P2P Final Report

Group Members:
- Hamza Zafar (03680906)
- Muhammad Bhatti (03681698)

Course: Peer to Peer Systems and Security

Project Module: Gossip

Group Number: 35

Process architecture

Since we are using Java NIO (Non-Blocking IO), our process architecture is inherently based on event loops. SocketChannels are registered with the Selector Object. Whenever a read or accept event is generated, the selector object returns us all the SocketChannels that need to be serviced.

Our Server Socket is registered with Selector for ACCEPT events, which means that whenever a new Peer wants to connect to Server Socket, an accept event will be generated by Selector, we can then simply call the accept function on our Server Socket to complete the connection establishment. This has a great advantage because using this approach the Server Socket does not block on accept call, instead the thread can do some other work. After accepting a new connection, we register the new SocketChannel with Selector Object for READ events. This means that whenever SocketChannel has some data that needs to be read a READ event is fired by the Selector. We can then service all the SocketChannels that have some data that need to be read.

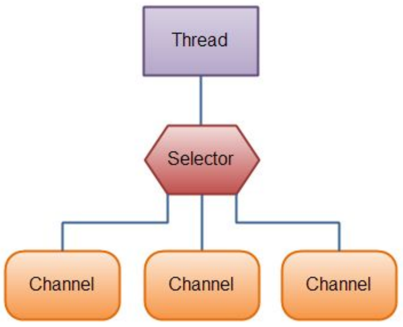


Figure: Java NIO Selector [2]

Additionally we are using NIO Buffers to interact with NIO Channels. Data is written from NIO Buffers into NIO Channels, similarly data is read from NIO Channels into NIO Buffers.

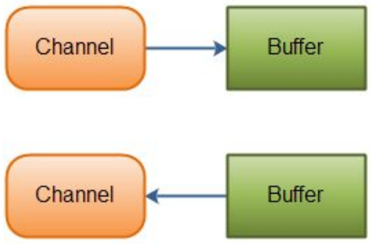
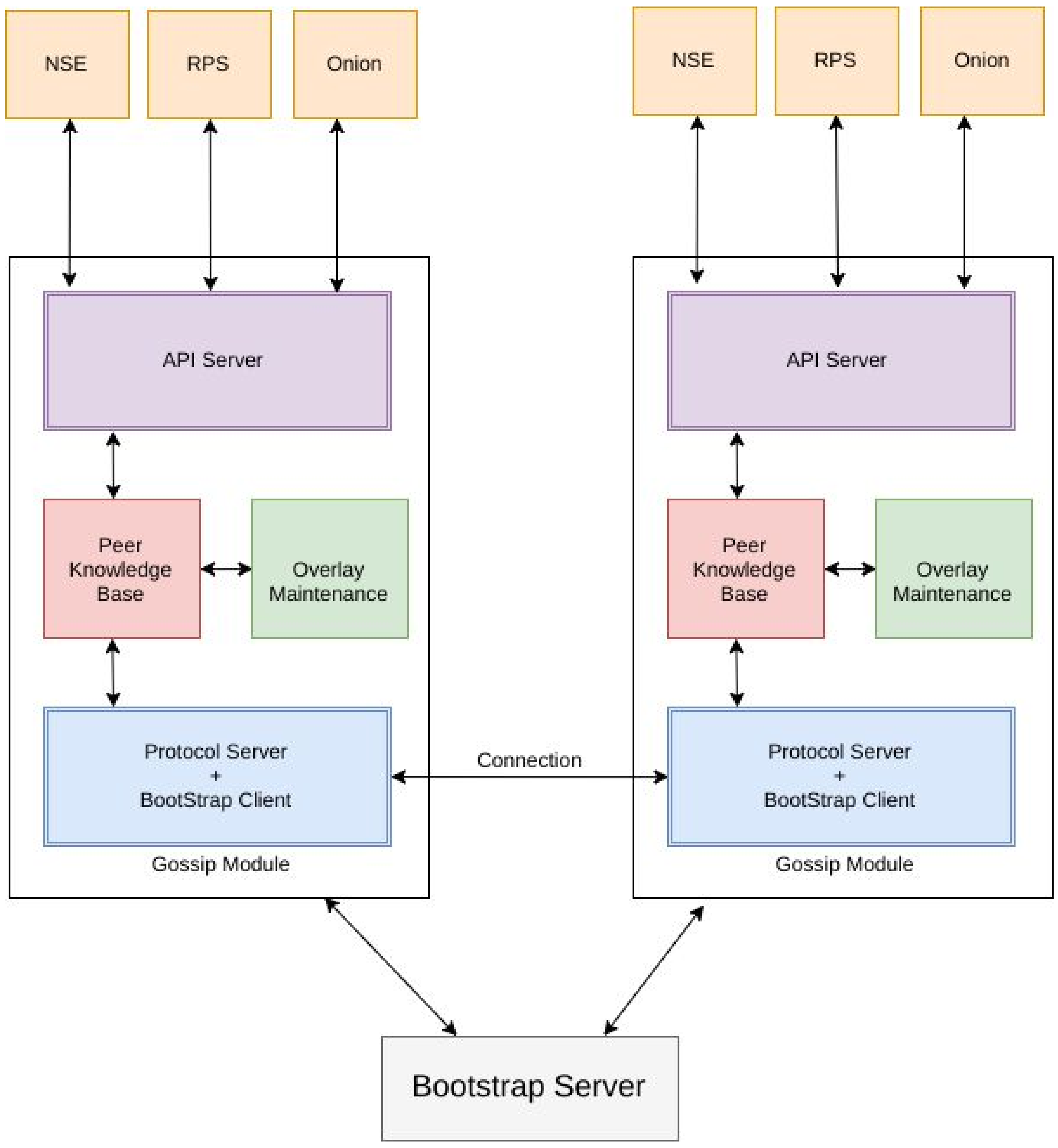


