Conservative vs. Optimistic Parallelization of Stateful Network Intrusion Detection [*]

Derek L. Schuff, Yung Ryn Choe, and Vijay S. Pai

Purdue University
West Lafayette, IN 47907
{dschuff, yung, vpai}@purdue.edu

Abstract

This paper presents two approaches to parallelizing the Snort network intrusion detection system (NIDS). One scheme parallelizes NIDS processing conservatively across independent network flows, while the other optimistically achieves intra-flow parallelism by exploiting the observation that certain intra-flow dependences are uncommon and may be ignored under certain circumstances. Both schemes achieve average speedup over 2 on four cores, with an average throughput over 1 Gbps on 5 traces tested.

Categories and Subject Descriptors D.1.3 [*Programming Techniques*]: Parallel programming; C.2.0 [*Computer-Communication Networks*]: Security and Protection

General Terms Performance, Security, Design

Keywords Snort, Parallelization

Background. Network intrusion detection systems (NIDSes) run on a server at the edge of a LAN to identify and log Internet-based attacks against a local network, such as attempts at buffer overruns, cross-site scripting, and denial-of-service. Unlike firewalls, which work by shutting off external access to certain ports, NIDSes can monitor attacks on externally-exposed ports used for running network services. The most popular NIDS is the open-source Snort, which identifies intrusion attempts by comparing every inbound and outbound packet against a *ruleset* [4]. Rules in the set represent characteristics of known attacks, such as the protocol type, port number, packet size, packet content (both strings and regular expressions), and the position of the suspicious content. Each new type of attack leads to new rules, with rulesets growing rapidly. The most recently-released freely-available Snort rulesets have over 4000 rules.

The processing required by a network intrusion detection system such as Snort is quite high, since the system must decode the data, inspect the data according to the ruleset, and log intrusions. These requirements limit Snort to an average packet processing rate of about 557 Mbps on a modern host machine (2.2 GHz Opteron processor) — just over half of the Gigabit link-level bandwidth. Consequently, it is not possible to deploy Snort directly at a high-end network access point that requires a data rate of 1 Gbps or more. To address this problem, various companies and researchers have proposed solutions based on clustering [1, 3, 5]. Clustered NIDS potentially allows high scalability, but requires the use of an expensive load-balancing switch.

Contributions. This paper presents and evaluates methods to parallelize Snort. Although an NIDS like Snort receives its input packet-by-packet, an NIDS must aggregate distinct packets into TCP streams to prevent an attacker from disguising malicious communications by breaking the data up across several packets. Additionally, an NIDS must process later packets in a given communication based on its analysis of earlier packets. For example, if a given sequence of characters represents a possible attack in the body of an HTML document but may appear normally in an image, the NIDS should not trigger an alert if an earlier packet indicated that this data transfer was an image. Such constraints are incorporated into Snort as *stream reassembly* and *flowbits*, respectively. All TCP data is reassembled into streams, and about 36% of rules require flow tracking (90% of which relate to NetBIOS). Both of these systems require packets to be processed in-order; the TCP reassembler should see the packets in the same order that they will be seen by the destination host to mimic its TCP stack behavior, and flowbits requires ordering because a packet which checks the flowbits state depends on all previous packets which may have set that state. However, any ordering or data sharing between the processing of separate packets only applies to packets in the same IP flow (not only TCP streams, but also source/destination communication pairs in other protocols). Although this paper specifically targets Snort, the parallelization challenges and strategies discussed apply to any intrusion detection system that uses TCP stream reassembly to merge packets together for inspection or preserves other state across different packets from the same flow.

