Packet Processing in Java

Ashley Hemingway

February 2015

Todo list

More here about main objective I think - check slides - mention DPDK before and link to background 3
Sort out subsection headings for introduction 4
Introduction 4
Background 8
Figure: Gantt chart would be very useful here 9
Add references 12
Add figure labels 12
Complete front page 12
Do final read through 12
Put through spell checker 12
Other useful applications 12
What font, size and line spacing? 12

Contents

1 Introduction (1 - 3 pages) 3

1.1 Motivation . . . 3

1.2 Objectives . . . 3

1.3 Contributions / Main idea for solving it . . . 4

2 Background (10 - 20 pages) 5

2.1 Network Components . . . 5

2.1.1 Network Packets . . . 5

2.1.2 Network Address Translator (NAT) . . . 6

2.1.3 Firewall . . . 7

2.2 Java . . . 8

2.2.1 JVM Networking . . . 8

2.2.2 JNI . . . 8

2.2.3 ByteBuffer . . . 8

2.3 I/O API's . . . 8

2.3.1 DPDK . . . 8

2.3.2 Netmap . . . 8

2.4 . . . 8

3 Project Plan (1 - 2 pages) 9

3.1 Time Plan . . . 9

3.1.1 Key Milestones . . . 9

3.1.2 Time Estimations . . . 9

3.1.3 Completed Milestones . . . 9

3.2 Fall Back Position . . . 10

3.3 Possible Extensions . . . 10

3.3.1 Off Heap Java Memory . . . 10

3.3.2 Limit Testing . . . 10

4 Evaluation Plan (1 - 2 pages) 11

4.1 Experiments & Outcomes . . . 11

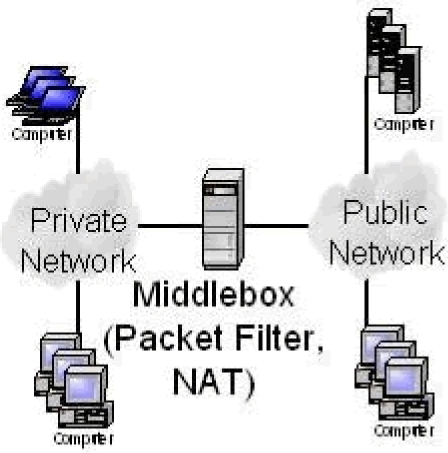
4.2 Functionality . . . 11

4.3 Usability . . . 11

1 Introduction (1 - 3 pages)

1.1 Motivation

As modern computing techniques advance, people are trying to find more generic solutions to problems which have been solved by native applications in the past. A main area of focus has been network middleboxes (network appliance), which are developed to manipulate network packets. Common examples of middleboxes are firewalls, network address translators (NATs) and load balancers, all of which inspect or transform network packets. In recent years, people have been developing a number of programmable middleboxes which allow these generic solutions to be used on a wide scale basis.



As middleboxes are mainly used for networking purposes, they are required to process network packets at line rate (i.e. at speeds which mean there is no backlog of packets to process). This requires the application to retrieve the packet from the network line, inspect and transform the packet in the desired way and then insert the packet back onto the network line, all within a time period sufficient enough to not cause a backlog. High performance implementations of such applications are available, but are written in native languages, predominately in C/C++. However, more and more high performance computing projects are been developed in Java and have succeeded in performing at similar speeds to C/C++ applications.

The main challenge is actually getting the I/O system for the Java application to run at line rate speeds, due to challenges with how the JVM (Java Virtual Machine) interacts with memory and the computer's kernel. Once this challenge has been overcome, there are no reasons why programmable middleboxes written in none native languages such as Java can exist within networking systems.

