A lightweight gRPC server for managing GNU Radio flowgraphs

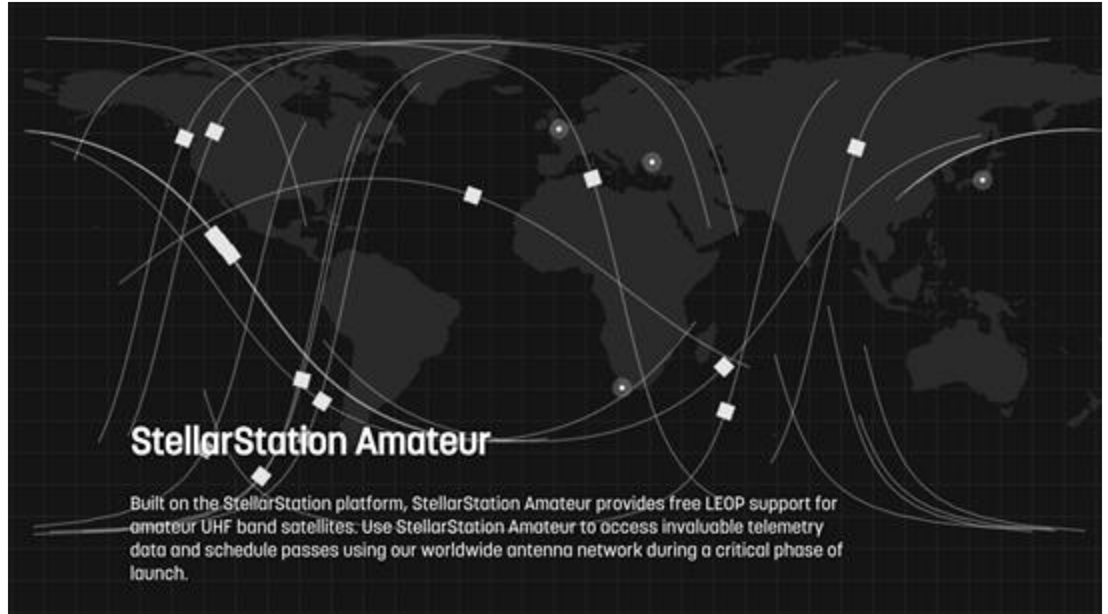
Reiichiro Nakano, Software Engineer at Infostellar

Overview

- Motivation behind Starcoder
- Starcoder
  - RPCs, gRPC and protocol buffers
  - Architecture
  - Key features
- Examples
  - Doppler shift correction
  - IQ Data streaming
  - AX.25 Transceiving
- Current state and Future work

StellarStation

- Worldwide satellite antenna sharing platform
- Currently focused on commercial UHF and S-band
- Also tracks amateur satellites - StellarStation Amateur



Stellarstation

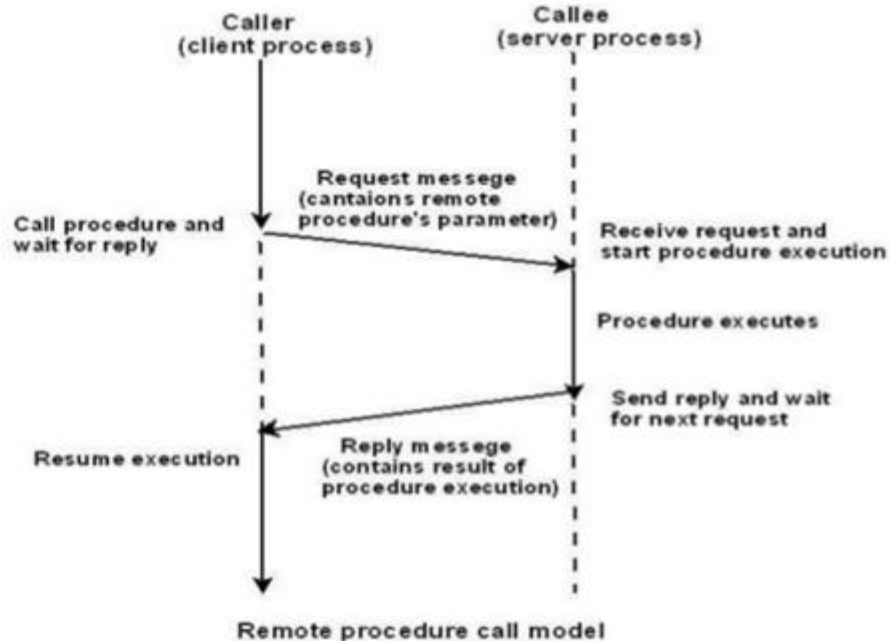
- A single groundstation must communicate with multiple satellites per day - all with different protocols!

