gRPC - A solution for RPCs by Google

Distributed Systems Seminar at Charles University in Prague, Nov 2016

Jan Tattermusch - gRPC Software Engineer

Google Cloud Platform

About me

- Software Engineer at Google (since 2013)
- Working on gRPC since Q4 2014
- Graduated from Charles University (2010)

Contacts

- jtattermusch on GitHub
- Feedback to jtattermusch@google.com

Motivation: gRPC

Google has an internal RPC system, called Stubby

- All production applications use RPCs
- Over 10^{10} RPCs per second in total
- 4 generations over 13 years (since 2003)
- APIs for C++, Java, Python, Go

What's missing

- Not suitable for external use (tight coupling with internal tools & infrastructure)
- Limited language & platform support
- Proprietary protocol and security
- No mobile support

What's gRPC

- HTTP/2 based RPC framework
- Secure, Performant, Multiplatform, Open

Multiplatform

- Idiomatic APIs in popular languages (C++, Go, Java, C#, Node.js, Ruby, PHP, Python)
- Supports mobile devices (Android Java, iOS Obj-C)
- Linux, Windows, Mac OS X
- (web browser support in development)

OpenSource

- developed fully in open on GitHub: https://github.com/grpc/

Use Cases

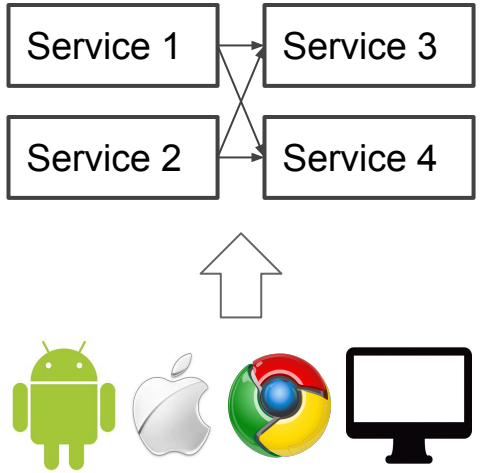
Build distributed services (microservices)

- In public/private cloud
- Google's own services

Client-server communication

- Mobile
- Web
- Also: Desktop, embedded devices, IoT

Access APIs (Google, OSS)



Key Features

- Streaming, Bidirectional streaming
- Built-in security and authentication
  - SSL/TLS, OAuth, JWT access
- Layering on top of HTTP/2 standard
  - Performance: Binary protocol, Stream multiplexing
  - Interoperability with 3rd party proxies, tools, libraries...
- Flow control
- Rich features
  - Load balancing, Tracing, Tooling ecosystem (cmdline tool)...

Detour: Google Protocol Buffers

- Lingua franca for representation of structured data at Google
- Provides an IDL and serialization format for gRPC (one can still opt-out)
- Open-sourced in 2008 and being improved since then
- Language & Platform Neutral
- Extensible (and backward compatible)
- Much more efficient than XML or JSON (space & parsing speed)

```
message Person {
 string name = 1;
 int32 id = 2;
 string email = 3;
 repeated PhoneNumber phones = 4;
}
```

