GO-HEP/XRootD

Mikhail Ivchenko (@EgorMatirov)

Izhevsk State Technical University

About GSoC 2018

- It's a global program organized by Google and focused on bringing more student developers into open source software development.
- 3 month programming project.
- 206 organizations.
- 1268 students were accepted in 2018.
- 1072 of them successfully completed their work.
- Pure-Go XRootD client implementation: https://goo.gl/ycYxAs.

What is XRootD?

Key features are:

- fault tolerant access to the remote data repositories;
- authentication / authorization via a set of plugins;
- support of parallel data transfer via several sockets.

Messages format

Request

- StreamID
- RequestID
- Parameters (16 bytes)
- Data length
- Additional data

matches

Response

- StreamID
- Status
- Data length
- Additional data

Why implement it in Go?

- Fast development
- High performance
- Easy to implement out-of-order responses handling via goroutines and channels
- Cross-platform

Goroutines

Lightweight threads

```
go f("param")
```

Channels

Pipes that connect goroutines.

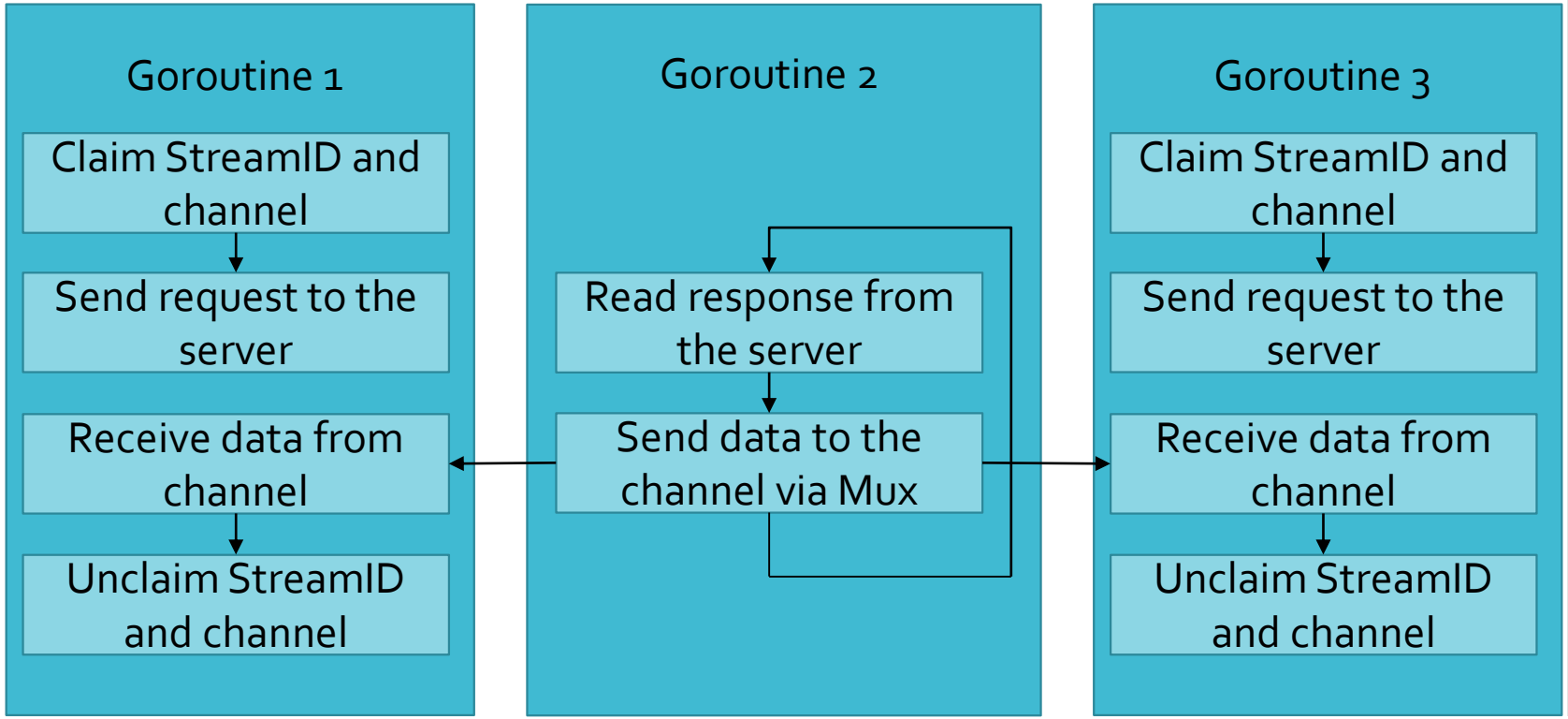
```
// Create a new channel with `make(chan val-type)`.
// Channels are typed by the values they convey.
messages := make(chan string)

// _Send_ a value into a channel using the `channel <-`
// syntax. Here we send `"ping"`  to the `messages`
// channel we made above, from a new goroutine.
go func() { messages <- "ping" }()

// The `<-channel` syntax _receives_ a value from the
// channel. Here we'll receive the `"ping"` message
// we sent above.
msg := <-messages
```

