Stress Testing Cluster Bro

Nicholas Weaver and Robin Sommer

Abstract

In previous work we have build a *NIDS cluster* as a scalable solution for realizing high-performance, stateful network intrusion detection on commodity hardware. Prototypes of our cluster, consisting of up to 10 PCs, are already operating at two major network sites.

In this work we are now gaging the scalability of our approach on the DETER testbed to identify potential performance bottlenecks when using larger number of nodes. Due to privacy concerns we can only use synthetic traffic for our evaluation and therefore start by building a new load-balancer element which can replicate small packet traces by several orders of magnitude. We then use this element to generate a network-load suitable for stress-testing the NIDS cluster from traffic captured on a single workstation.

While this approach cannot take into account many characteristics of site-specific live traffic, it still allows us to perform a first assessment of the cluster's underlying scalability hypothesis.

1 Introduction

We have currently modified the Bro Intrusion Detection System [2] to operate on a cluster of end host systems [5]. The goal is to take advantage of the significant potential parallelism present in the IDS problem [3] which observes that much of the work involved is per-flow independent, such as stream reassembly and application parsing. Additional work is per-host-pair independent, such as session structure-based analysis. Thus if we can take advantage of the many network streams present in actual traffic, we should be able to scale our rich IDS to very high data rates.

Our approach decomposes the analysis into multiple, independent processing elements running on commodity PC hardware. We use a single *manager* node which serves as a central point for alerts and responses (such as issuing ACL blocks to routers), one or two *proxy* systems which act to coordinate shared state between the IDSs, and a number of *sensor* nodes which interpret the actual traffic.

Additionally, a load balancer [4] is used to spread the traffic to the sensors based on a hash of the IP address pair. The load balancer ensures that all traffic associated with a pair of hosts always goes to the same sensor node, while distributing the entire stream across the group of sensors.

We are currently operating this IDS on two clusters of approximately 10 systems, one at a major university and one at a national laboratory. But we need to understand how the system will scale to larger clusters and how different traffic may stress a cluster differently from single systems. To further this, we need to build a framework for further testing.

Our framework for DETER consists of a Click module [1] to amplify a small test trace into a large volume of traffic, another Click module which implements our load balancer when read from a trace, and a series of scripts used to control experiments. Our initial testing used a highly stressful case composed entirely of HTTP traffic, which can cause a single-processor Bro to drop packets at rates barely exceeding 3600 packets per second.

In this particular case, we see indications of a *super* linear speedup when processing the trace on a cluster, as a cluster setup removes significant inter-flow state explosion effects. This speedup evaporates when the traffic volume is also scaled with cluster side.

2 Cluster Bro

Cluster Bro [5] begins with standard Bro with some modified policy scripts and roll assignments. One system is assigned to be the manager, the single node responsible for reporting all alerts and performing any response actions (such as activating ACL blocks). One or more systems are proxies. These proxies control shared data structures (such as for scan detection) and coordinate communication between cluster nodes.

The final Bro systems are sensor nodes. These sensor nodes process the individual communication between hosts. For more global analysis, such as scan detection, the sensor nodes communicate updates to the proxies. Likewise, alerts are reported to the manager node which performs aggregation.

Additionally, Cluster Bro requires a load balancer, either in software or hardware. The load balancer performs a simple hash on the (SRC IP, DST IP) tuple, and then sends the packet to the correct sensor by overwriting the destination MAC address.

3 Evaluation Strategy and Trace Selection

One significant problem in evaluating intrusion detection systems is evaluating on representative traces. Especially for systems which perform deep packet inspection, traces can be highly sensitive and can't easily be exported even to a different secure facility due to privacy and confidentiality concerns.

Since our work is attempting to find bottlenecks and performance artifacts, rather than comparing between multiple intrusion detections or site-specific evaluation, we can sometimes use synthetic traces. To create a synthetic trace, we begin with a small trace which triggers one or more known alerts, allowing us to verify correctness.

