1 Conclusion

This chapter summarises the achievements made throughout this project and then focusses on a number of possible extension to further the work presented.

Before this project there was no existing network middleboxes implemented in the Java language capable of fast packet processing. This project has gone a significant distance into achieving this feet through achieving a number of milestones:

- Thorough understanding of memory interactions between the JVM and native memory has been presented. A technique which hasn't been referenced before to access native structs directly from Java has been shown, potentially leading to a number of new applications which can take advantage of this.
- A Java based fast packet processing framework has been built on top of the exiting DPDK framework. By abstracting some of the complications of DPDK away from a user, fast development of applications is possible using Java.
- A few middlebox applications have been implemented using this new Java framework, showing the ease of use compared to the standard native option.
- Analysis of the performance of the new framework was presented comparing to it to C alternatives. Although falling short of the desired outcome, it shows that performance equalling is possible with more work.

In conclusion, a number of techniques have been presented to enable this project to continue and the framework to develop further. Even though the results fell short of the expectations under performance testing, enough promise can be seen to suggest that the expectation can be met in the future.

1.1 Future Work

This project focussed on a lot of background research and technique development to implement a new Java based fast packet processing framework to allow quick application development. This neglected a few key aspects which can be developed further, which are mentioned below, along with suggested improvement to the current framework to increase the performance.

- **Reduce mbuf pointer copying -** Currently when receiving a burst of packets, the underlying native methods copies the mbuf pointer and packet header pointer into a new memory location to be accessed via Java. This copying could be eliminated by directly pointing the mbuf array retrieved from the DPDK methods. This would require dynamically packing an array of mbufs which could prove challenging.
- **Improve socket locality -** A potential bottleneck within the current system is the use of affinity threads which are allocated without concern with processor sockets, memory sockets and NIC sockets. Making sure all of these align will increase efficiency.

- **More generic application starter -** A user has to hard-code a number of key aspects such as queue and port assignment. This could become vastly more user friendly if it could be done dynamically via control parameters, therefore also allowing for dynamic changing of the queue structure at run-time to aid in port which are dropping packets. This would also require the linking of the statistics profiler into the application instead of just simple output.
- **Packet object alternative -** By far the major bottleneck in the current system is the creation of millions of packet objects per second. This was done to keep the object orientated functionality of Java for easier use. However, this is one area which a different approach could rapidly increase the performance.
- **Framework extensions -** The current implemented framework exploits the key areas of DPDK while leaving numerous other features untouched and inaccessible. Extending the framework to cater for these features could allow for more complex applications to be built.