A Programmer's Guide to Modified Onion Routing and its Proof of Concept

Introduction:

This programmer's guide is in addition to the information provided in *A Report on Modified Onion Routing and its Proof of Concept.* This document discusses the basic architecture, the code layout, steps to run and test result. This document provides selective information to the programmer to enable her to make changes to the code and write test cases. If the programmer needs detailed architectural information including packet structure then she must look into the report mentioned above.

Basic Architecture:

The base architecture of the software is shown in the following diagram. Multiple applications can register against a single application proxy in the sending side. Similarly, multiple applications can register against a single exit funnel in the receiving side. Onion Proxy and exit funnel are designed such a way that they can be decoupled from host and run in dedicated proxy servers.

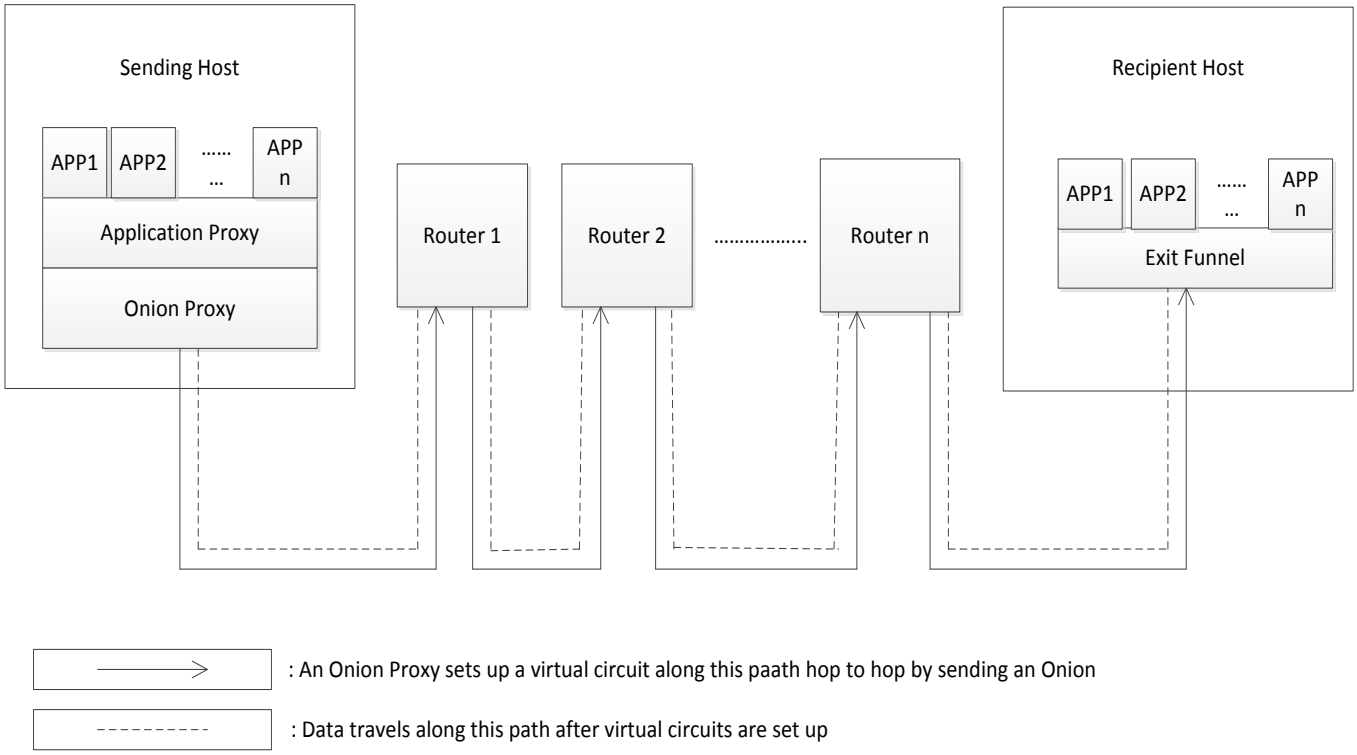

Figure 1- Basic Architecture Diagram of M. OR

Flow among Application, Application Proxy and Onion Proxy

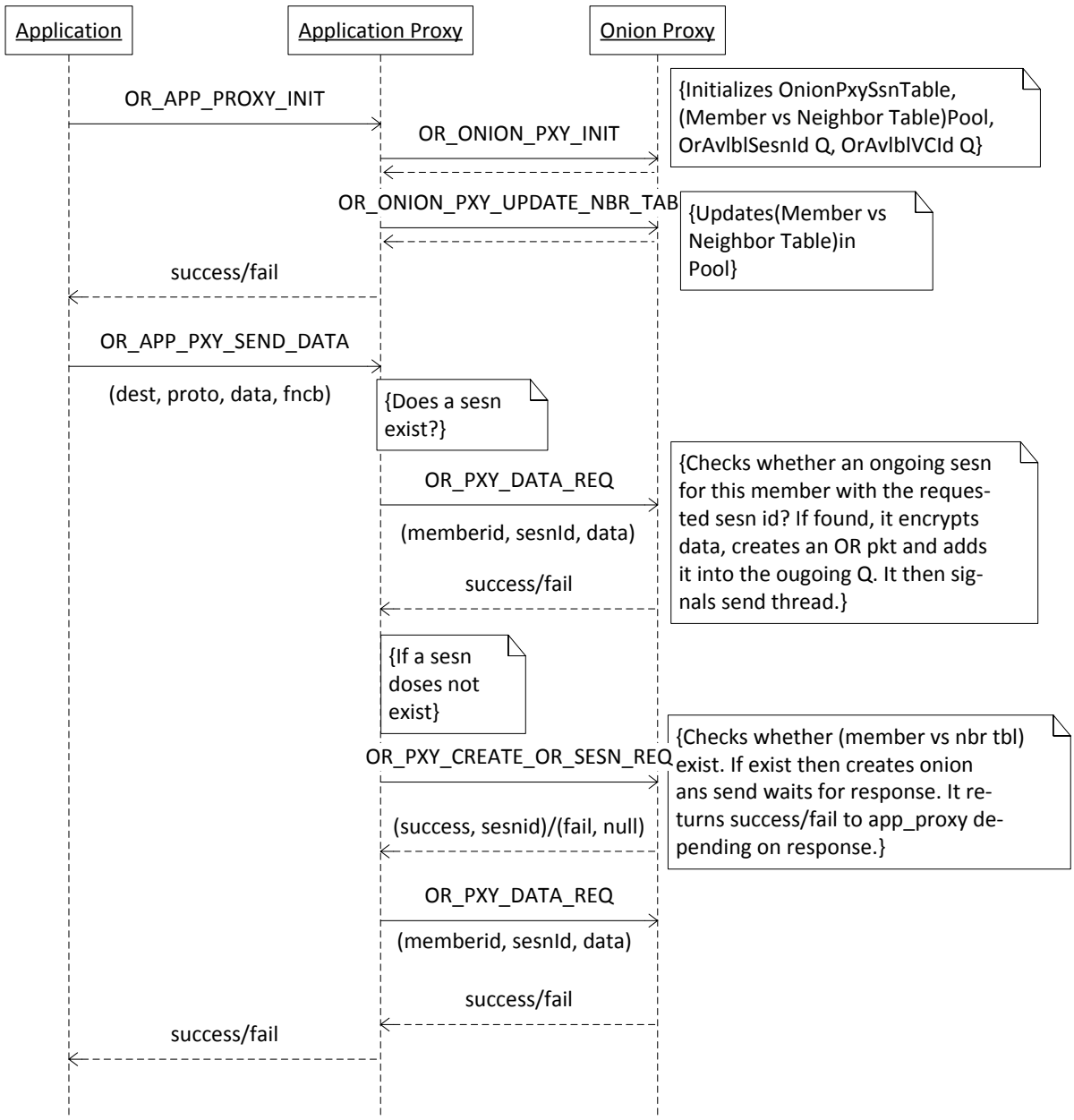

Figure 2 - Execution flow among App, App Proxy and Onion Proxy

Flow between Application/Receiver Proxy and Exit Funnel

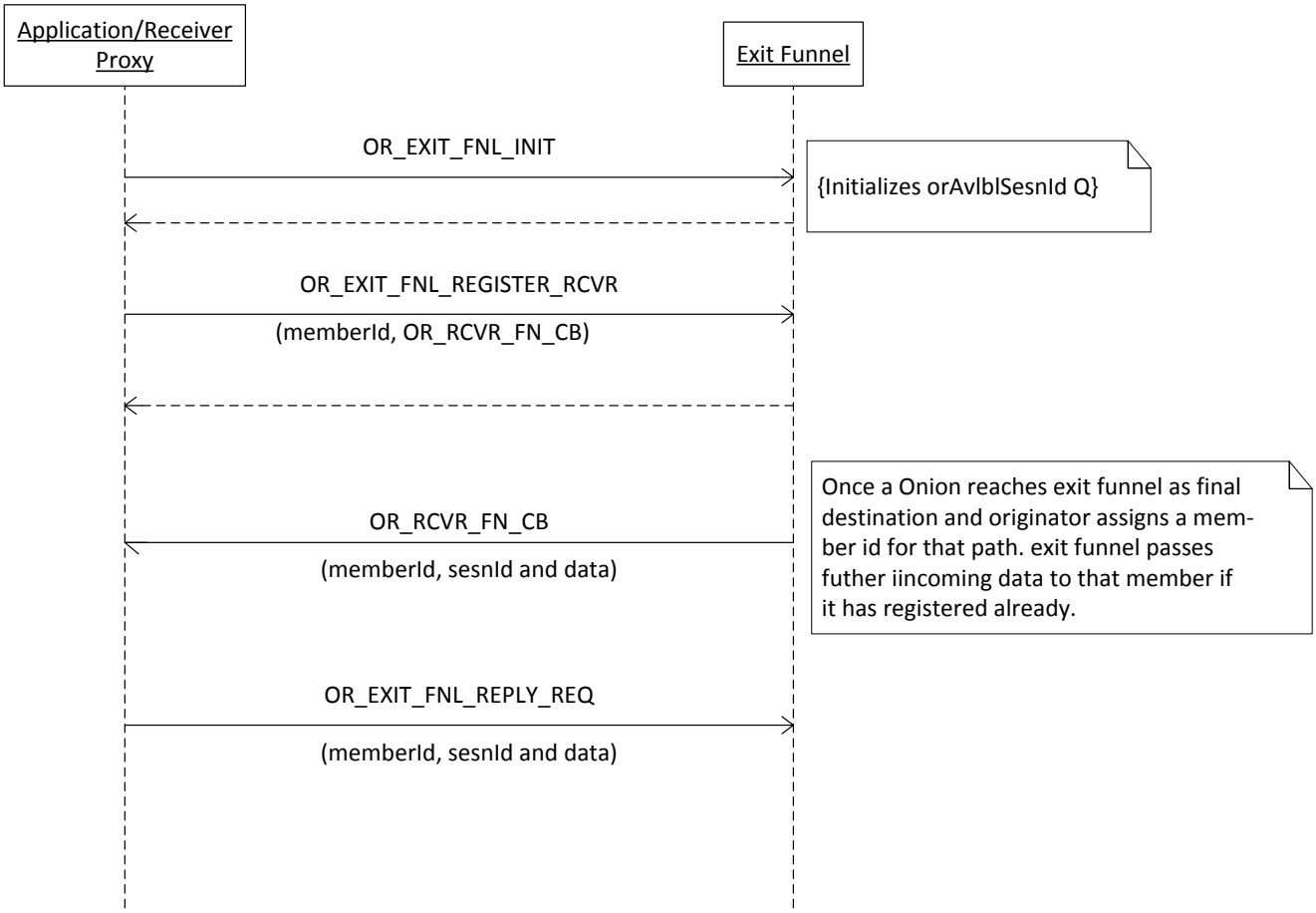

Figure 3 - Execution Flow among Application/Receiver Proxy and Exit Funnel

Configuration File (at the absence of an ORCS)

The absence of ORCS generates a requirement of a configuration file from which each member can read memberId, IP, port and public key. The file content look like the following:

```
Member Id ||   Device Type   ||      IP      || Port || Public Key
    2     ||  OR_ROUTER      || 127.0.0.1    || 3002 ||  14020204050a040a0a0204140a0a0a0a0a0a0a050a140a04040a0a0405040a0a
    1     ||  OR_ROUTER      || 127.0.0.1    || 3001 ||  0a0a0a0404140a140a020a0a040414040a140a04040a1414040a0a04040a040a
    3     ||  OR_END_DEVICE  || 127.0.0.1    || 3003 ||  140a0a04010a0a0a0a0a0a0a0a0a010a1404050a040a050a1402050a04141414
    4     ||  OR_END_DEVICE  || 127.0.0.1    || 3000 ||  04010a0104140a0a0a14040a140a040a0a040a0a140a0201050a0a041401010a
```

