Einführung in gRPC

Das moderne Toolkit für Microservices-Kommunikation

Thomas Bandixen

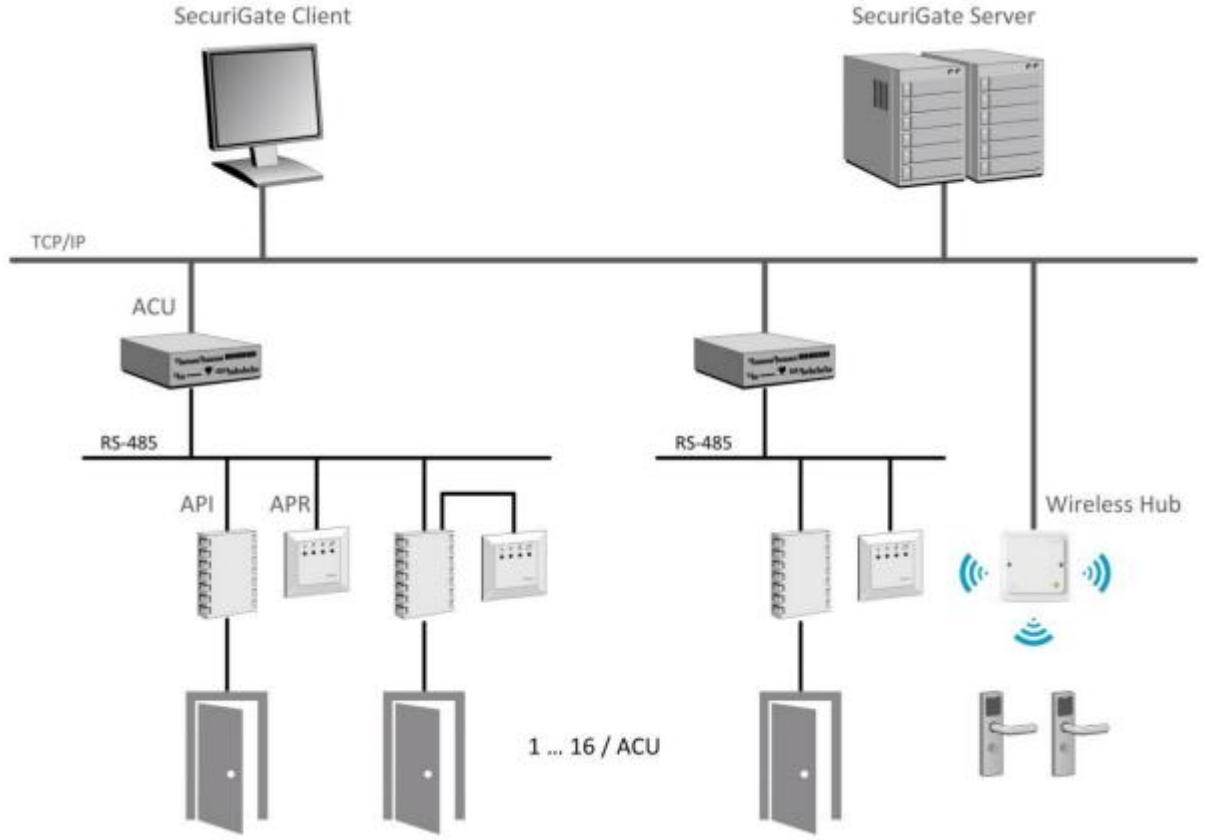
- Senior Consultant
- Trivadis AG
- thomas.bandixen@trivadis.com

Agenda

- Initial situation
- Introduction to gRPC
- gRPC live in action
- Why gRPC
- Conclusion.