Packages

- client – client library code.
- cmd/xrd-cp – utility for copying files.
- cmd/xrd-fuse – mounts the remote directory.
- cmd/xrd-ls – lists remote directory content.
- cmd/xrd-srv – serves data from a local directory over the XRootD protocol.
- internal/mux – library to dispatch responses using StreamID.
- internal/xrdenc – library to simplify messages encoding \ decoding.
- server – server library code.
- xrdfs – structures representing the XRootD-based filesystem.
- xrdfuse – implementation of the FUSE API.
- xrdio – File type implementing various interfaces from the io package.
- xrdproto – implementation of the protocol.

internal/mux

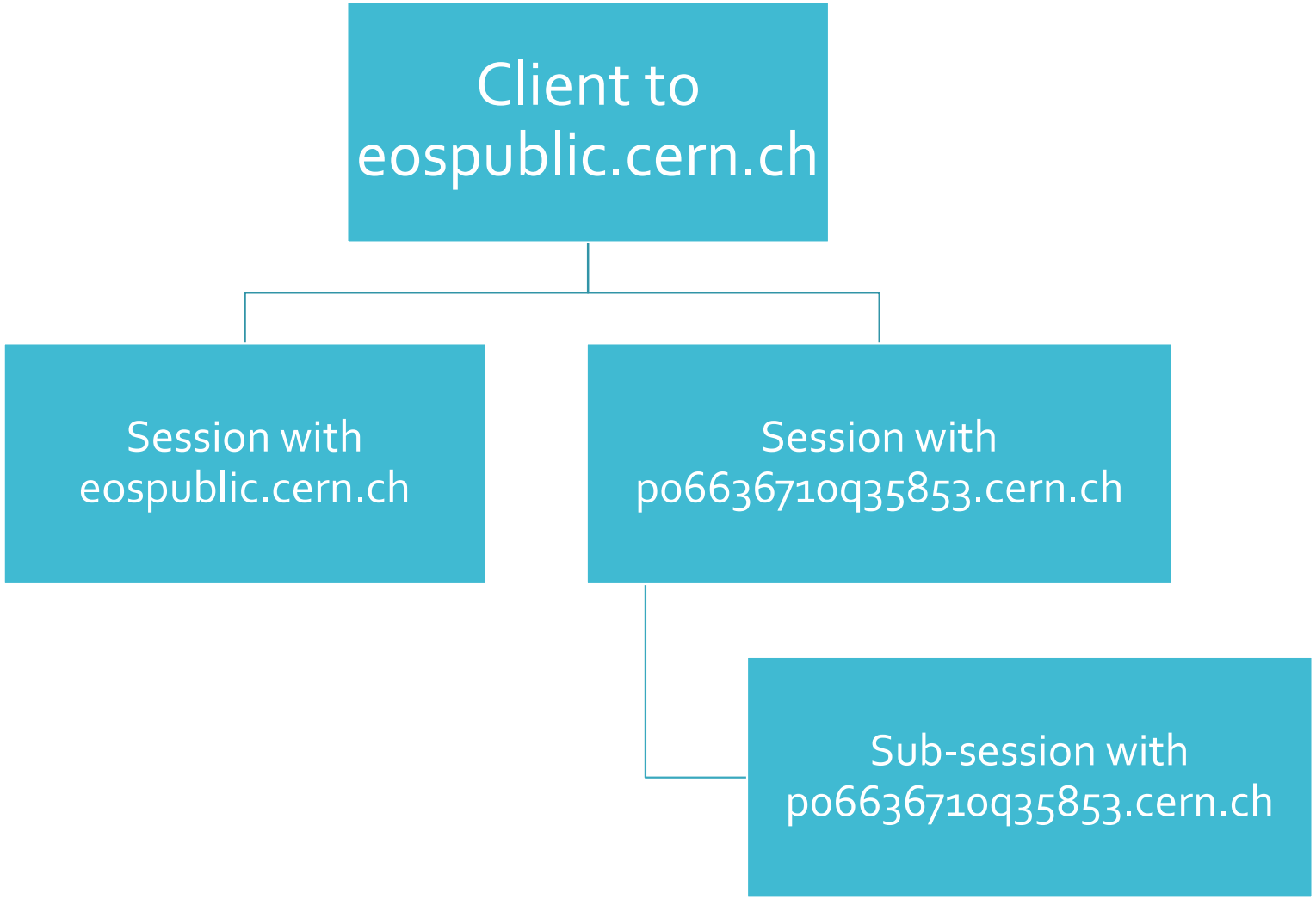
- Claim StreamID and a channel associated with it.
- Send a data to the channel via StreamID.
- Unclaim StreamID and the channel.



