Gigabit NCAP: High-Performance Packet Sniffing on Commodity Hardware

Sergei Egorov
esl@fidelissec.com

Gene Savchuk
savchuk@fidelissec.com

Fidelis Security Systems, Inc.
www.fidelissec.com

Abstract

We have developed a high-performance Network Content Analysis Platform (NCAP) suitable for a variety of applications requiring access to all layers of network traffic including the content of TCP/IP network data exchanges. NCAP is capable of operating on fully saturated Gigabit traffic using commodity hardware (multiprocessor Intel/Linux boxes with Gigabit NICs). NCAP architecture is scalable; it allows for effective utilization of multiple CPUs in SMP configuration by breaking up the network sniffing and analytical applications into several modules communicating via IPC. While very few technologies used in NCAP are particularly new by themselves, we believe that the architectural solution in general and in many details is original and produces excellent results in realistic high-speed network environments. The white paper describes the architecture in detail and provides benchmarking results and comparison with relevant components of popular open-source solutions.

Introduction

Many network content analysis applications are based on a component providing effective and accurate reconstruction of network data exchanges. To accomplish this, the component should be able to capture individual packets traveling through the network with the help of the network interface card operating in the promiscuous mode, decode the packets uncovering the underlying transport layer (IP), merge fragmented packets, track the ongoing bi-directional data exchanges (sessions), and, for TCP sessions, reassemble both sides of each data session, making their entire content available for the content analysis layer.

Such reconstruction is complicated by several factors. One of the major factors is speed: Modern networking equipment supports the latest Gigabit Ethernet standard, so many network segments operate on effective speeds reaching 700-800 Mbps or higher. To keep up with such a connection, the sniffing component should be extremely fast so that every packet is captured and there is enough time left for analysis of its content (individually or as a part of the session). Another limiting factor is accuracy: The sniffer, being a passive application, does not have all the information needed to reconstruct all traffic in all cases (to do so, it should have access to internal state of the communicating hosts). The situation becomes even more complicated if the sniffer

needs to analyze Full Duplex stream or asymmetrically routed traffic—several related network streams should be captured via separate NICs and analyzed as a single communication channel.

Existing open-source and proprietary solutions for this problem fall short on many counts. The effective ones rely on special hardware such as IBM's PowerNP network processor; those that do not are too slow and inaccurate to be useful in realistic high-speed network environments.

To solve this problem, we designed a solution (the Network Content Analysis Platform, or NCAP) that does not rely on any special hardware. The extensive testing performed on real and artificial traffic [LL99, CCTF] demonstrates that NCAP is fast enough and accurate enough to rival solutions based on custom hardware.

NCAP Architecture Overview

NCAP (Network Content Analysis Platform) provides packet sniffing, defragmentation, decoding, IP and TCP session tracking, reassembly and analysis of layers 2-7 at Gigabit speeds. In addition to these functions, it features a unified event processing backend with temporary event storage and event spooler. NCAP software hosted on a 2.5GHz Dual Intel Xeon box with Gigabit Intel NICs has enough processing speed to handle saturated Gigabit lines with 7-layer processing and several content analysis modules. This is possible because of maximally effective use of all hardware, most importantly NICs and CPUs.

NCA Platform is designed to take full advantage of multiple CPUs, providing maximum scalability for sophisticated content analysis algorithms. This scalability is achieved by breaking the full application to multiple independent modules and connecting them via flexible IPC mechanisms, suitable for the given configuration. The Platform's API has the following methods of connecting the processing modules:

- **Inline.** The packet analyzer is compiled together with the framework to the same executable and takes its time share in the main packet processing cycle. This method is most suitable for single-processor hardware.
- **Packet-level parallel.** After being decoded and initially processed by the IP and TCP reassemblers, the packet is made available for further analysis to a separate process using a circular queue. Up to 32 external analyzers can be attached to a single queue. There is also an option to set up several independent queues, with round-robin packet distribution between them.
- **Stream-level parallel.** The TCP stream reassembler puts the reassembled stream data into a circular stream queue. This queue serves the programs designed to analyze the content of an entire client-server conversation. Up to 32 external analyzers can be connected to a single queue. Also, multiple queues can be configured, with round-robin distribution between them.

Both inline and external Content Analysis components generate events by calling up the central Event Processing component via a message-based API. The Event Processing component runs in a separate process with regular priority; it gets events from the input queue and writes them to the temporary file storage. The persistent event storage is needed to withstand network outages with minimal information loss.

The Event Processing component is designed to minimize the possible effect of DoS attacks against the sniffer itself. It reacts to a series of identical or very similar events by compressing the entire series into one "combined" event that stores all the information in compressed form; for identical events, the combined event will contain information from a single event together with the event count.