Initial situation

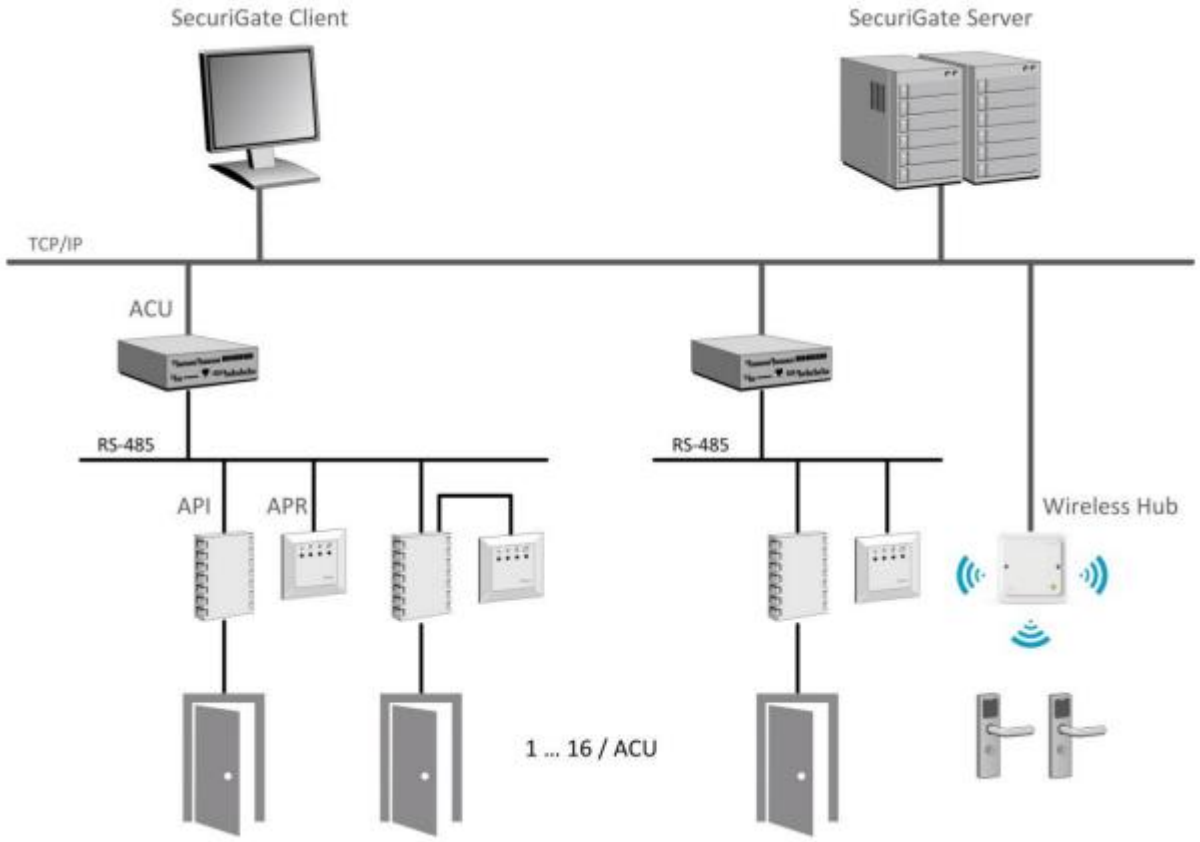
- Distributed systems
- Polyglot services
- Load balanced.



Source: SecuriGate

Initial situation

- ~1’000 ACUs
- 4 Communication Servers (load balanced)
- A lot of traffic
- High CPU usage.



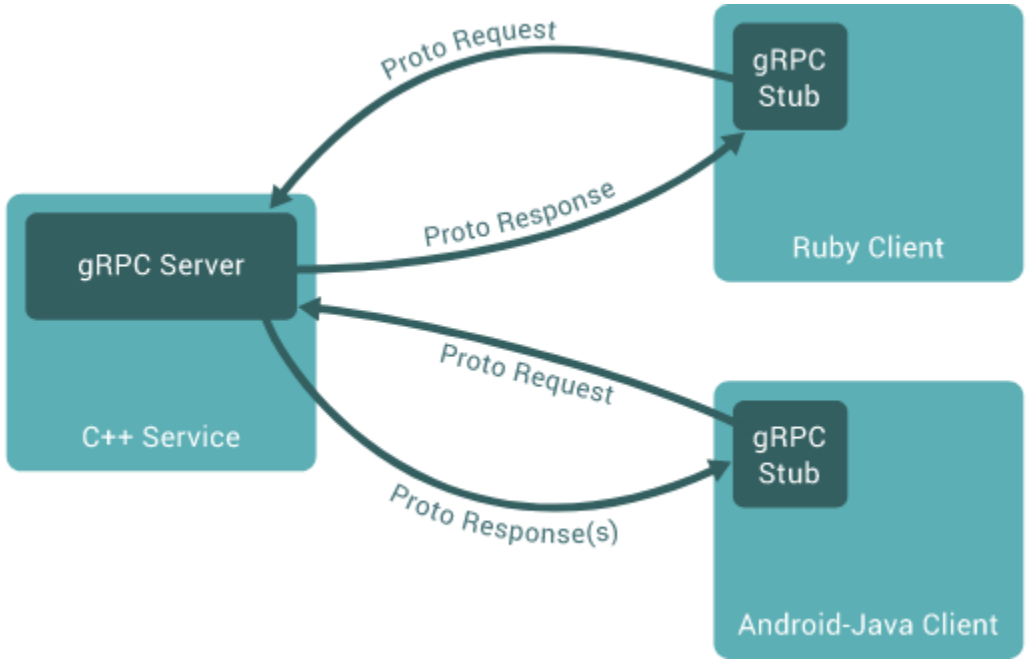
Source: SecuriGate


gRPC
A high performance, open-source universal
RPC framework

Introduction to gRPC

- Pronounced as “Jee-Arr-Pee-See”
- Is a high performance, open source RPC framework
- It helps in eliminating boilerplate code
- It helps in connecting polyglot services.