The strategies studied in this paper take different approaches to parallelizing the Snort NIDS: one conservative and one optimistic. The conservative scheme, called the *flow-concurrent* parallelization, exploits concurrency by parallelizing ruleset processing on a flow-by-flow basis. All packets are initially received in the order in which they appear on the network. The thread that receives them inspects the IP headers to determine the flow to which the packet belongs and then steers that packet to the appropriate processing thread based on whether or not that flow has already been assigned to a thread. Since each given flow is only processed by one thread at any given time, the dependences required for proper stream reassembly and flow tracking are maintained easily. This scheme works well if there are enough independent flows, but provides no benefits if all packets are from the same flow. The latter case is not a likely situation in a high-bandwidth edge NIDS, but does represent a limitation of this scheme.

The alternative parallelization is an optimistic variant on flow concurrency. This scheme starts with the basic flow-concurrent parallelization but then has the ability to dynamically reassign a flow to a different thread even while earlier packets of the flow are still

[*] This work is supported in part by the National Science Foundation under Grant Nos. CCF-0532448 and CNS-0532452.

PPoPP'07 March 14–17, 2007, San Jose, California, USA.
ACM 978-1-59593-602-8/07/0003.

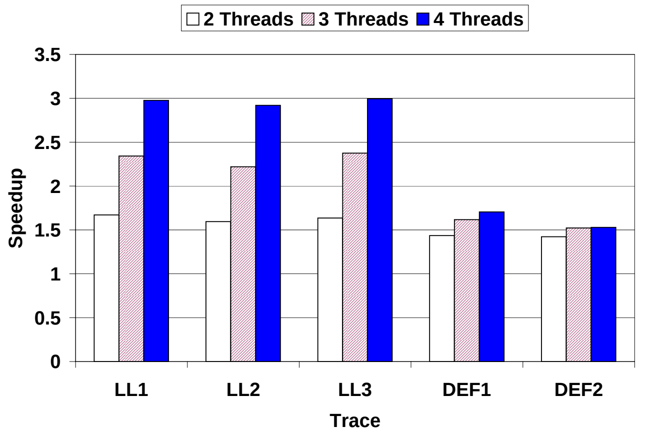

Figure 1. Parallel speedup for conservative parallelism

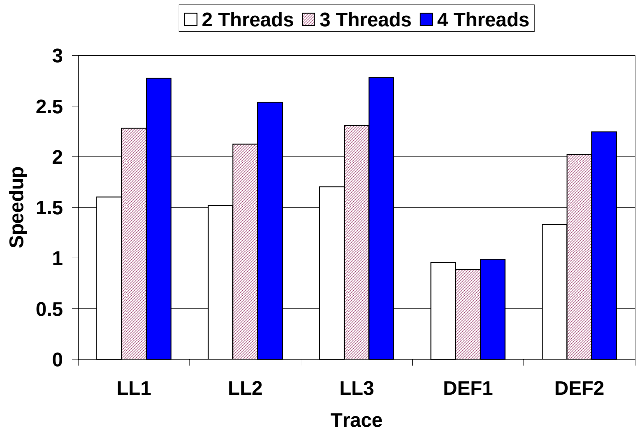

Figure 2. Parallel speedup for optimistic parallelism

being processed, potentially exploiting parallelism even with just one flow. This optimistic version relies on two key observations. First, TCP stream reassembly will still take place even if a stream is broken at some arbitrary point; reassembly is triggered by various flush conditions, one of which is a timeout. It is also easy to force additional flushes if needed for correctness. Consequently, any unprocessed earlier packets will still go through stream reassembly at their thread even though later packets are being reassembled and processed in another thread. Second, most packets do not match rules that use flowbits tracking, so enforcing ordering across all packets in a flow just to deal with a few problematic rules may be too restrictive. To precisely deal with the rules that do use flowbits, the optimistic system stalls processing in any packet that sets or checks flowbits unless it is the oldest packet in its flow. This condition is checked by adding per-flow reorder buffers. This system is optimistic in the sense that it reassigns threads under the assumption that the actual use of flowbits is uncommon, but is still conservative in maintaining correct ruleset processing without requiring rollbacks and redundant processing.

