Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata

Joshua S. White[a], Thomas Fitzsimmons[b], Jeanna N. Matthews[c]

[a]Wallace H. Coulter School of Engineering
[b,c]Department of Computer Science
{[a]whitejs, [b]fitzsitd, [c]jnm}@clarkson.edu
Clarkson University, Potsdam, NY USA

ABSTRACT

Given competing claims, an objective head-to-head comparison of the performance of both the Snort® and Suricata Intrusion Detection Systems is needed. In this paper, we present a comprehensive quantitative comparison of the two systems. We have developed a rigorous testing framework that examines the performance of both systems as we scale system resources. Our results show that a single instance of Suricata is able to deliver substantially higher performance than a corresponding single instance of *Snort*. This paper describes in detail both the testing framework capabilities, tests performed and results found.

Keywords: Intrusion Detection Systems, *Snort*, Suricata, Benchmark

1. INTRODUCTION

Any modern organization that is serious about security deploys a network intrusion detection system (NIDS) to monitor network traffic for signs of malicious activity. Setting up and running a NIDS takes a lot of specialized knowledge. The work that this paper describes is intended to guide those in some of the considerations that should be taken into account.

The most widely deployed NIDS system is Snort®, an open source system originally released in 1998. *Snort* is a single threaded system that uses a set of clear text rules to instruct a base engine how to react when particular traffic patterns are detected. In 2009, the US Department of Homeland Security and a consortium of private companies provided substantial grant funding to a newly created organization known as the Open Information Security Foundation (OISF), to build a multi-threaded alternative to *Snort*, called Suricata. Despite many similarities between *Snort* and Suricata, the OISF stated it was essential to replace the older single-threaded *Snort* engine with a multi-threaded system that could deliver higher performance and better scalability. Key *Snort* developers argued that Suricata's multi-threaded architecture would actually slow the detection process.

Given these competing claims, an objective head-to-head comparison of the performance of *Snort* and Suricata is needed. In this paper, we present a comprehensive quantitative comparison of the two systems. We have developed a rigorous testing framework that examines the performance of both systems as we scale system resources through 24 cores. We compare Suricata to both single instance and multi-instance *Snort*. We document how changes in rulesets and worloads affect the results. Our results led directly to substantial improvements in the ability of Suricata to scale to large numbers of cores.

Network Intrusion Detection Systems (NIDS) capture and inspect network traffic for signs of malicious or unauthorized activity. NIDS can be compared on many dimensions including performance, scalability, accuracy, ability to detect different kinds of attacks, ease of use, the responsiveness of the development community to requests for new features, documentation quality, and many others. In this paper, we focus specifically on comparing the performance and scalability of two of the most widely deployed NIDS: *Snort* and Suricata. Keeping up with all the traffic on a busy network is a performance intensive activity. If the NIDS is unable to keep up with the traffic in real-time, then uninspected packets are either dropped causing problems for legitimate traffic or allowed to flow causing problems for security.

In both *Snort* and Suricata, a base engine is controlled by a set of rules. Each rule describes network activity that is considered malicious or unwanted by specifying the content of network packets. Each rule also specifies an action to be taken in case a packet is suspect, such as raising an alert or dropping the packet. The base engine must read each raw packet from the network and quickly compare it against all the rules loaded into the system to determine what action to take.

In this section, we expand a bit on the history of both *Snort* and Suricata.

1.1 *Snort*

Snort is a free open source, NIDS. Originally released in 1998 by Martin Roesch as a lightweight cross-platform network sniffing tool (around 1200 lines of code), it has evolved into a powerful and full-featured intrusion detection and prevention product. *Snort* is a successful example of the open source development methodology in which community members contribute source code, bug reports, bug fixes, documentation and related tools.

In November 1999, Roesch published "*Snort*: Lightweight Intrusion Detection for Networks" at the 13th Annual LISA Conference. He detailed his work creating a pattern matching system for the output of the *Snort* sniffer.[2] Soon after pre-processors for protocol normalization where added to the *Snort* engine. This allowed a single rule to be applied to any variation of a protocol.

Today, *Snort* is one the most popular security tools of all time.[4–7] According to the *Snort* web site, it is actually the most widely deployed intrusion prevention technology in the world.[3]

Sourcefire Inc., a company founded in 2001 by Martin Roesch, manages the development of *Snort* and offers commercial products based on *Snort*. They also research the newest threats and develop rules for detecting them. The newest rules are available by paid subscription immediately and then released to all registered users after 30 days.

1.2 Suricata

In 2009, the US Department of Homeland Security, along with a consortium of private companies, provided substantial grant funding to a newly created organization, the Open Information Security Foundation (OISF). The grant was to build an alternative to *Snort*, called Suricata. Suricata was first released in 2010 and we worked primarity with version 1.2 released in January 2012.

Although all code is original, Suricata developers have made no attempt to disguise the many ways in which they are borrowing from the *Snort* architecture. They readily acknowledge *Snort* as "our collective roots". Suricata can even be used with the same rule sets used by *Snort*.

1.3 *Snort* vs. Suricata

With the wide success of *Snort*, it is natural to wonder what would motivate the development of another similar open source system. One of the primary reasons was concern for the performance limits of *Snort's* single threaded architecture. When *Snort* was built, it was designed to run on the most popular computers of the time, 32-bit single core systems. While many improvements have been made to *Snort* over the past 14 years, the base engine has remained single-threaded.