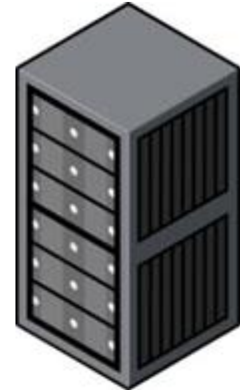
Motivation behind Starcoder

- Our ground station software (written in Go) did all the work: from managing satellite passes, moving the rotator, communicating with the cloud, and digital signal processing.
    - Got only as far as writing realtime decimation and FIR filtering in Go
- Eventually, we realized we *had* to use GNURadio.
- **How do we easily and programmatically start and stop multiple GNURadio flowgraphs and interact with them?**
    - Interacting primarily means: sending commands to the flowgraph, and getting packets from the flowgraph.

Remote Procedural Calls (RPC)

Go application





Alternatives explored

- One solution we looked into was the built-in ControlPort and ZMQ blocks.
  - Does not solve our problem of starting and stopping multiple flowgraphs: still needed to call command line from within our program.
  - Did not want to manage multiple Thrift and ZMQ connections.
  - PMTs passed from ZMQ Sinks are serialized using GNURadio's arbitrary serialization format. Calling language would need to know how to deserialize it.

Starcoder

- A lightweight gRPC server for managing GNU Radio flowgraphs in production

- gRPC high performance, open-source universal RPC framework

gRPC and protocol buffers

(greeter.proto)

Define the RPC service, its available procedures, and their corresponding inputs and outputs in the protocol buffer format.

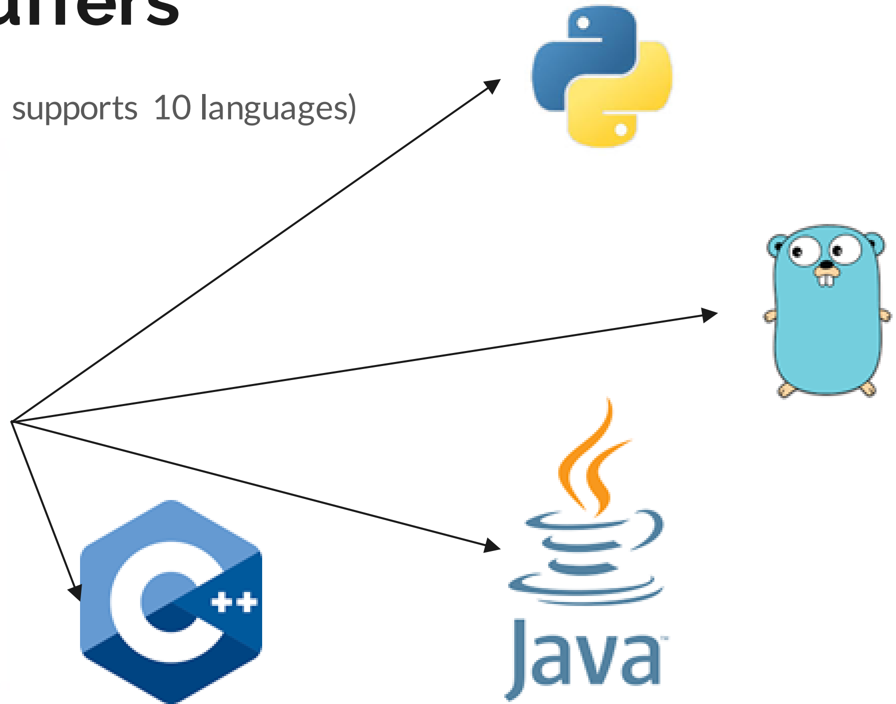
- Protocol buffers are "Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler."

```
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (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 and protocol buffers

Compile into your language of choice (currently supports 10 languages)

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



Clients can now call gRPC functions in the language of their choice

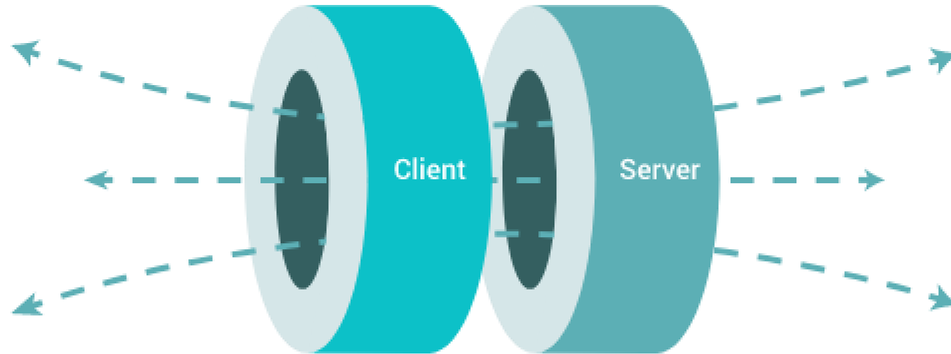
```
// The greeting service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
  // Sends another greeting
  rpc SayHelloAgain (HelloRequest) returns (HelloReply) {}
}

