Introductions
About gRPC

Client-server remote procedure call; functions+messages

OSS version and future of Google’s RPC

● Development on GitHub

Three implementations (C99, Java, Go), official hand-written APIs for ten languages

● Embedded to mobile to server
● Data center, mobile to cloud, cloud to cloud, device to device

Website: grpc.io  GitHub: github.com/grpc
About Myself

gRPC Java Tech Lead

In relation to IETF draft: work with Abhishek and Louis; Jayant in my manager

- Joined team after prototypes and early protocol
- Contributed to current protocol and provided feedback

Have HTTP background
Features & Protocol
Example Service

Protobuf IDL as example (gRPC core is marshaller agnostic)

```protobuf
service HelloService {
    rpc SayHello (HelloRequest) returns (HelloResponse);
}

message HelloRequest {
    string greeting = 1;
}

message HelloResponse {
    string reply = 1;
}
```
Protocol Features

- **Status:** canonical code + text message
  - Not HTTP status code. gRPC-defined
- **Cancellation propagation, Deadline propagation**
- **Streaming:** 0-to-many requests, 0-to-many responses
  - Bi-directional, full duplex, flow controlled, in order, best effort
  - Think “message-based TCP”
  - Is a natural scoping mechanism (e.g., notifications, locks)
- **Metadata (headers and trailers)**
  - Extension mechanism
  - Additional error information
- **Misc:** TLS with mutual auth, message compression
Example Service

Protobuf IDL as example (gRPC core is marshaller agnostic)

```protobuf
service HelloService {
  rpc SayHello (HelloRequest) returns (HelloResponse);
}

message HelloRequest {
  string greeting = 1;
}

message HelloResponse {
  string reply = 1;
}
```
Example Service (Streaming)

Protobuf IDL as example (gRPC core is marshaller agnostic)

```protobuf
service HelloService {
  rpc SayHello (stream HelloRequest) returns (stream HelloResponse);
}

message HelloRequest {
  string greeting = 1;
}

message HelloResponse {
  string reply = 1;
}
```
Protocol Features

- **Status**: canonical code + text message
  - Not HTTP status code. gRPC-defined
- **Cancellation propagation, Deadline propagation**
- **Streaming**: 0-to-many requests, 0-to-many responses
  - Bi-directional, full duplex, flow controlled, in order, best effort
  - Think “message-based TCP”
  - Is a natural scoping mechanism (e.g., notifications, locks)
- **Metadata (headers and trailers)**
  - Extension mechanism
  - Additional error information
- **Misc**: TLS with mutual auth, message compression
Basic Flow

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metadata (Headers) →</td>
<td></td>
</tr>
<tr>
<td>Message →</td>
<td></td>
</tr>
<tr>
<td>Half Close →</td>
<td></td>
</tr>
<tr>
<td>← Metadata (Headers)</td>
<td></td>
</tr>
<tr>
<td>← Message</td>
<td></td>
</tr>
<tr>
<td>← Metadata (Trailers) + Status</td>
<td></td>
</tr>
</tbody>
</table>
HTTP Mapping

RPC method: POST /namespace.ServiceName/MethodName
Metadata: Headers and Trailers
Messages: Length-prefixed frames in body (5 byte header)
  ● Reverse Proxyable

HTTP/1.1 semantics, but needs some edge features
  ● Trailers
  ● Concurrent request and response (bi-direction)
  ● Cancellation
Built on HTTP/2

Frame-based Multiplexing; substantially amortized cost of TLS

- Byte-based flow control (gRPC converts to message-based)
- Graceful connection shutdown

Still permits limited resource servers
Implementation Features

Messages from KBs to 100s MBs

Pluggable name discovery

Client-side load balancer

Reflection

Conversion to REST (with Protobuf and via Proxy)

and more
Status Codes

- OK
- CANCELLED
- UNKNOWN
- INVALID_ARGUMENT
- DEADLINE_EXCEEDED
- NOT_FOUND
- ALREADY_EXISTS
- PERMISSION_DENIED
- UNAUTHENTICATED
- RESOURCE_EXHAUSTED
- FAILED_PRECONDITION
- ABORTED
- OUT_OF_RANGE
- UNIMPLEMENTED
- INTERNAL
- UNAVAILABLE
- DATA_LOSS