Interim Report

Group 31
Zain ul Abideen & Furqan Shakoor
Module: Onion Forwarding

Process Architecture

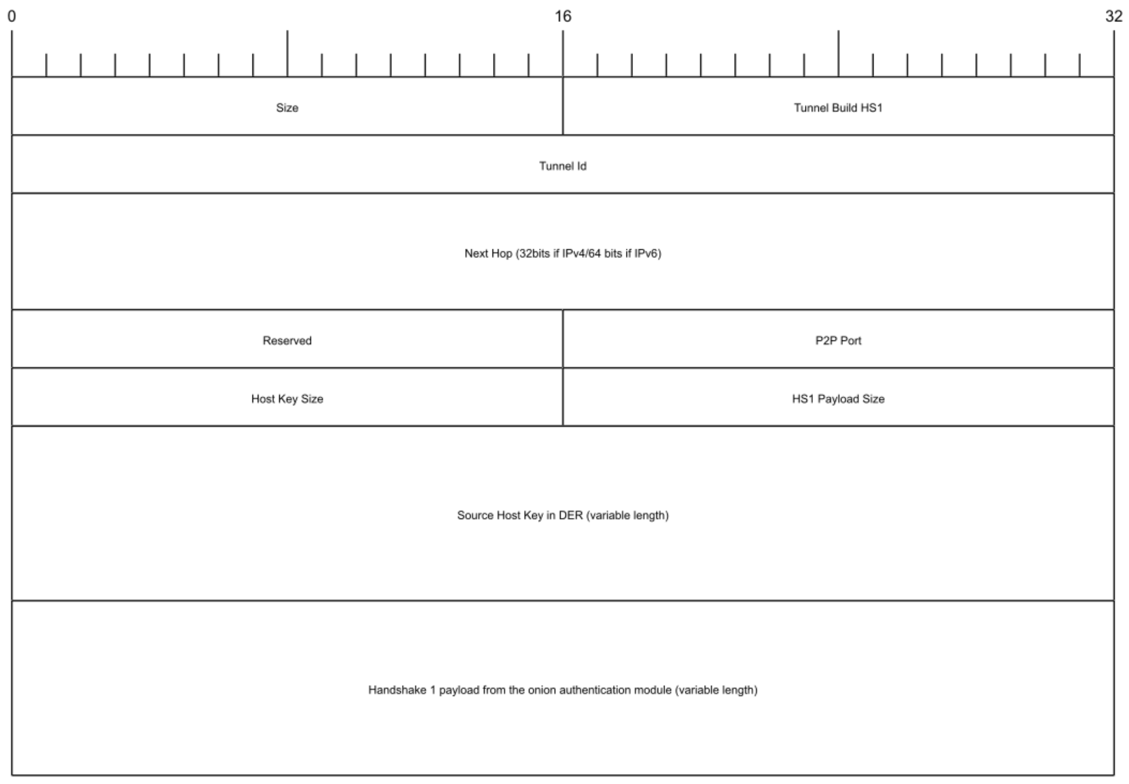
We will have a multi threaded architecture where each request will be handled by a different thread. In order to reduce the cost of constructing and destroying threads, we will use a python thread pool. The main thread will poll the API and P2P endpoints to listen for new requests. Any new request will be handled by one of the threads from the thread pool. We can keep the size of the thread pool fixed. The advantage of this approach is that we have an efficient utilization of resources with multiple threads and handling each task in a different thread is easier.

P2P/Inter-module protocol

Message Formats

In order to avoid analysis on the size of packets, all the packets in this module will have a fixed size of 64KB. Any packets smaller than that will have padding added to them.

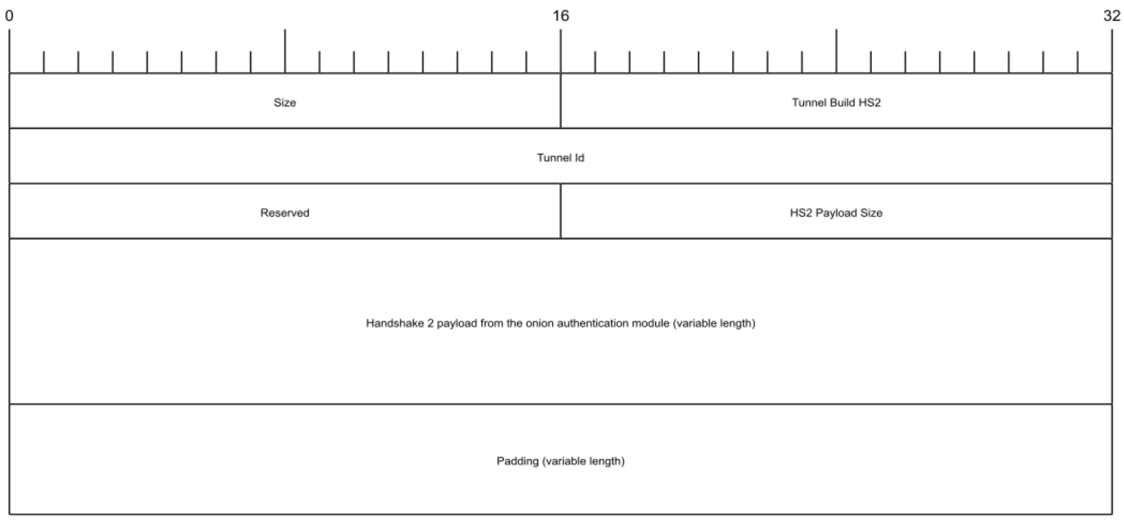
Tunnel Build HS1


0
16
32
Size
Tunnel Build HS1
Tunnel Id
Next Hop (32bits if IPv4/64 bits if IPv6)
Reserved
P2P Port
Host Key Size
HS1 Payload Size
Source Host Key in DER (variable length)
Handshake 1 payload from the onion authentication module (variable length)

This message will be used to build tunnels. Tunnel Id is a random id generated by the peer initiating the construction of the tunnel. This Id will be used to refer to this tunnel in the future.