internal/mux usage example

```
mux := New()
defer m.Close()
// Claim channel for response retrieving.
id, channel, err := m.Claim()
if err != nil {
    // handle error.
}
// Send a request to the server using id as a streamID.
go func() {
    // Read response from the server.
    // ...
    // Send response to the awaiting caller
    // using streamID from the server.
    err := m.SendData(streamID, want)
    if err != nil {
        // handle error.
    }
}
// Fetch response.
response := <-channel
```

Client and sessions



Security

Authentication:

- Happens at the beginning of the session (after login).
- Uses a list of default authentication providers (krb5, unix, host).
- It's possible to add custom authentication providers.

Signing:

- Server sends a list of the signing requirements.
- Client signs a request if necessary.

Error handling

- Server handles errors by redirection or error responses.
- Client handles I/O errors by redirection to an "initial session". (TODO: follow protocol in determination of "initial session").
- Any error response is returned to the caller of the client method.

Example of client usage

```
client, err := NewClient(context.Background(), addr, "gopher")
if err != nil {
    log.Fatal(err)
}
fs := client.FS()
```

```
stat, err := fs.Stat(context.Background(), path)
if err != nil {
    log.Fatal(err)
}
log.Println(stat.EntrySize)
```

```
file, err := fs.Open(context.Background(), path, xrdfs.OpenMode(0),
    xrdfs.OpenOptionsOpenRead)
if err != nil {
    log.Fatal(err)
}
```

```
data := make([]byte, 4)
if _, err := file.ReadAt(data, 0); err != nil {
    log.Fatal(err)
}
log.Println(data)
```

Server

- Is able to listen on multiple addresses (e.g. different ports).
- Goroutine is launched for each of the incoming connections.
- It's possible to implement various backends (local FS, remote FS).
- Still WIP.

Supported platforms

- Developing happens under Windows and Linux.
- CI testing is launched under Windows and Linux.
- Does not depend on cgo itself, should be usable at any OS and architecture supported by Go (Linux, Windows, macOS, *BSD) .
- FUSE-based file system in supported only on Linux.

TODO

- GSI authentication;
- PWD authentication;
- SSS authentication;
- getfile, putfile, set and readv requests;
- server implementation.

- https://github.com/go-hep/hep/issues/170.