Interim report

Jan-Cedric Anslinger Jörn von Henning

June 30, 2018

1 Project architecture

In our project, we use TCP for "formal communication" to communicate messages where we need to be sure that the receiver is getting the signal and UDP to send the actual content of the connection. As a result, we need to run at least a TCP-Listener and a TCP-Writer on each peer. Additionally, there is for each "tunnel-connection" a UDP-Listener and a UDP-Writer. To enable a good performance, we run these processes asynchronously in a multi-threaded architecture so that there is no blocking happening concerning the different connections and tasks of a peer. The peer itself is handling all incoming messages based on event-loops. For incoming TCP- and UDP-messages, there is a separate channel running for each of them so that these messages are directly forwarded to the controller. The controller can then decide how to handle these messages. In golang, it's possible to create these event-loops by running an asynchronous listening-function which is forwarding the received values into a channel. By using a range function, we can then access all values in the channel so that the controller can do his job.

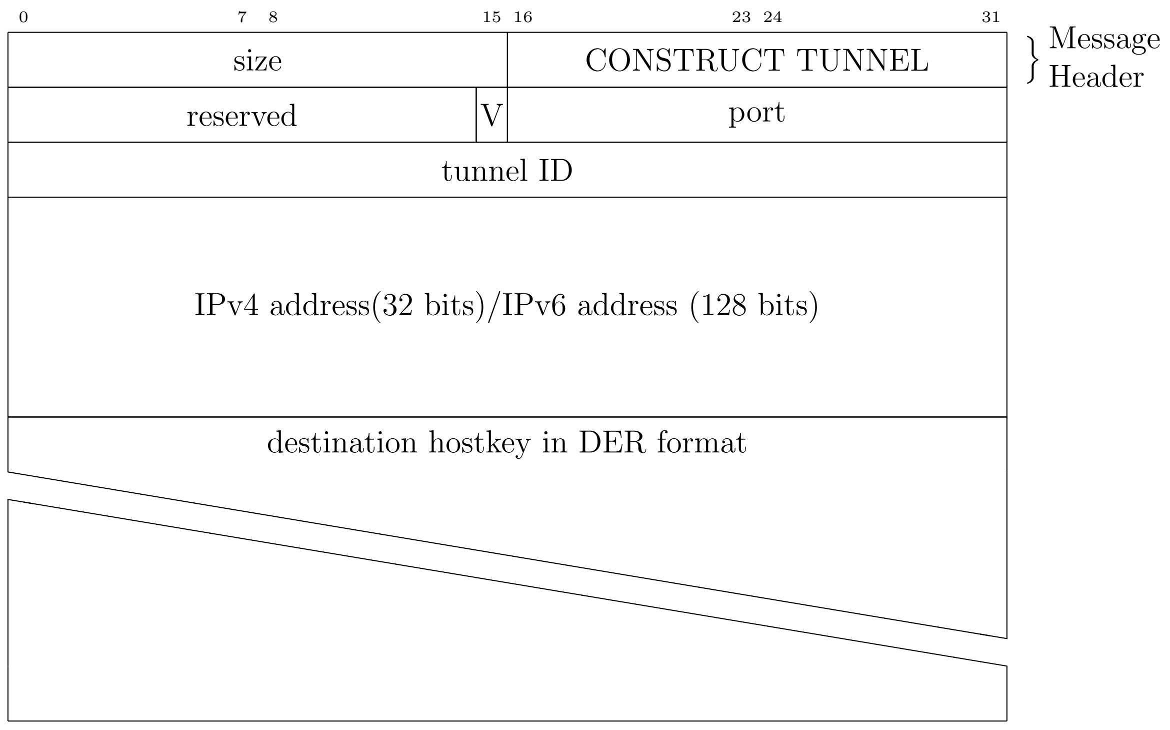
2 Inter-module protocol

2.1 Message types

Currently we have following messages defined. More are to come.

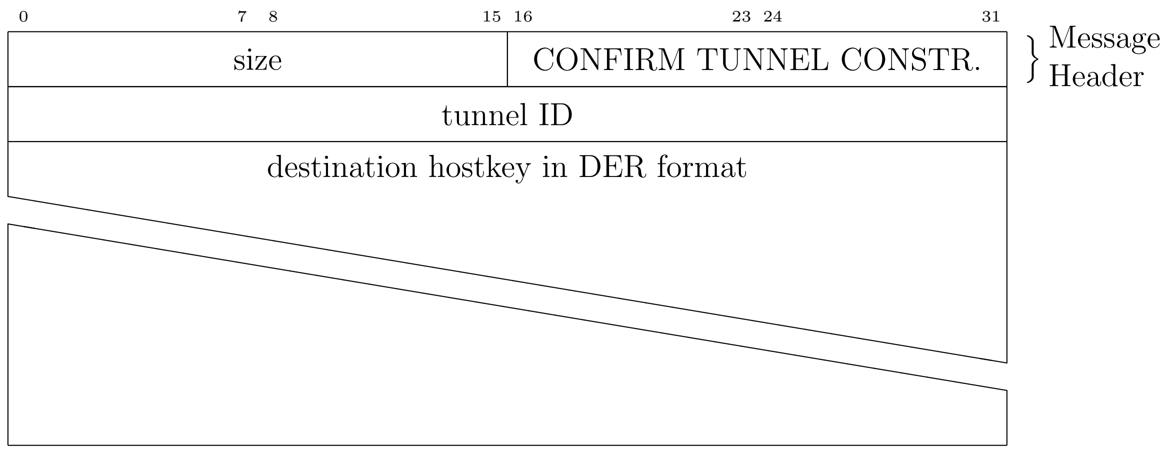
2.1.1 CONSTRUCT TUNNEL

This message is sent by the onion module in order to construct a tunnel between two peers. This message is identified by `CONSTRUCT TUNNEL` Message Type (567). The version of the network address is specified by the flag V; it is set to 0 for IPv4, and 1 for IPv6. Moreover, this message contains a tunnel ID, the port the onion module it is listening for incoming UDP messages, the identity and network address of the destination.


