Progress Report 3 (GSOC ‘16)

The project now has been completely moved to this repository. Each scenario now has its own status report showing the success/failure of each test and their respective description. Comments regarding the current progress and suggestions can be made here.

Scenarios Completed from Linux repo

Selective Acknowledgements

I have verified the support for selective acknowledgements through the test for fast retransmit. However, a more rigorous test will be done with assertions for `tcp_info`, but currently using them is creating an issue which I have addressed later in the report.

Fast Retransmit

One successful test for this scenario has been done. I don’t think any variations can be brought in this scenario as the concept itself is straightforward and easy to check.

Early Retransmit

I have currently made one successful test for this scenario, however I am still going through RFC5827 as more study can be done for this case (Plus I am enjoying it :P).

Additional scenarios from the proposal

Simultaneous Connection close

Summary
In this scenario, I check for the case when both the sender and receiver simultaneously close their connections. The conclusions which can be drawn are -

- ➢ After the client sends a **FIN-ACK**, the sender first has to **ACK** before sending out a **FIN** after **close()**'ing the connection.
- ➢ It is not at all possible for the sender to send out **FIN** before **ACK**, even though both sender-receiver simultaneously close their connections. This seems an odd behavior to me.

Urgent Pointer

I have figured out where in packetdrill is the code for urgent pointer, will be using it to complete the 2 additional scenarios mentioned in the proposal.

Current problems

Silly window avoidance

So since FreeBSD uses delayed acknowledgements (until now I have encountered a max of `100 ms`), I wanted to check if Nagle's algorithm is being used by the sender (just for curiosity). So if it does, data will be queued in the sender's buffer until **ACK** is received from the other side. I tried checking this by sending a small amount of data segment (less than **MSS**). However, when we use **PUSH**, the data is instantly sent to the receiver's application, without being queued inside sender's buffer. I was wondering if there could be some way in which we can queue the data in the sender's buffer (and receiver's as well) until it reaches **MSS**, or if it receives an **ACK** from the other side. I can also try switching to some other testing mechanism (say **netperf**) for testing some specific scenarios.

An interesting problem while working with assertions (Linux)

I was playing around with assertions for `tcp_info` on Linux, and I came across an interesting observation for fast-retransmit. The test is pretty simple, but something strange was happening here. The **MSS** is specified to be `1000`, however, on pushing segments of length `1000`, I got these errors -

```
script packet:  0.200699 . 1001:2001(1000) ack 1
actual packet:  0.200599 . 1001:3001(2000) ack 1 win 229
```

This seems strange as the actual packets are of size `2000`.
Earlier when I used to test for **PMTU** discovery, I always used to get errors when pushing data segments in multiples of **MSS** (Linux inclusive, even in Linux I haven't been successful in getting an **ICMP** message notifying of the **PMTU**).
On the other hand, this test makes successful use of `tcpi_retrans` option which fails for FreeBSD, as pointed out in the next section.

Patch for packetdrill

So currently while using the following assertions, I was initially getting some errors -

```
%{
assert tcpi_unacked == 5
assert tcpi_sacked ==  0
assert tcpi_retrans == 1
}%
```

I have made a small attempt at getting my hands dirty with the source code for `code.c` and `tcp.h`, and till now have arrived at the following patch, though it doesn't seem to work at the moment as it can be seen in this log. The values of `tcpi_sacked`, `tcpi_unacked` and `tcpi_sacked` always remain `0`, which is strange. Having a look at the error logs, it seems that only the previously used values of `tcp_info` are available for testing, and indeed on using them as assertions, the code just works fine. I

also intend to add more options for `tcp_info` taking reference from this document and making an effective use of these assertions while testing. Also that since this document points to making use of similar options in **netperf**, we can anytime turn to it for testing these specific `tcp_info` options.

Timeline

Start	End	Task
	15 July	All the remaining scenarios from Linux repository are done once the above mentioned observations gets resolved.
16 July	31 July	Attempt at completing the additional scenarios mentioned in proposal. ➢ **Simultaneous connection close** is done. ➢ **Urgent Pointer** will be done by tomorrow. ➢ Once I figure out a way of buffering data instead of simply **PUSH**'ing, most of the scenarios will be covered.
1 Aug	11 Aug	Attempt at patching packetdrill by adding a new mode of testing in which remote host will not need an instance of packetdrill running.
12 Aug	14 Aug	Code review
15 Aug		End of coding (soft)
23 Aug		End of coding (hard)