Figure: Java NIO Channels and Buffers [3]

To understand the NIO Ecosystem, we read the Java NIO Book by Ron Hitchens [1]
As a starting point, we also used sample code snippets published in the NIO Book.

Inter-module protocol



Bootstrap Server:
Firstly, one of the most important role in our gossip module is that of the bootstrap server. In short, a bootstrap server allows new peers joining a swarm to get information about some peers in the network. The bootstrap server maintains a list of all online peers. Whenever a new peer joins the network it requests the Hello Message to bootstrap server. The bootstrap server responds with a GOSSIP PEER LIST message. The new peer now has knowledge of some of the peers in the network, and connects to them. Since all peers participate in sharing information about their neighbors, the new peer gets known across the network/swarm, and learns about new peers.

Protocol Server:
The protocol server has a server socket which is used to accept connection requests from other peers. Protocol Server module also a Bootstrap Client module, when the gossip module is started, protocol server requests the list of peers from Bootstrap Server over a Datagram Channel (UDP based). Protocol Server interacts with the Peer Knowledge base, whenever a new connection is established with a new peer, it stores the reference of Peer's SocketChannel in Peer Knowledge base. Protocol Server uses Java NIO Selector (Event loop) to service all connected peers.

API Server:
The API server has a server socket which is used to accept connection requests from other modules for example NSE, Onion etc. API Server interacts with the Peer Knowledge base, whenever a new connection is established with a new module , it stores the reference of module's SocketChannel in Peer Knowledge base. API Server uses Java NIO Selector (Event loop) to service all connected modules.

Overlay Maintenance:
Overlay Maintenance module is used to maintain the DEGREE of peer. DEGREE is the number of peers a given peer should have. After every 10 seconds by default (delay could be changed by changing the **peer_list_send_delay** setting in conf file) the overlay maintenance thread queries the Peer Knowledge Base to get a list of all connected peers, it then broadcasts this list of connected peers. In this way peers can learn about new peers. Additionally, it checks if the number of connected peers is less than the DEGREE, then it opens connections to Peers IPs learnt from other Peers.

Peer Knowledge Base:
Peer Knowledge Base contains references to all connected peers and all connected modules. It also contains a cache of messages which are waiting for validation from modules. Older Entried in cache are discarded by newer entries. We have just 3 threads in our project (Protocol Server, API Server and Overlay Maintenance), therefore Peer Knowledge Base acts as point of data exchange between these threads. We have made sure that accesses to data structures in Peer Knowledge Base is highly synchronized.

