# On the Performance of String Search Algorithms for Deep Packet Inspection - Followup

Kieran Hunt

March 7, 2016

## 1 Processing Speed vs Input Length

In a previous paper by Hunt (2016), a selection of DNS traffic was used as input for the comparison between algorithm processing time and input length. The average length of a packet from that dataset was near 110 bytes. While that length made sense for dns traffic - notoriously low bandwidth usage - it did mean that a true reflection of algorithmic performance could not be obtained for longer length packets. A dataset of randomly generated DNS traffic was produced with a maximum length of 1500 bytes (the maximum transmission unit for ethernet) and a mean length of 770 bytes. The new set of packets is hereafter referred to as *Dataset C*. More information can be found in Table 1.

| Min | First Quartile | Median | Mean | Third Quartile | Max |
|---|---|---|---|---|---|
| 44 | 407 | 763.5 | 770.9 | 1136 | 1500 |

Table 1: A summary of the input lengths in *Dataset C*

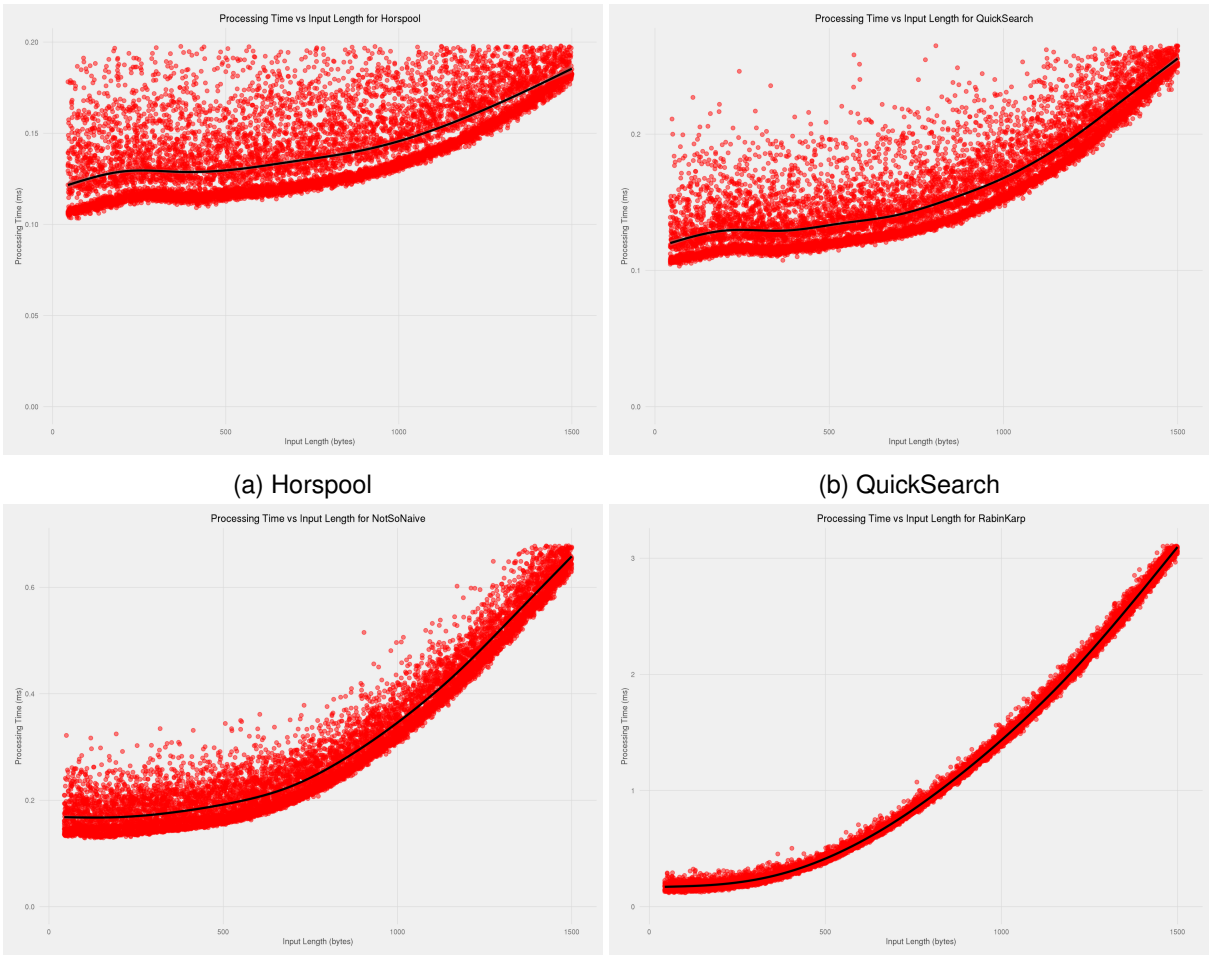
Each of the four algorithms was tested for the same rules as in Hunt (2016) and with *Dataset C* as input. Figure 1 shows a comparison between each of the algorithms.

From Figure 1, it is clear that the processing times for Horspool and QuickSearch (Subfigures 1a and 1b respectively) do not rise as quickly as those of NotSoNaive and RabinKarp (Subfigures 1c and 1d respectively) for inputs of greater length. Both NotSoNaive and RabinKarp show an exponential rise in processing times.