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

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

Python client:

```python
import grpc_library

response = grpc_library.SayHello(grpc_library.HelloRequest(name="user"))
print(response)
# grpc_library.HelloReply{“message”: “Hello user”}
```

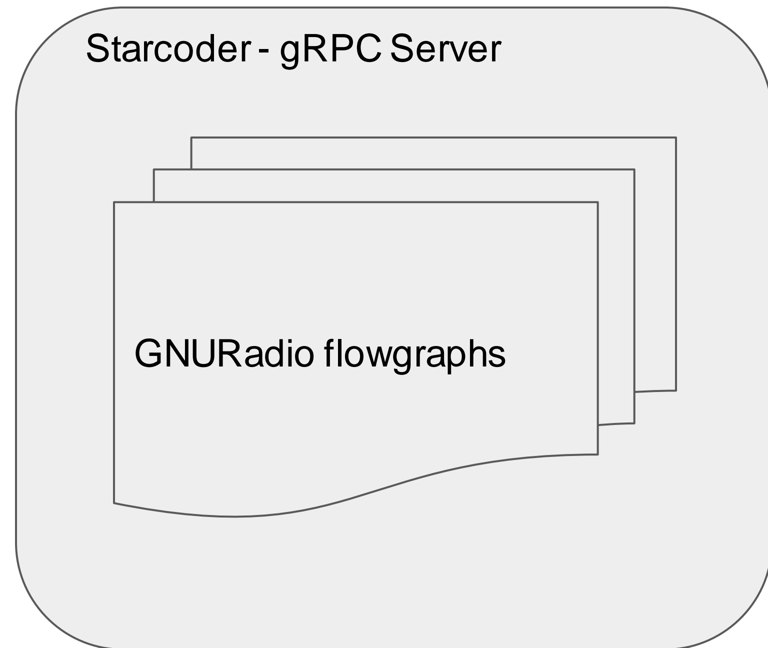
gRPC and protocol buffers

gRPC natively supports bidirectional streaming




```
rpc BidiHello(stream HelloRequest) returns (stream HelloResponse){
}
```

Starcoder Architecture




```
// The GNURadio process manager service definition.
service Starcoder {
  // Runs a flowgraph and streams back
  rpc RunFlowgraph (stream RunFlowgraphRequest) returns (stream RunFlowgraphResponse) {}
}
```

Starcoder Architecture

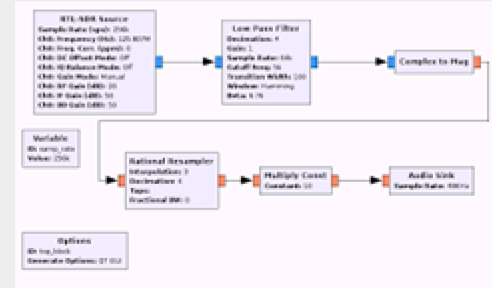
Client Program

First RunFlowgraphRequest:
- Contains name of flowgraph to run
- Contains initialization parameters

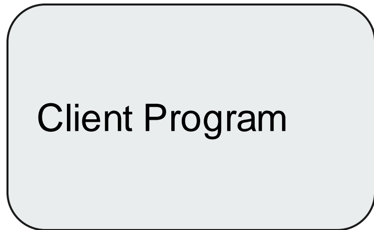
All Succeeding RunFlowgraphRequest:
- PMT messages directly to flowgraph

