Tradeoffs for Web Communications Fast Analysis[1]

Olivier Paul, Jean-Etienne Kiba

GET/INT LOR Department, 9 rue Charles Fourier, 91011 Evry cedex - France
Olivier.Paul@int-evry.fr, Jean-Etienne.Kiba@int-evry.fr

Abstract. In this paper, we explore a novel approach to perform a probabilistic fast analysis of web communications. Instead of relying on pattern matching algorithms, we look at simple network and transport level parameters and try to infer what happens at the application level. Our approach provides the ability to perform a trade-off between analysis speed and precision that could prove useful for some traffic analysis applications.

1. Introduction

The appearance of new threats (e.g. worms, DDOS attacks) has led network operators to provide intrusion detection services to their customers. In this paper we consider one of the challenges implied by this activity; the ability to monitor communications within operator networks. This task can be considered challenging for several reasons (limited traffic analysis abilities, large amounts of traffic, limited ability to introduce new mechanisms). We focus on HTTP based communications because they constitute one of the largest aggregate of packets on the Internet. The goal of this paper is to present techniques that would enable such communications to be analyzed in the network while complying with the aforementioned limitations.

2. Measurement Information

Our goal is to check whether network or transport level information could be used to infer application level operations. In a first part we examine how protocols might render this operation difficult. HTTP [1] exchanges can be viewed at several levels. At the lowest level, the HTTP protocol is based on a request-response protocol where each request attempts to perform an HTTP operation on an object at the server. We later call this level micro-session level. Information in HTTP 1.1 messages is organized into information elements called headers. Although HTTP 1.1 defines more than 40 different headers, requests and responses usually only use a few them.

HTTP 1.1 provides the ability for web clients and servers to multiplex several HTTP request-responses exchanges over a single TCP connection. Among persistent connections we can additionally distinguish between connections using pipelined

[1] This work is funded by IST DIADEM Firewall and GET DDOS projects.

requests and regular connections. Pipelined connections are used by the client to perform several requests without waiting for an answer from the server. This ability is however limited by the structure of html documents. Therefore micro-sessions can be distinguished at the network level by either looking at:

- Connections set-up and ending in the case of non persistent connections.
- Request-Response session patterns [3] in the case of non-pipelined persistent connections. These patterns can be found by observing TCP sequence numbers.
- Request-Response session patterns in the case of persistent pipelined connections. However only the first micro-session can be distinguished from other exchanges.

An interesting question is thus whether pipelined, persistent connections are supported in the real life. [3] shows that most browsers are either unable to use persistent-pipelined connections or configured by default to avoid using them.

3. Method and Objects size inference

Our assumption is that objects sizes can be inferred from network or transport level measurements. Several factors can play a role in making this process more difficult. For example at the transport level, measurement information includes HTTP headers.

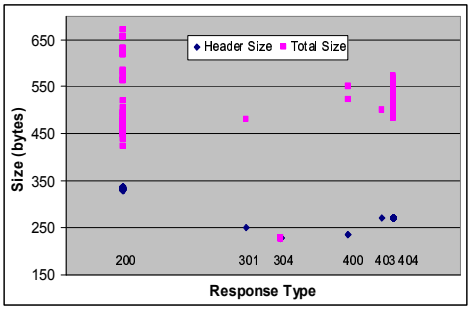

Fig. 1. Header and total sizes for various types of responses.

Figure 1 provides the relation between header sizes, types of response and total sizes in the case of our web server. These values where obtained by capturing responses packets from the server over several hours. Six types of responses (identified by code numbers) were captured.

Figure 1 shows that 200 (OK) responses can be distinguished from other responses by looking at the total size. Some 200 responses have a size that collides with other types of responses. However objects associated with these responses constitute less than 1% of existing objects. 304 (not modified) responses can also be distinguished from other responses by looking at the total size.

Additionally, tests show that persistent connections include additional HTTP headers. These headers have a fixed size (57 bytes). As a result knowing whether a connection is persistent is sufficient to deduce the influence of the persistence on the HTTP header size. This knowledge can be obtained by looking at multiple connections establishment-teardown over short periods of time. **Table 1** provides the relation between response sizes and codes.

