g-RPC

What, **W**hy, **W**here and **H**ow to use it…

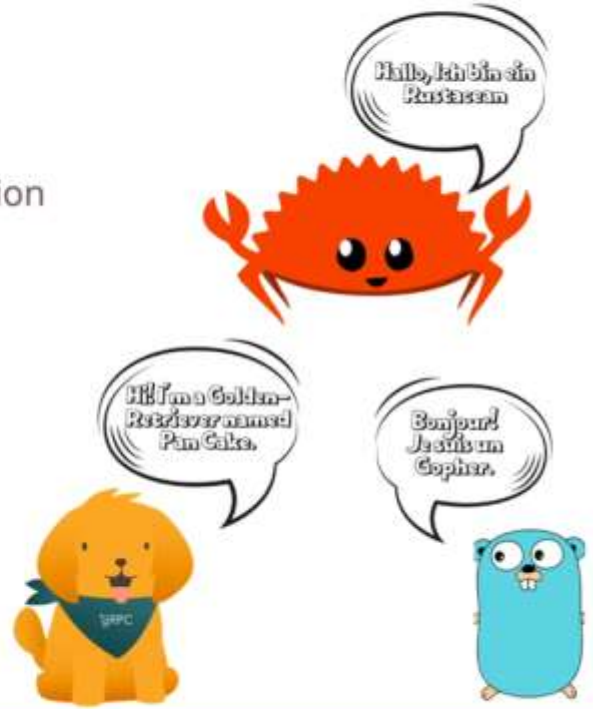
Mohammed Raihan Ullah
Lecturer, IICT, SUST
Email : raihan-iict@sust.edu

The motivation of gRPC

TECH SCHOOL

COMMUNICATION BETWEEN DIFFERENT LANGUAGES

- Back-end and front-end are written in different languages
- Micro-services might be written in different languages
- They must agree on the API contracts to exchange information
  - Communication channel: REST, SOAP, message queue
  - Authentication mechanism: Basic, OAuth, JWT
  - Payload format: JSON, XML, binary
  - Data model
  - Error handling



The motivation of gRPC

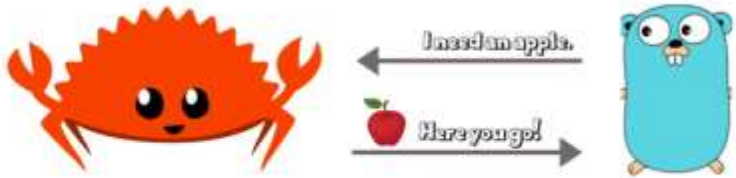
TECH SCHOOL

COMMUNICATION SHOULD BE EFFICIENT

- Huge amount of exchange messages between micro-services
- Mobile network can be slow with limited bandwidth

COMMUNICATION SHOULD BE SIMPLE

- Client and server should focus on their core service logic
- Let the framework handle the rest



What is gRPC?

TECH SCHOOL

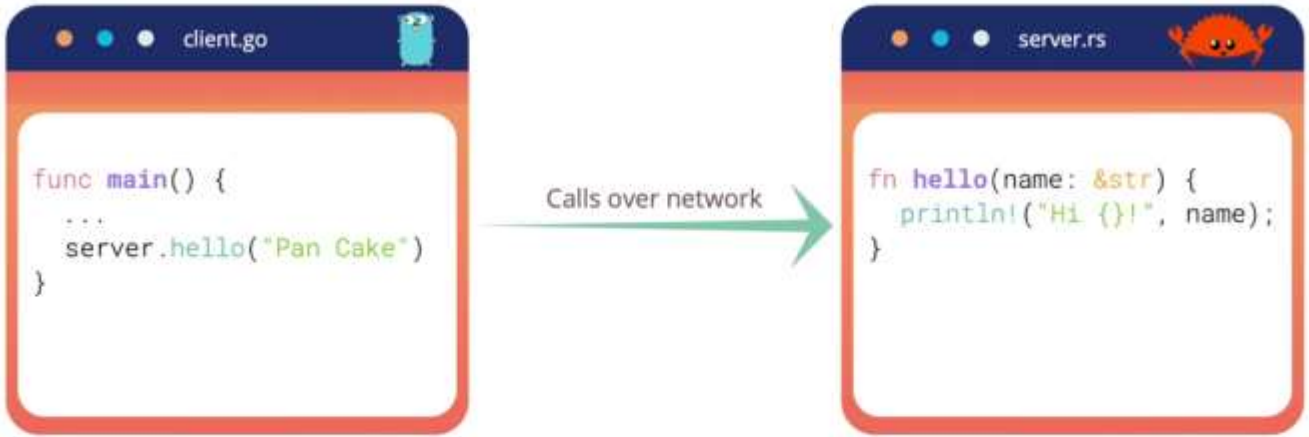
- gRPC is a high-performance open-source feature-rich RPC framework
- gRPC is originally developed by Google
- Now it is a part of the Cloud Native Computing Foundation - CNCF
- **g** stands for different things in each gRPC release: gRPC, good, green, glorious, game, gon
  - https://github.com/grpc/grpc/blob/master/doc/g_stands_for.md
- RPC stands for **R**emote **P**rocedure **C**alls