The task of comparing multiple network packets against a large list of intrusion detection rules certainly appears to be a highly parallelizable task for which modern multi-core systems would be well suited. Still, there has been substantial public debate about the need for a multi-threaded engine. OISF president, Matt Jonkman, argues that a multi-threaded engine is essential for high performance and scalability, but that there has been no commercial motivation for anyone in the *Snort* community to invest in rewriting the core engine from scratch.[23] He stated that "the money goes to management/forensics consoles, rules, and big fast boxes".[24] On the other hand, Martin Roesch, founder of *Snort* and Sourcefire, argued that the multi-threaded architecture would actually slow detection rather than make it faster.[23] The performance of Suricata's specific implementation of a multi-threaded architecture has also been criticized. For example, Roesch had called Suricata "a clone of *Snort* that performs worse at taxpayer's expense" and SourceFire's Vulnerability Research Team has reported that the performance of Suricata "isn't just bad; it's hideously, unforgivably bad".[24]

Beyond questions about performance and the fundamental need for a multi-threaded architecture, Matt Jonkman, OISF founder, cites concerns about the *Snort* development process[26] as significant factors in the development of Suricata. * Unlike many open source projects that have commercial offerings, Sourcefire has not forked a separate commercial version of *Snort* under a dual-license model.[27] As a result the community version of *Snort* is tightly controlled by Sourcefire and it is difficult to get insight into the *Snort* development process, bug tracking information or future roadmap. In addition, developers outside Sourcefire have a difficult time submitting bug fixes. Jonkman offers up the example of Will Metcalf, now OISF QA lead, who discovered 35+ major bugs in *Snort* over the past 3 years. When Metcalf tried to contribute code to

*Jonkman clarifies that the various reasons cited for the development of Suricata are not necessarily the same as what motivated the Department of Homeland Security (DHS) and many private companies to contribute backing to the project.[26] Instead, it was both government and industry need for innovation in the IDS world.[28]

fix these bugs, he reports that his contributions went unacknowledged by the *Snort* development team and that a number of the flaws still remain. Any developers that do succeed in contributing code to *Snort* must also sign over the rights to that code to Sourcefire rather than simply releasing it under an open source license. In the case of Suricata, the OISF owns the copyright and the software can not be commercialized because the foundation is a non-profit organization.[26] [†]

1.4 Other Considerations

While the focus of this paper is performance and scalability, it is also worth briefly mentioning some other differences between *Snort* and Suricata. There are certainly differences in features. For example, Suricata includes some innovative features such as the ability to automatically detect common network protocols even when they are used on non-standard ports. Similarly, *Snort* contains features unavailable in Suricata, including some features required in classified environments such as the option of hiding the rules being used for inspecting network traffic. In addition, since Suricata is newer and less widespread, there are fewer resources documenting installation procedures and configuration options than there are for *Snort*.

2. RELATED WORK

In 2011, Day and Burns compared the performance and accuracy of *Snort* and Suricata through 4 cores using VMware virtual machines, but concluded that additional study was needed to examine performance at even larger numbers of cores.[13] Indeed, our study revealed surprising results above 4 cores and led to substantial improvements in the performance of Suricata.

Another paper with similar objectives to our own was "A Comparitive Analysis of *Snort* and Suricata Intrusion Detection Systems" by Eugene Albin.[14] Albin presents three experiments in comparing the performance of *Snort* and Suricata: using live network traffic, static pcap files, and testing ruleset functionality using Pytbull. Albin also used a VMware ESXi hosted virtual machine for the majority of his work and states that this may have contributed to the potential skewing of packet rates when compared to a physical machine.

One of the chief differences between Albin's work and this research, was an unquantified and varying component of background traffic that limits the repeatability of his tests. For example, Albin's second experiment shows a graph comparing the performance of both *Snort* and Suricata showing live network traffic varied between runs. Albin points out that there was no consistency across runs in terms of traffic type and traffic volume. Albin's use of PytBull in his later experiments also appears to have also been done while connected to a live network, causing traffic encountered to be potentially susceptible to the same inherent network variability.

3. METHODOLOGY

The goal of this paper is to present a thorough, repeatable, quantitative, apples-to-apples comparison of the performance of *Snort* and Suricata. Our experiments were conducted on a flexible hardware platform that allowed us to scale the hardware resources actively available. We ran our experiments with pcap traces to ensure repeatability and are providing all of our scripts and traces to encourage others to run the test suite on their own hardware.

In this section, we describe in detail the ways in which we varied each aspect of our testing. We varied the number of cores used, the rulesets used, the configuration of each IDS and the workload used. The script we developed for our testing consists of approximately 3000 LOC and is available at our project webpage.[22] Our basic methodology could easily be extended to include more workloads via additional pcap traces, additional rulesets and additional configuration settings. The same test suite could also be used on any hardware platform for which *Snort* and Suricata are available.

For each test, we capture a variety of metrics including packets per second (PPS) as processed by each IDS, the amount of memory used by each IDS process and the CPU utilization. To capture PPS measurements, we use builtin reporting functions in each IDS. For *Snort*, we enabled the perfmonitor function as described in the perfmon performance profiling guide.[36] Suricata has a similar functionality known as statistical logging that is enabled by default. Using the *Snort*.conf and Suricata.yaml files we set both engines to output statistics to their respective log files in 1 second intervals. To capture memory usage and CPU utilization measurements, our scripts parsed the output of the standard ps command on Linux and extracted the memory usage and CPU usage information for each IDS process.

[†]The OISF views Suricata as the first of many projects and intends to be a safe place for long term open source projects to reside without developers needing to fear that the project will become closed or their contributions unacknowledged.

3.1 Varying the Hardware Resources

The test system consisted of an AMD 8439 Opteron class processor and a 4 socket by 4 Memory Bank motherboard. Table 1 describes the details of the hardware platform we used. We were able to vary the number of CPU cores actively used in each experiment.

Table 1. Details Of Hardware Testing Platform

Server Specifications	
RAM	RAM 8 x 8GB 240Pin DDR3 PC3 10600 RAM
CPU	2 x CPU AMD Opteron 6234 Interlagos 2.4 GHz 12-Core
L1 Cache	128KB Per Core
L2 Cache	12MB Per CPU
L3 Cache	16MBPer CPU
Motherboard	ASUS K3PE-D16
CPU Bus	AMD HyperTransporttm 3.0 Max 124.8 GB/s Aggregated Bandwidth
Disk	6 x 2TB Western Digital Black

We encourage others to use our testing scripts and pcap files to run tests in their own hardware environment.This is the best way to gauge actual performance differences before choosing an IDS system.

