Predicting the Resource Consumption of Network Intrusion Detection Systems

Holger Dreger
Siemens AG, Corporate Technology
holger.dreger@siemens.com

Anja Feldmann
Deutsche Telekom Labs/TU-Berlin
anja@net.t-labs.tu-berlin.de

Vern Paxson
ICSI/LBNL
vern@icir.org

Robin Sommer
LBNL/ICSI
robin@icir.org

ABSTRACT

When installing network intrusion detection systems (NIDSs), operators are faced with a large number of parameters and analysis options for tuning trade-offs between detection accuracy versus resource requirements. In this work we set out to assist this process by understanding and predicting the CPU and memory consumption of such systems.

Categories and Subject Descriptors: I.6.5: Model Development.

General Terms: Measurement, Security.

Keywords: NIDS, Performance Model.

1. INTRODUCTION

Operators of network intrusion detection systems (NIDSs) face significant challenges in understanding how to best configure and provision their systems. The difficulties arise from the need to understand the relationship between the wide range of analyses and tuning parameters provided by modern NIDSs, and the resources required by different combinations of these.

In addition, a NIDS must operate in compliance with *soft real-time* constraints, to issue timely alerts or blocking-directives for intrusion prevention. Such operation differs from *hard* real-time in that the consequences of the NIDS failing to "keep up" with the rate of arriving traffic is not catastrophe, but rather *degraded performance* in terms of some traffic escaping analysis ("drops") or experiencing slower throughput (for intrusion prevention systems that only forward traffic after the NIDS has inspected it).

Soft real-time operation has two significant implications in terms of predicting the resource consumption of NIDSs. First, because NIDSs do not operate in hard real-time, we eschew performance evaluation techniques that aim to prove compliance of the system with rigorous deadlines (e.g., assuring that it spends no more than T microseconds on any given packet). Given the very wide range of per-packet analysis cost in a modern NIDS, such techniques would severely reduce our estimate of the performance a NIDS can provide in an operational context. Second, soft real-time operation also means that we cannot rely upon techniques that predict a system's performance solely in terms of aggregate CPU and memory consumption: We must also pay attention to *surges* in CPU load, for understanding the degree to which in a given environment the system would experience degraded performance (packet drops or slower forwarding).

In our experience the operational deployment of a NIDS is often a trial-and-error process, for which it can take weeks to converge on an apt, stable configuration. In our work[1] we set out to assist operators with understanding the resource consumption trade-offs available to them when operating a NIDS that provides a large number of tuning parameters and analysis options. In this context, a particular difficulty regards how resource consumption intimately relates to the specifics of the network's traffic—such as its application mix and its changes over time—as well as the internals of the particular NIDS in consideration.

SIGMETRICS'08, June 2–6, 2008, Annapolis, Maryland, USA.
ACM 978-1-60558-005-0/08/06.

2. METHODOLOGY

We begin towards our goal by devising a general NIDS resource model to capture the ways in which CPU and memory usage scale with changes in network traffic. We then use this model to predict the resource demands of different analysis depths for specific environments. Finally, we develop an approach to derive site-specific NIDS configurations that maximize the depth of analysis given predefined resource constraints.

2.1 Modeling NIDS Resource Usage

When modeling the resource consumption of a NIDS, our main hypothesis concerns *orthogonal decomposition*: i.e., the major subcomponents of a NIDS are sufficiently independent that we can analyze them in isolation and then extrapolate aggregate behavior as the composition of the contributions from these individual components. In a different dimension, we explore the degree to which we can estimate the system's overall resource requirements by extrapolating from fine-grain sampled network traffic coupled with coarse-grain traffic summaries. Even though this simplification disregards many of the internals of a NIDS's operation, we experimentally assess our claim and find that in general it holds for the quite-complex Bro NIDS [3]. While we do observe some complications due to the unusually high degree (compared to other NIDS) of flexibility that the Bro system provides to the operator, our study shows that in practice these issues do not tend to be of major concern.

If orthogonal decomposition holds, we can systematically analyze a NIDS' resource consumption by capturing the performance of each subcomponent individually, and then estimating the aggregate resource requirements as the sum of the individual requirements. We partition our analysis along two axes: type of analysis, and proportion of connections within each class of traffic. We find that the demands of many components scale directly with the prevalence of a given class of connections within the aggregate traffic stream. This observation allows us to accurately estimate resource consumption by characterizing a site's traffic "mix." Since such mixes change over time, however, it is crucial to consider both short-term and long term fluctuations.

[1]For a more detailed discussion see [1].

We stress that, by design, our model does *not* incorporate a notion of detection quality, as that cannot reasonably be predicted from past traffic. We focus on identifying the types of analyses which are *feasible* under given resource constraints. With this information the operator can assess which option promises the largest gain for the site in terms of operational benefit, considering the site's security policy and threat model.

