Early Recognition of Encrypted Applications

Laurent Bernaille, Renata Teixeira

HAL Id: hal-01097556

https://inria.hal.science/hal-01097556

Submitted on 19 Dec 2014

Early Recognition of Encrypted Applications

Laurent Bernaille and Renata Teixeira

Université Pierre et Marie Curie - LIP6-CNRS
Paris, France

Abstract. Most tools to recognize the application associated with network connections use well-known signatures as basis for their classification. This approach is very effective in enterprise and campus networks to pinpoint forbidden applications (peer to peer, for instance) or security threats. However, it is easy to use encryption to evade these mechanisms. In particular, Secure Sockets Layer (SSL) libraries such as OpenSSL are widely available and can easily be used to encrypt any type of traffic. In this paper, we propose a method to detect applications in SSL encrypted connections. Our method uses only the size of the first few packets of an SSL connection to recognize the application, which enables an early classification. We test our method on packet traces collected on two campus networks and on manually-encrypted traces. Our results show that we are able to recognize the application in an SSL connection with more than 85% accuracy.

1   Introduction

Accurate classification of traffic flows is an essential step for network administrators to detect security threats or forbidden applications. This detection has to happen as early as possible, so that administrators can take appropriate actions to block or control the problem. Given that the simple inspection of IANA-assigned port numbers is no longer a reliable mechanism for classifying applications [1], many campus or enterprise networks now use content-based mechanisms. These mechanisms search the content of packets for well-known application signatures [2–4]. Although very effective and accurate, content-based mechanisms are easy to evade by using encryption. To make matters worse, Secure Sockets Layer (SSL), which can easily be used to encrypt any application communication, is widely available. In this paper, we design a classifier able to detect the underlying application in encrypted SSL connections.

Before constructing a classifier for encrypted traffic, we characterize the use of SSL on two campus networks by studying packet traces. This characterization sheds light on the prevalence of SSL in today's networks, the SSL versions in use, and the types of application that use encryption. We see an increase in the use of SSL, which coincides with the surge of new applications that use SSL. For instance, Bittorrent clients (Azureus and uTorrent) now offer SSL encryption as a way to hide from content-based blocking of peer-to-peer applications. These factors indicate that SSL usage will continue to increase.

Our classifier for encrypted traffic builds on two observations. First, SSL does not modify significantly the the number of packets, their size, and their inter-arrival time [5, 6]. Second, TCP connections can be classified based on flow-level information such

as duration, number of packets and mean inter-arrival time [7–11]. We extend the classifier presented in [12, 13] to identify applications in encrypted SSL connections. This classifier, which we refer to as *early identification*, identifies the application associated with a TCP connection using only the first few packets in the connection. Our method to identify encrypted traffic involves two steps. First, we detect SSL traffic. Then, we apply early identification to the first encrypted application packets to recognize the underlying application. With this method, we recognize encrypted applications with more than 85% accuracy.

After comparing our traffic classification method with previous work in Section 2, we present the packet traces we used to train our classifier and to evaluate our mechanism in Section 3. Section 4, briefly introduces SSL, describes a content-based approach to identify SSL connections, and characterizes SSL usage in our traces. Section 5 presents our classification mechanism and Section 6 evaluates it. We conclude in Section 7 with a summary of our contributions and a discussion of future directions.

2   Related Work

Although any flow-level classifier [7–11, 13, 14] can potentially identify encrypted applications, this paper is the first to design an application-recognition mechanism for encrypted traffic and test it on real SSL traffic. We choose to extend the classifier presented in [12, 13], because it can recognize the application associated with a TCP connection early. All of the other classifiers rely on statistics on the whole connection, which prevents them from being used online. An alternative method [15] uses connectivity patterns for each host in the network. This approach could work for encrypted traffic, but its goal is different from ours: it finds services associated to hosts, whereas we classify single TCP connections.

