PRODUCT HOW-TO: Use multicore flow processing to boost network router/security appliance throughput

Robert Truesdell

4/1/2010 11:05 PM EDT

Adding the Netronome NFP-3240 network flow processor to a standard x86 platform allows network router/switch and security appliances to perform regular expression matching at 10 to 40 Gbps, without the assistance of specialized hardware. In many network and security appliances, the need for regular expression matching (RegEx) is an essential requirement, specifically for deep packet inspection (DPI) applications such as intrusion detection and prevention systems (IDS/IPS), content firewalls, virus scanning, data loss prevention (DLP), and lawful intercept applications.

Many appliance manufacturers for these network security applications are frequently confronted with the decision to integrate regular expression capability via specialized hardware, or leverage multicore x86 processors and use software packages and libraries, such as the Perl Compatible Regular Expression (PCRE) library.

In either case, accelerated network processing is required to reach the 10 to 40 Gbps data rate that many network applications demand. In most instances, specifically with standard Linux applications, developers prefer the use of software packages such as PCRE for two primary reasons.

First it is a widely adopted package across the open source community and security applications, and second is it is a free technology, unlike that of specialized regular expression hardware. The challenge with a software-implemented solution is meeting performance requirements.

Appliance manufacturers are developing network and security appliances requiring 5 to 10 Gbps of security processing today, with rates rapidly moving to 40 and 100 Gbps.

These requirements would typically convince appliance manufacturers of the need for specialized RegEx hardware; however, a new trend is evolving throughout the network appliance industry, mostly due to significant advances in standard Intel x86 processors and Netronome's network flow processors.

Many current network appliance designs are built on single or dual-socket Xeon quad-core processors operating at up to 3.0Ghz frequencies. The x86 instruction set is ideal for complex data processing such as regular expression matching, and these designs are supporting up to 1 to 3 Gbps of regular expression matching on network traffic without the assistance of network flow processors.

Adding regular expression hardware to the design via a PCIe card will not increase the network throughput as these designs are network I/O constrained. With the recent release of the Westmere Xeon CPUs, which supports six-dual threaded CPU cores, processing capacities are tripled with costs staying relatively low although network I/O remains a problem.

The new challenge which emerges from this increase in processing capacity is pushing the network I/O capability to meet the full processing potential of these x86 processors.

Heterogeneous network processing

To improve the network I/O performance in embedded x86 designs to 10 to 40 Gbps, a heterogeneous processing architecture is required. In this model, a network flow processor, such as the NFP-3240, front ends the x86 CPUs (**Figure 1 below**).

This network co-processor relieves the x86 of computationally burdensome tasks that reduce its overall performance such as complex flow processing and transitioning packets across CPU and

memory locations.

Within this architecture, the network flow processor is also responsible for applying hardware driven policies to flows which include x86 load balancing, flow redirection to specific matching engines, filtering, or fast-path/cut through, among other actions.