3.2 Varying the Configuration

We have tested both *Snort* and Suricata in their default, out-of-the-box configurations to establish a baseline. We also experimented with varying some of the configurarion settings. In the results section, we refer to performance optimized configurations of both *Snort* and Suricata and in this section we describe in detail what that entails.

In the case of Suricata, configuration modifications were done by changing some end-user configuration parameters in the suricata.yaml file and not by compiling in any special acceleration options, such as PF_Ring, that may have added additional performance. The parameter max-pending-packets specifies the maximum number of packets that Suricata can process simultaneously. It has a default value of 50. In the performance optimized tests of Suricata, we set this value to its maximum 65535 packets. This setting is the maximum hard limit for this value due to the packet pool being a lockless ringbuffer that can contain USHRT_MAX. USHRT_MAX is the maximum value that can be stored in an unsigned short variable. Setting this value to its maximum is supposed to increase performance substantially on a multi-core/threaded system as indicated in discussions on the Suricata wiki/mailing lists.[14, 16, 18–20]

Suricata additionally allows for the run-modes to be changed. In the performance optimized tests of Suricata, we have changed the run-mode from its default value of auto to autofp. AutoFP or Automatic Flow Pinned mode is an Intel technology for a multi-threaded environment that can guarantee all processes related to a single packet of data reside on a single core.[18] If a packet requires multiple threads either for pre-processing or rule comparison, overall performance can be hindered if the threads reside on physically different cores. This is caused by time delays necessary for copying the data between cores. AutoFP ensures that if packets are all part of a single flow they will be processed on the same core if the process is specifically doing flow based work.

In the case of *Snort*, we switched the ac-bnfa-nq (Aho-Corasick Binary NFA) mode that *Snort* uses to search the payload for specific strings to simple ac (Aho-Corasick) mode. According to many sources this method uses more memory then the default mode but when used can increase performance significantly.[30–33] We also modified *Snorts* max-pkt-time and set it to 1000, which mean that any packet taking more then 1000 usec to process is dropped. In many situations when a network has large bursts of traffic or sensors don't have enough available memory this modification could potentially result in a large number of dropped packets.[30, 34, 35]

3.3 Varying the Rulesets

Another important factor when running an IDS like *Snort* or Suricata is the choice of ruleset.

Emerging Threats is an open source community that was initially created to support an open *Snort* ruleset. Currently this group produces rulesets compatible with both *Snort* and Suricata. The Emerging Threats Open Ruleset (ET-Open) consists of contributions from community members and is freely available for download.[8]

Emerging Threats, also produces a professional ruleset (ET-Pro). In the ET-Pro ruleset, each item contains a rule portion that is optimized for *Snort*, a rule portion that is optimized for Suricata and an alert portion that is shared by both engines. Items in the ET-Pro set do not necessarily migrate to the ET-Open set overtime. They are a seperate set that are optimized for *Snort* and Suricata by the Emerging Threats team. A home user license for the ET-Pro ruleset is currently $35 annually and a enterprise user license is $500 annually per sensor. They offer volume licenses on a case by case basis for organizations requiring over 10 sensors.

The Sourcefire Vulnerability Research Team (VRT) produces the official rule set for *Snort*. New rules released by the VRT are free to the community after approximately 1 month. They are available immediately upon release through various subscription models. Personal licenses cost $29.99 and business licenses start at $399/sensor.[29]

Table 2 summarizes the rulesets we used in our testing.

Table 2. Ruleset Details

Ruleset Name	**Version**	**# Of Rules**
ET-Open Ruleset	6953	16179
ET-Pro Ruleset	8101424752816199	13154
Snort VRT Ruleset	10-2011	18038

3.4 Varying the Workload

In order to vary the workload presented to the IDS, we used a variety of pcap trace files. The pcap traces we used came from two sources. Our largest data set consisted of the 2010 iCTF Conference "Attacks Against Litya" network capture that consisted of 67GB of captured network traffic (23.7 GB compressed). This network capture was taken during the conference contest in which participants attacked the fictitious "nation of Litya, ruled by the evil Lisboy Bironulesk."[15] The scenario and network service design forced the participants to attack the infrastructure of Litya, much like a critical infrastructure nation state cyber attack.

The second set of pcap files we created outselves by using tcpdump to capture runs of specific PytBull traffic.[9] Pytbull is an open source IDS testing framework. It allows the user to test specific payloads and traffic against different IDS and different rulesets. Pytbull was used to target specified payloads, allowing us to analyze the performance of *Snort* versus Suricata. The traffic generated by each of the tests were captured in pcap format and replayed for each configuration by our testing script.

Table 3 summarizes the set of pcap files we used in our testing.

Table 3. PCap File Details

PCap Test Name	**Description**	**# Of Packets**	**Size**
iCTF Conf. Workload	Attack Workload	818178784	67 GB
Client-side Attacks	Client-side Download Attacks	3786	6.1 MB
Test Rules	Basic Ruleset Testing	245167	22.5 MB
Bad Traffic	Non-RFC Compliant Packets	2152	3.7 MB
Fragmented Packets	Various Fragmented Payloads	2459	4.4 MB
Multiple Failed Logins	Track Multiple Failed Logins	1200	1.8 MB
Evasion Techniques (ET)	Detection of Various (ET)	52323	21.6 MB
Shell Codes	Ability to Detect Shell Codes	8200	15.4 MB
Denial Of Service	Ability to Detect DoS Attempts	3312	2.3 MB
Pytbull All	TCPDump of All Pytbull Runs	308067	63.2 MB