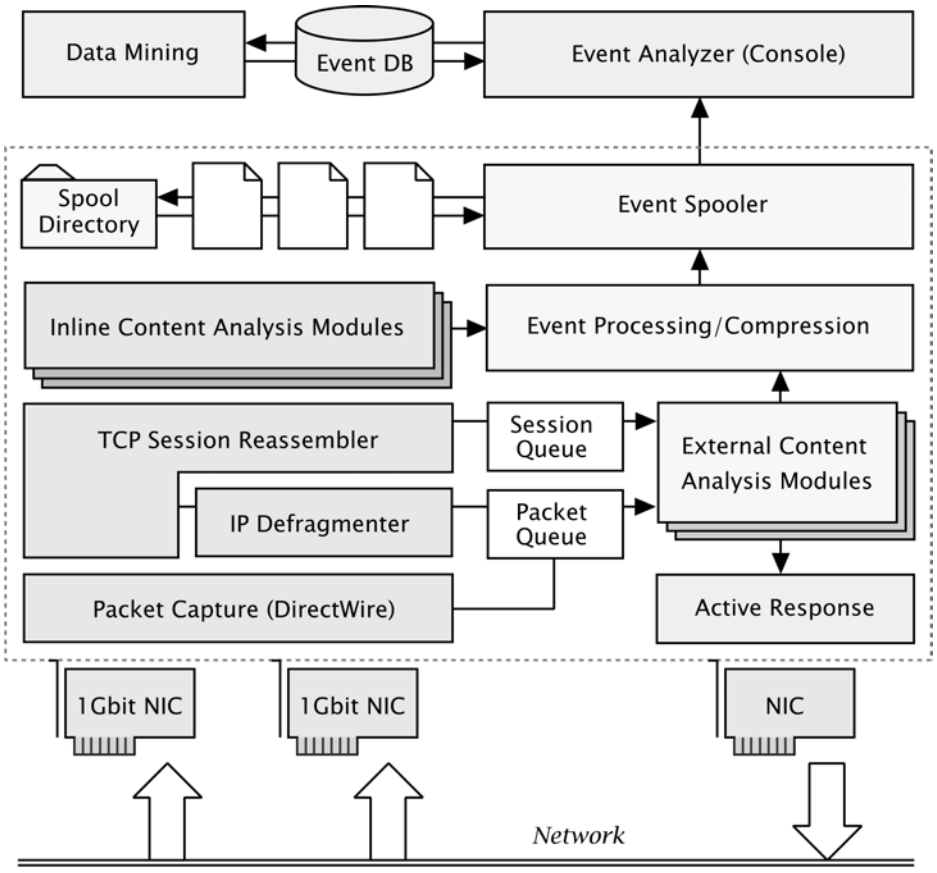

Fig.1. NCAP Architecture

The information collected by the Event Processor is sent to its destination (a separate event analysis component such as Data Mining Console) by an Event Spooling component. The Event Spooler keeps track of new events as they are written into a spool directory. Each new event is encrypted and sent to one or more destinations. The Event Spooler runs as a separate low-priority process.

The rest of the white paper describes in depth the critical NCAP components: Packet Capture, IP Defragmenter, and TCP Session Reassembler. NCAP-based content analysis modules will be described in separate white papers[1].

[1] Our rule-based content analysis module is based on SNORTRAN compiler [SNORTRAN].

Packet Capture Component (DirectWire)

NCA Platform's Packet Capture technology (DirectWire) has been developed to satisfy specific requirements for a Gigabit-capable network sniffer. In single-NIC half-duplex mode, DirectWire technology offers up to 2x speedup over conventional packet capturing methods on stock hardware (`libpcap` on a Linux/Intel box with Gigabit Intel NICs). This speedup is achieved by keeping time-consuming activities such as hardware interrupts, system calls and data copying to a minimum, thus leaving more time to packet processing. Real-life network traffic is heterogeneous. Usual packet size distribution, based on our experience, tends to have maximums at about 80 bytes and 1500 bytes. The packet rate distribution over time is highly uneven. Unlike the legitimate destination host, a network sniffer has no ability to negotiate packet rates according to its needs. Therefore, it should be designed to provide adequate buffering for the traffic being sniffed, thus guaranteeing an adequate processing window per each packet.

Each hardware interrupt potentially causes a context switch, a very expensive operation on a modern Intel CPU. To keep interrupts to a minimum, DirectWire relies on customized Intel NIC drivers making full use of Intel NIC's delayed-interrupt mode. The number of system calls is reduced by taking advantage of the so-called "turbo" extension to packet socket mode supported by latest Linux kernels (`PACKET_RX_RING` socket option).

When used to their full potential, modified drivers and turbo mode provide the fastest possible access to the NIC's data buffers; polling at 100% capacity causes only about 0.001 interrupt/system call per captured packet (amortized). To deal with momentary surges in traffic, DirectWire allocates several megabytes for packet buffers. Large buffers also reduce packet loss caused by irregular delays introduced by the IP defragmenter and TCP reassembler.

Unlike standard `libpcap`, DirectWire packet capture can operate in FD/SPAN modes using multiple NICs, providing support for full session reassembly. Packets coming from multiple NICs operating in promiscuous mode are interleaved by polling several packet buffers simultaneously. The polling strategy used by DirectWire does not introduce additional context switches or system calls; each buffer gets its share of attention calculated by a built-in load-balancing algorithm.

The DirectWire component is implemented as several load-on-demand dynamic libraries. The "general-purpose" library processes an arbitrary number of NICs. There are also versions with hard-coded parameters optimized for 1(HD mode) and 2(FD mode) NICs, as they are used most often in network appliances. The programming API for DirectWire resembles PCAP (full compatibility is impractical because of functional differences). The general-purpose library accepts interface initialization strings with multiple interfaces (e.g., "`eth1:eth3:eth5`").