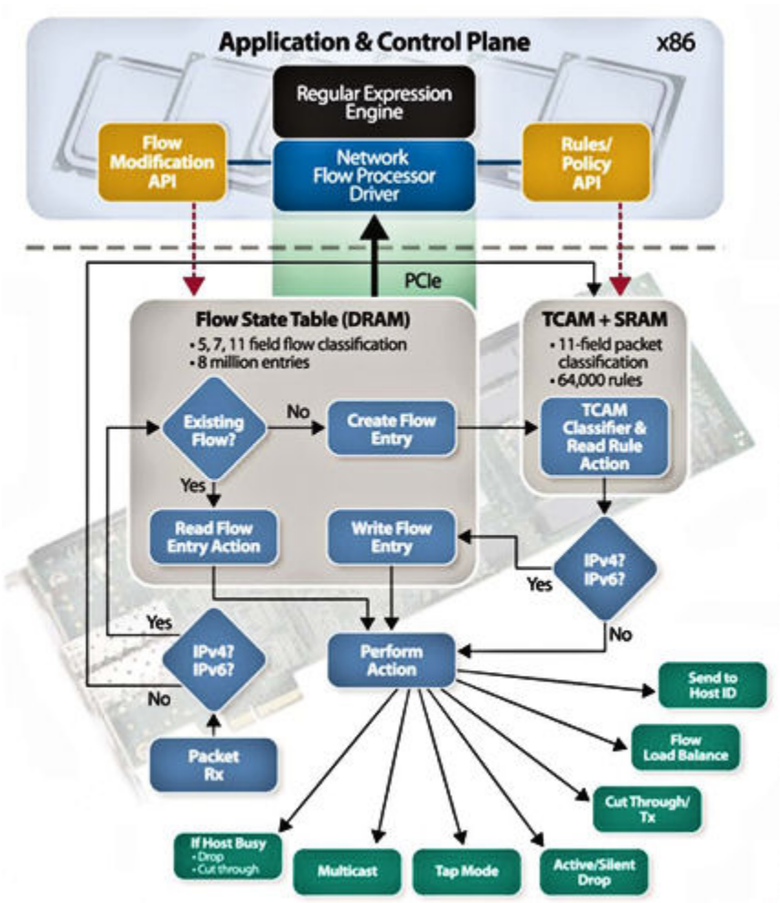

Figure 1. In a heterogeneous network processing architecture, the network co-processor relieves the x86 of computationally burdensome tasks that reduce its overall performance such as complex flow processing and transitioning packets across CPU and memory locations.

Introducing the Heterogeneous Multicore Architecture

Meeting the challenges of complex data processing, such as regular expression matching on packet payloads at 10 Gbps and beyond requires the assistance of a network co-processor.

Adding a regular expression processor may relieve the x86 CPUs from the task of complex matching, but it does not help with network I/O rates and therefore does not increase the appliance processing capacity past 1 to 3 Gbps of matching.

Adding a network flow processor can relieve the x86 CPUs of a number of networking tasks, thus creating more CPU cycles for regular expression processing while simultaneously increasing network throughput. The x86 CPUs and network flow processor(s) are coupled over a high-speed, virtualized PCIe gen2 interface supporting a raw bandwidth of 40 Gbps.

Front ending the x86 CPUs with network flow processors via PCIe cards allows for flow preprocessing to occur before matching or DPI is applied to each flow or packet. Flow preprocessing, in this context, applies to the identification and classification of a flow based on the 5, 7 or 11-tuple of the received packet and applying the programmable action to the flow.

Flows can be delivered to the host by load balancing across many parallel application instances or they can be placed into a specific application instance or CPU core. Other actions include silently dropping, actively rejecting or passing the flow via cut-through.

As such, the policies or actions associated with each flow can be dynamically changed or updated at the discretion of the programmer (i.e. the results of regular expression matching, DPI, signature matching or any other analysis).

This type of flexibility is essential in situations where, for example, a specific portion of a flow such as the first 500 bytes, are of interest for inspection.

After regular expression matching is complete on the data of interest all subsequent packets belonging to the flow can be filtered or cut-through at the network flow processor layer which conserves valuable PCIe bandwidth and reduces x86 CPU cycles.

Other flow modification use cases include event driven updates where the programmer chooses to modify the action associated with the flow based on the match of an expression.

Upon matching, the application could update the hardware-driven action associated with that flow to now redirect to a separate regular expression engine or traffic recording application.

For inline applications, a match can be an indication of malicious traffic traversing the device. In this instance, the hardware-driven policy associated with the malicious flow could be updated to drop all packets belonging to that flow.

An additional way to improve efficiency is to classify the flow as a certain application and then redirect the flow to a signature database populated with patterns specific to application identified within the flow.

This approach allows the system to run many engines, each with a unique and specialized signature database versus running many engines with large, identical databases. **Figure 2 below** illustrates the architecture driving the flow modification process.