Reference:
https://grpc.io/about/
https://www.cncf.io/

What is Remote Procedure Calls?

TECH SCHOOL

- It is a protocol that allows a program to
  - execute a procedure of another program located in other computer
  - without the developer explicitly coding the details for the remote interaction
- In the client code, it looks like we're just calling a function of the server code directly
- The client and server codes can be written in different languages



How gRPC works?

TECH SCHOOL

- Client has a generated stub that provides the same methods as the server
- The stub calls gRPC framework under the hood to exchange information over network
- Client and server use stubs to interact with each other, so they only need to implement their core service logic

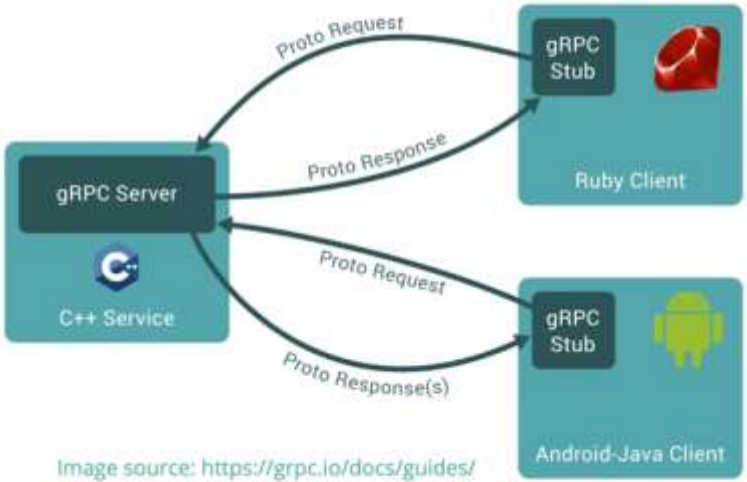


Image source: https://grpc.io/docs/guides/

How stubs are generated?

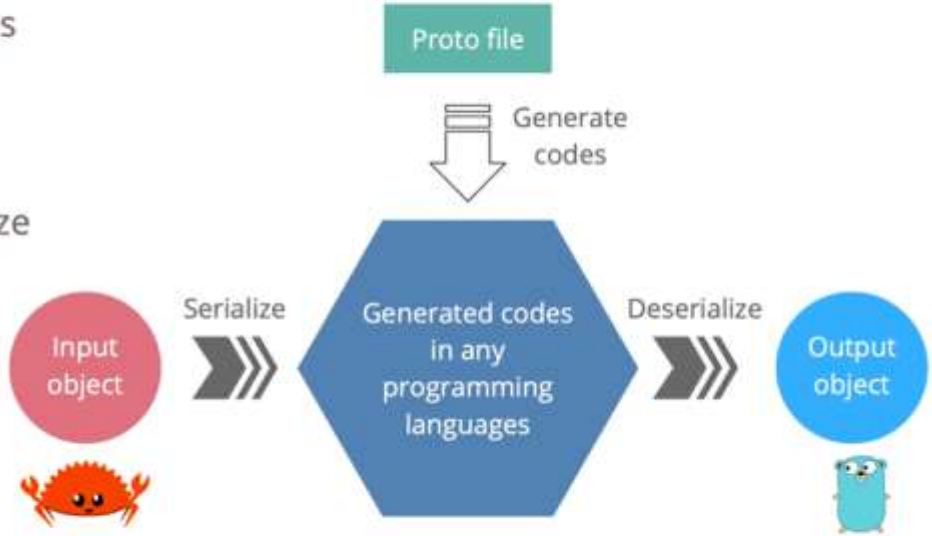
TECH SCHOOL

- API contracts description
  - The services and payload messages are defined using Protocol Buffer
- Server and client stubs are generated by the
  - Protocol Buffer compiler
  - gRPC plugins of each language