Our measurements of real traffic and simulated traffic with a TCP-oriented model for distribution of packet arrival times demonstrated that improvements to packet buffering and pick-up increase the time slot for packet processing by 20% on average. On the same traffic this leads to a 30%-50% decrease in packet loss ratio (PLR) in the critical 0.5-1 Gbps zone, allowing the sensor to handle 1.5 times or more load given the same PLR cut-off and traffic saturation levels.

The benchmarking results shown below were prepared using the data collected on the following hardware: Entry-level server (Intel 845GE chipset with 33Mhz/32bit PCI bus), 2.4 Ghz Xeon processor with 400Mhz FSB, DDR266 memory and built-in Intel PRO/1000 NICs (82540EM chip). The actual throughput numbers shown are based on an assumption that packets

are 1500 bytes long; such an assumption is needed because we established that performance figures for both libraries used in our tests mostly depend on packets-per-second rate, not on bits-per second. Although an average packet length in realistic saturated Gigabit traffic is usually less than 1500 bytes (200-500 bytes depending on a variety of factors), the average bit in a 1Gbps data stream is most likely to be transferred by a packet of maximal length (in our experiments, probability of being in a large packet was 0.7-0.9). We did not use jumbo frames in our benchmarks.

No matter what assumptions one makes regarding the average packet size, the relative numbers remain the same. The benchmarking results only give a picture of the relative performance of the packet capture components; the overall performance of a complete system depends on the entire data processing pipeline and a multitude of other factors, including hardware choices.

Three series were measured for each library using different per-packet delays to model packet processing: 10mks, 5mks, and 3mks. The chart shows that DirectWire provides significant performance gains when the processing window decreases to 5mks and less (as network load increases). The 3mks window is very small, but sufficient for NCAP working in SMP mode to do decoding, defragmentation, TCP tracking/reassembly and offloading of the packet to one of several SNORTRAN-based packet processors. In the Full TCP Stream Processing mode, the NCAP does not need to offload individual packets, consuming even less processor time.

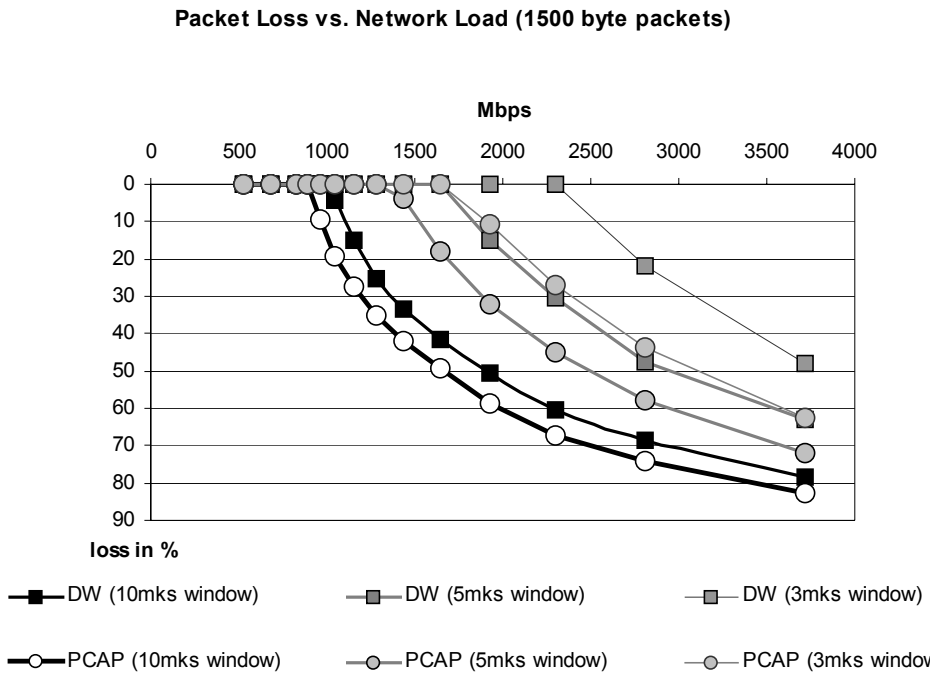

Fig.2. DirectWire vs. `libpcap` (see Appendix for actual numbers).

IP Defragmenter Component (FSS Defrag)

NCAP's IP defragmenter has been developed from the ground-up to satisfy specific requirements for a network sniffer. Multi-purpose IP defragmenters have been designed under the assumption that the traffic is legal and that fragmentation is rare. A network sniffer serving as a base for a packet inspection application has to work under heavy loads and be stable in the presence of DoS attacks. In addition to providing fast and robust packet reassembly, it has to detect and react to illegal fragments as soon as they arrive. This guarantees that the packet inspection application has low reaction latency and can withstand attacks specially designed to bring down 'standard' IP stacks. The IP Defragmenter for a network sniffer should also provide the following configurable options: minimum fragment size, maximum number of fragments per packet, maximum reassembled packet size and packet reassembly timeout. The IP Defragmenter should perform equally well on any fragment order.