We then amplify this trace using a small Click module. The amplification consists of duplicating the packet stream by an amplification factor. Each new packet stream has the SRC and DST addresses hashed to new values. The resulting streams are interleaved, but on a slightly staggered basis to keep the streams out of phase, and written to a new file.

For performing our tests itself, we modified our Click load balancer to read from an amplified trace file. This reading uses a programmable rate limit expressed in packets per second. By creating a smooth packet flow, we eliminate small buffering errors and packet bursts from affecting end-hosts, causing this test to focus solely on system resources used to process the traffic.

We also created some custom shell scripts to manage the cluster, launch experiments, and collect results.

This technique is suitable for discovering and evaluating bottlenecks in our system, but it is *NOT* sufficient to know whether our system will scale without bottlenecks on real traffic. Without access to real traffic traces, with full packets, we can only state that we have removed known bottlenecks, not that our system will be without bottlenecks when deployed in the real world.

Our initial testing for this work used an HTTP trace. This trace, collected on a user's local system, is 1215 packets and 653930 bytes. It is composed almost entirely of HTTP sessions, which can be particularly stressful for Bro to analyze. This trace included two "attacks", suspicious URL requests for `../../../../etc/motd` and `../../../etc/passwd` and several benign requests. Thus we can easily check to see if the trace was correctly processed: count the total number of sensitive URL alarms and make sure it matches the amplification factor. If Bro had dropped packets, this adds to the alarm count through `content-gap` errors. Thus we can quickly verify that Bro correctly processed the packet stream.

The cluster itself used one manager node, one proxy node, and a variable number of sensor nodes. Additionally, we performed tests using a single Bro sensor without the cluster. We used an almost-complete set of Bro analyzers for the cluster, including scan detection, dynamic protocol detection, and HTTP request and reply analysis. Our particular trace highly stresses the HTTP reply analyzer, as removing this analyzer improves Bro's performance by over a factor of 2.

4 Evaluation Results and Discussion

For our evaluation on DETER, we used PC3080 systems as the nodes for the Bro cluster. These systems are 3 GHz, dual CPU pentium-4 based Xeon systems with hyperthreading enabled, all on a single, high bandwidth Nortel stack. For the traffic generating systems, we used the PC3000 systems which are of comparable performance and located on the same Nortel stack.[1]

Our initial test (Table 1) used a 500$\times$ multiplication of the HTTP stream and a single sending node. Going from a single Bro instance (without cluster extensions) to a two-sensor cluster showed an expected slowdown in processing per node. But we were initially puzzled by the super-linear speedup observed in the HTTP stresstest as the number of nodes increased. We believed this is due to interference effects or state exhaustion.

If we take the single stream and multiply it by a factor of 100 before sending it to a single processor Bro instance, Bro is able to handle 4900 pps before it starts dropping traffic. When we change it to a factor of 500, Bro is only able to handle 3600 pps. This appears due to interference effects as multiple HTTP analyzers cause state explosion effects or some linear processing as a function of the number of outstanding streams.

We confirmed this hypothesis by creating a 500x multiplied stream which acted to *concatenate* the original stream sequentially rather than interleaving the traffic. Bro was capable of handling this linearly-interleaved

[1]When performing high-bandwidth experiments on DETER, it is critical to ensure that important links don't cross between switch stacks.

Number of Sensor Nodes	Maximum Processing Rate	Speedup
Standalone Bro	3600 pps	
2 Sensors	6800 pps	1.8×
4 Sensors	14400 pps	4.0×
8 Sensors	35400 pps	9.8×

Table 1: The Scalable Performance of Cluster Bro on the 500x multiplied HTTP trace

Number of Sensor Nodes	Multiplication Factor	Maximum Processing Rate	Speedup
Standalone Bro	400	3800 pps	
2 Sensors	800	6600 pps	1.7×
4 Sensor	1600	13100 pps	3.4×
8 Sensors	3200	25500 pps	6.7×
16 Sensors	3200	55700 pps	14.7×
24 Sensors	3200	88400 pps	23.2×
31 Sensors	3200	84400 pps	22.2×

Table 2: The performance of Cluster Bro when the number of streams is scaled with the size of the cluster.

stream at only 3600 pps. Thus the slowdown is probably the result of state exhaustion or linear processing due to data structures being populated but not deleted when connections closed, rather than cache effects, as cache effects wouldn't cause a slowdown on the concatenated stream.