Message Formats

Module Message Type	**Message Code**
GOSSIP ANNOUNCE	500
GOSSIP NOTIFY	501
GOSSIP NOTIFICATION	502
GOSSIP VALIDATION	503
Gossip internal Messages	**Message Code**
GOSSIP HELLO	504
GOSSIP PEER LIST	505

HELLO

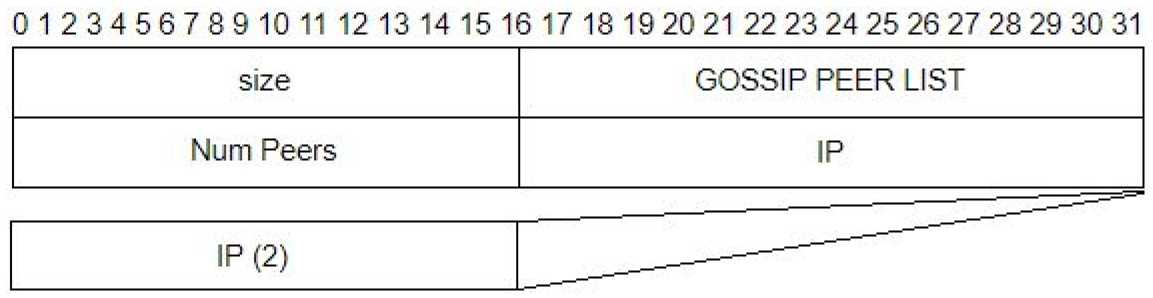
When peers connect to Bootstrap Server or other peer modules, they send the Hello message after the establishment of connection. Hello Message is sent by both, the connection initiator and the receiver. We are using Source IPs as Peer Identifiers, therefore the entities can verify the authenticity of other entities. If the source ip in Hello Message does not match with the Socket's address then the connection is terminated.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

size	GOSSIP HELLO
Source IP	

PEER LIST

Peer List Message is sent by the Bootstrap Server to the new Peer in swarm. It contains the IPs of all Peers in the swarm, Bootstrap server shuffles the list of IPs before sending. Peer List Message is also used by Overlay Maintenance Thread to gossip the IPs of connected peers. 'Num Peers' indicate the number of IPs that the message contains. Each IP is represented as a 32-bit int.



Cleanup Procedures:

When Peer disconnect from Protocol Server of Modules Disconnect from API Server a read event is generated in Event Loop. The call to read function returns -1 which indicates that the remote entity has closed the connection, we then proceed to remove all the related data from Peer Knowledge Base.

Dependencies:

- Open JDK 1.8
- Apache Maven 3.3.9
- External Libraries used:
    - Apache Commons Cli
    - Apache Commons Configuration
    - JUnit 4.12

  We have added dependencies in pom.xml, simply run `mvn install` to download all dependencies. Additionally we are using Maven Assembly Plugin which packages the dependencies in a single Jar File. Therefore we don’t have to add them in class path. More details in following Sections.

Install and Run:

Run `mvn install` and the follow the steps in following guides

- Procedure For Testing The Overlay Maintenance Of Peers

https://gitlab.lrz.de/zafar/P2P-gossip/blob/master/OVERLAY_MAINTENANCE_TEST.md

- Gossip Announce, Validation, Notification, Notify testing
  https://gitlab.lrz.de/zafar/P2P-gossip/blob/master/END_TO_END_GOSSIP_TEST.md

To run the unit tests, please run `mvn -Dtest=* test`

Future Work

- Proof of work for authentication, currently we have IP Addresses as peer identities.
- We had a plan to add a check if the Source IP in Peer's Hello Message corresponds to the Address of the Socket, in this way we could verify the authenticity. Since we were running multiple peers on a single machine using testing and development phase, we encountered a strange issue. We started Peers which used different IP Addresses from the private address space, when we called the function to get socketaddress, it always returned 127.0.0.1 even though if the address was 127.0.0.2, this problem did not arise when we ran Peers on different machines. Therefore, for the time being we are not verifying the authenticity because we are running multiple peers on a single machine. A possible solution is to use Docker to orchestrate a swarm of Peers, each peer in a separate container which uses a different IP.

Work Division and Effort

- Hamza Zafar
    - Implemented Protocol Server
    - Implemented the Overlay Maintenance
    - Implemented the Peer Knowledge Base and Cache
    - Designing the Code Structure
    - Setting up the build tool
    - Wrote integration test for Bootstrap
    - Wrote a mock module for testing purposes
    - Reading Java NIO book
    - Between 120 to 150 hours in total

- Muhammad Bhatti
    - Implemented Bootstrap server
    - Implemented the API server
    - Wrote JUnit tests
    - Reading Java NIO book
    - Between 100 to 120 hours in total

References:

[1] http://javanio.info/
[2] http://tutorials.jenkov.com/images/java-nio/overview-selectors.png

[3] http://tutorials.jenkov.com/images/java-nio/overview-channels-buffers.png