To satisfy these goals, we developed a defragmenter with low per-fragment overhead. Focusing on per-fragment, rather than on per-packet overhead, is necessary to handle DoS attacks flooding the network with illegal and/or randomly overlapping fragments. Minimization of per-fragment overhead is achieved by lowering the cost of initialization/finalization phases and distributing the processing evenly between the fragments. As a result, invalid fragment streams are recognized early in the process and almost no time is spent on all the fragments following the first invalid one. Minimizing initialization/finalization time also had a positive effect on the defragmenter's performance on very short fragments, used in some DoS attacks targeted at security devices. This improvement is attributed to better utilization of buffering capabilities provided by the NIC and a packet capture library.

To measure the defragmenter's performance, we developed a fragment generation utility that can be programmed to generate fragments of different sizes showing a wide range of invalid fragmentation (sizes out-of-bounds, overlap, missing fragments). Combining the real-life and artificially generated traffic allowed us to identify and eliminate bottlenecks in the defragmenter's code and make optimal choices for internal parameters including timeouts and hash functions.

The charts below show the results of FSS Defrag benchmarks on a single-CPU 2GHz Intel Xeon server with 1GB RAM. On all fragment sizes and for both valid and invalid fragments, FSS Defrag demonstrated throughput above 1Gbps, reaching as high as 19Gbps on large invalid fragments. Note that on invalid fragments, the defragmenter's early invalid fragment detection pays off, leading to 6-fold performance gains. Also note that IP fragment order has no impact on FSS Defrag's performance.

For comparison, the same benchmarks were run on open-source Snort v2.0's defragmenter [RO99]. Snort v2.0's defragmenter scored consistently lower on all benchmarks (3 times slower on average). Low throughput on small fragments and invalid fragments is a serious bottleneck that may affect the ability of the whole packet inspection application to handle heavy loads and withstand DoS attacks on Gigabit networks.

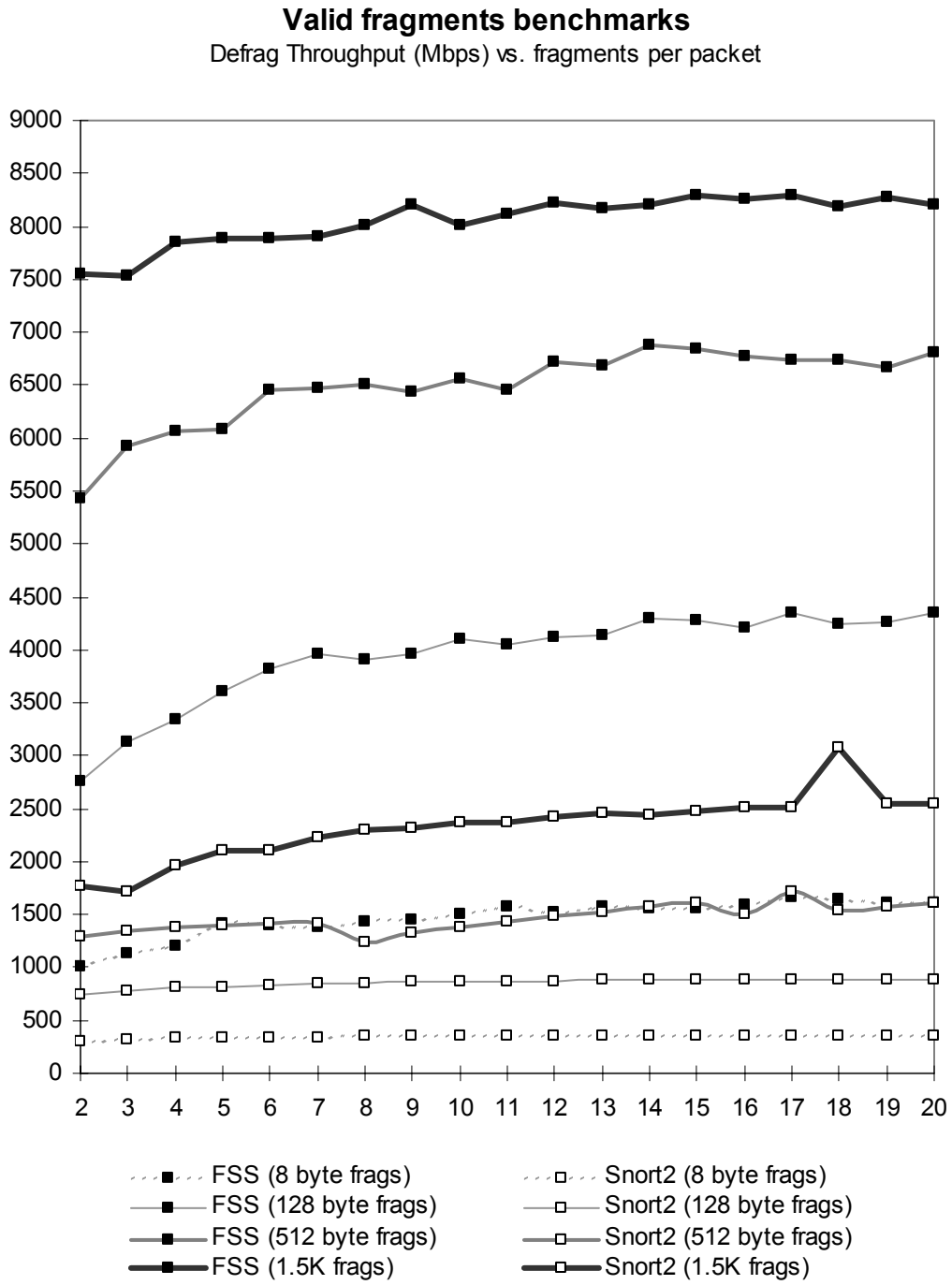

Fig.3. FSS Defrag's vs. Snort 2 defragmenter's performance on valid fragments.