Why gRPC uses Protocol Buffer?

TECH SCHOOL

- Human-readable Interface Definition Language (IDL)
- Programming languages interoperable:
  - Code generators for many languages
- Binary data representation:
  - Smaller size
  - Faster to transport
  - More efficient to serialize / deserialize
- Strongly typed contract
- Conventions for API evolution
  - Backward & forward compatibility
- Alternative options
  - Google flatbuffers
  - Microsoft bond



What languages are supported by gRPC?

TECH SCHOOL

- 10 officially supported languages
  - Pure implementation: Go, Java, NodeJS
  - Wrap C-gRPC core: C/C++, C#, Objective-C, Python, Ruby, Dart, PHP
- Many other unofficial libraries: Swift, Rust, TypeScript, Haskell, etc.

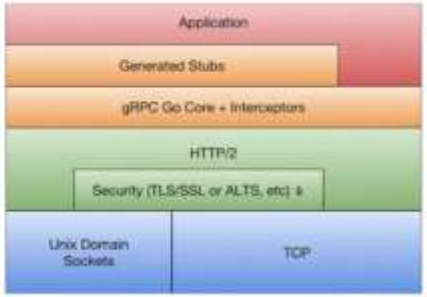
Reference:
https://grpc.io/about/
https://github.com/grpc

What makes gRPC efficient?

TECH SCHOOL

GRPC USES HTTP/2 AS ITS TRANSFER PROTOCOL

- Binary framing
  - More performant and robust
  - Lighter to transport, safer to decode
  - Great combination with Protocol Buffer
- Header compression using HPACK
  - Reduce overhead and improve performance
- Multiplexing
  - Send multiple requests and responses in parallel over a single TCP connection
  - Reduce latency and improve network utilization
- Server push
  - One client request, multiple responses
  - Reduce round-trip latency



The gRPC Go stacks
Image source: https://grpc.io/blog/grpc-stacks/

HTTP/1.1 vs HTTP/2 speed comparison
Source: http://www.http2demo.io/

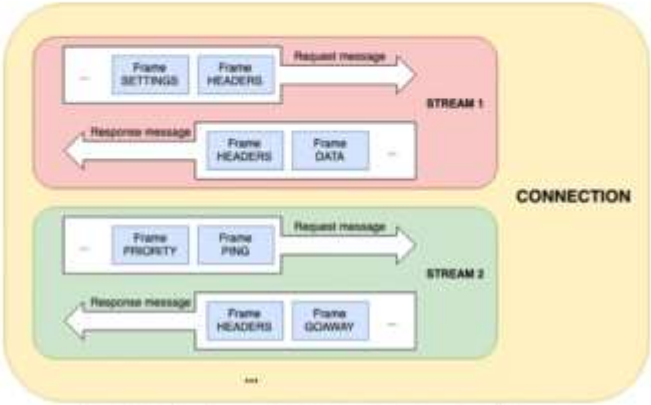
Reference:
https://developers.google.com/web/fundamentals/performance/http2

How HTTP/2 works under the hood

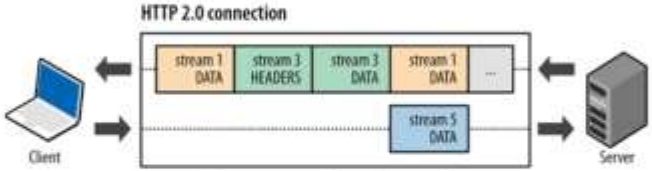
TECH SCHOOL

- Single TCP connection carries multiple bidirectional streams
- Each stream has a unique ID and caries multiple bidirectional messages
- Each message (request/response) is broken down into multiple binary frames
- Frame is the smallest unit that carries different types of data: HEADERS, SETTINGS, PRIORITY, DATA, etc.
- Frames from different streams are interleaved and then reassembled on the other side

