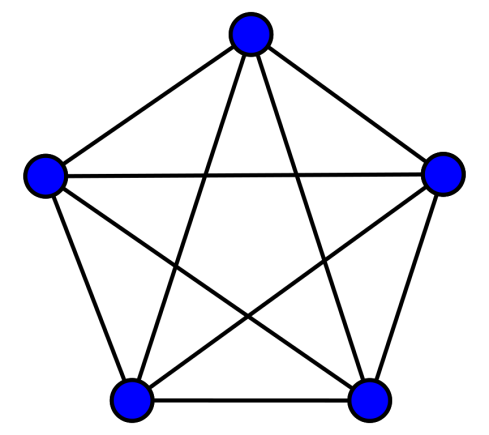
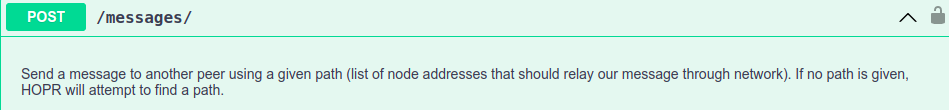
**Test Plan - Messages**

**K5 Graph**

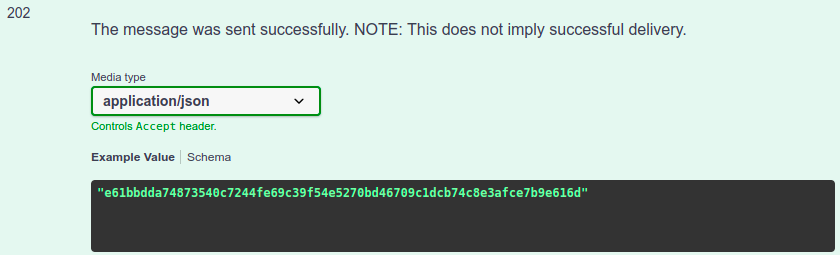


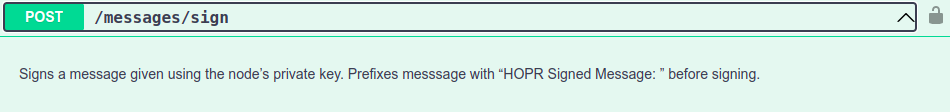
All the HOPR graph nodes are interconnected to each other at all times, meaning that each HOPR node has a Websocket server running and other HOPR nodes connected to it act as Websocket clients. So for a k5 graph we will have 5 Websocket servers running and 5 Websocket clients connected to each, making up of 25 Websocket clients.

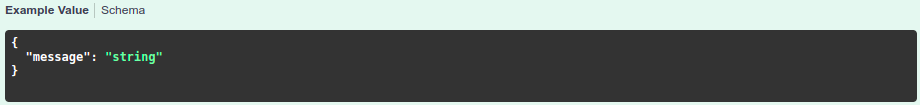
For interacting and controlling the nodes, simple HTTP REST requests are used.