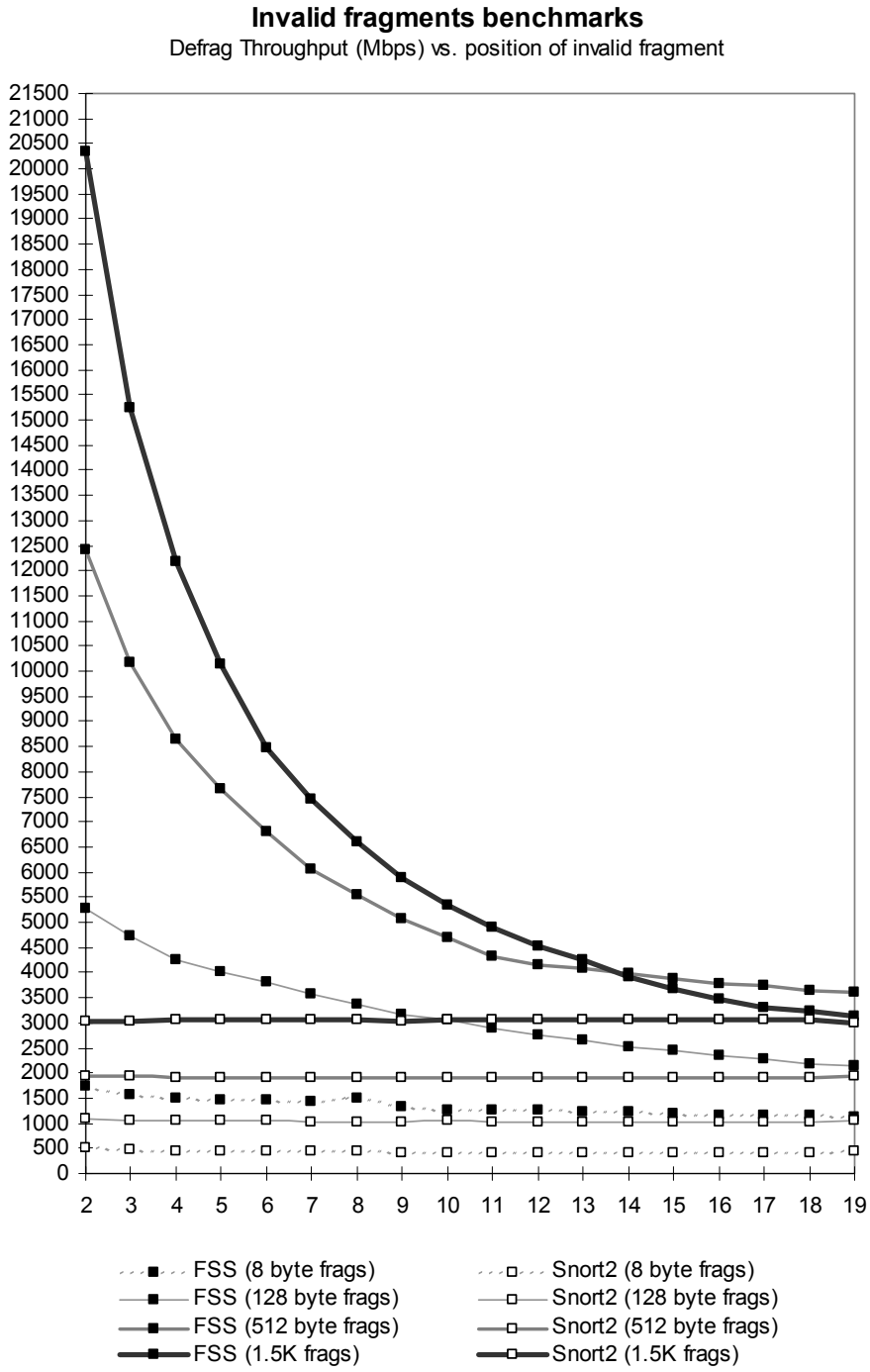

Fig.4. FSS Defrag's vs. Snort 2 defragmenter's performance on invalid fragments.

TCP Reassembler Component (FSS Reassembler)

NCAP's TCP reassembler tracks TCP sessions, keeps a list of information describing each open session, and concatenates packets belonging to a session so that the entire content of the client and server streams can be passed to upper levels of content inspection. Since NCAP's job is to provide multi-layer reassembly and content inspection, partial solutions like "deep" packet inspection, handling of only one side of a full-duplex connection, or reassembling arbitrary regions within the data stream to improve the chances of probabilistic detectors are not adequate.

