1 Initial language comparison

Before any implementation or specific design considerations were undertaken, an evaluation of the performance of C, Java and the Java Native Interface (JNI) was carried out. Although data from existing articles and websites could be used for Java and C, there was no existing direct comparisons between them and the JNI, therefore custom tests were carried out.

The JNI is inherently seen as a bottleneck of an application (even after its vast update in Java 7).

article on this

As this application would be forced to use the JNI, numeric values of its performance was helpful to evaluate the bridge in speed required to be overcome.

reasons why JNI is slow

1.1 Benchmarking Algorithm

As discussed previously , there are always advantages and disadvantages of any algorithm used for benchmarking. In order to minimise the disadvantages, an algorithm was used which tried to mimic the procedures which would be used in the real application, just without the complications. Algorithm **??** shows that the program basically creates 100,000 packets individually and populates their fields with random data, which is then processed and returne in the 'result' field. This simulates retrieving low-level packet data, interpreting and acting upon the data, and then setting data within the raw packet.

ref this

Algorithm 1 Language Benchmark Algorithm

```
1:  function Main
2:      for i = 1 to 100000 do
3:          p ← Initialise Packet
4:          PopPacket(p)
5:          ProPacket(p)

6:  function PopPacket(Packet p)                    ▷ Set data in a packet
7:      p.a ← randomInt()
8:      p.b ← randomInt()
9:      p.c ← randomInt()
10:     p.d ← randomInt()
11:     p.e ← randomInt()

12: function ProPacket(Packet p)                    ▷ Process a packet
13:     res ← p.a * p.b * p.c * p.d * p.e
14:     p.result ← res
```

For the JNI version, the same algorithm was used, however, the *PopPacket* method was carried out on the native side to simulate retrieving raw packet data. The *ProPacket* method was executed on the Java side with the result been passed back to the native side to be entered back into the packet structure.

Timing within the algorithm for all variations was carried out between each iteration. This firstly eliminated any initial start-up time associated with the application which is common with the JVM. Secondly, any calls for time stamps to the system would be miminised as 100,000 iterations would occur in-between them.

1.2 Results

Each language had the algorithm run 1,000 times in order to minimise any variations due to external factors. Figures show that C was considerably quicker than Java, while Java using the JNI was extremely slow.

ref this

expand on this

1.3 Further Investigation

Due to the very poor performance of the JNI compared to other languages, further investigations were carried out to find more specific results surrounding the JNI.

Is this relevant