To further narrow down the cause, we performed a detailed profile on a single (no multiplication) and a 100× multiplied stream. In this profile, almost all the additional time is due to iterations over data structures. Thus it is clear that some non-constant-time data structure usage is what results in the slowdown for larger multiplication factors.

To further examine the superlinear effects, we created a second testcase. In this testcase, the duplication factor was scaled linearly with the number of nodes up to 8 nodes and the interleaving factor was increased.[2] Additionally, we shifted from one sending node to four sending nodes to increase the sending bandwidth.

When the workload per node is scaled up with the number of nodes, the superlinear speedup evaporates and a minor sublinear speedup is observed instead, as the cluster no longer benefits from having smaller data structures. When the workload is no longer scaled with cluster size, the performance again resumes its superlinear growth.

Until the 31 sensor case, the critical bottleneck was probably in the sensors themselves, as they would drop packets and therefore produce an incorrect analysis. Thus for moderate and smaller clusters, we appear to have good scalability. Even for the 24 sensor case, the cluster would work reliably up until the sensors started dropping packets.

The 31 sensor case, however, had all sensors reporting 0 drops, but the manager's alarm log did not include all the alarms. This suggests that we have hit a scalability limit in the communication or aggregation, either on the manager or the proxy, when we use this many sensor nodes. Additionally, the 31 sensor case is unreliable, sometimes it will succesfully log all alarms at a much higher data rate and sometimes would drop alarms. We reported the data rate which the cluster would reliably log all alarms. We plan on investigating this further.

5 Summary

We have built a framework for evaluating the scalability of the Bro cluster implementation using Deter and amplified synthetic traces. We have shown that if the total amount of work remains constant, for a particular case we can even see a super-linear speedup as resource exhaustion effects, but this speedup is reduced to sublinear when we scale the workload with cluster size.

It is important to remember, however, that because we are using a synthetic trace, we can only use this to discover and evaluate potential bottlenecks. This offers no guarantee that operational traffic won't display different performance artifacts. It politically difficult to generate and access full-packet traces from production networks, but such traces are essential if we wish to validate our system for operational environments.

[2]We ran out of disk space on the click sending nodes, preventing us from creating larger traces.

6 Acknowledgments

This work is sponsored by the US Department of Energy Office of Science and the National Science foundation under grants STI-0334088 and NRT-0335290. Special thanks to Vern Paxson, Brian Tierney, Matthias Vallentin, and Mark Dedlow for help on the Bro cluster.

Parts of this work were produced under the auspices of the Institute for Information Infrastructure Protection (I3P) research program. The I3P is managed by Dartmouth College, and supported under Award number 2003-TK-TX-0003 from the U.S. Department of Homeland Security, Science and Technology Directorate. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security, the Science and Technology Directorate, the I3P, or Dartmouth College.

References

[1] R. Morris, E. Kohler, J. Jannotti, and M. Frans Kaashoek. The click modular router. In *Symposium on Operating Systems Principles*, pages 217–231, 1999.

[2] V. Paxson. Bro: A system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[3] V. Paxson, R. Sommer, and N. Weaver. An architecture for exploiting multi-core processors to parallelize network intrusion prevention. In *Proceedings of the IEEE Sarnoff Symposium*, May 2007.

[4] Lambert Schaelicke, Kyle Wheeler, and Curt Freeland. Spanids: a scalable network intrusion detection loadbalancer. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 315–322, New York, NY, USA, 2005. ACM Press.

[5] M. Vallentin, R. Sommer, J. Lee, C. Leres, V. Paxson, and B. Tierney. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *RAID 2007 (to appear)*.