1.2 Objectives

More here about main objective I think - check slides - mention DPDK before and link to background

With the main aim of implementing a Java application which can process network packets at line speed with similar comparisons to C/C++ applications we defined a number of milestone objectives:

- Understand similar application and API's written in C/C++ which process at line speeds and how the implementations can be exploited for Java applications
- Exploit two Java features which allows for better I/O performance: ByteBuffer's and the Java Native Interface (JNI)
- Implement basic middlebox applications in Java such as a firewall and a NAT
- Compare Java implementations to those which are pure Java and pure C/C++

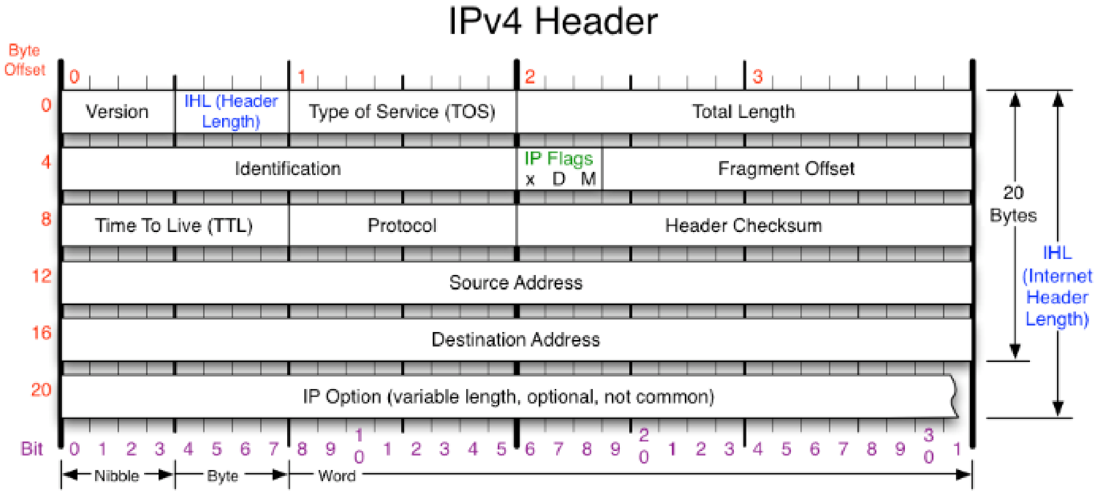
1.3 Contributions / Main idea for solving it

2 Background (10 - 20 pages)

2.1 Network Components

2.1.1 Network Packets

A network packet is responsible for carrying data from a source to a destination. Packets are routed, fragmented and dropped via information stored within the packet's header. Note: in this report packets and datagrams are interchangeable.



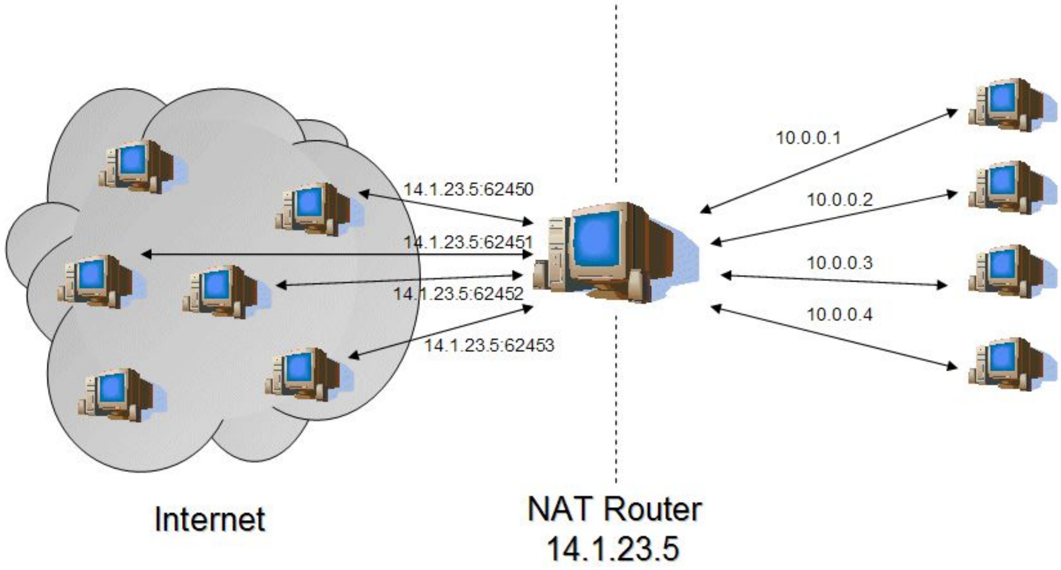
- Version - IP version number (set to 4 for IPv4)
- Internet Header Length (IHL) - Specifies the size of the header since a IPv4 header can be of varying length
- Type of Service (TOS) - As of RFC 2474 redefined to be differentiated services code point (DSCP) which is used by real time data streaming services like voice over IP (VoIP) and explicit congestion notification (ECN) which allows end-to-end notification of network congestion without dropping packets
- Total Length - Defines the entire packet size (header + data) in bytes. Min length is 20 bytes and max length is 65,535 bytes, although datagrams may be fragmented.
- Identification - Used for uniquely identifying the group of fragments of a single IP datagram
- X Flag - Reserved, must be zero
- DF Flag - If set, and fragmentation is required to route the packet, then the packet will be dropped. Usually occurs when packet destination doesn't have enough resources to handle incoming packet.
- MF Flag - If packet isn't fragmented, flag is clear. If packet is fragmented and datagram isn't the last fragment of the packet, the flag is set.
- Fragment Offset - Specifies the offset of a particular fragment relative to the beginning of the original unfragmented IP datagram