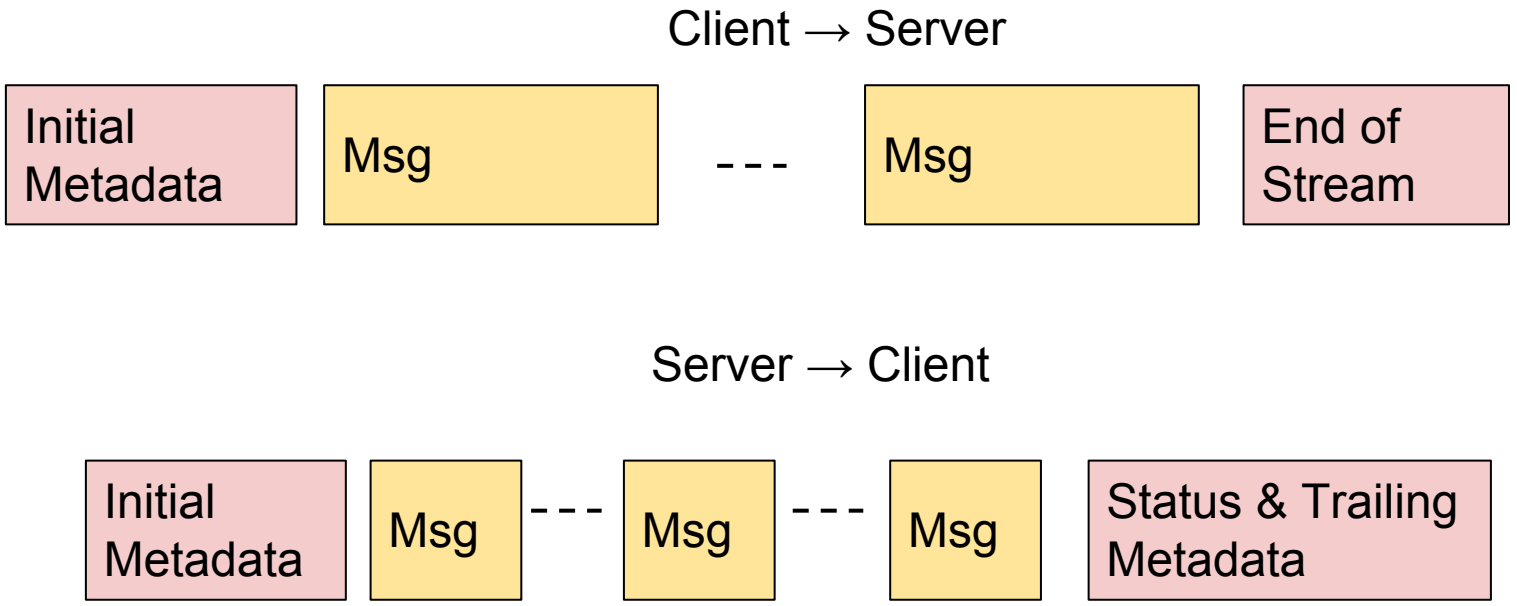
Protocol Buffers: Messages

```
message Person {
  string name = 1;
  int32 id = 2;
  string email = 3;
  repeated PhoneNumber phones = 4;
}

message PhoneNumber {
  string number = 1;
  PhoneType type = 2;
}

enum PhoneType {
  MOBILE = 0;
  HOME = 1;
  WORK = 2;
}
```

Protocol Buffers: Services

```
service Greeter {
 rpc SayHello (HelloRequest) returns (HelloResponse) {}
}

service RouteGuide {
 rpc GetFeature(Point) returns (Feature) {}
 rpc ListFeatures(Rectangle) returns (stream Feature) {}
 rpc RecordRoute(stream Point) returns (RouteSummary) {}
 rpc RouteChat(stream RouteNote) returns (stream RouteNote) {}
}
```