Building a reassembler that satisfies these goals is not an easy task; we could not find any open-source project that includes a reassembler sophisticated enough to handle the intricacies of real-life packet streams. The problems faced by a packet inspector's reassembler are quite different from those of TCP/IP stacks: Packets seen by a sniffer NIC in promiscuous mode do not come in the expected order, so traditional state diagrams are of little use; standard timeouts need to be adjusted due to various delays introduced by taps and routers; there is not enough information in the packet stream to calculate internal states of the client and server, and so on.

A TCP stream reassembler for a packet sniffer should operate in a harsh environment of the modern network better than any 'standard' TCP/IP stack. The design should include TCP SYN flood protection, memory overload protection, etc. The design is further complicated by the fact that there are several different TCP/IP stack implementations with their own idiosyncrasies. And, the TCP/IP stream reassembler for a packet sniffer should be fast.

FSS Reassembler relies on NCAP packet capture layer (DirectWire), allowing it to watch any number of NICs simultaneously, interleaving data taken from different network streams. This functionality is crucial for reliable reassembly of both client and server data in Full-Duplex TCP stream and/or asymmetrically routed packets, because of each stream's dependency on the other for session control information.

FSS TCP reassembler can operate in 3 modes:

1. **Session tracking only**. This mode suits applications that only need to track TCP packet's direction (client->server or vice versa) and validity. In SMP setting, direction information is made available to recipient applications via a packet-level API.
2. **Session tracking and Partial TCP stream reassembly**. The initial parts of client-server conversations are collected in buffers limited by a configurable cutoff value. In SMP setting, the reassembled stream is made available to recipient applications via a stream-level API. This mode is designed for application logging initial segments of TCP sessions containing malicious packets such as Intrusion Detection systems (IDS). The default cutoff value is 8KB for a server part of the conversation and 8KB for the client part.
3. **Session tracking and Advanced TCP Stream reassembly**. Client-server conversation is collected into pre-allocated buffer chains. By default, up to 1600KB of every conversation is collected (800KB per direction). The size parameter is configurable and may be increased as needed. Reassembled streams are made available to recipient applications in SMP setting. A special algorithm watches for 'TCP Sequence skip' effects usual for long TCP sessions and distinguishes them from malicious and out-of-window packets. This mode delivers precise

stream reassembly; it is designed for content scanning applications, where the reassembled stream is further decomposed/decoded layer-by-layer and analyzed for content.

The FSS Reassembler's design is a descendant of our experimental reassembler based on simplified state transition diagrams reminiscent of Markov Networks. In the original design, each socket pair was mapped to a separate finite state automaton that tracks the conversation by switching from state to state based on the type of the incoming packet, its sequence number, and its timing relative to the most recent "base point" (the previous packet or the packet corresponding to a key transition). Since the reassembler has to deal with out-of-place packets sometimes (request packet coming after the reply packet), transitions cannot rely exclusively on packet type. At each state, the automaton keeps several "guesses" at what the real state of conversation might be, and chooses the "best" one on the basis of the incoming packet. Whichever "guess" can better predict the appearance of the packet is taken as the "best" characterization of the observed state of the conversation and new "guesses" are formed for the next step.

The current implementation of the FSS Reassembler is a simplified and streamlined modification of the original design based on extensive testing on real networks. Planning and transitions are hard-coded; parameters are fixed and inline-substituted to allow for code optimization. The resulting reassembler satisfies our requirements for robustness and quality. Its average throughput is 1.5-2 Gbps on normal traffic. It goes down to 250 Mbps on specially prepared SYN flood/DoS attacks, when the average packet length equals 80 bytes. Benchmarks below are run on single-CPU 2Ghz Intel Xeon box with 1GB RAM. For comparison, Snort 2.0 stream4 engine was tested under the same conditions2.

As the measurement results demonstrate, FSS Reassembler is fast enough to deal with fully saturated 1Gbps traffic. Combined with a separate packet-level inspection process running on a second CPU in SMP configuration or one or more separate TCP Stream decoders/analyzers, NCAP provides the basis for a wide range of Gigabit-capable network monitoring solutions. In contrast, publicly available open-source solutions like Snort's stream4 require cheats and tricks to help them to keep up with Gigabit traffic on commodity hardware. In Snort 2 example, this means restricted default settings (client only, several well-known ports) and artificial filters such as 'HTTP flow control' processor, ignoring as much as 80% of the traffic in default mode. Our experiments with Snort 2 settings made clear that stream4's throughput is a real bottleneck; allowing more packets in just changes the way Snort drops packets from 'predictable' to 'random'.

---

[2] In reassembly tests, Snort 2.0's TCP reassembler (stream4) does not provide functionality comparable to modes 2 and 3 of our reassembler. Stream4 reassembles only partial pseudo-random chunks of a TCP stream and performs packet-level inspection on these chunks. To provide an approximation of FSS reassembler functionality for throughput measurements, stream4's 'reinjection' mechanism has been disabled. Also, stream4 reassembles only the client side of the conversation by default and does it only for a predefined set of ports. This configuration has also been changed to reassemble both client and server streams on every port.

