Accelerating Regular Expression Processing Using Hardware DFA Engines

Abstract - The processing of regular expressions (regexes) constitutes a powerful and common detection mechanism in most of network intrusion detection system (IDS). Yet because regular expressions come with significant overheads in terms of both memory and CPU cycles, security network managers have traditionally been conservative in using them.

In this abstract we present a design, implementation and a preliminary evaluation of a framework to drastically reduce the performance cost of regular expressions. Our objective is to make the processing of regular expressions a commodity to effectively remove its performance cost constraint from the network analytics. Using hardware-based deterministic finite automatons (DFA), we use our framework to accelerate computational-expensive regular expressions found in Bro, the popular language-rich general-purpose IDS [1]. Our evaluation shows that regular expression matching in Bro can be accelerated by a factor of up to 37 times, resulting in overall system throughput improvements of up to 15 times depending on the amount of regular expressions processing found in the input specifications.

Keywords: Intrusion Detection System, regular expression, high performance, DFA.

1 The Regex Cache

Illustration 1 presents the design of our regex acceleration engine. At the core of our design, there is a caching facility used as a communication mechanism between the hardware and the application. Instead of passing the results computed by the hardware directly to the application, a cache acts as a rendezvous used for both hardware and software to store and retrieve regex processing results. Upon arrival, a packet is first handled by a dispatcher. Based on its IP tuple (source and destination IP addresses and port numbers), worker 1 performs a look up operation to a table of flow states (TFS) to retrieve the flow context associated with the packet. If the look-up operation returns no result, then a new flow state is created, initialized, and stored in the TFS. Worker 1 queues the packet and the flow state into a hardware queue; the hardware DFA engines process each element of the queue and scan the packets in search for any of the regular expressions loaded in the DFA. The results—expressed in terms of packet offsets where the regexes are found—are stored into the cache. The hardware also queues the new flow state into a queue which is then processed by worker 2. Worker 2 is responsible to update the new state of the flow in the TFS.

When the packet is processed by the network analyzer, the handling of regular expressions is done via a look-up operation to the cache. If a result is obtained, then the network analyzer uses the returned offsets to identify the location of the regexes within the packet; otherwise, the cache yields a miss and the network analyzer falls back to the software DFA engines.

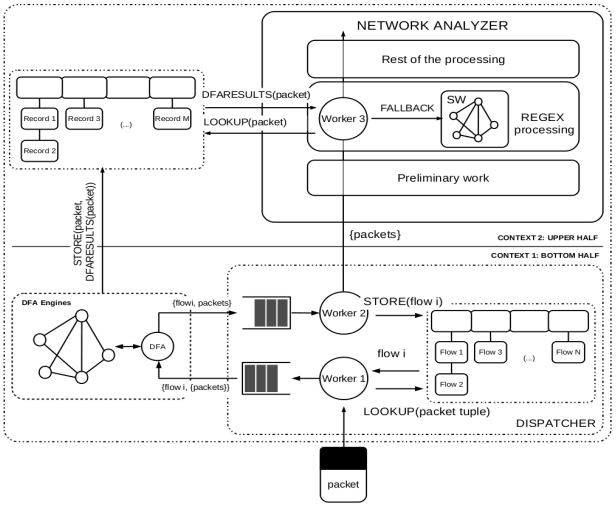

Illustration 1: Work flow design of the regex acceleration engine

2 Performance Evaluation

We implemented the regex acceleration engine on the Cavium NICPro2 network processor [4]. The NICPro2 comes with 16 MIPS64 cores at 600MHz running Linux and 16 DFA independent hardware thread engines. To test the performance under a variety of input scenarios (single versus multiple connection and small versus large packet size), we use two traces, one made of numerous short-lived HTTP connections (browse.pcap) and another one consisting of a single long-lived HTTP connection (single.pcap). For the latter, we also generate multiple versions of it by changing the MTU settings of the network card to produce a variety of traces with different average packet sizes:

Trace	# of connec-tions	# of packets	# of bytes	Average packet size	Type of traffic
browse.pcap	89	8208	5MB	722	Web browsing of sites
single.pcap	1	43377	43MB	Variable, various traces	Large single file download

2.1 Performance Gains: Processing of Regular Expressions

We first consider the cost of processing regular expressions in isolation, without considering the rest of the processing made by Bro. Illustration 7 presents results for various scenarios by dissecting the cost of hardware processing into its two components: the cost of submitting

a buffer to the hardware DFA engine and the cost of fetching a result from the cache.