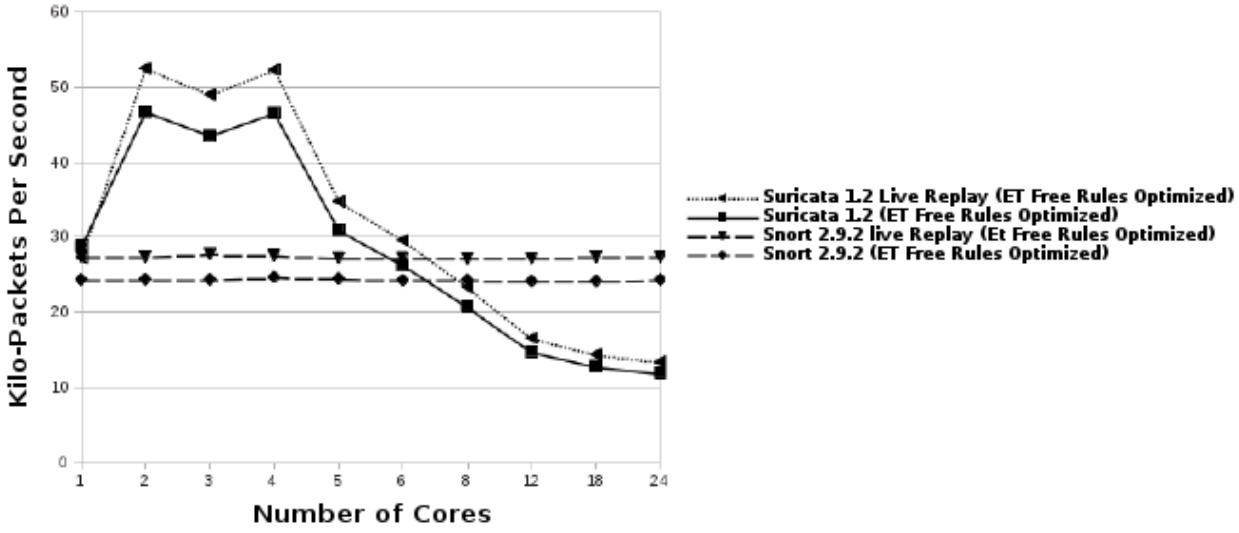

Figure 1. Baseline - Default configuration with ET-Open ruleset, ICTF 2010 dataset

4. RESULTS

4.1 Tests

We ran a total of 8600 tests. Specifically, we tested 10 workloads (as shown in Table 3), 4 rulesets configurations (ET-Open, ET-Pro, VRT and No ruleset), 4 IDS configurations (default and performance optimizied configurations of both *Snort* and Suricata as discussed in section 3.2) and 10 settings for the number of available cores (1,2,3,4,5,6,8,12,18,and 24). Each of these 1600 variations were run 5 times each. In addition to these 8000 tests in which each IDS processed packets directly from the pcacp files, we also ran another 600 tests in which we used a separate machine to read the pcap files and play them onto the network segment for processing by the IDS.

At the time of our testing, we used the most recent versions of *Snort* and Suricata were pulled from available repositories and used. The versions used were as follows: *Snort* v2.9.2, and Suricata v1.2.

4.2 Out of the Box Defaults

Figure 1 shows a baseline comparison of *Snort* and Suricata. Both *Snort* and Suricata are shown in their default out-of-the-box configuration and they are both using the exact same set of rules, the ET-Open Ruleset. The workload used is the 2010 iCTF Conference trace. The y-axis shows the average Packets Per Second (PPS) while the x-axis shows the number of CPU cores used.

To validate our methodology of using pcap files, we also ran our tests using a separate machine to replay the pcaps onto the network at various speeds rather than reading the pcap file directly into the IDS under tests. These live traffic tests were done by replaying packets from the ICTF 2010 capture at the rate which they were originally captured at. The packets were rewritten to make use of the 10.10.1.0/24 network configuration that our systems were set to. Both *Snort* and Suricata were set to monitor this IP range and our interface was set to promiscuous mode. As the tests show our maximum Suricata performance was around 95,000 PPS. [‡] We attribute this jump in performance compared to Suricata running in PCAP read mode to issues we suspect exist with both read latency and disk I/O.

Overall, there is little difference in the results with live traffic replay. Reading the traces from the pcap eliminates the need for a separate replay machine and thus reduces the complexity of the experimental infrastructure. It is also interesting to

[‡]Many IDS Studies use PPS as the primary metric for measuring performance of a system. Some however, may prefer a metric of Gbps especially when deploying a system in a known network configuration. For our live tests we replayed traffic from a recorded dataset which had varying packet sizes. We set a fixed maximum transmission unit (MTU) size of 9600 Bytes for our 10GigE Interface. In the event that all packets being replayed were at the MTU of 9600 Bytes it would take approximately 139,000 PPS to reach 10 Gbps.

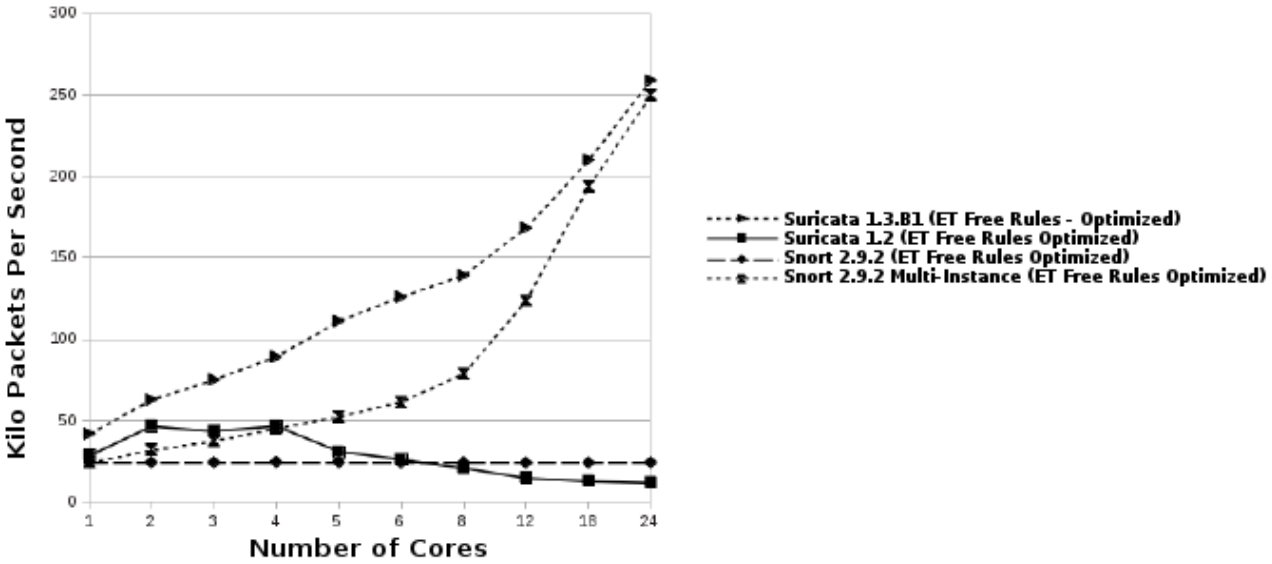

Figure 2. Various Configurations - *Snort* Running In Multi-instance Mode and Suricata Development Version