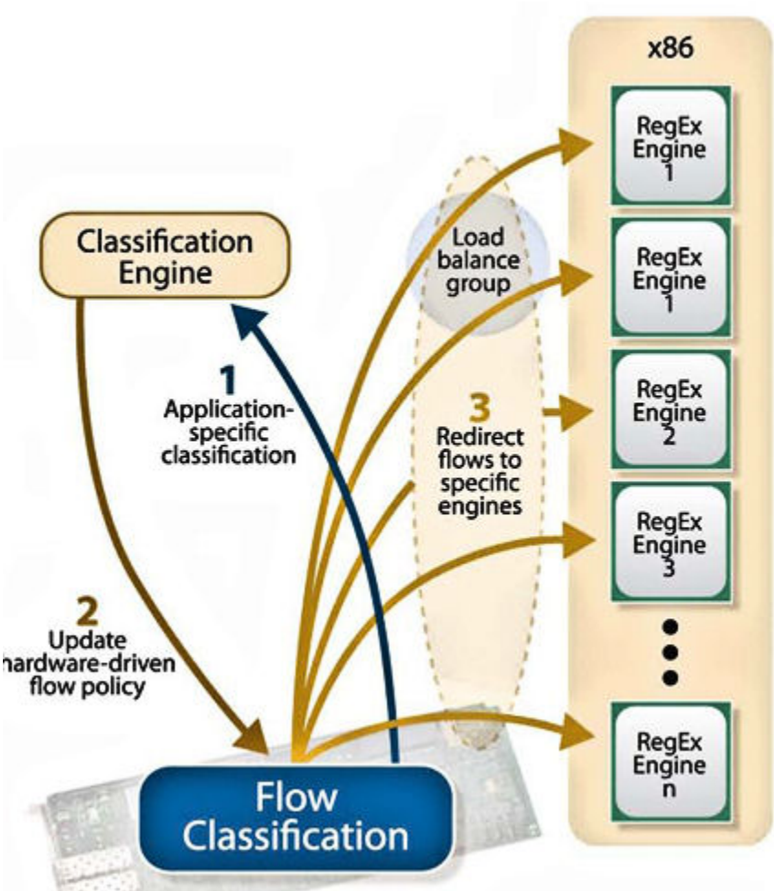

Figure 2. How a network flow processor architecture drives the flow modification process.

For example, suppose a DPI or classification engine were to identify webmail traffic. The engine could then redirect the flow to another engine looking specifically for document watermarks or other signs of sensitive data passing over unauthorized channels.

For data delivery, coupling the front facing network flow processor (**Figure 3, below**) with a zero-copy driver allows for high speed data delivery at 10 Gbps line rates directly into the application memory space over PCIe.

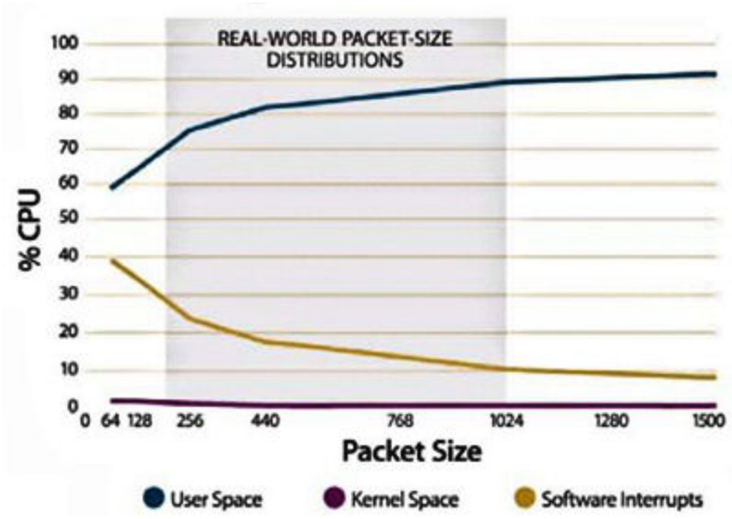

Figure 3. CPU utilization: packet capture using network flow processors.

This not only improves the data delivery rate compared to the standard architecture (**Figure 4, below**), but also drastically improves latency through the system which is crucial for inline applications.

An added benefit of utilizing a zero-copy driver is the reduced number of CPU cycles required to get packets in and out of the application instances. The immediate benefit of the heterogeneous architecture is a 10x increase in network I/O performance for x86 applications.

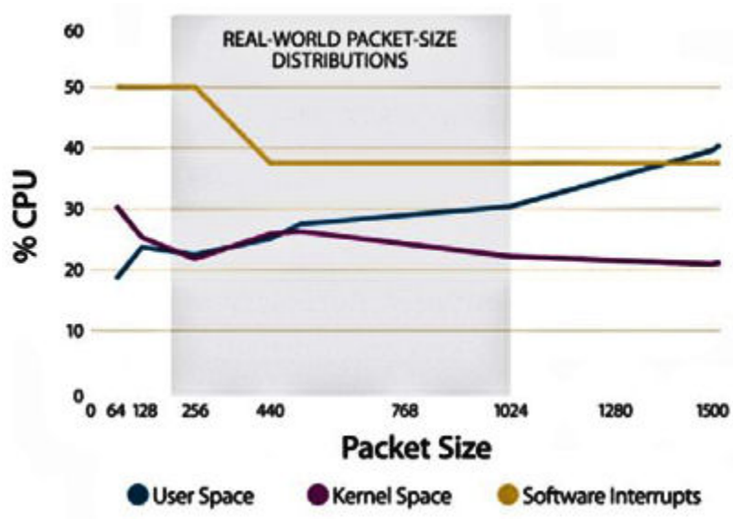

Figure 4. CPU utilization: packet capture using standard hardware.