RunFlowgraphResponse:
- PMT messages directly from flowgraph

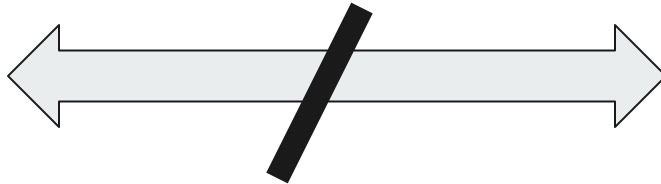
Starcoder - gRPC Server



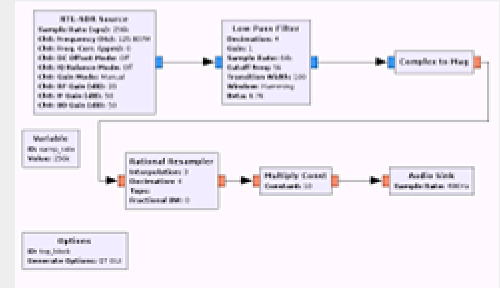
Starcoder Architecture



Client decides to close the stream



Starcoder - gRPC Server



Starcoder Architecture

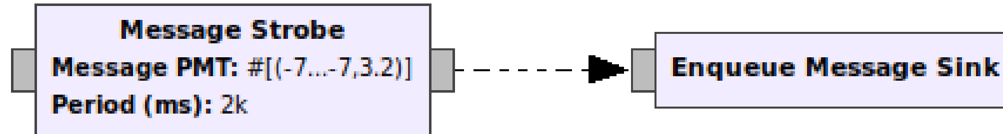
Starcoder - gRPC Server

Client Programs

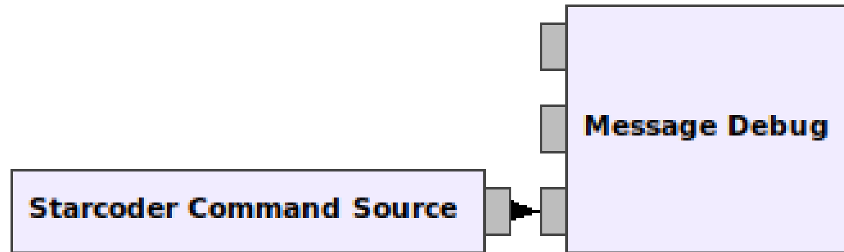
GNURadio flowgraphs

- Enqueue Message Sink Block
- Starcoder Command Source Block

Enqueue Message Sink block



Starcoder Command Source block



PMTs as Protocol buffers

- Communication with flowgraphs and Starcoder clients happen through PMTs.
- GNURadio's asynchronous messages are PMTs, but other languages don't know what PMTs are!
- Protocol buffers can be compiled to any language they support
- **Starcoder contains a one-to-one mapping between PMTs and protocol buffers!**
    - Conversion is done in C++ (see proto_to_pmt.cc and pmt_to_proto.cc)

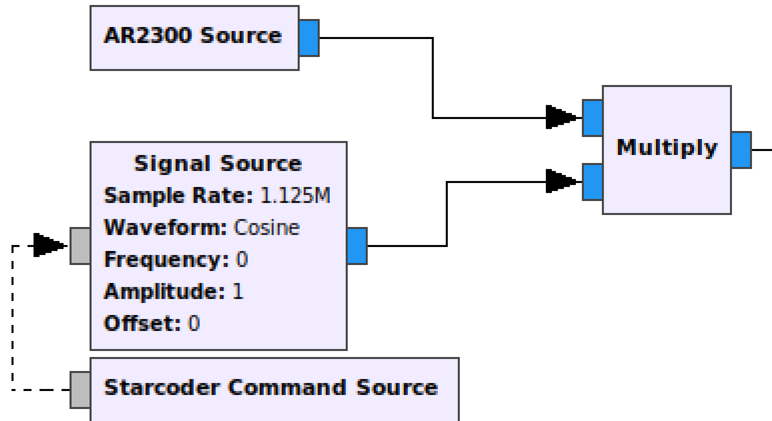
```
// A GNURadio PMT (Polymorphic Message Type)
message BlockMessage {
  oneof message_oneof {
    bool boolean_value = 1;

    string symbol_value = 2;

    int64 integer_value = 3;

    double double_value = 4;

    Complex complex_value = 5;

    Pair pair_value = 6;

    List list_value = 7;

    UniformVector uniform_vector_value = 9;

    Dict dict_value = 10;

    bytes blob_value = 11;
  }
}
```

- Starcoder server is written in Go
- Flowgraphs are compiled to Python
- Instead of calling a bash process, we “embed” the Python interpreter in Go (CPython)
  - Can call Python functions and instantiate Python classes.
- Go -> C (CPython interpreter via C-API) -> Python (Flowgraph level) -> C++ (GR scheduler level)
- Can call flowgraph methods start(), stop(), and wait() from Go
- Lets Starcoder hook in directly to the Starcoder command source and message sink blocks.
  - We register/receive a C queue to/from the block (register_starcoder_queue(), get_starcoder_queue_ptr())
  - The sink sends up messages received by the block - Starcoder waits on these queues to send them back out through gRPC
  - Starcoder sends commands to the C queues of the command source blocks - The block waits on this queue to propagate messages to downstream blocks.