note that when reading the pcap files directly there is no possibility of dropped packets because the system reads the input as fast as it can process it and no faster. This improves repeatability of the tests. In the remainder of the tests, we read packets directly from the pcap traces.

4.3 Performance vs. Defaults

Based on Albin's study and on other known opinions about the performance of *Snort* and Suricata, we expected that Suricata would not perform as well as the mature *Snort* on a single core, but would eventually out-perform *Snort* as additional cores were made available. However, Suricata performed slighly better than *Snort* at 1 core, reflecting substantial efforts at tuning.

Snort's performance is flat as we add additional cores because the single threaded architecture is unable to take advantage of the additional cores. Suricata's performance initially increases as cores are added, but peaks between 2 and 4 cores and then drops off substantially as more cores are added. We investigated this non-intuitive result with the Suricata development team and indeed, a substantial problem was found and fixed. Specifically, several modifications to Suricata's flow engine which resulted in large improvments in scalability past four cores.[37]

These improvments included the replacement of a series of ordered lists which handled packet queing with hash tables created using Jenkins Hashing. Doing this added to the CPU usage but in turn resulted in much faster packet queing. In addtion to this the previous Suricata 1.2 version used two seperate hash tables for both the thresholding and tag engines. Each of these tables were controlled by a single lock. Since each packet results in a lookup as does each tag these frequent lookups created a bottlneck. This has been addressed in version 1.3 Beta 1 with the introduction of a fine grain locking model where both tagging and thresholding use the same hash table. In addition our work spured the OISF developers to added lock profiling code to the Suricata codebase which will allow for easier future profiling of the engine.

Snort, by default is a single-threaded, single-instance archtitecture. However, it is also possible to run *Snort* in mult-instance mode using scripts provided by Metaflows which include PF_Ring support for *Snort*.[38] These scripts include a modification of PF_Ring to enable load balancing for *Snort*.

Figure 2 compares single-instance *Snort* 2.9.2 and Suricata 1.2 as shown in Figure 1 to multi-instance *Snort* 2.9.2 and a development build of Suricata (1.3B1) that contains the improvements described above. There are substantial changes in the performance of both *Snort* and Suricata as a result. Both multi-instance *Snort* and Suricata 1.3B1 demonstrate an ability to scale well with added cores. At 24 cores their performance is quite close with Suricata out-performing *Snort* by as much as 9,299 Packets Per Second on all core settings.

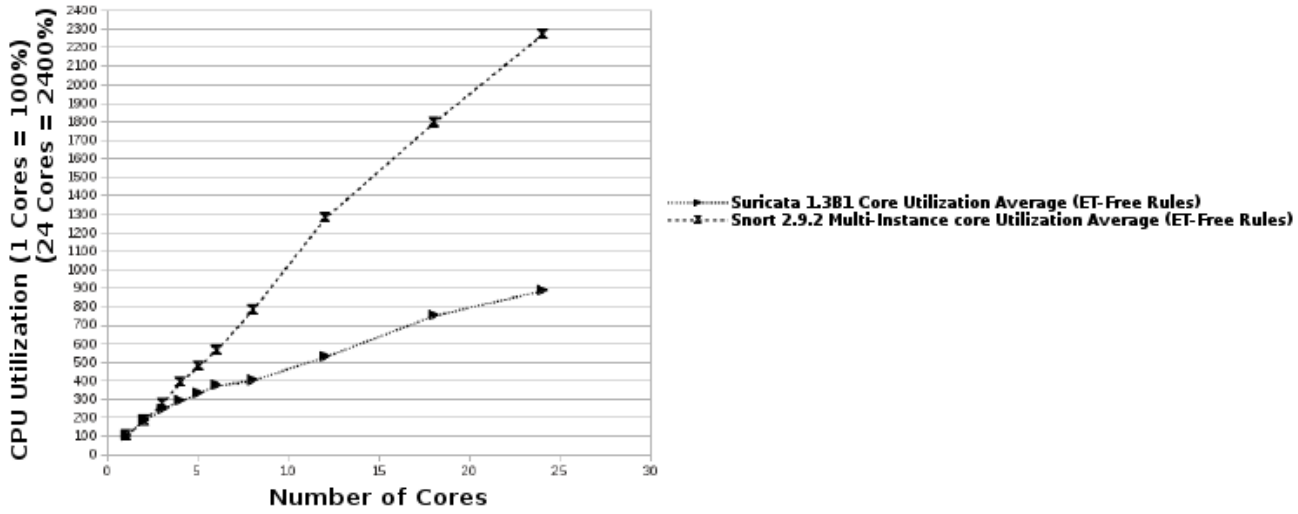

Figure 3. Impact on Average CPU Utilization — ET-Open with Default and Performance Optimized configurations, ICTF 2010 Dataset

In Figure 3, we shift from examining average packets per second to CPU utilization. The y-axis shows average CPU utilization. With 24 cores, the maximum value is 2400% or 24 cores are 100% utilized. The results of the using the ET-Open Ruleset with both multi-instance *Snort* 2.9.2 and Suricata 1.3B1. *Snort* uses substantially more CPU resources especially above 5 cores. Similarly, in Figure 2, shows result using the ET-Open Ruleset for both *Snort* and Suricata. *Snort* uses more memory, especially above 12 cores.