- Time To Live (TTL) - Limits the datagrams lifetime specified in seconds. In reality, this is actually the hop count which is decremented each time the datagram is routed. This helps to stop circular routing.
- Protocol - Defines the protocol used the data of the datagram
- Header Checksum - Used for to check for errors in the header. Router calculates checksum and compares to this value, discarding if they don't match.
- Source Address - Sender of the packet
- Destination Address - Receiver of the packet
- Options - specifies a number of options which are applicable for each datagram. As this project doesn't concern these it won't be discussed further.

2.1.2 Network Address Translator (NAT)

As a routing device, a NAT is responsible for remapping an IP address to another by altering the IP datagram packet header. NAT's have become extremely important in modern networking systems due to IPv4 address exhaustion, allowing a single public IP address to map to multiple private IP addresses. This is particularly useful in large corporations where only a limited public network connection is required, meaning that all private IP addresses (usually associated with a single machine) are mapped to the same public IP address. A NAT will make use of multiple connection ports to identify which packets are for which private IP address and then re-assign the packet header so the internal routers can forward the packet correctly. As can be seen by Figure (number) each internal address is mapped to via the port number associated with the external address. NAT's are generally implemented as part of a network firewall as they inspect the datagram packets for malicious data and sources.

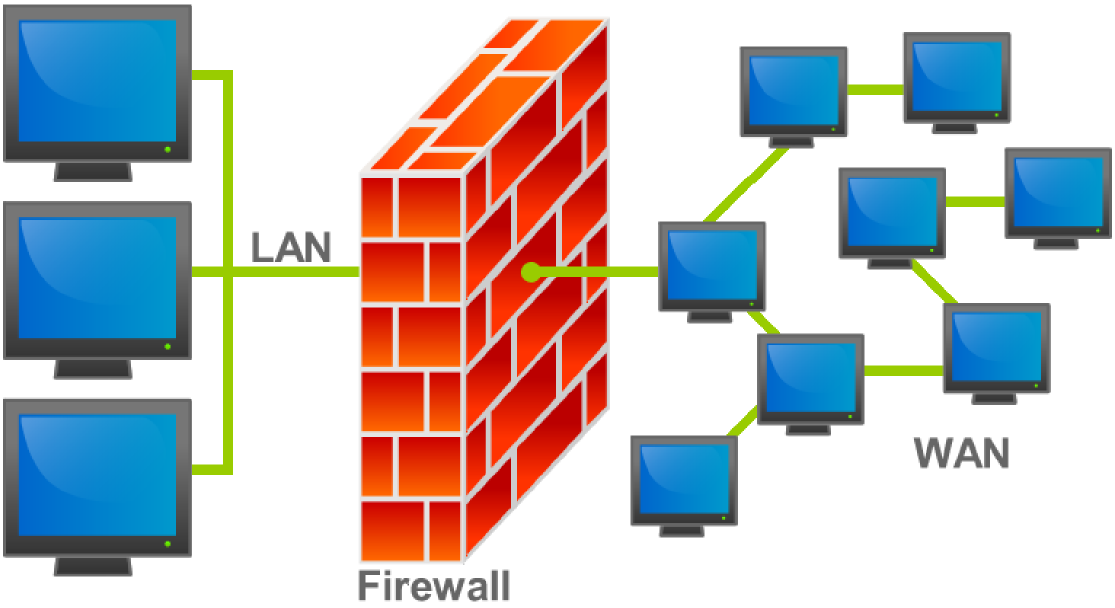
Private IP Address	**Public IP Address**
10.0.0.1	14.1.23.5:62450
10.0.0.2	14.1.23.5:62451
10.0.0.3	14.1.23.5:62452
10.0.0.4	14.1.23.5:62453