The indirect benefits are two-fold in that a greater amount of traffic can now be received by the application while increasing the CPU cycles available to process the traffic.

Case Study: Using the NFP-3240 with a Heterogeneous Multicore

There are many networking and security applications requiring RegEx, perhaps the most widely known application is Snort. Snort is an open source IDS/IPS which is built on the PCRE library.

At its core, Snort is a pattern matching engine. What differentiates Snort from other pattern matching applications is its signature database. The signatures used in Snort are specific to identifying and alerting on security threats observed in network traffic.

Other differentiating features within Snort include the ability to reassemble full TCP sessions, protocol standardization via preprocessors and many other features.

As opposed to writing a custom PCRE application which provides matching capability and performance sampling, this case study is based on using Snort which provides both of these capabilities. Snort is delivered with a standard configuration which includes support for the Stream preprocessor, HTTP preprocessor and many others which are not required for implementing a simple pattern matching application.

These preprocessors can simply be removed from the configuration and are not required for basic RegEx. The next component to examine is the Snort rule set. Again, the rules packaged with Snort are specific to identifying security threats, but they can be modified. Below is an example of a Snort rule:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"WEB-MISC
/etc/passwd"; content:"/etc/passwd"; nocase; metadata:service http;
classtype:attempted-recon; sid:1122; rev:6;)
```

Simplified, this rule is looking into HTTP traffic for access to the /etc/passwd file. Upon Snort detecting this, an alert is generated and the traffic is optionally dropped. The rule could very easily be rewritten as follows:

```
alert tcp any any -> any any (msg:"Access to /etc/passwd"; content:"/etc/passwd";
sid:1; rev:1;)
```

The rewritten rule has been simplified to match against the content "/etc/passwd" for sessions using any TCP port and any IP address. While this can be considered an inefficient way to rewrite the rule, it clearly illustrates how Snort can be used to match against any packet content.

In addition to specifying custom expressions or patterns within Snort rules, there is also the ability to specify IP addresses or subnets, IP protocol and TCP/ UDP ports as criteria for matching.

The above example uses the content keyword in the rule which performs a simple string match; however, complex regular expressions can also be used with the "pcre" keyword. The following example is a simple demonstration of the usage of PCRE to identify a social security number of the form XXX-XX-XXXX:

```
alert tcp any any -> any any (msg:"Leaking a Social Security number"; pcre:"/[0-9]
{3}-[0-9]{2}-[0-9]{4}/"; sid:1; rev:1; )
```

At this point, a system user/manager could populate the rules configuration with an unlimited number of expressions of interest. As described above, customers can run multiple instances of the same rules database or run instances with unique rules specialized for a specific type of traffic.

In either case, the network flow processor is capable of load balancing flows across each instance or process each flow to determine which instance the flow should arrive at.

Throughout the performance assessment, eight instances of Snort were initiated with an identical pattern database consisting of 995 random signatures plus five signatures which were created with the intention of being matched with test traffic.

Each instance of Snort used an identical configuration which includes only performance profiling and the signature database. The test platform includes two quad core Xeon processors operating at 2.83 GHz. The platform also includes two network facing PCIe acceleration cards with network flow processors.

Within this system, the network flow processors are responsible for packet acquisition off the wire, flow preprocessing, load balancing, and zero-copy delivery, while the x86 CPUs are responsible for regular expression matching and control plane tasks.

The performance objective for the system is to support 150,000 successful matches per second while at 100 percent traffic capacity. A traffic profile made up of TCP and UDP sessions with average packet sizes of 440 Bytes and a random IP addressing scheme was used.

To begin pushing the system to near full processing capacity, a sustained 6.6 Gbps of the test traffic was sent into the system. The performance profiler for the application reported 824 Mbps per application instance.

Once the system was under load the same traffic profile was repeated at a rate of 150,000 packets per second, however the payloads were modified to include data which would successfully match against 5 of the 1000 signatures in the database.

The distribution of matching versus non-matching traffic was selected to prove effectiveness and accuracy of the whole pattern matching system while under high network throughputs.

As shown in **Figure 5 below** adding 150,000 packets per second increases the network throughput to 7.1 Gbps, which is evenly load balanced across eight instances of the regular expression engines.

This yields a throughput measurement of roughly 908 Mbps and 18,750 successful matches per second for each engine instance. This performance is based on traffic being processed by the regular expression engines and does not account for traffic that could be filtered or cut-through on the network flow processor which would increase total system performance further.

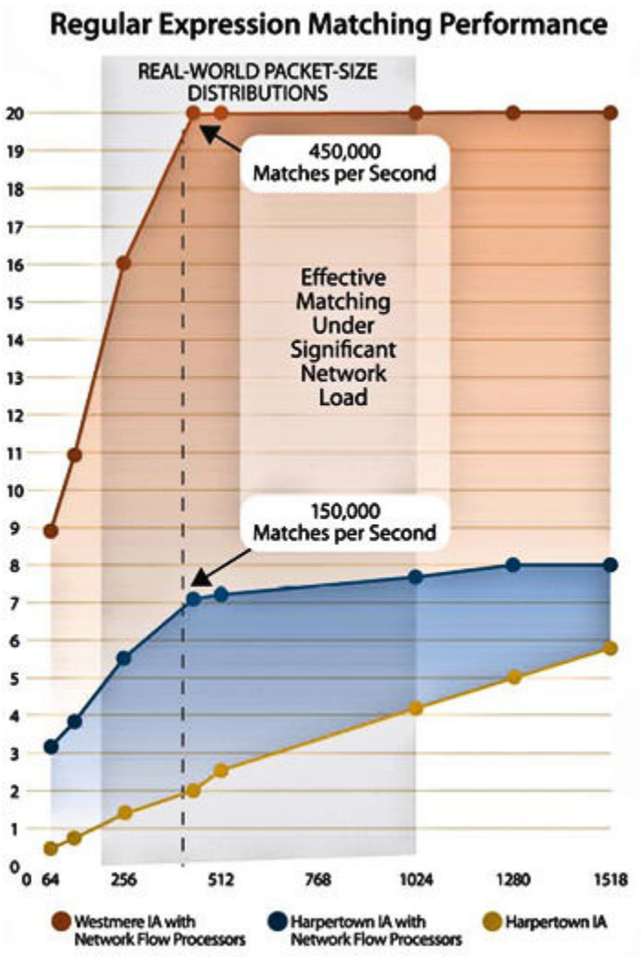

Figure 5. Measured pattern matching throughput (Gbps)

When compared to a standard dual quad core x86 system without network flow acceleration, the accelerated platform shows a 3.5x improvement. As previously stated, the bottleneck in the standard architecture is the inefficiencies in handling high-packet and throughput rates. Adding RegEx hardware would not improve the situation as the RegEx capability of the x86 CPU is not the limiting component.

With the processing improvements that come with the Westmere CPUs and the advancements in network flow processors, the above metrics are expected to triple - based on preliminary testing. The 3x improvement increases the pattern matching throughput capability beyond 20 Gbps, with total system flow processing capability of up to 40 Gbps when including filtered and cut-through traffic.

The Westmere CPUs would allow a customer to run up to 24 RegEx engines, assuming one engine per core. The network flow processor would then load balance flows across the 24 engines or optionally place flows into specific engines.

Conclusion

Regular expression matching is an essential ingredient to most security appliances. While many manufacturers seek the assistance of RegEx hardware to assist with this task, the heterogeneous multicore architecture proves to be the most effective for network and security applications.

Adding network flow processors to standard x86 platforms provides network and security equipment manufacturers with a complete solution to support regular expression matching at 10 to 40 Gbps, without the assistance of specialized RegEx hardware or designs.

With the release of the Westmere CPUs and the NFP-3240 flow processor, manufacturers will have a complete solution to meet 40 Gbps requirements without changing their underlying hardware and software design. While the network flow processor is responsible for flow classification, actions and packet delivery, it simultaneously relieves the x86 CPU of such tasks, making more cycles available for complex matching.

Robert Truesdell *is a Field Application Engineer for Netronome, holds a BS in Computer Science from the University of Pittsburgh. Prior to Netronome, Robert worked as a Network Engineer at Concurrent Technologies.*