4.4 Varying Rulesets

It is worth noting that differences in ruleset can cause substantial difference in performance. System administrators are warned of the possible impact of adding poorly written custom rules. Figure 5 shows the results of our rule variation testing. With more recent versions of both *Snort* and Suricata we tested the iCTF 2010 workload against the engines running both the ET-Free Rules and ET-Pro rulesets.

We saw some difference between the rulesets though not as much as we expected even though ET-Open rules are contributed by community members and ET-Pro rules are professionally tuned. The ET-Open set is still vetted by the Emerging Threats team and would presumably not contain any excessively compute expensive rules. Figure 5 also shows that given the small performance boost that the ET-Pro ruleset gave *Snort* it was able to come within 1,581 Packet Per Second of Suricata running the ET-Free Ruleset. For short the overall improvment between ET-Free and ET-Pro was 7,718 PPS while Suricata saw an improvment of 49,918 PPS.

4.5 Varying Workload

We experimented with different workloads and their effects on each IDS. For all previous figures, we used the iCTF pcap trace, but here we compare two runs with traces of individual Pytbull tests.[9] Figure 5 shows the results of running *Snort* with the ET-Open ruleset at 1 core and Suricata with the ET-Open ruleset across 1, 4, and 24 cores with different workloads.

As we saw in our baseline results in Figure 1, these results demonstrate that Suricata's PPS throughput peaks at 4 cores on most workloads. This is most notable on the iCTF and Evasion Techniques workloads as shown in Figure 6. The Evasion Techniques, included in Pytbull, represent a bundle of attacks, with slight modifications, made to evade an IDS. This can include obfuscating the payloads, protocol violations, overlapping packet fragments, and more. This set of techniques is made to challenge the IDS capabilities. Across almost every workload, Suricata has better performance with the exception of the Evasion Techniques workload at 1 and 24 cores.

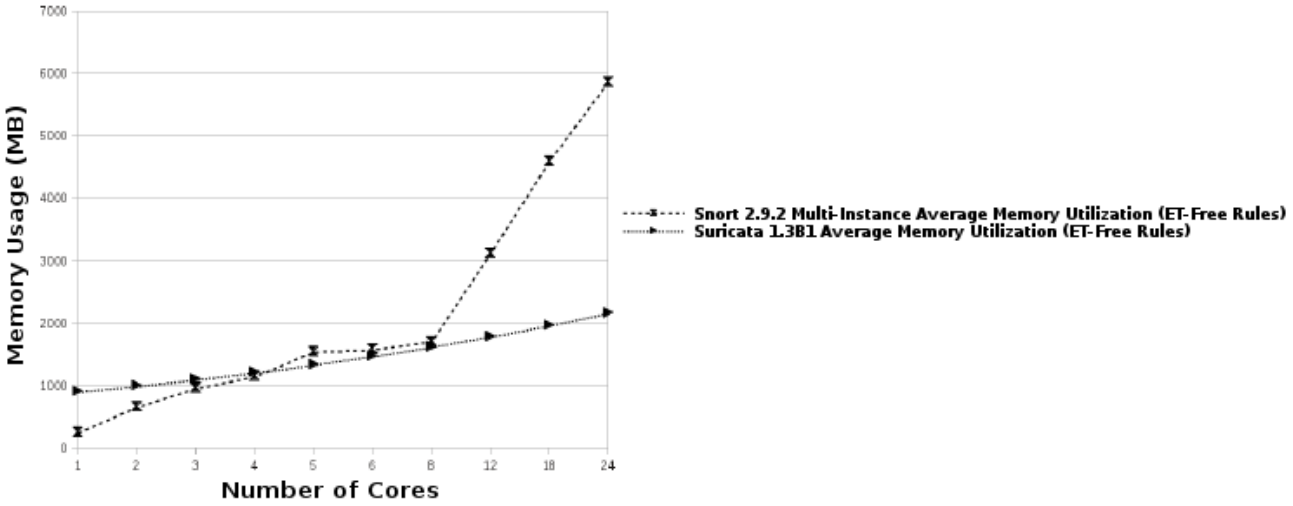

Figure 4. Impact on Average Memory Utilization - Et-Open with Default and Performance Optimized configuration, iCTF 2010 Dataset

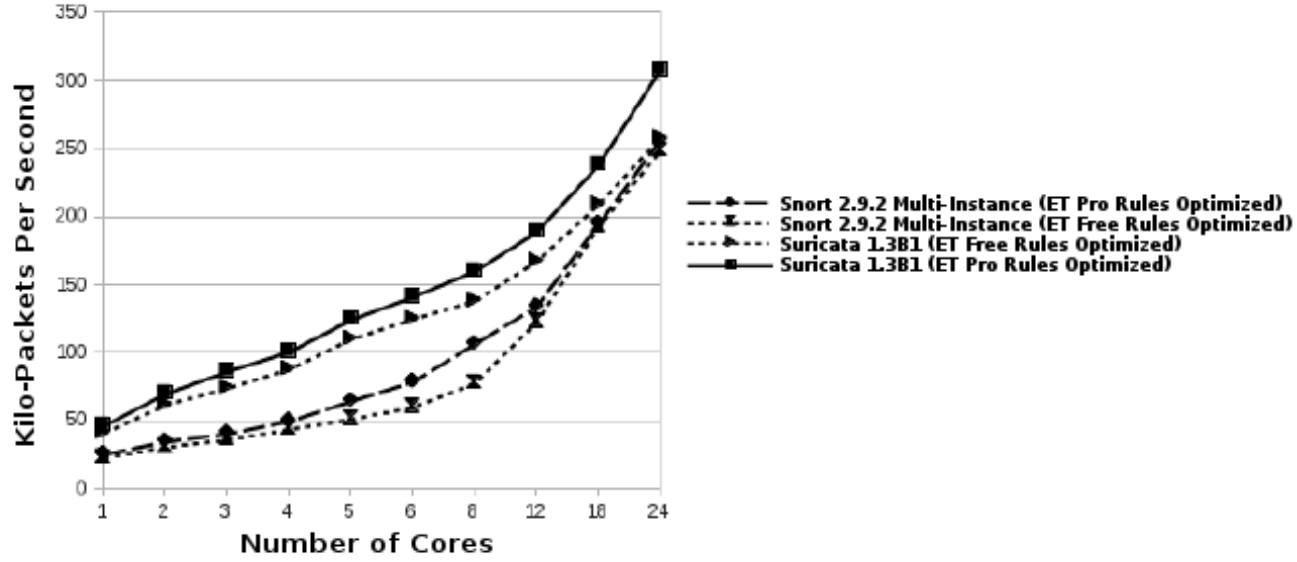

Figure 5. Various Configurations - *Snort* Running In Multi-instance Mode and Suricata Development Version

