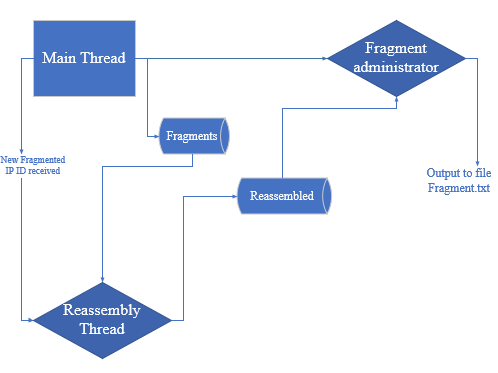
**Intro:**

In this project an IP Packet reassemble was added to detect attacks like the teardrop attack. An overview of how this was built is that the main thread that handled the parsing, filtering, and displaying of packets, received the added functionality of being able to detect fragmented packets. This main thread would first start a thread that handled the reassembled packets, and when a detected fragmented packet was received then it would start a new thread for every new IP ID received. This new thread would handle reassembling of the packets. The main thread would also have a concurrent queue where it stored the fragments for the packets, as well as another concurrent queue where the reassembled fragments would be stored. The reassembled fragments were sent to a file called Fragment.txt for easier viewing. The full functionality is shown in the following graph:



**Main Thread:**

The main thread of the program was taken from the original project 1 code. This code was modified by first changing the class that parsed the IP address to detect when a fragmented packet was received, and to calculate the checksum of the packet. Another addition made was to start a thread before parsing packets that would handle reassembled fragmented packets. This thread was started from a new class that was added called FragmentAdministrator. The values that were passed to it were two concurrent queues. The first has all the fragments that haven’t been reassembled yet, and the second is all the reassembled packets. Another value of time AtomiInteger was passed to it, and this would be used to tell the thread when the main thread was done, so that it would finish as well. When the main thread received a fragmented packet it first checks in a vector to see if the IP ID received matches one it has previously received. If it doesn’t then it starts a thread with the class IPFragmentAssembler. This new class gets two concurrent queues. The first has all the fragments that haven’t been reassembled yet, and the second has all the reassembled packets. If it has received that IP ID before then it will store it in the concurrent queue as map interface with a key of the IP ID, and the value of the parsed ip packet.

**IPFragmentAssembler:**

This class was made to handle the reassembly process of a single IP ID packet fragment. The constructor of this class creates a timestamp of when it was started, which correlates with when the first packet of that IP ID was received. It will also create a treemap object that will hold all its IP fragments. When this class is started as a thread, it will first check if it has successfully reassembled the packet, or it has timed out. If it hasn’t then it will check how long it has been since it started by comparing the timestamp to the current time, if more than 10 seconds have passed then it will set the timeout value to exit the loop in the next iteration.

The concurrent queue of IP fragments is checked for the first value. If the first value has the IP ID that the thread is assigned to then it will grab, it and remove. The fragment retrieved is then checked to see if it is either the first or the last. This can be found since the first fragment will have the more fragment flag set, and an offset of 0. While the last fragment will have the more fragment flag unset, with an offset greater than 0. The first and last packets are stored in their own variables, while all the other fragments are stored into a treemap object. The reassembly process will not start until the first and last packet have been received. Once these packets are received then it is checked if they can be put together without any in between packets, whether there is an overlap or not. If it overlaps it will flag it however. If they can be then they are put together, and the header of the first packet is used.

If the first and last packet can’t be put together, then a function called packetReassembly is called. This function receives a fragment, a treemap object of fragments, an array of bytes to reassemble the payload, and a value that checks how many bytes the current fragment has. The function will then check to see if it can find a fragment in the treemap object that follows the fragment it received, whether it be overlapping or not. If it overlaps it will flag it however. If it can it will extract the payload of the packets and concatenate them together, it will then call itself recursively. This is done until either a packet can’t be found in the treemap object, or the last packet is reached.

When the last packet is reached then it will check if it can put them together, and if it can then it will concatenate the payloads and then return a CompleteFragment object. The purpose of this object is to return whether reassembly was a success, and the payload if it was. If the reassembly process was successful then the payload of the fragment is concatenated with the header of the first packet. Also, if at any point the size of the payload exceeds 64k then it is also flagged. The completed packet is then placed into a model called FragmentModel. The timestamp, and the treemap of fragments is also stored into the model. Furthermore, if any events such as packet overlapping, or the payload exceeding 64k happen then the appropriate sid is also stored in this model. The sid of overlapping is 2, and the sid of exceeding 64k is 3. If neither of these happened, then an sid of 1 is set meaning the reassembly process was perfect. If while retrieving the packets the thread times out, then a fragment model is again created with the packets received and an sid of 4 which signifies timing out. The fragment model is then placed into the concurrent queue of reassembled fragments, and the thread exits.

**FragmentAdministrator:**

This classes main task was to handle the output of the reassembled packets. When this class is started as a thread, it will check if the reassembled packet queue has any items in it. If it does then it will retrieve and remove them. Whenever it retrieves an item, it will record the IP ID of the packet. It will then check if the sid of the reassembled fragment and if its 2 or 4 then it will print either overlap detected for 2 or time out detected for 4. It will then print the reassembled packet.

The secondary task of this class is to flush the queue of any IP ID packets that have already been reassembled. Since it stores all of the IP ID’s of the finished packets, it will loop through this list and remove fragments that have already finished.

**Sample Output:**