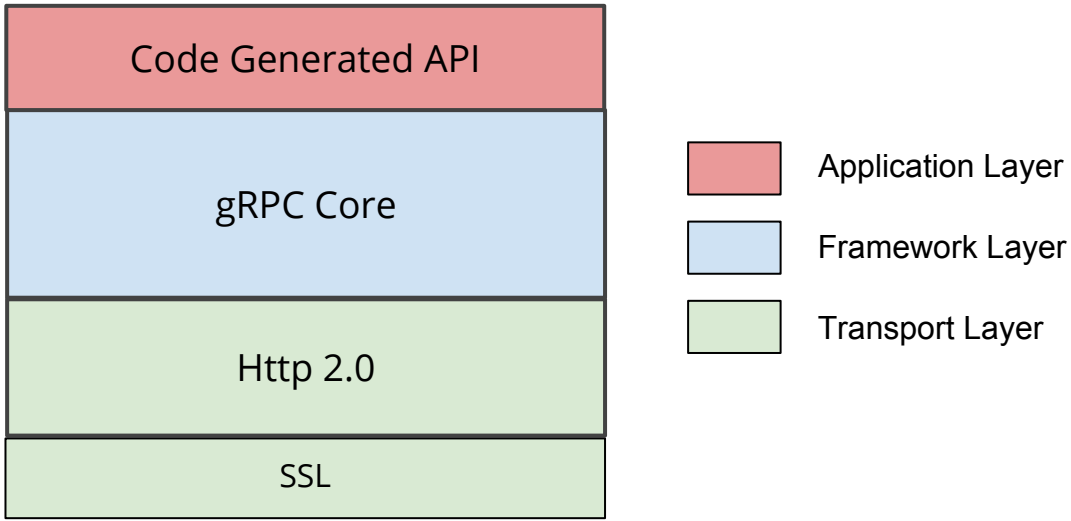
gRPC Concepts: Core Protocol



Architecture: Native stack

Full stack implementations

- C/C++
- Java
- Go

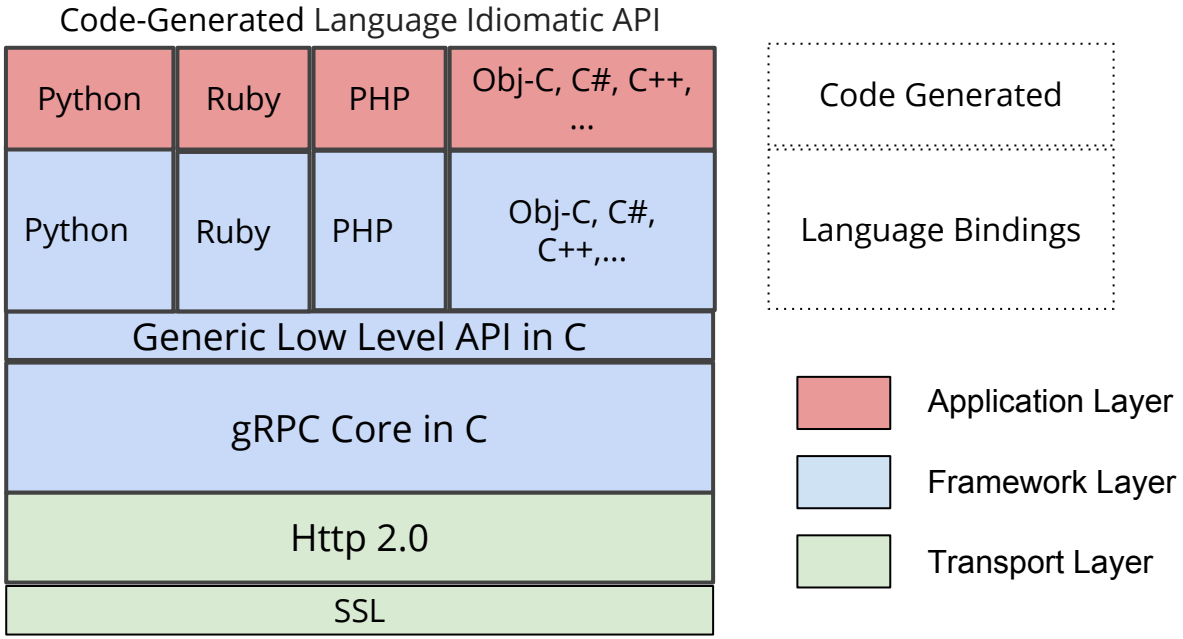


Architecture: "Wrapped" stack

C#, Node.js, Ruby, PHP, Python, Obj-C

Benefits

- Code sharing
- Interoperability
- Performance
- Security
- Team structure



Example: C# client

```
Channel channel = new Channel("127.0.0.1:50051", ChannelCredentials.Insecure);

var client = new Greeter.GreeterClient(channel);


String user = "you";

var reply = client.SayHello(new HelloRequest { Name = user });


Console.WriteLine("Greeting: " + reply.Message);
```

Example: C# server 1

```
Server server = new Server

{

    Services = { Greeter.BindService(new GreeterImpl()) },

    Ports = { new ServerPort("localhost", Port, ServerCredentials.Insecure) }

};

server.Start();
```

Example: C# server 2

```
class GreeterImpl : Greeter.GreeterBase
{
    // Server side handler of the SayHello RPC
    public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)
    {
        return Task.FromResult(new HelloReply { Message = "Hello " + request.Name });
    }
}
```

Example: C# server streaming

```
var call = client.SubscribeForUpdates(request);

var responseStream = call.ResponseStream;

while (await responseStream.MoveNext())

{

    SubscribeResponse update = responseStream.Current;

    Console.WriteLine("Received update: " + update.ToString());

}
```

Example

Tutorials in all languages are available on http://grpc.io

Current Status

We've launched GA in August 2016!