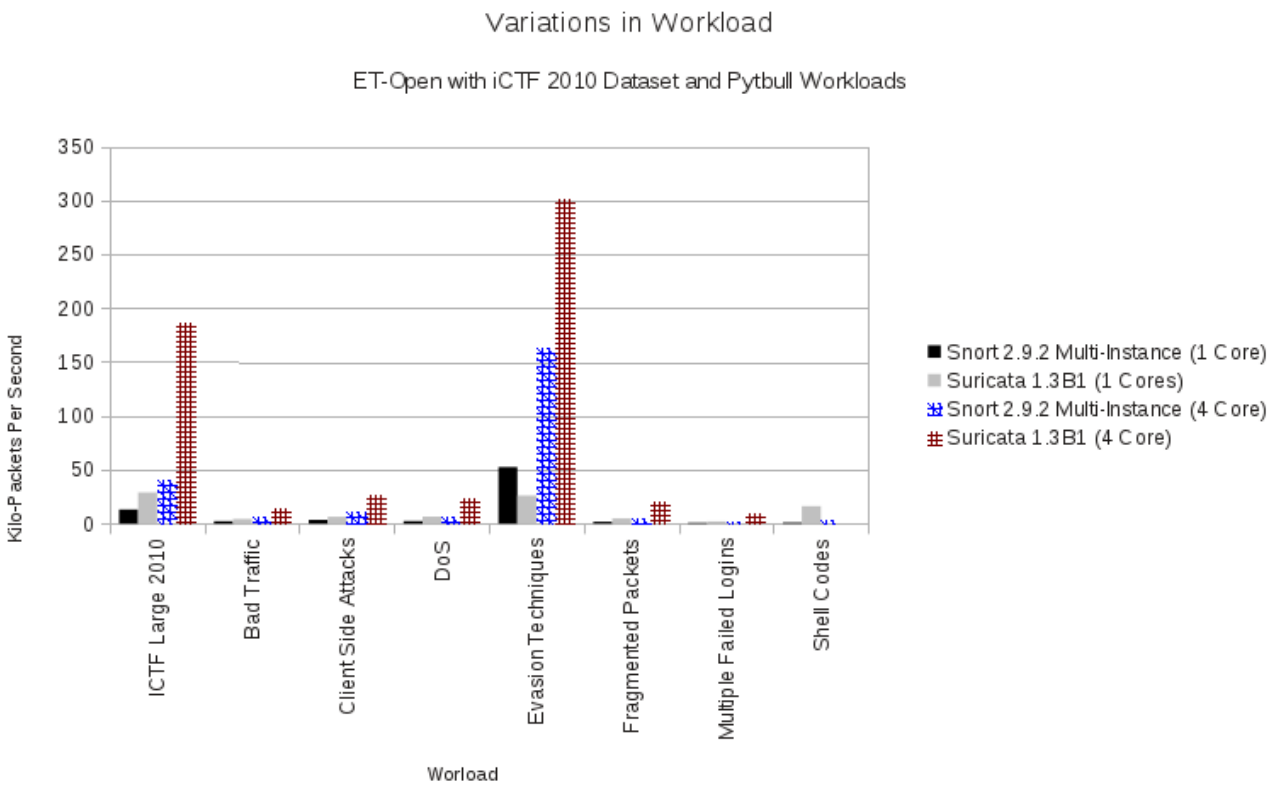

Figure 6. Variations in Workload - ET-Open, iCTF 2010 Dataset and Pytbull Workloads

5. CONCLUSION

Our results demonstrate substantial problems with scalability for the the versions of both *Snort* and Suricata tested when run in the default configurations. For *Snort*, we demonstrate the scalability achieved with a multi-instance configuration. For Suricata, our testing lead to substantial increase in scalability with a maximum of 52,550 PPS in version 1.2 and 258,912 PPS in version 1.3 Beta 1. With these changes, Suricata outperforms *Snort* by as little as 9,299 Packets Per Second at 24 cores and by as much as 64460 PPS at 6 cores. Specifically, our results show Suricata outperforming *Snort*, even in a single core, where we expected *Snort* to have an advantage. Interestingly, Suricata is also able to achieve lower average memory usage and lower average CPU utilization as well. Finally, we explored variations in rulesets and in workload, but find that our base conclusions held across these variations.

6. FUTURE WORK

The results we presented in this paper could be easily be augmented with tests on additional hardware platforms, with trace of additional workloads and using different configurations of *Snort* and Suricata. We plan to continue this work ourselves, but we are also hoping that other groups will take our testing infrastructure and use it in their own environments - with their own rulesets and workloads. We would like to found a repository of results to which others could contribute.

A community respository containing results of independent performance comparisons across many environments could help substantially in demystifying the choice of IDS for many organizations. We hear claims of substantially higher performance than we observed in our local tests, but when we ask questions about the configuration or details of the hardware required, few details are available. *Snort* considers this type of information on its products to be proprietary and it is therefore difficult to verify independently. Running *Snort* in a multi-process, parallel configuration on multiple cores requires special preprocessing and postprocessing glue code/scripts and the hardware necessary to achieve packet per second ratings in the

Gbps range appears to require hardware priced in the range of tens to hundreds of thousands of dollars. [§] The main *Snort* web page mentions 400,000 registered *Snort* users and we wonder what percentage are running *Snort* in such customized configurations. One goal of our paper is to provide repeatable performance experiments in more standard/commodity environments.