Experimental Results. The parallel NIDS is tested using 3 packet traces from the 1998-1999 DARPA intrusion detection evaluation at MIT Lincoln Lab and 2 from the Defcon 9 Capture the Flag contest [2, 7]. The Lincoln Lab traces (LL1–LL3) are simulations of large military networks. Because they were generated specifically for IDS testing, (including anomaly-based detection systems, which require realistic traffic models to be useful) the traces have a good collection of ordinary-looking traffic content and also contain attacks that were known at the time. The Defcon traces (DEF1, DEF2) are logs from a contest in which hackers attempt to attack and defend vulnerable systems. Consequently, these traces contain a huge amount of attacks and anomalous traffic, representing a sort of pathological case for intrusion detection systems. Both parallelizations use most of the same packet processing code as the current Snort (version 2.6), with minor modifications to make certain code segments re-entrant and well-synchronized using Pthreads. The resulting NIDS is evaluated on a 1U rack-mounted Sun Fire X4100 x86-64 Linux system with two dual-core Opteron processors (four processor cores in total).

Figure 1 shows the performance speedup for the conservative parallelization scheme, which achieves substantial speedups on all 3 LL traces, achieving 73–83% of the theoretical ideal linear speedup for 2–4 threads and achieving 2.9–3.0 speedup at 4 threads. All 3 traces see processing rates in excess of 1 Gbps with 4 threads; two of the traces achieve this rate with 3. The peak processing rate is 1.7 Gbps. The two factors that limit performance in these cases are a small amount of imbalance (occasionally more than one thread ran out of work at the same time) and synchronization and data transfer overheads (primarily in the form of cache-to-cache transfers between processors). In contrast, the Defcon traces have poor speedup because of their unusual composition. One in particular (DEF2) has almost no flow concurrency; in fact, for much of the trace there is only one active flow, so no flow-based parallelization scheme can hope for any significant improvement. This makes it a good candidate for improvement using the optimistic parallelization strategy. Figure 2 shows the speedup for this method, which indeed showed benefits over the conservative method; performance improved by about 50% for 4 threads to achieve a factor of 2.2 parallel speedup and a peak traffic rate over 2 Gbps. Since the Lincoln Lab traces already have good flow concurrency, optimistic flow reassignment provides no benefit for them; in fact, their performance is degraded by 7–13% compared to the conservative method because of the overhead of maintaining the reorder buffers and the extra synchronization required, limiting speedup to 2.8 on four cores.

Summary. Both schemes see an average traffic rate of just over 1 Gbps for the 5 traces, nearly doubling the performance of the serial version with only a slight increase in hardware cost and no increase in space. Either parallelization allows the benefits of high-performance intrusion-detection without relying either on higher clock frequencies (which are reaching a stage of diminishing returns) or costly and space-consuming load balancers. Additional detail can be found in the full paper [6].

References

[1] F5 Networks. Securing the Enterprise Perimeter – Using F5's BIG-IP System to Provide Comprehensive Application and Network Security. White paper, Oct. 2004.

[2] J. W. Haines, R. P. Lippmann, D. J. Fried, E. Tran, S. Boswell, and M. A. Zissman. 1999 DARPA Intrusion Detection System Evaluation: Design and Procedures. Technical Report 1062, MIT Lincoln Laboratory, 2001.

[3] Radware Inc. FireProof Security Activation. White paper, Sept. 2004.

[4] M. Roesch. Snort – Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, pages 229–238, 1999.

[5] L. Schaelicke, K. Wheeler, and C. Freeland. SPANIDS: A Scalable Network Intrusion Detection Loadbalancer. In *Proceedings of the 2nd Conference on Computing Frontiers*, pages 315–322, 2005.

[6] D. L. Schuff, Y. R. Choe, and V. S. Pai. Conservative vs. optimistic parallelization of stateful network intrusion detection. Technical report, Purdue University, 2007.

[7] Shmoo Group. Defcon 9 Capture the Flag Data, Sept. 2001.