Peer-to-Peer-Systems and Security

Chair of Network Architectures and Services TUM

SoSe 2016

Supervisor:
Sree Harsha Totakura
Lukas Schwaighofer
Prof. Dr.-Ing Georg Carle

Students:
Hannes Dorfmann
Dorel Coman

Interims report

DE-RO Team working on **RPS** module

Process Architecture

We decided to use Netty[1], which uses event loops to dispatch events. Furthermore, Netty extends the concept of event loops with the concepts of pipelines: Instead of having a single handler attached to an event it allows you to attach a whole sequence of handlers which can pass their output as input to the next handler. We even go a step further by using RxNetty[2], a Reactive Stream extension on top of Netty. In short: Event handlers in Netty are composable and composition is at the very core of most maintainable and extendable code.

Our RPS layer runs multi-threaded since Netty per se is multithreaded. However, that doesn't mean that Netty requires a thread per request. Therefore, Netty is able to handle more concurrent connections with less available memory compared to a thread-per-request approach. With less threads running on your server the operation system will be less busy doing context switches and other thread related overhead. This can lead to a performance increase and allows us to guarantee a high throughput of incoming requests.

Authentication

To ensure that we are talking to the correct module will sign each message's data payload with HMAC[3]. Every peer using our RPS module uses the same secret key for hashing and hash verification. If the Hash of an incoming message doesn't match we will discard this message and close the connection permanently (see exception handling section).

Message Format

Get local view of a peer:

size	type = 542

No DATA needed for this type of message.

Response to return the local view of a peer:

size	type = 543
HMAC of DATA	
DATA (list of local view)	

[1] http://netty.io/
[2] https://github.com/ReactiveX/RxNetty
[3] https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

As serialization format for DATA we are going to use Google's Protobuf [4] protocol. The message format of the peer's local view will be defined later.

Exception Handling

Protocol errors and corrupt data are caught when parsing incoming messages. In that case we close the connection permanently because we assume that it is a malicious peer or attacker because we rely on the fact that TCP is reliable so no incorrect message can be received from a non malicious client.

[4] https://developers.google.com/protocol-buffers/