Final Report

16 Onions

Josef Stark Charlie Groh

August 17, 2017

1 General

We are the team "16 Onions" consisting of Josef Stark and Charlie Groh, and our goal is to develop a prototype implementation of the Onion module.

2 Application internals

In this section we will describe the general structure of the application, the network protocol we implemented, as well as an overview over how the source code is organized.

2.1 Network protocols

For the communication between distinct onion instances we decided to use both TCP and UDP as underlying protocols in order to avoid reinventing the wheel, since they both fulfill the respective requirements perfectly.

Control messages, i.e. messages for authentication, tunnel construction, tunnel destruction and heartbeat are transferred and forwarded over TCP, because for those messages it is very important that they actually arrive or that we get feedback if one of those messages could not be delivered to the target, so we can react in an appropriate manner—e.g. assume that the corresponding node went down and construct an alternative route. TCP satisfies these requirements as it acknowledges the reception of messages, resends messages if necessary and reports a failure if a message still didn't provoke an acknowledgement after a few retries. If we used UDP instead, we would have had to re-implement all of these features by hand.

User data messages, i.e. messages containing VoIP and cover data, are transferred and forwarded over UDP, since for those a short delay is a requirement which UDP can satisfy. UDP is packet based and does not acknowledge the arrival of packets at all, so the lower delay that this causes comes at the price of possibly losing some packets which are not resent and therefore never reach their target, without the sender being informed about the loss. This is acceptable for VoIP and cover data.

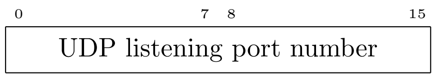
Other modules can use this module to send user data messages (UDP), while control messages are only used internally by the module for tunnel management tasks.

2.1.1 Network abstraction layer

At the lowest level of our application lies the so-called multiplexer, which abstracts away low-level protocol details and offers a clean interface for message interchange between nodes. The core tunnel mechanisms exclusively use this interface. The main tasks of the multiplexer are:

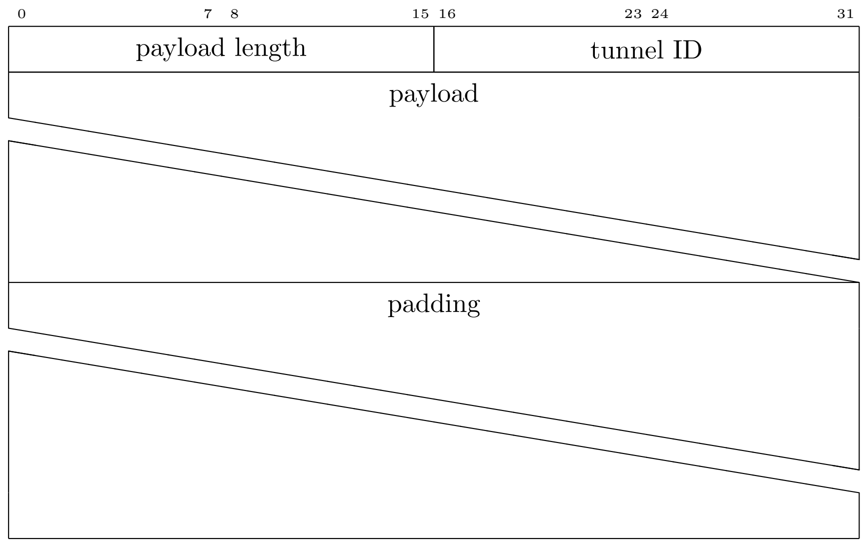
- Establishing and terminating control data channels to other peers using TCP
- Establishing and terminating user data channels to other peers using UDP
- Padding/Unpadding messages to/from a fixed, configurable size while offering transport functionality of variable-sized payload
- Correctly assigning and relating incoming and outgoing messages to the tunnel they belong to

Upon receiving the request to connect to a certain peer, the multiplexer first establishes a TCP connection to that peer for transporting control data and then immediately sends a packet to that peer containing the UDP port on which it is listening for user data:



(TCP message)

This is the only message type that is exchanged on its own. After it has been sent, messages are only sent upon request of the core tunnel mechanism. They can be transmitted over TCP or UDP, which is controlled by a flag of the corresponding function. They have the following structure:



(TCP or UDP message)