0
7
8
15
16
23
24
31
size
CONSTRUCT TUNNEL
Message Header
reserved
V
port
tunnel ID
IPv4 address(32 bits)/IPv6 address (128 bits)
destination hostkey in DER format

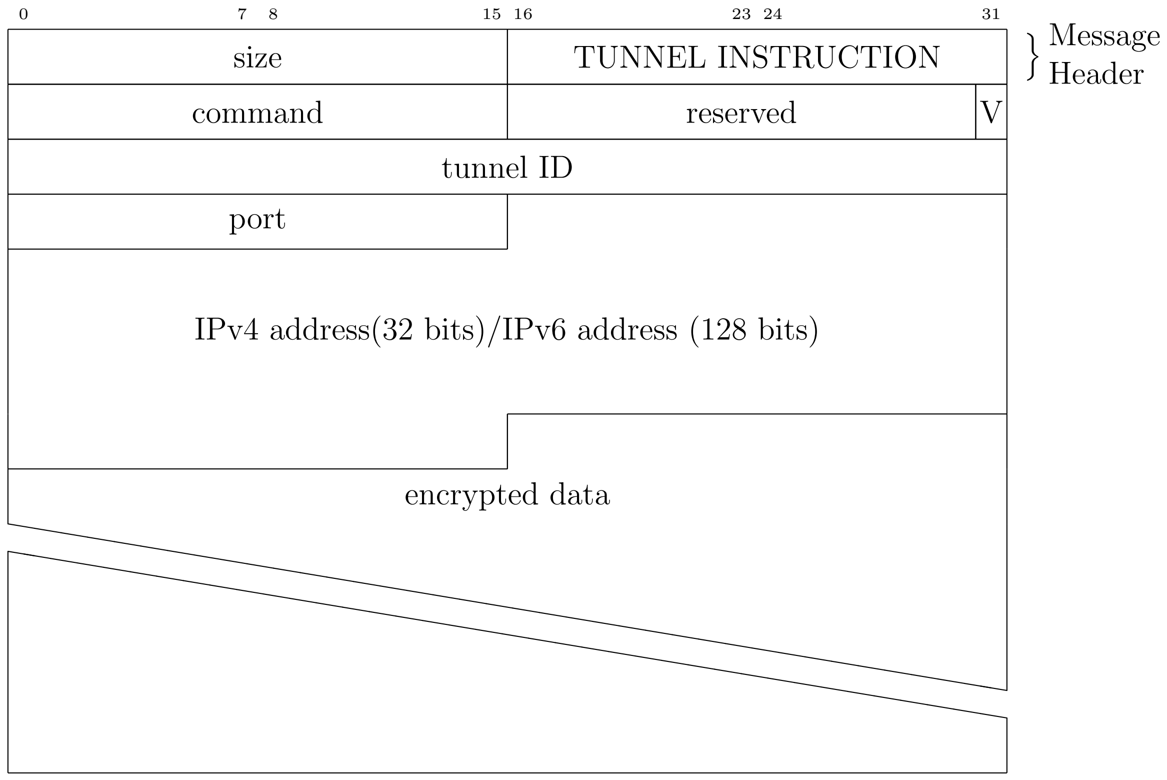
2.1.2 CONFIRM TUNNEL CONSTRUCTION

This message is sent by the onion module in order to confirm the requested tunnel construction. This message is identified by `CONFIRM TUNNEL CONSTRUCTION` Message Type (568). It contains a tunnel ID and the identity.



2.1.3 TUNNEL INSTRUCTION

This message is sent by the onion module in order to request other peers to executed the specified command. This message is identified by `TUNNEL INSTRUCTION` Message Type (569). The version of the network address is specified by the flag V; it is set to 0 for IPv4, and 1 for IPv6. Furthermore, this message contains the command, which has to be executed by the receiving peer, the tunnel ID, the TCP Port of the final destination peer (inside the currently constructed tunnel), the network address of the final destination peer and data which is encrypted with the public key of the subsequent peer (hence, one can forward the message until the last peer of the already builded tunnel has received the message).



2.2 Error handling

Concerning error handling, we are still working on a solution. Our current approach is to implement an event loop with a separate error-channel for

TCP- und UDP-messages. This channel is available for all process so that if an error occurs somewhere, it gets forwarded by the channel to the error handler which can then decide how to handle the problem and if other peers or modules should be contacted. The benefit of this solution is that errors concerning the UDP-connections won't stop the peer from working. Only problems concerning the actual runtime of the peer itself or the TCP-connection can interrupt the service.