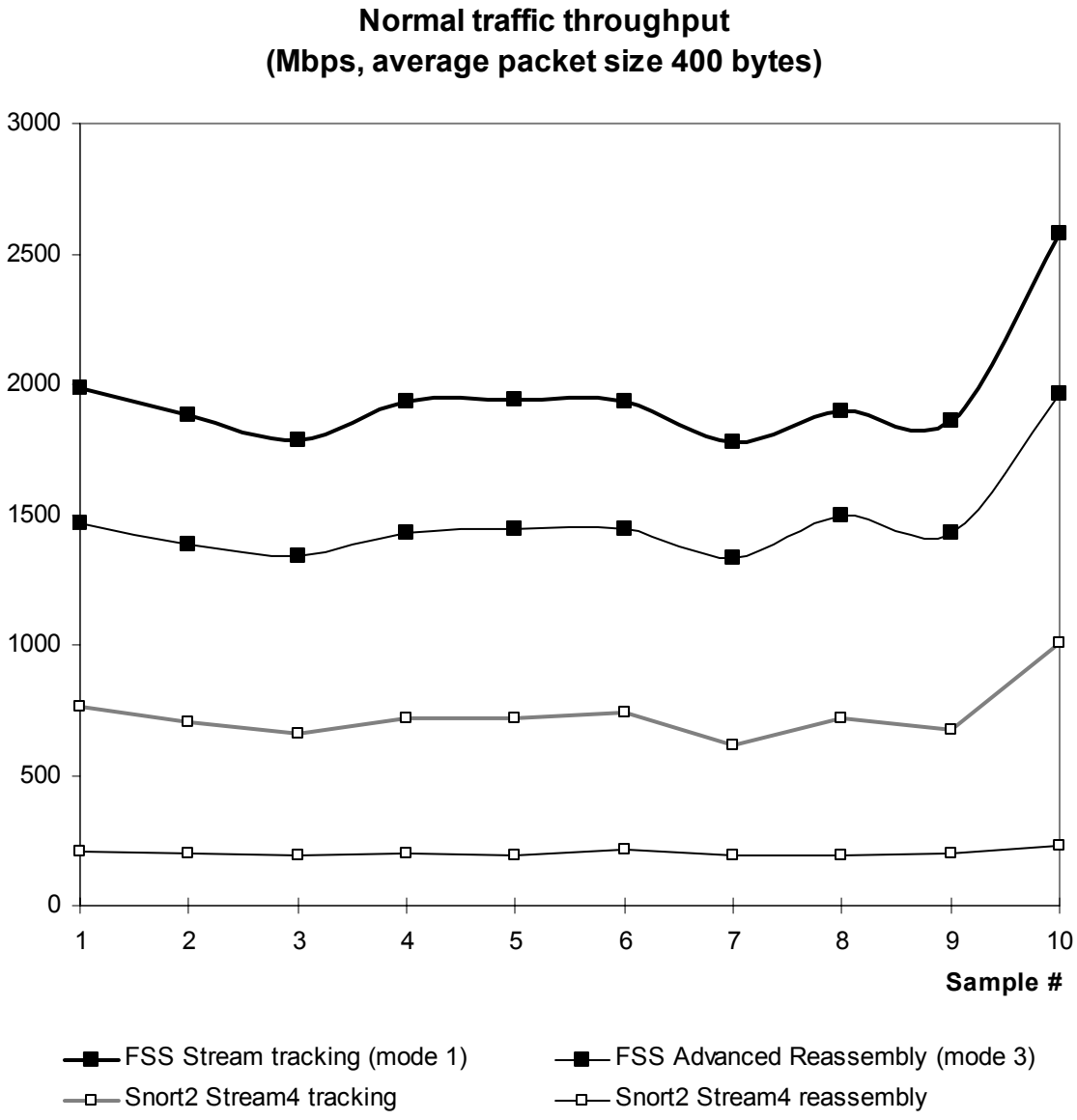

Fig.5. FSS Reassembler vs. Snort 2's stream4 performance on normal traffic.