Horspool and QuickSearch also show initial humps in processing speed around the 250 byte mark. This can be attributed to the overheard associated with each packet. For packets of more than 250 bytes, the processing speed relies more on the length of the packet then the initial - per-packet - overhead. This initial hump is not evident on either NotSoNaive nor RabinKarp; the reason for this is twofold. First, the speed of these algorithms is more affected by the input length, because of this an initial hump would be surpassed by the time it takes to process each packet. Second, the vertical scale of these two graphs is much larger than that Horspool or QuickSearch. Owing to that a small hump earlier on is less visible on a larger scale.

## 2 Processing Speed vs Number of Matches

For Section 1, *Dataset C* was created. It was designed in such a way that the number of matches in the input would not affect the processing speed. The processing speed was thus only affected by the length of the input and the algorithm used. In order to compare the processing speed against different




(a) Horspool

(b) QuickSearch

(c) NotSoNaive

(d) RabinKarp

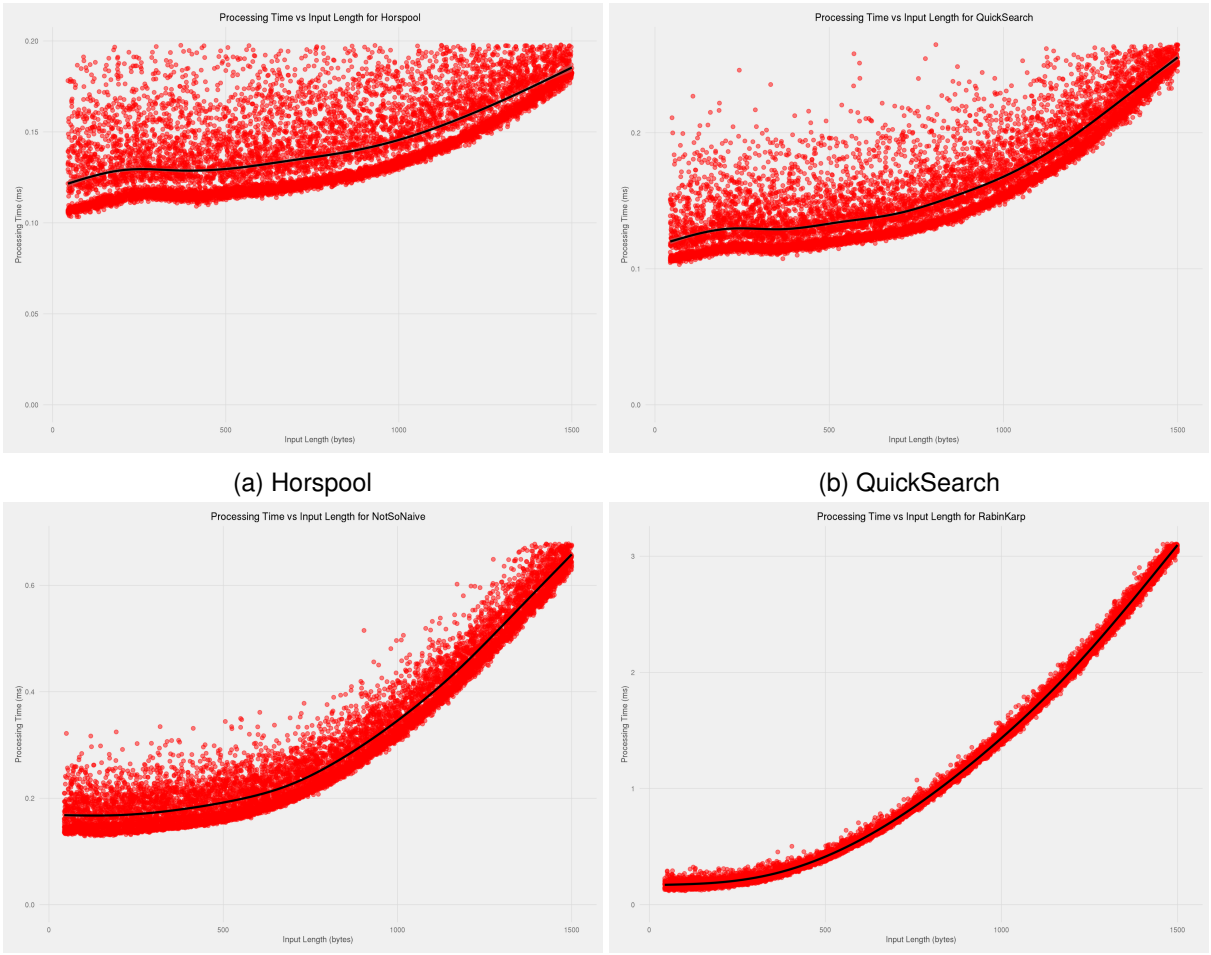
Figure 1: Mean processing times versus input length for the Horspool, QuickSearch, NotSoNaive, and RabinKarp algorithms for *Dataset C*.

numbers of matches, another Dataset needed to be created. *Dataset D* contains 10000 DNS packets. Each 1500 bytes long. The payload in *Dataset D* contains a mixture of randomized text and repeated strings of the rules. Each packet therefore contains between 0 and 1500 bytes of characters which are known to match with the rules used. This input data isolates the number of matches from the length of the input.

# 3 Parallelism

# References

K. R. Hunt. On the performance of string search algorithms for deep packet inspection. *who knows*, 2016.




(a) Horspool

(b) QuickSearch

(c) NotSoNaive

(d) RabinKarp

Figure 2: Mean processing times for each algorithm vs number of matches. *Dataset D*.