Introduction to gRPC



Protocol Buffers

- Structure the data in .proto files
  - Methods are structured as *services*
  - Protocol buffer data is structured as *messages*
- “protoc” generates your code to:
  - Populating
  - Serializing
  - retrieving message types
- Java, C++, Python, Objective-C, C#, Android Java, Ruby, JavaScript and Go.

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

gRPC Concepts

- gRPC has four kinds of service method
  - Unary RPC
  - Server streaming RPC
  - Client streaming RPC
  - Bidirectional streaming RPC.

```
rpc SayHello(HelloRequest) returns (HelloResponse){
}
rpc LotsOfReplies(HelloRequest) returns (stream HelloResponse){
}
rpc LotsOfGreetings(stream HelloRequest) returns (HelloResponse) {
}
rpc BidiHello(stream HelloRequest) returns (stream HelloResponse){
}
```

gRPC Concepts

- Deadlines/Timeouts
- RPC termination
- Cancelling RPCs
- Metadata
- Channels.

gRPC live in action

Demo

https://github.com/tbandixen/Basta-2019-gRPC-Demo

Why gRPC

REST

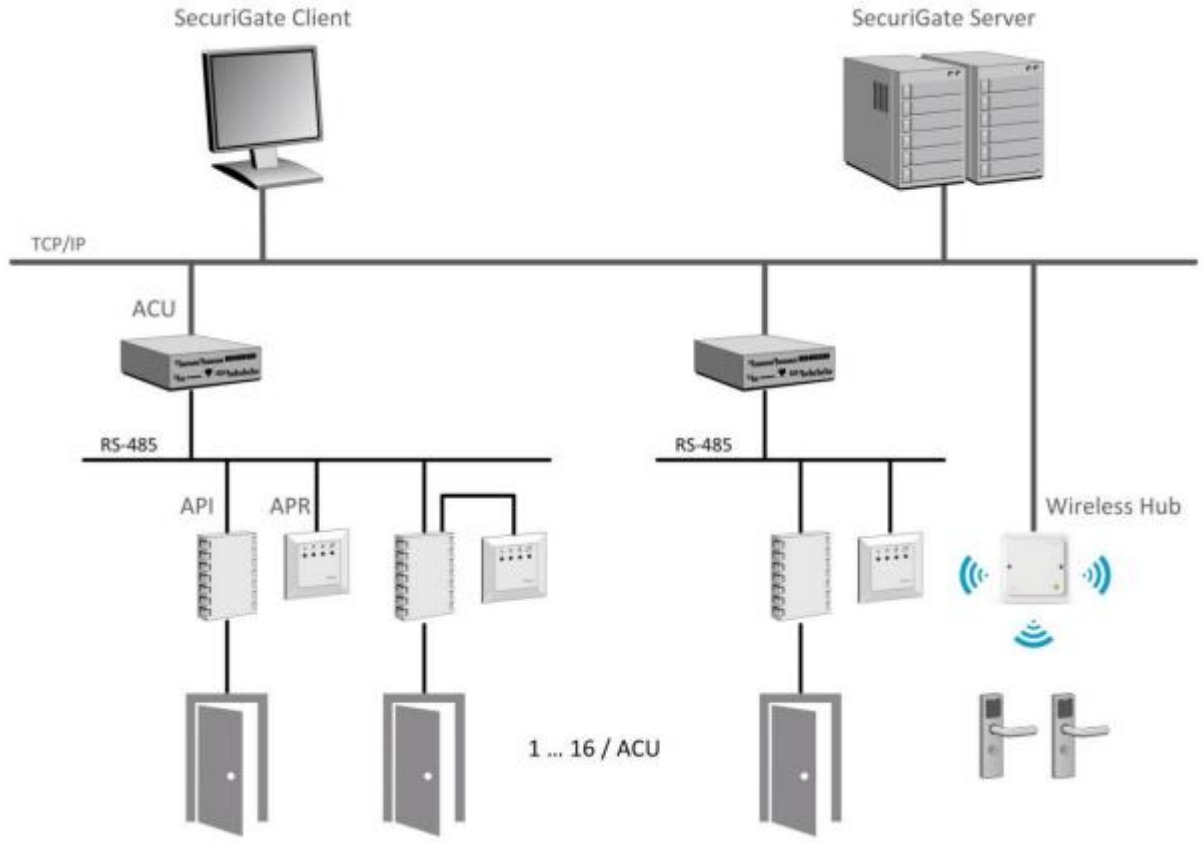
- Since about 2000
- Design concept
- JSON (text)
- HTTP 1.1

gRPC

- Since about 2015
- Contract-based communication
- Protobuf (binary)
- HTTP/2

Conclusion

- Significantly lower traffic
- CPU load strongly reduced.



Source: SecuriGate

Thank you for your attention

and enjoy the Basta!

Thomas Bandixen
thomas.bandixen@trivadis.com