere in the definition, but their numbering
must reflect the order in which they were added
```

```
  phones @1 :List(PhoneNumber);

  struct PhoneNumber { #Compound types without an static ID cannot be renamed, as automatic
IDs are determinstically generated
    number @0 :Text;
    type @1 :PhoneType = mobile; #Default value

    enum PhoneType {
      mobile @0;
      landline @1;
    }
  }
}
```

Values in Cap'n Proto messages are represented in binary, as opposed text encoding used by "human-readable" formats such as JSON or XML. Cap'n Proto tries to make the storage/network protocol appropriate as an in-memory format, so that no translation step is needed when reading data into memory or writing data out of memory.[note 1] For example, the representation of numbers (endianness) was chosen to match the representation the most popular CPU architectures.[3] When the in-memory and wire-protocol representations match, Cap'n Proto can avoid copying and encoding data when creating or reading a message and instead point to the location of the value in memory. Cap'n Proto also supports random access to data, meaning that any field can be read without having to read the entire message.[4]

Unlike other binary serialization protocols such as XMI, Cap'n Proto considers fine-grained data validation at the RPC level an anti-feature that limits a protocols ability to evolve. This was informed by experiences at Google where simply changing a field from *mandatory* to *optional* would cause complex operational failures.[5][note 2] Cap'n Proto schemas are designed to be flexible as possible and pushes data validation to the application level, allowing arbitrary renaming of fields, adding new fields, and making concrete types generic[6] Cap'n Proto does, however, validate pointer bounds and type check individual values when they are first accessed.[4]

Enforcing complex schema constraints would also incur significant overhead,[note 3] negating the benefits of reusing in-memory data structures and preventing random access to data.[7] Cap'n Proto protocol is *theoretically* suitable[8] for very fast inter-process communication (IPC) via immutable shared memory, but as of October 2020 none of the implementations support data passing via shared memory.[9] However, Cap'n Proto is still generally considered faster than Protocol Buffers and similar RPC libraries.[10][11]

Networking

Cap'n Proto RPC is network aware: supporting both handling of disconnects and promise pipelining, wherein a server pipes the output of one function into another function. This saves a client a round trip per successive call to the server without having to provide a dedicated API for every possible call graph. Cap'n Proto can be layered on top of TLS[12] and support for the Noise Protocol Framework is on the roadmap.[13] Cap'n Proto RPC is transport agnostic, with the mainline implementation supporting WebSockets, HTTP, TCP, and UDP.[14]

Capability security

The Cap'n Proto RPC standard has a rich capability security model based on the CapTP protocol used by the E programming language.[15]

As of October 2020, the reference implementation only supports level 2.[13]

Adoption

Cap'n Proto was originally created for Sandstorm.io, a startup offering a web application hosting platform with capability-based security. After Sandstorm.io failed commercially, the development team was acqui-hired by Cloudflare;[16] which uses Cap'n Proto internally.[17]

While Cap'n Proto has code generators in a variety of languages, the immaturity of the implementations and relatively smaller number (compared to Protocol Buffers or FlatBuffers) is frequently cited as a barrier to adoption.[18][19]

Notes

1. Unlike Apache Arrow, Cap'n Proto's in-memory values are not suited for sharing mutable data (https://news.ycombinator.com/item?id=14248780)
2. Marking a field as required was removed from Protocol Buffers 3 (https://developers.google.com/protocol-buffers/docs/proto3).
3. *Assuming the data has already been allocated* (e.g. in network buffers, read from disk) access becomes O(1). Additional serialization/deserialization steps (as required to inspect values) would limit performance to O(n).

References

1. Varda, Kenton. "Cap'n Proto Schema Language" (https://capnproto.org/language.html). Archived (https://web.archive.org/web/20150317184515/https://capnproto.org/language.html) from the original on 2015-03-17. Retrieved 2020-09-05.