The new binary framing layer in HTTP/2 enables stream multiplexing



Logical structure of stream, message and frame within a single HTTP/2 connection



Frames are interleaved

Image source: https://developers.google.com

HTTP/2 vs HTTP/1.1

TECH SCHOOL

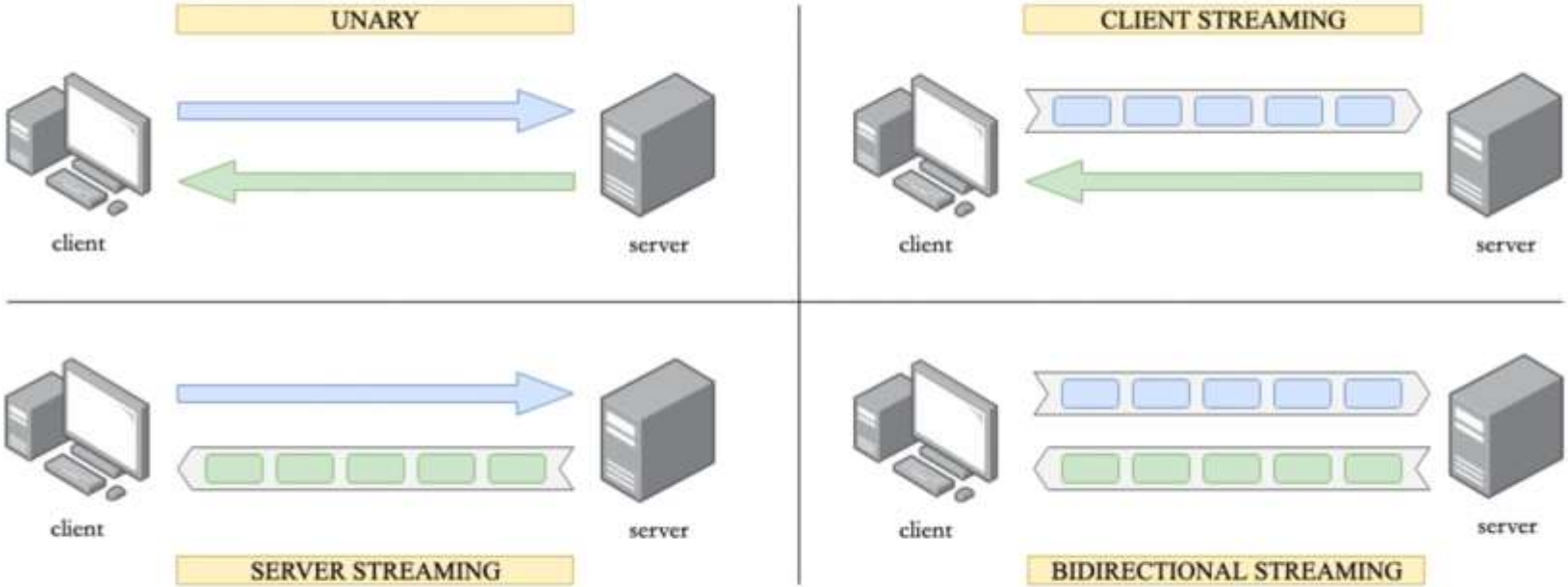
	HTTP/2	HTTP/1.1
TRANSFER PROTOCOL	Binary	Text
HEADERS	Compressed	Plain text
MULTIPLEXING	Yes	No
REQUESTS PER CONNECTIONS	Multiple	1
SERVER PUSH	Yes	No
RELEASE YEAR	2015	1997

Multiple requests over HTTP/2 and HTTP/1.1 connections

Image source: https://blog.cloudflare.com/

4 types of gRPC

TECH SCHOOL



gRPC vs REST

TECH SCHOOL

FEATURE	GRPC	REST
Protocol	HTTP/2 (fast)	HTTP/1.1 (slow)
Payload	Protobuf (binary, small)	JSON (text, large)
API contract	Strict, required (.proto)	Loose, optional (OpenAPI)
Code generation	Built-in (protoc)	Third-party tools (Swagger)
Security	TLS/SSL	TLS/SSL
Streaming	Bidirectional streaming	Client → server request only
Browser support	Limited (require gRPC-web)	Yes

Reference:
https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison
https://github.com/grpc/grpc-web

Where gRPC is well suited to?