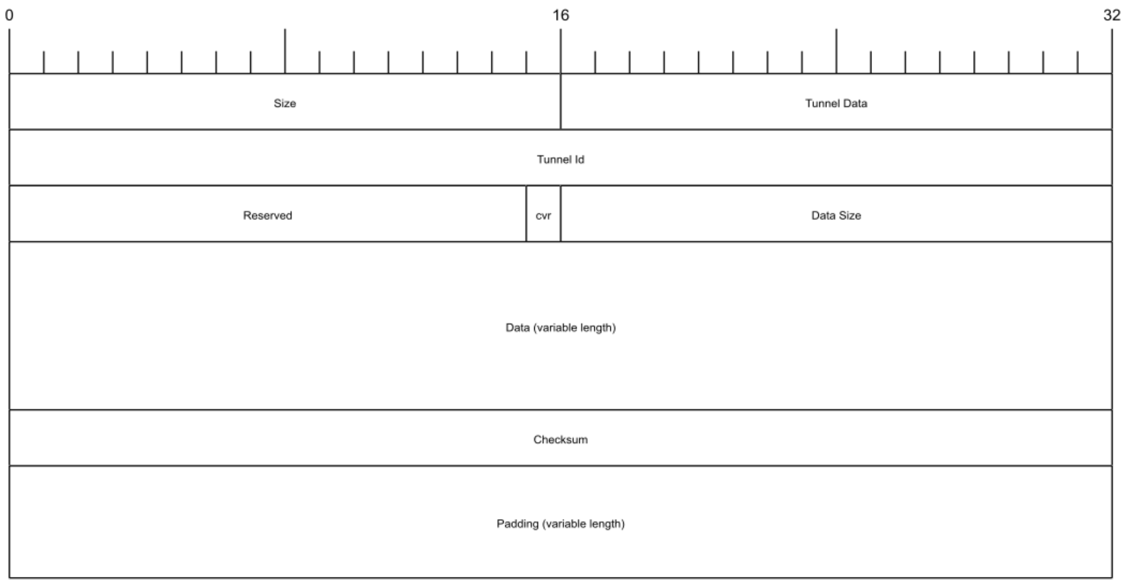
This packet will be sent iteratively through a single tunnel by the tunnel originator. When this packet reaches the edge of the tunnel, the last node will take a look at the value of “Next Hop” and send the packet to that node. This packet also contains the Host Key of the source and the handshake payload from the onion authentication module so that a symmetric key can be established.

Tunnel Build HS2


0
16
32
Size
Tunnel Build HS2
Tunnel Id
Reserved
HS2 Payload Size
Handshake 2 payload from the onion authentication module (variable length)
Padding (variable length)

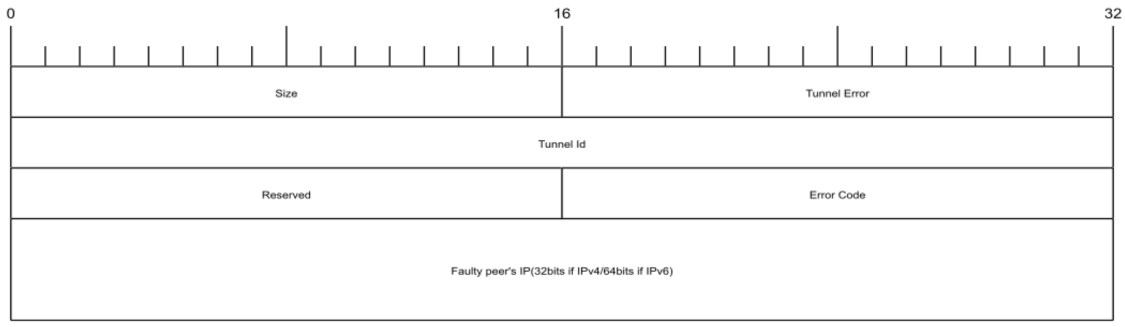
This message is a response for Tunnel Build HS1. It will go back to the source node through the same tunnel the handshake 1 came from and will establish a symmetric key between the sender and the source of the tunnel.

Tunnel Data



This message will be used to send data through an established tunnel. It will contain the tunnel Id through which the data is to be sent. The bit “cvr” is a boolean, which if set means that the data in this packet is cover traffic.

Tunnel Error



This packet will be sent by a peer to a message originator in case an error occurs during the construction of a tunnel or while sending a message.

Authentication

In order for the onion layer to communicate with any other peer, it must first establish a tunnel with them which comprises of a handshake. If the peer is not running the correct module on that port then the handshake response would either not be received or it would be malformed; in either case, the tunnel will fail to form and a “Tunnel Error” will be sent back through the tunnel to the message originator. After a peer has been identified as faulty, the message originator can simply discard that peer and ask RPS for another random peer.

Exception Handling

Any errors like malformed data or failure to get a response during tunnel building and sending data will be propagated back to the message originator via a “Tunnel Error” message. As soon as the message originator gets this packet, it will drop the faulty peer and form a new tunnel including a new random peer. This mechanism will handle peers going down and broken connections. For checking data integrity we have a field called “checksum” in the tunnel data packet. Checksum will be a hash of the data field. If the computed checksum at the recipient does not match the one in the packet, the packet will be dropped and we will not consume any corrupted data.