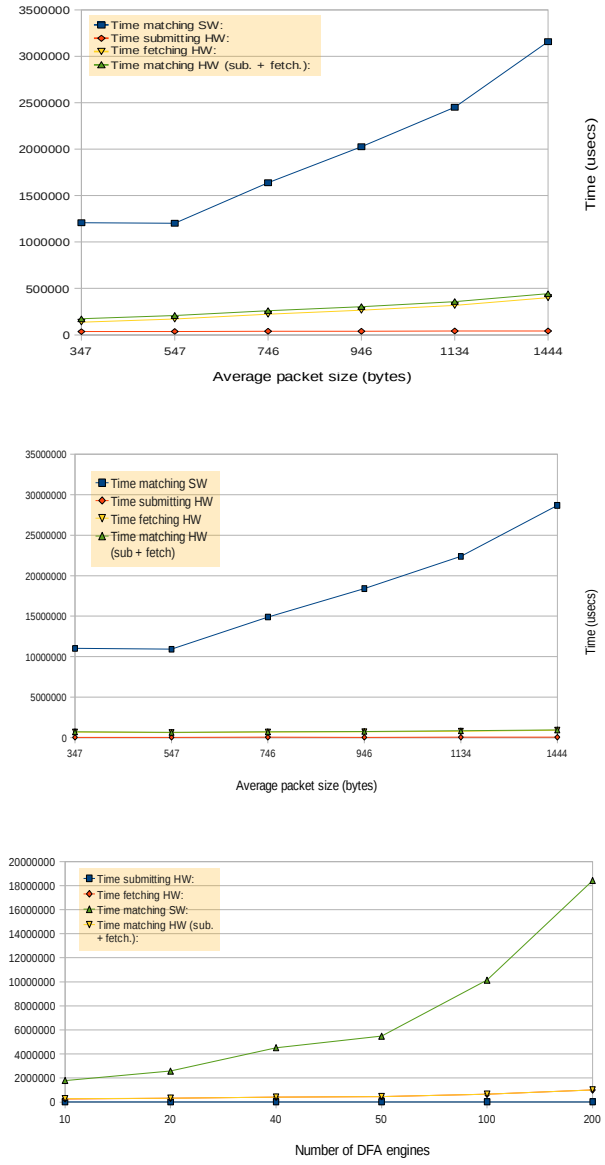

Illustration 2: cost of processing regular expressions dissected by time spent submitting a request and fetching a response for (up) single.pcap with 10 software DFA engines, (center) single.pcap with 100 software DFA engines and (bottom) browse.pcap with average packet size of 722 bytes; all input specifications have 1000 regular expressions;

When stretching the input scenarios, we find that for 200 software DFA engines and with the file single.pcap at maximum average packet size (1444 bytes), performance gains of 37 times were achieved.

2.2 Performance Gains: System-Wide

We evaluate now the Bro system-wide gains obtained using the regex acceleration engine. Illustration 3 provides performance gains for the two considered scenarios, short-lived multiple connections and long-lived single connection. Results are qualitatively similar, with the latter showing higher absolute performance gains. This can be attributed to the fact that for long-lived connections, Bro needs to spend less time setting up connections, which is a type of work that involves no regex processing.

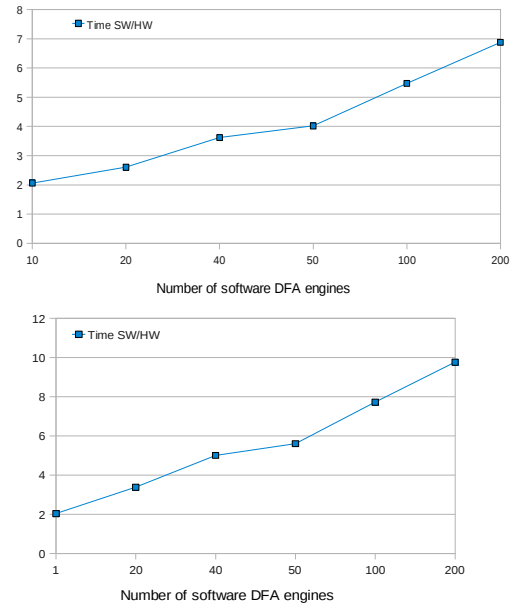

Illustration 3: overall system performance gains for (up) browse.pcap (average packet size of 722 bytes) and (bottom) single.pcap (with average packet size of 746 bytes). All input specifications have 1000 regular expressions

Illustration 4 presents aggregated results of performance gains as a function of average packet size and number of software DFA engines. Performance gains of up to 15 times are achieved for large average packet sizes (1444 bytes) and large number of software DFA engines (200).

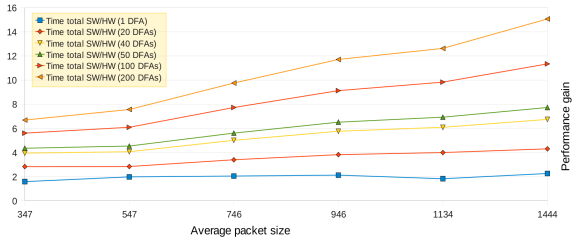

Illustration 4: Overall performance gains for single.pcap, for various packet sizes and various numbers of software DFA engines

References

[1] V. Paxson, “Bro: A System for Detecting Network Intruders in Real-Time,” Computer Networks, 31(23-24), pp. 2435-2463, December 1999.

[2] S. Dharmapurikar and J. W. Lockwood, “Fast and scalable pattern matching for network intrusion detection systems,” IEEE Journal on Selected Areas in Comm., 24(10):1781–1792, 2006.

[3] H. Dreger, A. Feldmann, M. Mai, V. Paxson, R. Sommer, “Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection,” USENIX Security Symposium, June 2006.

[4] “OCTEON XL NICPro2 Accelerator Board Family,” Cavium Networks, Product Specification.