7. ACKNOWLEDGEMENTS

We would like to thank members of the *Snort* and Suricata communities for answering our many questions and for feedback on earlier versions of this work. We would especially like to thank Matt Jonkman, Victor Julien and Will Metcalf of the OISF and Eugene Albin from the Naval Postgraduate school. We would like to thank Bivio Networks for answering a number of questions about their products and how they run parallel instances of *Snort*. Finally we would like to thank a members of the *Snort* community that preferred to remain anonymous but whom answered questions regarding *Snort* conguration, optimization and preferred deployment.

REFERENCES

1. http://snort.org, December 2011.
2. M. Roesch, "Snort: Lightweight Intrusion Detection for Networks", 13th Annual Systems Administration Conference (LISA), November 1999.
3. "Snort: The De Facto Standard for Intrusion Detection and Prevention", http://www.sourcefire.com/security-technologies/open-source/snort, December 2011.
4. SecTools.org, "Top 125 Security Tools", http://sectools.org, December 2011.
5. Infoworld.com, "The Greatests Open Source Software of All Time", http://www.infoworld.com/d/open-source/greatest-open-source-software-all-time-776?source=fssr.
6. J. Carr, "Snort: Open Source Network Intrusion Prevention", Esecurity Planet, http://www.esecurityplanet.com/prevention/article.php/ 3681296/Snort-Open-Source-Network- Intrusion-Prevention.htm, June 2007.
7. J. Koziol, "Intrusion Detection with Snort", Sams Publishing, May 2003.
8. Emerging Threats, http://emergingthreatspro.com, December 2011.
9. Pytbull, http://pytbull.sourceforge.net, December 2011.
10. Sysbench, http://sysbench.sourceforge.net/docs/, December 2011.
11. S. Demaye, "Suricata-vs-snort", http://www.aldeid.com/wiki/Suricata-vs-snort, December 2011.
12. OISF, http://www.openinfosecfoundation.org/projects/ suricata/wiki, December 2011.
13. D. Daye and B. Burns, "A performance analysis of snort and suricata network intrusion detection and prevention engines", ICDS 2011, The Fifth International Conference on Digital Society, 2011.
14. E. Albin, "A Comparative Analysis of Snort And Suricata Intrusion Detection Systems", Naval Postgraduate School, Dudley Know Library, September 2011.
15. iCTF pcap Dataset, "Full Packet Capture of (Attack Against Litya)", International Capture The Flag, http://ictf.cs.ucsb.edu/data/ictf_f2010/ ictf2010pcap.tar.gz , December 2010.
16. https://redmine.openinfosecfoundation.org/projects/ suricata/wiki/Suricatayaml, December 2011.
17. http://lists.openinfosecfoundation.org/pipermail/ oisf-users/2011-August/000820.html, December 2011.
18. http://lists.openinfosecfoundation.org/pipermail/ oisf-users/2011-February/000447.html, February 2011.
19. http://lists.openinfosecfoundation.org/pipermail/ oisf-users/2011-August/000820.html, August 2011.
20. http://lists.openinfosecfoundation.org/pipermail/ oisf-users/2011-December/001109.html, December 2011.
21. http://www.bivio.net/public_pdfs/Bivio_7000_DS.pdf, December 2011.
22. Snort and Suricata Performance Comparison, http://www.clarkson.edu/class/cs644/ids/, December 2011.

[§]One concrete datapoint we recevied comes from Bivio. The Bivio 7500 Series appliance uses hardware stream splitting and hardware application load balancing with custom FPGA's to split *Snort* across 48 Cores (i.e. multiple instances of *Snort*). They report around 10 Gbps of throughput. This 48 core solution is 8U in size, consisting of 4 x 2U appliances ganged together, consumes 2400 Watts of power, and costs around $100,000 per appliance. This configuration is not a *Snort* certified solution, but rather a solution developed especially for the DOD community.

23. B. Whaley, “Snort, Suricata creators in heated debate Are the open source projects irrevocably damaged?”, Network World US, July 2010.
24. J. Vijayan, “DHS vendors unveil open source intrusion detection engine”, Computerworld, July 2010.
25. E. Messmer, “Is open source Snort dead? Depends who you ask”, http://www.networkworld.com/news/2010/072010-is-snort-dead.html, NetworkWorld, July 2010.
26. M. Jonkman, Personal Email Communication, February 13 2010.
27. Valimaki, Mikko., “Dual Licensing in Open Source Software Industry,” Aalto University, Systemes d’Information et Management, Vol. 8, No. 1, pp. 63-75, 2003,
28. Department of Homeland Security Host Program, “Suricata as an Exemplary example of DHS innovation”, http://www.cyber.st.dhs.gov/host/
29. Snort, VRT Subscription options, http://www.snort.org/vrt/buy-a-subscription/, February 2012.
30. GameLinux.org, “Some Notes on Making Snort Go Fast Under Linux”, http://www.gamelinux.org/ page id 284, December 2011.
31. http://lists.emergingthreats.net/pipermail/emerging-sigs/2011-January/011641.html, January 2011.
32. Mikelococo.com, “Capacity Planning for Snort IDS”, http://mikelococo.com/2011/08/snort-capacity-planning/, August 2011.
33. Snort.org, “Snort 2.9.2 Manual, Section 2.1”, http://manual.snort.org/node16.html, 2011.
34. Snort.org, “Snort 2.9.2 Manual, Section 2.5”, http://manual.snort.org/node20.html, 2011.
35. http://seclists.org/snort/2011/q2/32, April 2011.
36. S. Sturges, “Using Perfmon and Performance Profiling to Tune Snort Preprocessors and Rules”, http://www.snort.org/assets/ 163/WhitePaper_Snort_PerformanceTuning_2009.pdf, November 2009.
37. V. Julien, “Suricata Scaling Improvements”, http://www.inliniac.net/blog/2012/05/29/suricata-scaling-improvements.html, May 2012.
38. Metaflows, http://www.metaflows.com/technology/pf-ring/, 2012