2.1.3 Firewall

Firewalls are generally the major applications which sits between the public and private network of a system. They provide packet filtering which controls which packets can enter the private network via establishing a set of rules which packets have to adhere to. Filtering can be based on a number attributes of the packet such as the source and destination IP address and port and the destination service. Firewalls can also offer a number of other useful features such as NAT's or dynamic host configuration protocol (DCHP). As well as providing protection on a network level, application layer firewalls exist which stop certain applications from sending or receiving a packet.

Within this project, the term 'firewall' is used for a network layer firewall which filters packets dependent on the source IP address. Any changes in this will be mentioned within the relevant sections.



2.2 Java

2.2.1 JVM Networking

2.2.2 JNI

2.2.3 ByteBuffer

2.3 I/O API's

2.3.1 DPDK

2.3.2 Netmap

3 Project Plan (1 - 2 pages)

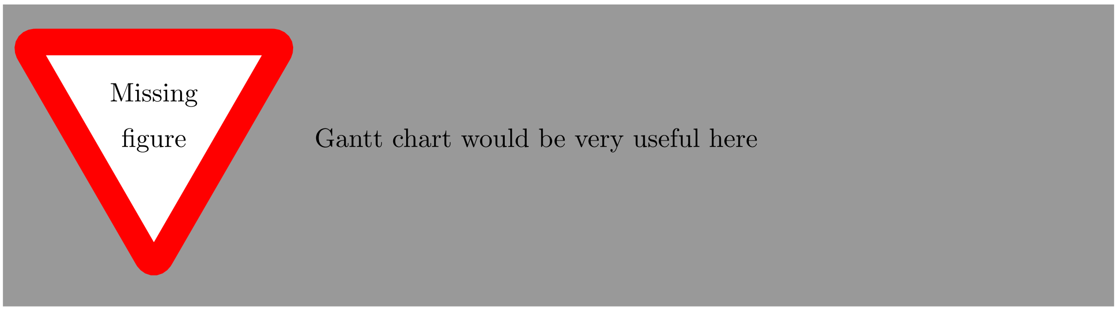
3.1 Time Plan

The given time plan outlined below provides me with a useful indication of how to measure the progress of the project, as well as dividing the whole project into smaller, more manageable sections which can be assessed individually. Some of the milestones have already been completed, while the others are expected to be finished within the given time period.

3.1.1 Key Milestones

- Done - Background reading on DPDK & Netmap
- Done - Understand workings off Java Native Interface (JNI) and Java memory
- Done - Get DPDK (and associated programs) working on local machine (or VM)
- In Progress - Link DPDK library with shared object files for JNI
- Awaiting - Implement basic IP address firewall
- Awaiting - Implement basic NAT
- Awaiting - Find similar implementations of firewall and NAT in C/C++ and Java
- Awaiting - Set up 2 independent machines to use for testing purposes
- Awaiting - Run tests on firewall and NAT to produces performance measurements
- Awaiting - Analyse results

3.1.2 Time Estimations



Gantt chart would be very useful here

3.1.3 Completed Milestones

As indicated, the background reading on the DPDK and Netmap libraries has been completed and the overview of the implementation has been assessed. The pros and cons has been outlined in the background section (give number). From this, I was able to get DPDL fully working on a virtual machine running locally on a personal machine, which allowed for a more in depth analysis of the implementation of DPDK.

I'm currently working on a way to generate shared object files from the C code used the link

DPDK with a Java application. Due to the complex build system which DPDK makes use of, there has been a number of issues which have arisen.