Table 1. Response code classification using response size for persistent connections.

Result	Response Size
200	RS >550 or 250<RS<460
304	240<RS <250
301, 400, 403, 404	460<RS<550

Using a similar methodology, we define a set of classification criterion in order to infer the method used in HTTP requests.

Figure 1 shows that 200 responses can carry HTTP headers whose size are not fixed. As a result using an average HTTP header size value to estimate objects sizes in the case of GET requests can lead us erroneous results. By looking more closely at HTTP headers we can classify header fields according to their behavior:

- Some headers values never change (e.g. response code, server id, accept range).
- Some header values change but have a fixed size (e.g. last modified and date).
- Some header sizes change depending on the document (e.g. content type, size).

As a result for a given object, the response size should remain constant. This means that by keeping the relation between response sizes and object sizes, we can get an exact estimate of objects sizes.

4. URI inference

For URI Inference, our goal is to use parameters such as the object size, the date and time at witch a request was performed or the IP address of the requesting client. The relation between these parameters can be found in log files on the web server.

The model we selected to perform inference operations is a Bayesian network. Bayesian networks are graphical models that can be used to represent causal relationships between variables. A Bayesian network is usually defined as an acyclic directed graph $G, G = (V, E)$, where V is a set of nodes and E the set of vertexes, a finite probability set (Ω, Z, P) and a set of variables defined on (Ω, Z, P), so that :

$$P(V_1, V_2, \mathrm{K}\ , V_n) = \prod_{i=1}^{n} P(V_i | C(V_i)) \quad (1)$$

where $C(V_i)$, is the set of causes for V_i in the graph.

The inference in a causal network consists in propagating unquestionable information within the network, in order to deduce how beliefs concerning the other nodes are modified. This propagation is related to causal relations between nodes.

In order to limit the resources used by our model, a first step was to aggregate possible parameter values. IP addresses were aggregated into country codes and date and time were also aggregated in some way. As the cost of inference in a Bayesian network increases exponentially with the number of variables we also evaluated the ability for each parameter (size, country codes, time, date) to explain URIs. This led us to the selection of country code and size variables (Bayesian network in Figure 3).

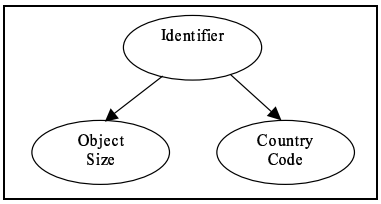

Fig. 2. Resulting Bayesian network.

5. Implementation and tests

A traffic analyzer was implemented as an extension to IPFilter [4] to capture HTTP sessions. Sessions are delimited as specified in section 2. When a session ends, the corresponding information (size, IP addresses, time, and date) is handed to a user space process and stored in a file.

Validation tests were performed using our departmental web server. This web server includes roughly 15k objects and receives 10k requests a day. Models were built using a one month log file including 309k requests. The validation was performed by simulating requests to the server using the same log file. Results are presented in **Table 2**. Among requests, only requests with a "200" response codes were used for method inference. Only inferred "GET", "200" requests were used for object size and URI inference.

Table 2. Ability to predict correct parameter values.

Parameter	Requests considered	Correct Responses	% Correct
Response Code	300986	288568	96
Method	228189	208347	92
Object Size	154289	150505	98
URI	154289	135212	88

An implementation of our inference process was performed in C in order to test its performance. The implementation was performed on FreeBSD using a 2.4Ghz Pentium Xeon (512KBytes cache, 1GBytes RAM). Results show that our inference process should be able to analyze roughly 1M requests per second using a 2.5Mb model. Assuming standard Internet traffic this would allow us to treat a 20Gb/s full duplex link.

References

1. R. Fielding and al. HTTP 1.1, RFC 2616. Internet Engineering Task Force, June 1999.
2. B. Krishnamurthy and al. PRO-COW: Protocol Compliance on the Web, USITS '01, 2001.
3. F. Donelson Smith and al., What TCP/IP protocol headers can Tell Us About the Web, ACM SIGMETRICS/Performance 2001, June 2001.
4. Darren Reed, IPFilter v 4.1.3, available at coombs.anu.edu.au/~avalon/, 2004.