The methods presented in [14, 16] are the only ones that share our goal of classifying encrypted traffic. As other flow-level classifiers, however, this mechanism also requires *all* packets in the connection before classifying it. In addition, we perform a measurement-based study that first characterizes the usage of SSL in two campus networks and then evaluates our mechanism against real SSL connections, whereas their classifier was only evaluated under simulated encrypted traffic.

3   Packet traces with encrypted traffic

Our study relies on two sets of data: packet traces collected at the edge of two campus networks and manually-generated traces. Packet traces allow us to characterize the usage of SSL in operational networks, when manually-generated traces help us validate our classification mechanism.

We used two one-hour traces collected at the edge of the Paris 6 network, in 2004 and in 2006 (referred to as P6-2004 and P6-2006, respectively) and a packet trace collected at the edge of the UMass campus in 2005. Both traces collected on the Paris 6 network contain packet payload, which allows the identification of SSL versions and options. The UMass trace only captures 58 bytes for each packet, because of privacy and security reasons. Fortunately, many packets do not contain any options (IP or TCP)

and, without options, 58 bytes capture four bytes of TCP payload, which is enough to accurately identify SSL connections and the versions they use. In the UMass trace, 50% of connections on SSL standard ports had 4 bytes of TCP payload.

To validate our method, we needed a ground truth, or SSL connections for which the underlying application is known. We use two methods to obtain this ground truth. First, we filter the Paris 6 traces to keep only connections directed to well known HTTPS and POP3S servers in the university. To extend our validation to other types of traffic, we also manually encrypted traces consisting of other applications. We replay packet traces over an encrypted tunnel and capture the resulting connections. We use three machines, say A, B and a controller. Machine A represents the server of the TCP connection and machine B the client. First, we establish a tunnel between A and B using stunnel[1]. Then, the controller parses an existing packet trace. For each packet in a TCP connection, if the packet was sent from the TCP server, the controller asks A to send the packet to B over the encrypted tunnel, otherwise it asks B to send it.

4   Analysis of SSL traffic

This section presents a brief background on SSL and a content-based method to identify SSL connections in packet traces. We end with a characterization of SSL in our traces.

4.1   Description of SSL

Secure Sockets Layer (SSL) provides authentication and encryption to TCP connections. SSL runs between the transport layer (usually TCP) and the application layer. Three different versions of SSL have been developed: SSLv2 [17], SSLv3.0 [18] and TLS [19]. As SSL version 2 (SSLv2) presents several security flaws, its use is now strongly discouraged. Its follow-up version is SSL version 3 (SSLv3.0). The latest version, Transport Layer Security (TLS or SSLv3.1), is the standard specified by the Internet Engineering Task Force (IETF) and is similar to SSLv3.0. The differences between SSLv3.0 and TLS are minor (for instance, types of ciphers supported, pseudo-random functions and padding policies) and do not affect the handshake or the packet sizes. Therefore, we use SSLv3 to refer to both protocols.

Figure 1 presents an SSL handshake for SSLv2. The exact messages exchanged differ for SSLv3, but the main steps remain the same. First, the client and the server negotiate the SSL version they are going to use and choose an encryption algorithm. Second, the server authenticates itself to the client (the client might do likewise if required by the server) and both peers negotiate an encryption key. Finally, they terminate the handshake, and can start exchanging application data over the encrypted channel.

4.2   Identifying SSL connections

Many applications can be detected based on a well-known signature (for instance "GET /index.html HTTP/1.1" for HTTP traffic). Unfortunately, there is no such pattern for SSL. The server Hello packet sets the configuration of the connection (in particular, version and encryption algorithm). Therefore, we can analyze this packet to verify if