3.2 Fall Back Position

If a number of unforeseen issues arise which impact on the progress of the project, I've outlined a few of the key milestones which can be omitted in order to allow the project to still be evaluated:

- Implement basic NAT
- Find similar implementations of firewall and NAT in C/C++ and Java
- Run tests on firewall and NAT to produces performance measurements

Obviously, the emission of the above milestones will result in a less accurate evaluation of the overall project, but it's better than having nothing to evaluate in the end. As seen above, limiting the testing to one type of middlebox (only the firewall) can still give good results. This therefore reduces the need to look for an implementation for the NAT in Java and C/C++, while meaning less tests will have to be carried out. I feel that the remaining milestones are definitely key to the project in order for an adequate evaluation to be carried out.

3.3 Possible Extensions

If time allows, there are a number of possible extensions which can be undertaken to advance the project further forward and produce more data which can be evaluated later.

3.3.1 Off Heap Java Memory

In order for DPDK to handle packets correctly, it has to insert these packets onto the outgoing 'ring'. To do this, the Java application has to place its serialised objects into memory which DPDK can access. Under the current implementation, the Java application will directly place the packets onto the 'ring' using the direct ByteBuffer allocate feature. The other option is to use off heap java memory which DPDK can access and place onto the 'ring' automatically. This off heap memory is slightly slower than on heap memory but much faster than disk memory, while off heap memory stores object serialised and therefore ready for transmission in packets.

Altering the application to make use of off heap Java memory and then testing the results could potentially offer a new and faster way of high performance packet processing in Java.

3.3.2 Limit Testing

Another interesting extension could be to test the limitations of the NAT and firewall applications implemented within this project, and therefore limitations of the underlaying implementation. In terms of the project, the limitations could be on the max packet throughput which it can handle without having to buffer the incoming packets.

4 Evaluation Plan (1 - 2 pages)

The sections below outline how the project will be evaluated in order to determine whether the initial objectives have been met and whether the final outcome can be deemed a success, even if it provides unexpected results.

4.1 Experiments & Outcomes

The main measure of success will be from the experiments which are to be carried out. Using the high performance Java packet I/O application which is currently been developed as part of this project, it will be compared to similar, already existing applications written in pure Java and in C/C++. This comparison will consist of 2 parts, firstly an application for a NAT and secondly a simple IP address firewall. These similar applications will then be benchmarked against each other and the results analysed.

Benchmarking will consist of setting up 2 independent computers, most likely standalone machines, in order to take advantage of the ability to overwrite the Intel network card drivers as required by the DPDK API. Testing can the be carried out on the firewall and NAT implemented in the 3 different contexts, measuring the latency between the time sent and time received of the network packet. As the testing will be carried out on the same machines, linked to the same network there is unlikely to be much variation in the network latency. This means that the variation in the times will be from the system processing speeds. To account for minor variations in the computing performance of the system, numerous iterations of the same test will be carried out, then taking statistical averages will provide the best final results in order to analyse correctly.

Analysis of the results will be mainly carried out via the use of graphs which allow for easy comparison of results. The expected outcome is that the application developed from within this project is of similar speeds to applications coded directly in C/C++ and much faster than those coded in Java. This is mainly because the kernel will be bypassed, but the use of the Java Native Interface is expected to slow down certain aspects of the application. However, as long as speeds which coincide with those of the line rate are reached, this will be acceptable. There is the possibility that expected speeds are well above those that are actually measured. Although this isn't ideal, the project shouldn't be considered a failure as it will still provide very useful feedback as to where future investigations should be focussed on.

4.2 Functionality

The functionality of the application will be evaluated to make sure it provides those features that are required. This will be done by manually accessing whether the provided application is capable of doing the features that we outlined in the objectives. This can obviously be done by assessing the firewall and NAT applications.

4.3 Usability

Add references

Add figure labels

Complete front page

Do final read through

Put through spell checker

Other useful applications

What font, size and line spacing?

References

http://www.neclab.eu/Projects/midcom.htm
http://windowsitpro.com/networking/what-types-network-address-translation-nat-exist