TECH SCHOOL

- Microservices
  - Low latency and high throughput communication
  - Strong API contract
- Polyglot environments
  - Code generation out of the box for many languages
- Point-to-point realtime communication
  - Excellent support for bidirectional streaming
- Network constrained environments
  - Lightweight message format

Reference:
https://docs.microsoft.com/en-us/aspnet/core/grpc/comparison

How to define a protocol message

TECH SCHOOL

- Name of the message: UpperCamelCase
- Name of the field: lower_snake_case
- Some scalar-value data types:
  - string, bool, bytes
  - float, double
  - int32, int64, uint32, uint64, sint32, sint64, etc.
- Data types can be user-defined enums or other messages
- Tags are more important than field names
  - Is an arbitrary integer
    - From 1 to 536,870,911 (or 2^29-1)
    - Except from 19,000 to 19,999 (reserved)
  - From 1 to 15 take 1 byte
  - From 16 to 2047 take 2 bytes
  - Don't need to be in-order or sequential
  - Must be unique for same-level fields

welcome.proto

```
syntax = "proto3";

message <NameOfTheMessage> {
  <data-type> name_of_field_1 = tag_1;
  <data-type> name_of_field_2 = tag_2;
  ...
  <data-type> name_of_field_N = tag_N;
}
```


Reference:
https://developers.google.com/protocol-buffers/docs/proto3#scalar

3 types of gRPC connection

- **INSECURE**

  Plaintext data. No encryption. Don't use it for production!

- **SERVER-SIDE TLS**

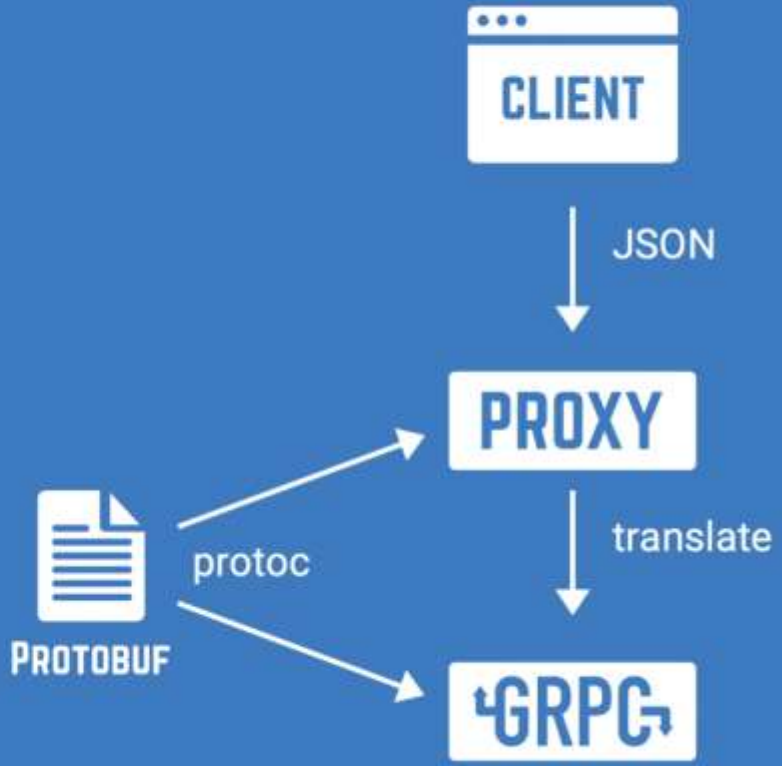
  Encrypted data. Only server needs to provide its certificate to client.

- **MUTUAL TLS**

  Encrypted data. Both server and client need to provide certificates to each other.

gRPC Gateway

- A plugin of protobuf compiler
- Generate proxy codes from protobuf
- Translate RESTful call to gRPC



References

- https://youtube.com/playlist?list=PLy_6D98if3UJd5hxWNfAqKMr15HZqFnqf