---

[1] http://www.stunnel.org
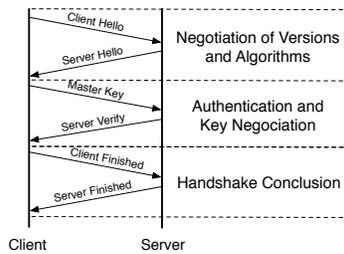
Fig. 1. Example of SSL handshake (SSLv2)

a connection is using SSL and determine the version. SSL connections rely on SSL records transported using TCP. Each SSL record starts with a SSL header that is sent unencrypted, and is followed by either SSL configuration options (during the SSL handshake) or by the encrypted application payload. The first two bits in SSLv2 headers are always 1 and 0, the following 14 bits contain the size of the SSL record and the third byte is the message type (1 for "Client Hello" and 4 for "Server Hello"). The first byte of SSLv3.x (i.e. SSLv3.0 or TLS) packets is the message type (22 for configuration records and 23 for records with encrypted application payload). The second and third bytes indicate the major and minor versions (3 and 0 for SSLv3.0 or 1 for TLS).

Let $bit_i[x]$ be the content of bit x in the payload of packet i in the connection, $bit_i[x : y]$ the integer represented by the sequence of bits from x to y, $Byte_i[z]$ the value of byte z, and $Size_i$ the payload size of packet i (computed from fields in IP and TCP headers: Internet Header Length, Total Length and Data Offset). We summarize the decision process to determine if a connection is using SSL and the associated version in the following algorithm:

```
If  bit₂[0] = 1 and bit₂[1] = 0 and bit₂[2 : 15] = Size₂ and Byte₂[3] = 4
    Connection is an SSLv2 connection
Else If Byte₂[1] = 22 and Byte₂[2] = 3
    If Byte₂[3] = 0
        Connection is an SSLv3.0 connection
    Else If Byte₂[3] = 1
        Connection is a TLS connection
    Else Connection is not using SSL
Else Connection is not using SSL
```

4.3 Description of SSL traffic

We applied our identification mechanism to three traces: P6-2004, P6-2006 and UMass. Table 1 shows the proportion of the SSL version found in each trace. We see that most SSL traffic consists of SSLv3.0 and TLS, although there are still a few instances of SSLv2 in the P6-2006 trace. By comparing the proportion of SSL connections in the P6 traces from 2004 and 2006 (4.6% and 8.6%, respectively), we see a sharp increase in the usage of SSL. This trend is supported by the SSL surveys achieved by netcraft [20] (in 2005 only, the use of SSL in the web servers they surveyed increased by 30%). On the UMass trace, this proportion is lower. This difference is because the P6 traces consist only of academic traffic, whereas the UMass campus also has a dorm and, therefore, contain many other types of applications (such as online games).

An interesting observation is the proportion of non-SSL traffic in connections using standard SSL ports (labeled as "SSL Port but not SSL"). We studied this traffic in detail.

In P6-2004, all non-SSL connections on SSL ports were non-encrypted traffic using port 443 (probably misconfigured web servers). In P6-2006, we still find this non-encrypted HTTP traffic, but also observe other types of traffic. For instance, the trace contains un-encrypted SIP traffic (VoIP connections from Instant Message softwares using port 443 to avoid firewalls), and HTTP connections using the CONNECT method. This method is used when a web client connects to an SSL web page using a proxy. However, the contacted servers were not proxies but web servers. It turned out these clients were try-ing to connect to SMTP servers using the web servers as TCP proxies, probably to send spam (this method works for misconfigured Apache servers with proxy capability). In the UMass trace, we also found Bittorent connections using port 443 to avoid firewalls.

Finally, using the detection method presented in Section 4.2, we evaluated the pro-portion of SSL on ports not usually associated with SSL ("SSL on non-SSL Ports"). This proportion is not negligible and is even increasing on the P6 network. This indi-cates that SSL is spreading to applications for which it was not formerly used.