Figure 4 - Configuration file content

Code Layout

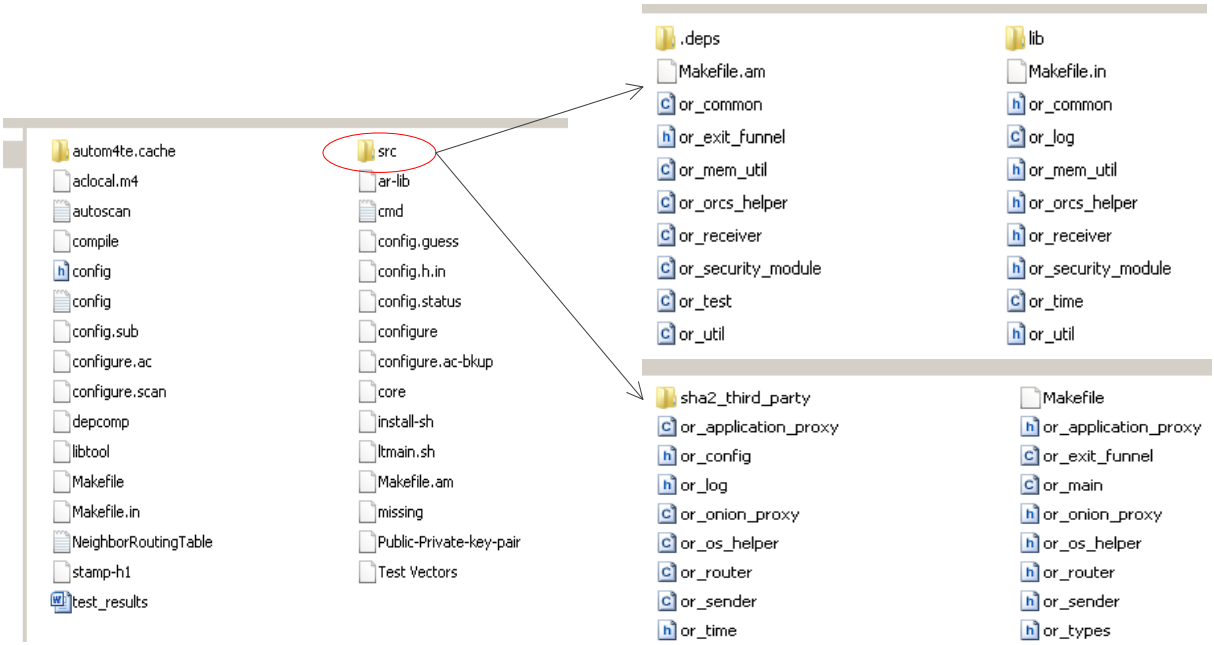

Figure 5 – Code Layout

API Files

The API files relevant to sending side and receiving side are respectively or_application_proxy.h and or_exit_funnel.h.

Log Module

Macros OR_LOG and OR_LOG_NO_EOL is defined in or_log.h file. The former prints log to input file and console with a timestamp and by inserting a newline by default at the end of the line. The later does not perform these two tasks. The logging can be done using three levels - OR_LOG_LEVEL_ERROR, OR_LOG_LEVEL_CRITICAL and OR_LOG_LEVEL_DEBUG. They are all enabled for now. If the programmer wants to selectively enable logs then she has to define three compiler flags with different levels. She will also have to make change in or_log() & or_log_no_eol() to compare 'level' with the values passed in these three flags.

Project Configuration Header File

The primary configuration parameters for the project are defined in or_config.h.

Type Header File

All primary type structures are defined in or_types.h.

Steps to Run

1. Go to project root.
2. Do make clean and make.
3. Open four terminals.
4. Start the program by choosing a role. The following commands are relevant when the configuration file shown above is used.
   src/m_onion_routing role=sender port=3000 logfile='DirPath'/or_snd_log.txt nwkLayoutFile='DirPath'/NeighborRoutingTable.txt

   src/m_onion_routing role=router port=3001 logfile='DirPath'/or_router1_log.txt nwkLayoutFile='DirPath'/NeighborRoutingTable.txt

   src/m_onion_routing role=router port=3002 logfile='DirPath'/or_router2_log.txt nwkLayoutFile='DirPath'/NeighborRoutingTable.txt

   src/m_onion_routing role=receiver port=3003 logfile='DirPath'/or_rcvr_log.txt nwkLayoutFile='DirPath'/NeighborRoutingTable.txt

5. The sender needs to be launched at the end as the connections are TCP based.
6. After launching the sender will send an onion, establish a VC based path and send the handshake message:
   "I am ur anonymous friend. Accept Greetings."
   The recipient responds by sending handshake response:
   "Hello anonymous friend. Greetings accepted."
7. After this sender process launces a command line app with the following options:
   Try Someting...
   a. Replay Onion
   b. Send a delay tolerant msg
   c. Send a delay intolerant msg
   d. Inject dummy/padding messages

8. User can try different options from the above list.

Test result

Test Case 1

On an average how many dummy messages are required to inject for a message to reach to the destination through 2 (2-1) threshold pool mix.

Result:

No of sender's message	Threshold pool mix (n[pool]-s[threshold])	Average No. of Dummy messages	No of trials
1	(2-1)	2.3	10 (3, 1, 1, 2, 3, 1, 2, 2, 6, 2)
2	(2-1)	3	5 (6, 3, 2, 3, 1)
1	(3-1)	4	10 (2, 3, 3, 12, 5, 0, 9, 4, 1, 1)
2	(3-1)	5	5(8, 17, 2, 5, 3)
1	(3-2)	6.4	10 (3, 7, 1, 15, 1, 1, 11, 7, 3, 15)
2	(3-2)	7	5(6, 10, 6, 4, 9)

Test Case 2

Replay an Onion.

Result:

Onion rejected by first router.

Test Case 3

Send 50 delay intolerant messages to recipient.

Result:

All messages acknowledged by recipient.

Test Case 4

Send multiple back to back delay tolerant messages to recipient.

Result:

Result covered by results of Test Case 1.

**********END**********