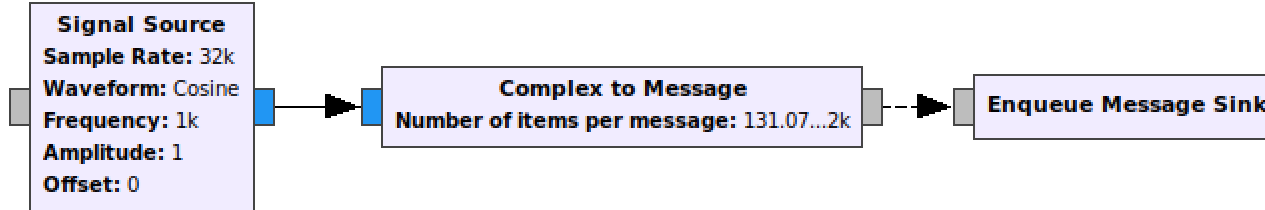
Starcoder key features and design decisions

- Fully manages the lifecycle of flowgraphs - compilation, executing, and stopping
- Uses gRPC as the RPC framework
- All interaction with a flowgraph is done through a single bidirectional streaming gRPC connection
- PMTs are converted to a well-defined language-neutral protocol buffer format
- Written in Go

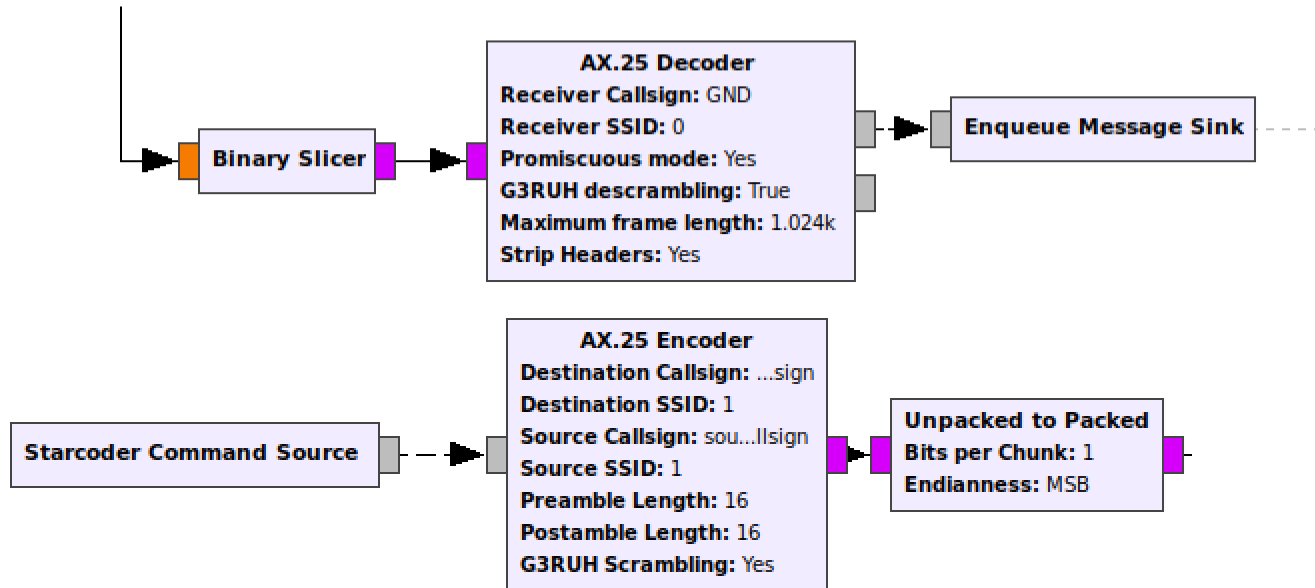
Examples - Doppler shift correction



Examples - Streaming back complex I/Q data



Examples - AX.25 Transceiving



Room for Improvement

- Non-existent documentation
  - If you're interested in using Starcoder *right now*, please contact us!
- Only supports PMTs.
  - To send back streaming data, we need to package them as PMTs.
- All flowgraphs run in the same process.
  - A malfunctioning flowgraph or buggy blocks can potentially mess with other running flowgraphs.
  - Solution: Run each flowgraph as a separate process so we can kill them when necessary.
- Flowgraphs only run while the stream is alive.
  - Optimized for Infostellar's use-case: multiple short-running flowgraphs
- Very interested in tighter integration with GNURadio.

Contact me: reiichiro@istellar.jp

Github repository: https://github.com/infostellarinc/starcoder