Trace	Total Connections	SSL Connections	SSLv2	SSLv3	TLS	SSL Port but not SSL	SSL on non-SSL Ports
P6-2004	0.5M	4.6%	0.6%	81.0%	18.4%	1.9%	1.1%
P6-2006	1.0M	8.6%	0.2%	53.2%	46.6%	1.1%	4.2%
UMass	1.7M	1.2%	0 %	48%	52%	5.0%	1.5%

Table 1. SSL versions

5 Classification Mechanism

Our characterization in the previous section shows that SSL traffic is increasing. We now present a methodology to identify the applications in SSL connections. Figure 2 describes our classification mechanism. This classifier takes as input a stream of packets from a TCP connection and outputs the application associated to the connection. It runs in three steps: recognition of SSL connections, detection of the first packet containing application data, and recognition of the encrypted applications.
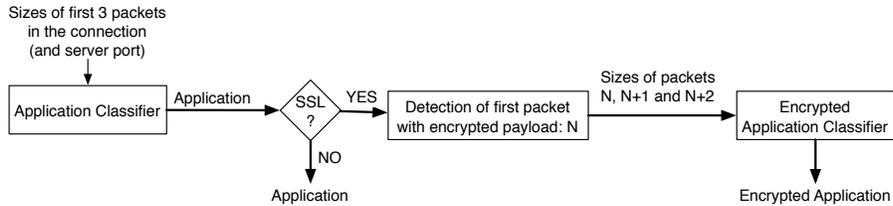

Fig. 2. Classifier Overview

5.1 Recognition of SSL traffic

We use early classification [13] to recognize SSL traffic. We construct this classifier in two phases: training and online classification. The training phase applies a clustering algorithm to a set of sample TCP connections, which we call the training data. We represent each connection in this data set in a P-dimensional space using the sizes and directions of the first P data packet ([13] presents a detailed analysis that justifies using

packet sizes instead of other features such as Inter-Arrival Time). Resulting clusters contain one or more applications. The online classification uses a heuristic to assign a TCP connection to one of the clusters and another heuristic to label it with one of the applications in the cluster.

For this study, we use a training data set composed of the following applications: Http, Ftp, Nntp, Pop3, Smtp, Ssh, Msn, Bittorent, Edonkey, SSLv2 and SSLv3. We apply a signature-based filtering method on the P6-2004 trace to select 500 random connections for Http, Ftp, Nntp, Pop3, Smtp, Ssh, Msn and Edonkey. Since the amount of identifiable Bittorent traffic in our traces is small, we manually generate a Bittorent trace from which we select 500 connections. Additionally, we use the method presented in Section 4 to select 500 SSLv2 and SSLv3 connections from the P6-2006 trace.

We applied our clustering mechanism to this training set. We use a clustering algorithm based on Gaussian Mixture Model [13]. We find that using the first three packets and 35 clusters gives good results (the method to choose the number of packets and clusters is described in [13]). To assign a new connection to a cluster, we compute the probability that this connection belongs to each cluster and choose the one with the highest probability. Finally, to label the connection, we test two methods: use the dominant application in the cluster (*Dominant* heuristic in [13], or label the connection according to the composition of the cluster and the server port it is using (*Cluster+Port* heuristic in [13]). We evaluate the efficiency of this classifier in Section 6.

5.2 Detection of the first data packet

After the classifier establishes that the connection is SSL, it analyzes the packets in the connection to find the first application packets. For SSLv2, the handshake can take four or six packets, depending on whether the client and the server share an encryption key that is still valid. To decide which handshake is used in a given connection, we check if the second packet sent by the client starts a key negotiation (as in figure 1).

For SSLv3, this detection is more difficult because the last packet in the SSL handshake may contain an SSL negotiation record as well as records with encrypted application payload. Therefore, to detect the first application packet in SSLv3 connections we inspect SSL records until we find the first record with content type equal to 23, which indicates an application payload. This inspection is not computationally intensive because the header of each record contains the size of the record. Figure 3 shows the distribution of the position of the first SSL packet that contains application data across all SSL connections. This result shows that there is never application data in the first two packets (which is expected from the RFC). This implies that it is safe to start inspecting TCP payloads after the third packet to identify the first application packet. This is convenient because that is what we need to detect SSL connections (as described in 5.1). Figure 3 also shows that application data may start at any packet between 3 and 12, which justifies the need for the online packet inspection. The number of packets in the handshake depends on whether the client and the server already share a session key. Besides, a packet can consist of several SSL handshake records and SSL implementations use different methods to regroup these records.
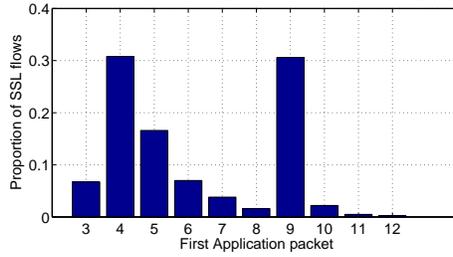
Fig. 3. Position of the first packet with application data