2. Denhardt, Ian (June 2019). "A Critique of the Cap'n Proto Schema Language" (https://zenhack.net/2019/06/25/a-critique-of-the-capnproto-schema-language.html). *zenhack.net*. Archived (https://web.archive.org/web/20190626022038/https://zenhack.net/2019/06/25/a-critique-of-the-capnproto-schema-language.html) from the original on 2019-06-26. Retrieved 2020-10-10.
3. Varda, Kenton. "Cap'n Proto: Introduction" (https://capnproto.org/index.html). *Cap'n Proto Homepage*. Archived (https://web.archive.org/web/20150317185148/https://capnproto.org/index.html) from the original on 2015-03-17. Retrieved 2020-11-09.
4. Varda, Kenton. "Cap'n Proto: Encoding Spec" (https://capnproto.org/encoding.html). *Cap'n Proto*. Archived (https://web.archive.org/web/20150317202732/https://capnproto.org/encoding.html) from the original on 2015-03-17.
5. Varda, Kenton. "FAQ § How do I make a field "required", like in Protocol Buffers?" (https://capnproto.org/faq.html#how-do-i-make-a-field-required-like-in-protocol-buffers). *Cap'n Proto*. Archived (https://web.archive.org/web/20150318011733/https://capnproto.org/faq.html) from the original on 2015-03-18. Retrieved 2020-09-05.
6. "Cap'n Proto: Schema Language" (https://capnproto.org/language.html#evolving-your-protocol). *capnproto.org*. Retrieved 2020-10-10.
7. "Cap'n Proto: Cap'n Proto, FlatBuffers, and SBE" (https://capnproto.org/news/2014-06-17-capnproto-flatbuffers-sbe.html). *capnproto.org*. Retrieved 2020-10-10.
8. Richardson, Corey (October 2016). "Robigalia: An Operating System for the Modern Era" (https://robigalia.gitlab.io/book/rosme.html#_capabilities_in_phoma). *robigalia.gitlab.io*. Archived (https://web.archive.org/web/20180915204543/https://robigalia.gitlab.io/book/rosme.html) from the original on 2018-09-15. Retrieved 2020-10-10.

9. Kenton, Varda (May 3, 2017). "Why is not intended that in-memory state be in Cap'n Proto / Protobuf objects?" (https://news.ycombinator.com/item?id=14252940). *Hacker News (news.ycombinator.com)*. Retrieved 2020-10-10.
10. Naughton, Chris (Aug 24, 2018). "Protocol Benchmarks" (https://github.com/ChrisMacNaughton/proto_benchmarks). *Github*. Archived (https://web.archive.org/web/20180830220546/https://github.com/ChrisMacNaughton/proto_benchmarks) from the original on 2018-08-30. Retrieved 2020-09-05.
11. Parimi, Dinesh (2019). "Datacenter Tax Cuts: Improving WSC Efficiency Through Protocol Buffer Acceleration" (https://people.eecs.berkeley.edu/~kubitron/courses/cs262a-F19/projects/reports/project4_report_ver2.pdf) (PDF). Archived (https://archive.today/ADmtF) from the original on 2020-09-05. Retrieved 2020-09-05.
12. "Cap'n Proto: Road Map" (https://capnproto.org/roadmap.html). *capnproto.org*. Retrieved 2020-10-10.
13. "Roadmap" (https://capnproto.org/roadmap.html#rpc-protocol-features). *Cap'n Proto*. 2021-03-13. Archived (https://web.archive.org/web/20150317192558/https://capnproto.org/roadmap.html) from the original on 2015-03-17.
14. "Cap'n Proto: C++ RPC" (https://capnproto.org/cxxrpc.html). *capnproto.org*. Retrieved 2020-10-10.
15. "RPC Protocol" (https://capnproto.org/rpc.html#protocol-features). *Cap'n Proto*. Archived (https://web.archive.org/web/20150318011754/https://capnproto.org/rpc.html) from the original on 2015-03-18.
16. Varda, Kenton (13 Mar 2017). "The Sandstorm Team is joining Cloudflare" (https://sandstorm.io/news/2017-03-13-joining-cloudflare). *Sandstorm.io*. Archived (https://web.archive.org/web/20170313180618/https://sandstorm.io/news/2017-03-13-joining-cloudflare) from the original on 2017-03-13. Retrieved 2020-09-05.
17. Zhi, Jiale (2013). "Introducing lua-capnproto: better serialization in Lua" (https://blog.cloudflare.com/introducing-lua-capnproto-better-serialization-in-lua/). Archived (https://web.archive.org/web/20140306184242/http://blog.cloudflare.com:80/introducing-lua-capnproto-better-serialization-in-lua) from the original on 2014-03-06. Retrieved 2020-09-05.
18. "gRPC in Production" (https://news.ycombinator.com/item?id=14824477). *Hacker News*. July 2017.
19. "reddit/r/rust: Learning Cap'n Proto RPC" (https://www.reddit.com/r/rust/comments/2yp7cb/learning_capn_proto_rpc/). 2015. Archived (https://web.archive.org/web/20150316044554/https://www.reddit.com/r/rust/comments/2yp7cb/learning_capn_proto_rpc/) from the original on 2015-03-16.