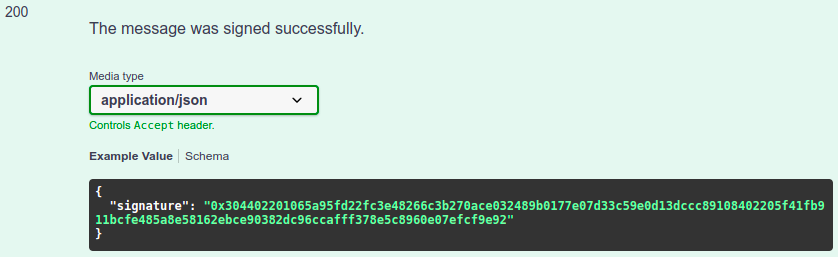


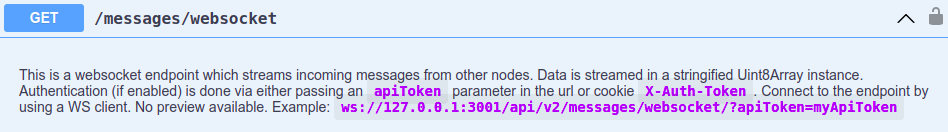














**Functional Test Scenarios**

1. **Send message between 2 nodes with no hops**

- Preconditions: Have a channel open between *node1* and *node2*

Step1: Send message from *node1* to *node2*

Step2: Check that the message is received on *node2*

- Check the stream of Uint8Array encoded numbers

Step3: Check that other nodes are not visited

Step4: Check that the message is not received by other nodes

Step5: ?Check that the balance is/not charged by the message?

1. **Send message between 2 nodes with one defined hop peerId**

- Preconditions: Have a channel open between *node1*, *node2*, and *node3*

Step1: Send message from *node1* to *node2* with *node3* as path

Step2: Check that the message arrives at *node2*

Step3: Check that *node3* was visited, and *node4* and *node5* are not visited

Step4: Check that the message is not received by other nodes

Step5: ?Check that the balance is/not charged by the message?

1. **Send message between 2 nodes with 1 random hop and no defined path**

Note: The random hops should be ignored in this case

- Preconditions: Have channels opened between *node1*, *node2*, *node4*, so that we can check that the random path chosen is through available *node4*. In the case there are channels open and funded between all k5 graph test setup nodes, then we have to check node 3, 4, and 5 just to see that the system has visited at least one node from those remaining 3 nodes.

Step1: Send message from *node1* to *node2* with 1 hop defined

Step2: Check that the message arrives at *node2*

Step3: Check that node4 was visited (or one of node 3, 4, or 5)

Step4: Check that the message is not received by other nodes

Step5: ?Check that the balance is/not charged by the message?

1. **Send message between 2 nodes with one defined hop and 1 random hops**

Note: The random hops should be ignored in this case

- Preconditions: Have channels opened between *node1*, *node2*, *node3* and at least one other node, just to see that the random hop does not go through other nodes.

Step1: Send message from *node1* to *node2* with *node3* as path and 1 hops

Step2: Check that the message arrives at *node2*

Step3: Check that 1 random hop is ignored and only *node3* is visited

Step4: Check that the message is not received by other nodes

Step5: ?Check that the balance is/not charged by the message?

**Edge Cases**

1. **Send reply back from *node2* to *node1***

- Preconditions: Have a channel open between *node2* and *node1*

Step1: Send a test message from *node1* to *node2*

Step2: Send a test message back from *node2* to *node1*

Step3: Check that the test message arrives at node1

Step4: Check that other nodes don’t receive the message

Step5: Check that other nodes are not visited

1. **Send test message between 2 nodes with a fund-depleted hop in between**

- Preconditions: make sure *node3* is out of funds

Step1: Send test message between *node1* and *node2* with *node3* as a defined path/hop

Step2: Check that message does not arrive at node2 (usually with a 5-10s timeout)

Step3: Perhaps check that node3 was visited (has to investigate what happens @node3 in this case and check against documentation)

**Notes**:

- For the provided test environment with 5 nodes, all the nodes are already funded but in a real word scenario the HOPR clients have to fund the nodes, so a few more scenarios could be built for those cases too.

- I have to still learn more about the system, and I’m sure I will be able to come up with more scenarios.

**End-to-end Testing**

Develop some end to end workflows to test some real-life use cases of the system.

1. Create some channels between certain nodes, fund them, send messages, etc

**Fuzz Testing**

1. Send 100 (or a certain fixed number) of messages between random nodes with random generated message (long and short) just to see what happens to the system

- Eventually, we can make a variation of this so that we will cover the following cases

1. Direct messages
2. Using defined hop address
3. Using a certain number of random hops chosen by the system. in this case its a bit more challenging to check where the hops are routed through the system, but its possible with a controlled k5 graph in the test environment for example, and we can go around over all the nodes and check the visited timestamp, make a difference between them and see which ones are newer compared to the last visitation.