2.2 Example NIDS Resource Usage

To assess our approach of modeling a NIDS's resource demands as the sum of the requirements of its individual components, and scaling linearly with the number of connections, we examine an example NIDS. Among the two predominant open-source NIDSs, Snort [4] and Bro [3], we chose to examine Bro for two reasons: *(i)* Bro provides a superset of Snort's functionality, since it includes both a signature-matching engine and an application-analysis scripting language; and *(ii)* it provides extensive, fine-grained instrumentation of its internal resource consumption; see [2] for the specifics of how the system measures CPU and memory consumption in real-time. Snort does not provide similar capabilities.

For our analysis we captured a 24-hour full-packet trace at the border router of a major university campus. This facility provides 10 Gbps upstream capacity to roughly 50,000 hosts at two major universities, along with additional research institutes, totaling 2-4 TB a day. The trace encompasses 3.2 TB of data in 6.3 billion packets and 137 million connections. 76% of all packets are TCP. We refer to this trace as `campus-full`.

We first verify that the NIDS' resource usage generally scales linearly with the number of connections on the monitored network link. Then we assess our hypothesis that we can consider the resource consumption of the NIDS's components as independent of one another. We find that for the Bro NIDS, both assumptions generally hold for CPU and memory. Doing so can however overestimate the memory demand for some components.

2.3 Resource Prediction

After validating that we can often compose NIDS resource usage in terms of per-component and per-connection scaling, we then employ these observations to expose tradeoffs for different configurations for operating a NIDS in a new network environment (or for improving the configuration in an existing environment). In addition, we can estimate when, for a given configuration and expectation of traffic growth, the NIDS's current computational resources will no longer suffice.

We start by devising a methodology for finding a suitable configuration based on a snapshot of an environment's network traffic. Based on this methodology, we implemented an automatic configuration tool, `nidsconf`, for the Bro NIDS. The tool analyzes a 20-minute network packet trace, sampled at a per-connection granularity, from which `nidsconf` determines a set of Bro configurations, including sets of feasible analyzers and suitable connection timeouts. These configurations enable Bro to process the network's traffic within user-defined limits for CPU and memory.

Now that we can identify appropriate configurations based on a detailed packet-level trace, we turn to estimating the long-term performance of such a configuration. Such extrapolation is crucial before running a NIDS operationally, as network traffic tends to exhibit strong time-of-day and day-of-week effects. Thus, a configuration suitable for a short snapshot may still overload the system at another time, or unnecessarily forsake some types of analysis during less busy times.

For this purpose we require long-term, coarser-grained logs of connection information as an abstraction of the network's traffic.

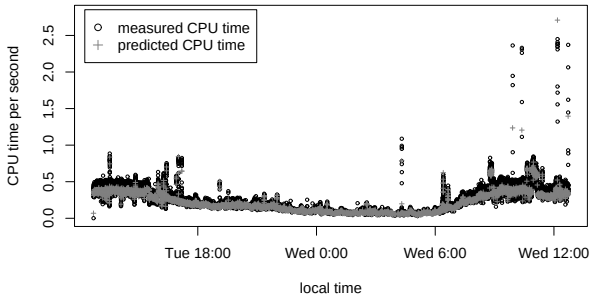

Figure 1: Measured CPU time vs. predicted CPU time.

Such logs can, for example, come from NetFlow data, from traffic traces with tools such as tcpreduce [6], or from the NIDS itself (Bro generates such summaries as part of its generic connection analysis). Such connection-level logs are much smaller than full packet traces (e.g., $\ll 1\%$ of the volume), and thus easier to collect and handle. Indeed, some sites already gather them on a routine basis to facilitate traffic engineering or forensic analysis.

Figure 1 shows the per-second consumption for processing the `campus-full` trace: projected by using connection logs plus the analysis results on a 20-minute packet trace, as predicted by `nidsconf` (crosses), versus actual per-second CPU consumption exhibited by running a basic Bro configuration (circles). Overall, the predicted CPU time matches the variations in the measured CPU time quite closely.

3. CONCLUSION

In this work we set out to understand and predict the resource requirements of network intrusion detection systems. We develop a methodology to *automatically* derive NIDS configurations that maximize the systems' detection capabilities while keeping the resource load feasible, and sketch a tool we built that derives realistic configurations for the open-source Bro NIDS.

An interesting avenue for future work is predicting what kind of equipment a network link requires to be *fully* analyzed, rather than estimating the amount of analysis one can afford with a given set of resources. This is not as straight-forward as it might initially seem because—as for example [5] demonstrates for the Snort and Bro NIDSs—performance can vary greatly between seemingly quite similar hardware platforms. Such effects make it tricky to model the actual benefit that, e.g., a CPU upgrade would provide.

4. REFERENCES

[1] H. Dreger. *Operational Network Intrusion Detection: Resource-Analysis Tradeoffs*. PhD thesis, TU München, 2007.

[2] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer. Operational Experiences with High-Volume Network Intrusion Detection. In *ACM Computer and Communications Security*, 2004.

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

[4] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proc. Systems Administration Conference*, 1999.

[5] R. Sommer and V. Paxson. Enhancing Byte-Level Network Intrusion Detection Signatures with Context. In *ACM Computer and Communications Security*, 2003.

[6] `tcp-reduce`. `http://ita.ee.lbl.gov/html/contrib/tcp-reduce.html`.