- Basic features in all languages + stable API
- Easy installation
- Stability
- Baseline performance
- In production with Google APIs: Cloud Bigtable, Cloud PubSub, Speech, ...
  - Client libraries available in several languages
- In production with various apps: Allo, Duo
- Used by many external companies/projects:
  - OSS: etcd, Docker containerd, cockroachdb
  - Square, Netflix, YikYak, Carbon 3D, Lyft
  - Cisco, Juniper, Arista

What's Next

Exciting times are coming:

- Usability improvements
- Better performance
- More Google APIs accessible through gRPC
- More internal Google services running on gRPC
- More external adoption
- Bigger ecosystem around gRPC (Google, OSS)
- Rich features

What's next: Rich Features

- Command Line Tool
- Tracing
- Load Balancing
- Retries
- Customizable name resolution
- Compression
- Resource Limits
- RPC Fairness
- ...

Performance

- Different priority for different languages
  - "scalable languages": C++, Java, Go, C#
- What we measure
  - Latency & Throughput
  - Unary & Streaming
  - 8core & 32core
- Public dashboard continuously populated with benchmark results
  - data based on freshest upstream/master
  - see improvements, track regressions

Performance: cont'd

Latency (secure connection)

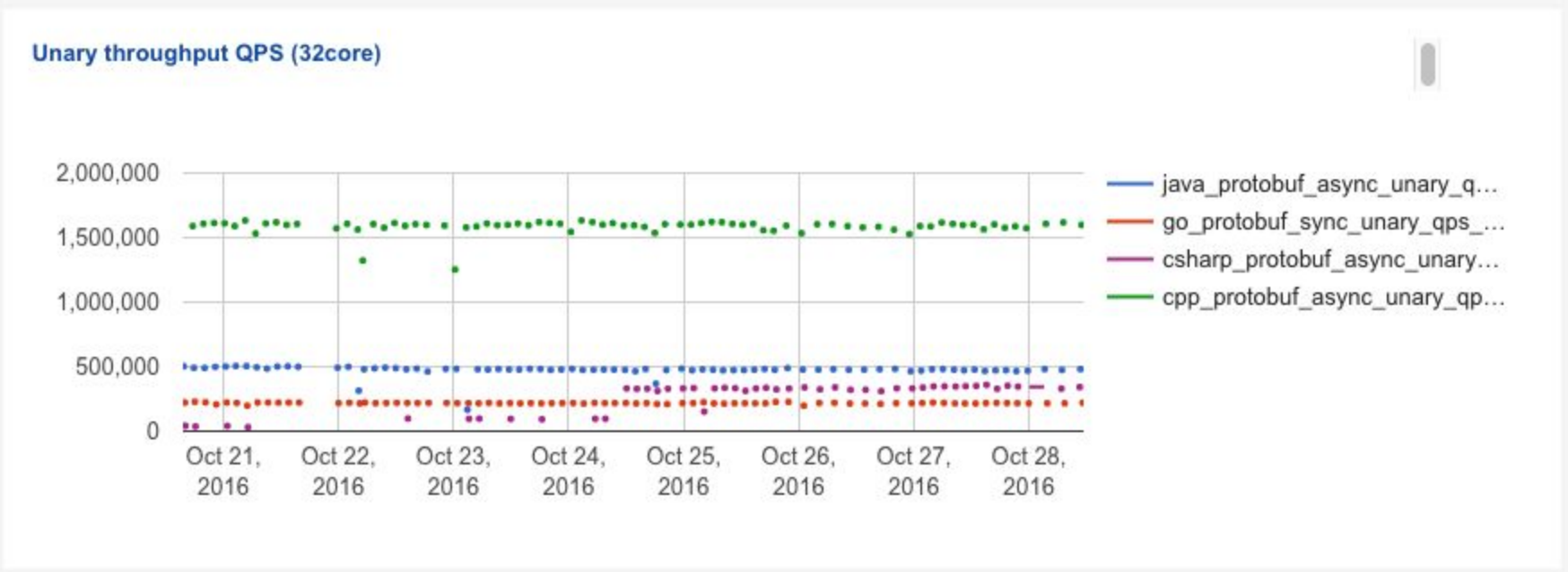
- Unary: Sub 1ms latency for all languages (C++ 200µs)
- Streaming: C++ 150µs

Throughput (secure connection)

