A Comparison of String Search Algorithms for Deep Packet Inspection

Kieran Hunt

September 9, 2015

Abstract

Contemporary deep packet inspection systems often rely on custom hardware or entrenched ideas about the string search algorithms used. These algorithms have mathematically provable time or space complexities however not much is empirically known about their performance on real-world packet datasets. We felt that some string search algorithms could produce results that differed from their theoretical performance within the context of packet inspection. Furthermore, we sought to show that even algorithms with similar theoretical performances could produce differing practical results. Our approach was to reimplement a variety of the established string search algorithms and run them through a diverse set of tests with both real-world and constructed datasets. Our tests found that the Bloom filter was the fastest overall, Rabin-Karp was the most memory efficient and that the Nave algorithm was the slowest. Furthermore we found that, although the Bloom and Cuckoo filters have the same theoretical time complexity, the cuckoo filter was almost twice as slow as its counterpart. These findings help to show which algorithms perform best in practice and can help future algorithm designers to improve on the current approaches. In practice we show which algorithms a designer of a deep packet inspection system should consider implementing based on our findings.

1 Introduction

Deep packet inspection (DPI) forms part of the packet filtering that takes place on a computer network. For the general routing of a packet it is only necessary to look as far as the packet's headers. DPI takes the inspection further by closely examining the packet's payload data; its purpose is to detect data in a packet that is interesting to the network administrator. Generally, data that is interesting to the administrator is viruses, spam, incomplete or corrupted protocols, and obvious intrusions.

Deep packet inspection is very prevalent today. From Internet Service Providers (ISP) to Corporate Networks. Packets are inspected for a plethora of reasons.

Today, many approaches exist to perform deep packet inspection. The approaches can be split into two categories: hardware and software. A hardware implementation generally requires the use of expensive and custom systems such as Field Programmable Gate Arrays (FPGAs). Hardware is generally faster than software (should cite) but has slower compile times and does not scale as well. Software, like Bro and Snort, is generally run on Commercial off-the-shelf (COTS) machines or on virtualised infrastructure. Because it is not limited by the underlying hardware, the software implementations are able to scale both up and down to match the needs of the network.

Bro[1] and Snort[2] are two platforms designed for network analysis.

Deep packet inspection is traditionally done within an intrusion detection system (IDS) and as such needs to be fast enough that it can match the speed of the network. Failing to at least match the network speed will results in a slowdown of traffic flowing into the network. Failing to meet the network speed could also mean that uninspected packets will be queued, if the DPI system continues to not be able to keep up with the incoming packets the system will usually start dropping packets. This result is certainly undesired as it negatively impacts the quality of services experienced by users on the network.

Talk about packet inspection with respect to string search. Compare searching through a book to searching through a packet.

2 Algorithms

A bit about each of the algorithms. Their time and space complexities. Talk about each of their strengths a weaknesses. Try to guess which might be good at DPI.

Talk about why we chose those algorithms. Talk about any that we didn't choose.

2.1 Naïve

In order to objectively compare algorithms we implemented a control algorithm. Named Naïve, this[3] algorithm has a $O(n^2)$ time efficiency.

2.2 Aho-Corasick

[Aho and Corasick, 1975]

2.3 Bitap

[Baeza-Yates and Gonnet, 1992]

[1] `https://www.bro.org/`

[2] `https://www.snort.org/`

[3] `https://github.com/KieranHunt/dpi_algorithms/blob/master/src/dpi_interface/naive.rs`

2.4 Bloom

[Bloom, 1970]

2.5 Boyer-Moore

[Boyer and Moore, 1977]

2.6 Cuckoo

[Fan et al., 2014]

2.7 Horspool

[Horspool, 1980]

2.8 Rabin-Karp

[Karp and Rabin, 1987]

2.9 Knuth-Morris-Pratt

[Knuth et al., 1977]

3 Testing Environment

Give a run-down of the system used.

The Rust language was used to implement all of the search algorithms. Rust was selected because it is fast and safe. Both qualities are important in a real-time packet inspection scenarios. For interacting with both packet captures and live capture handles we used the Rust pcap[4] library. This library provides a wrapper around libpcap[5]. Through pcap, we are able to easily retrieve the packet data which can in turn be run through the various algorithms (See part 2).

Each algorithm was implemented in Rust. All but the Bloom[6] [Bloom, 1970] and Cuckoo[7] [Fan et al., 2014] filters were were implemented by the authors.

Talk about the pcap format.

The data used in the tests was real-world packet data collected from Rhodes University ('s proxies? firewalls?). The libpcap library, in conjunction with tcpdump to collect and save the packet information. Rhodes University actively collects and records the packets into and out of its network. The data collected is grouped by month. Our dataset spanned from September 2013 to July 2015.

[4] `https://github.com/ebfull/pcap`
[5] `http://www.tcpdump.org/`
[6] `https://github.com/nicklan/bloom-rs`
[7] `https://github.com/seiflotfy/rust-cuckoofilter`

Table 1: Captured Packet Data Sizes

Year	Month	#Packets	Size on Disk (GB)
2013	September		5
	October		23
	November		20
	December		22
2014	January		30
	Febraury		30
	March		51
	April		30
	May		0.34
	June		17
	July		23
	August		15
	September		24
	October		28
	November		30
	December		12
2015	January		22
	Febraury		25
	March		29
	April		13
	May		31
	June		27
	July		0.448

Because of the long period of time for which the packets were captured, the amassed capture files are very large. According to Table 1, the same captures ranged from a few megabytes to over fifty gigabytes in size.

Each algorithm was run over each month's packets for the same search string.

4 Results

Have some tables of general speeds across the various algorithms/datasets. Same for memory.

5 Analysis

Compare results with each other.

Compare results against the theoretical outcomes.

6 Conclusion

Talk about going forward. Future work.

References

[Aho and Corasick, 1975] Aho, A. and Corasick, M. (1975). Efficient string matching: An aid to bibliographic search. *Communications of the ACM*.

[Baeza-Yates and Gonnet, 1992] Baeza-Yates, R. and Gonnet, G. (1992). A new approach to text searching. *Communications of the ACM*.

[Bloom, 1970] Bloom, B. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*.

[Boyer and Moore, 1977] Boyer, R. and Moore, J. (1977). A fast string searching algorithm. *Communications of the ACM*.

[Fan et al., 2014] Fan, B., Andersen, D., Kaminsky, M., and Mitzenmacher, M. (2014). Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*.

[Horspool, 1980] Horspool, N. (1980). Practical fast searching in strings. *Software: Practice and Experience*.

[Karp and Rabin, 1987] Karp, R. and Rabin, M. (1987). Efficient randomized pattern-matching algorithms. *IBM Journal of Research and Development*.

[Knuth et al., 1977] Knuth, D., Morris, J., and Pratt, V. (1977). Fast pattern matching in strings. *SIAM Journal on Computing*.