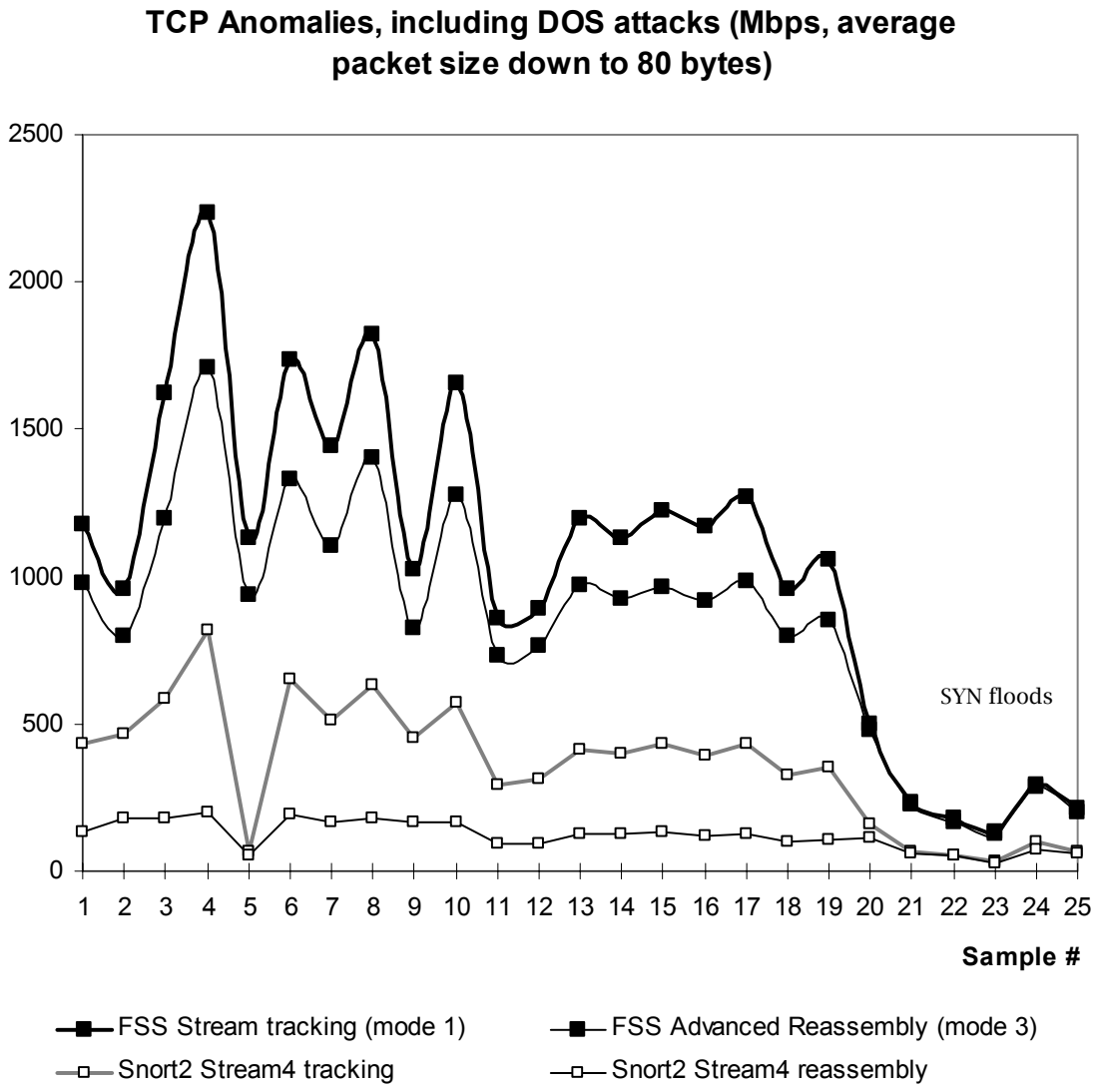

Fig.6. FSS Reassembler vs. Snort 2's stream4 performance on DoS attacks.

Appendix A

The table below shows the results of libpcap vs. DirectWire benchmark. For DirectWire, the relative time taken by system calls is shown together with percentage of lost packets.

Mbps	pkt/sec	libpcap %lost			DirectWire %lost (%syscalls)					
		10mks	5mks	3mks	10 mks		5 mks		3 mks	
525	45871	0.0	0.0	0.0	0.0	(17.5)	0.0	(17.5)	0.0	(17.5)
678	59259	0.0	0.0	0.0	0.0	(13.6)	0.0	(13.6)	0.0	(13.6)
823	71942	0.0	0.0	0.0	0.0	(9.7)	0.0	(11.2)	0.0	(11.2)
887	77519	0.1	0.0	0.0	0.0	(6.6)	0.0	(10.4)	0.0	(10.4)
962	84033	9.5	0.0	0.0	0.0	(0.0)	0.0	(9.6)	0.0	(9.6)
1048	91532	19.2	0.0	0.0	4.2	(0.0)	0.0	(8.8)	0.0	(8.8)
1153	100755	27.6	0.0	0.0	14.9	(0.0)	0.0	(8.0)	0.0	(8.0)
1282	112044	35.0	0.0	0.0	25.1	(0.0)	0.0	(7.2)	0.0	(7.2)
1440	125786	42.0	3.9	0.0	33.2	(0.0)	0.0	(4.3)	0.0	(6.4)
1647	143884	49.5	17.8	0.0	41.4	(0.0)	0.0	(0.0)	0.0	(5.6)
1923	168067	58.6	32.0	10.7	50.5	(0.0)	14.9	(0.0)	0.0	(4.1)
2300	201005	67.1	45.0	27.2	60.6	(0.0)	30.5	(0.0)	1.1	(0.0)
2808	245398	74.1	57.8	43.6	68.6	(0.0)	47.6	(0.0)	21.9	(0.0)
3722	325203	82.8	71.9	62.8	78.5	(0.0)	63.0	(0.0)	48.2	(0.0)

References

[LL99] MIT Lincoln Laboratory, 1998/1999 DARPA Off-Line Intrusion Detection Evaluation, http://www.ll.mit.edu/SST/ideval/

[CCTF] The Shmoo Group, Capture the Capture The Flag, http://www.shmoo.com/cctf/

[RO99] Martin Roesh. Snort: Lightweight intrusion detection for networks, in Proceedings of the 13th Systems Administration Conference. 1999, USENIX.

[SNORT] Snort.org, http://www.snort.org

[SNORTRAN] Sergei Egorov, Gene Savchuk, SNORTRAN: An Optimizing Compiler for Snort Rules, http://www.fidelissec.com/snortran