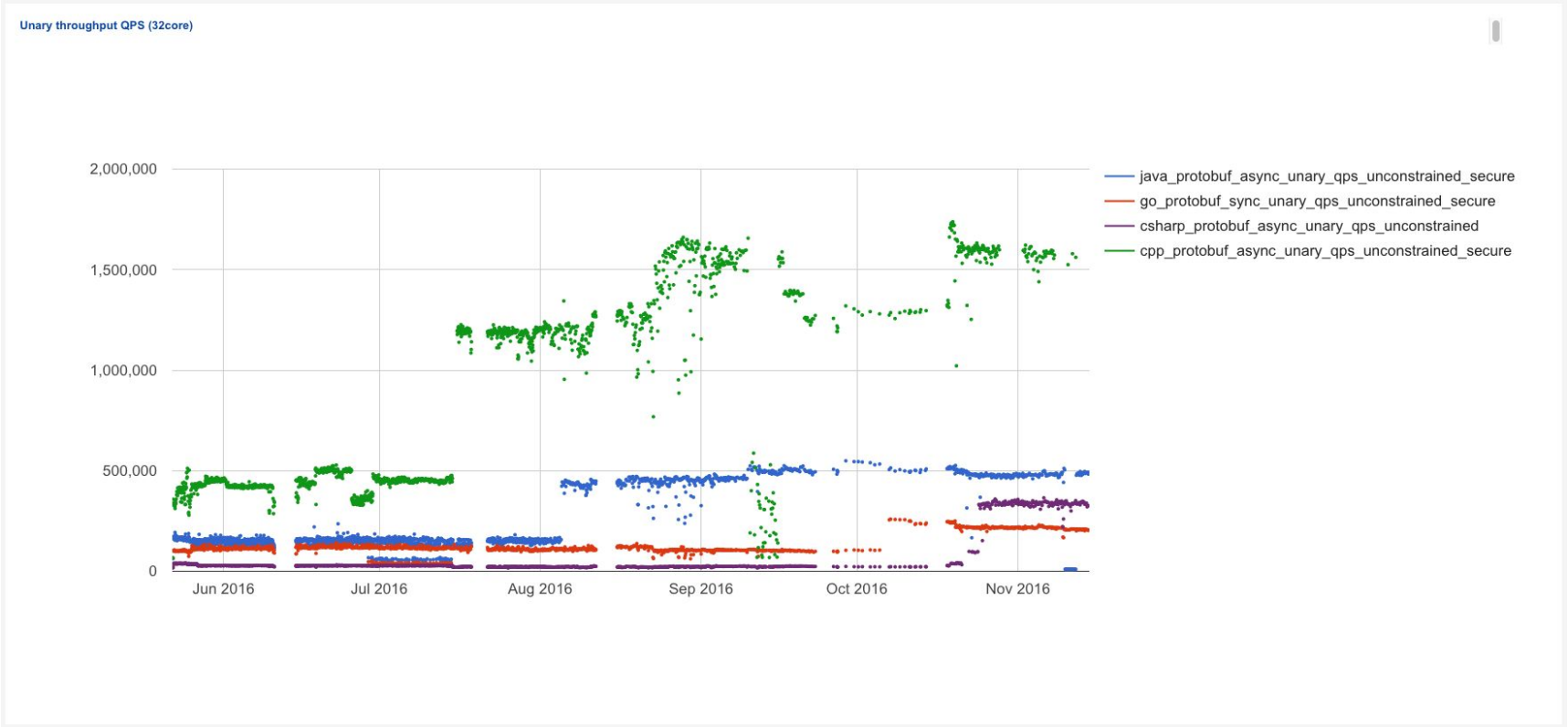
- Unary 8core: 370K QPS (C++)
- Unary 32core: 1.5M QPS (C++)
- Streaming 32core: 3.5M QPS (C++)

https://performance-dot-grpc-testing.appspot.com/explore?dashboard=5760820306771968

Performance: Current



Performance: Improvement over 6 months



Quiz

gRPC: What does "g" stand for?

"g" stands for

v1.0.0 - gRPC

v1.1.0 - good RPC

.....

Contributing

- https://github.com/grpc
- BSD licensed
- We welcome pull requests

Contact us:

- grpc-io@googlegroups.com
- Website: http://grpc.io

Protobuf

- https://github.com/google/protobuf

Opportunities

Google Summer of Code

Papers (e.g. on performance)

Build your own services & apps!

Questions?

Thanks!