5.3 Application Identification

Once we have identified an SSL connection and the packets corresponding to the application, we recognize the encrypted application based on the sizes of its packets. SSL can use different encryption algorithms, which will modify packet sizes differently. Table 4(a) shows the most common encryption algorithm used in the P6-2004 and P6-2006 traces. The specifications for SSL allows for more than 50 encryption methods. However, we can see that the five most common algorithms account for more than 98% of SSL connections.

Cipher	P6-2004	P6-2006
RC4_128_MD5 (0x04)	79.7%	66.0%
DHE-RSA-AES256-SHA (0x39)	5.9%	13.6%
AES_256_SHA (0x35)	1.0%	10.4%
RC4_128_SHA (0x05)	9.7%	7.0%
RC4_40_MD5 (0x03)	2.0%	2.1%
Other	< 2%	< 1%

(a) Proportion of each cipher
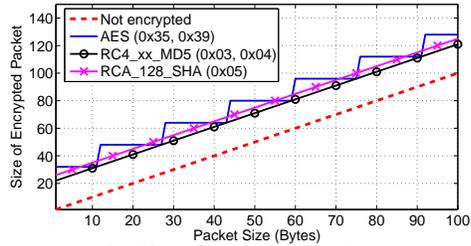
(b) Size of encrypted packets

Fig. 4. SSL ciphers

To evaluate the influence of encryption algorithms on the sizes of exchanged packets, we design a small application that sends packets with different sizes over an SSL connection. Figure 4(b) shows the relationship between the size of application payload and the final size of the ciphered packet depending on the encryption algorithm (we focus on packets with less than 100 bytes, but the evolution remains the same for larger values). This figure shows that encryption mechanisms increase the size of the packets by an amount that depends on the cipher. For RC4 based ciphers, this increase is fixed: 25 bytes for RC4_128_SHA and 21 bytes for the other two. For AES-based ciphers, the increase varies by steps, because they use blocks.

These results are encouraging because even if encryption alters packet sizes, this change is limited and predictable. The most accurate method to decide on the size of the original packet, is to look up the encryption method in the handshake packets and transform the size of application packets accordingly. However, for the five most common ciphers this method is overkill because the increase varies from 21 to 33 bytes. Therefore, instead of keeping track of the cipher, we use a simple heuristic to decide

on the size of the original packet: subtract 21 from the size of the encrypted packet regardless of the cipher.

We apply the same method of section 5.1 to the transformed sizes of the first packets with encrypted data to decide on the encrypted application. We extend the *Cluster+Port* labeling heuristic to take into account SSL-specific ports: we use 443 for HTTPS, 993 for IMAPS and 995 for POP3S.

6   Evaluation

In this section, we first evaluate our method to recognize SSL on the P6-2006 trace. Then, we validate our method to recognize encrypted applications on real Https and Pop3s traffic extracted from the P6-2006 trace and on manually encrypted connections. The test sets do not include any of the training samples.

6.1   Recognition of SSL traffic

To evaluate the accuracy of our classifier to recognize SSL connections, we use two metrics: the proportion of connections accurately classified for all applications in our test data set (True Positives) and the proportion of connections that are wrongly labeled with each application (False Positives). Our test set consists of 50,000 connections, with more than 2,000 connections for each application. Table 2 presents both metrics for our two labeling heuristics: Dominant and Cluster+Port (defined in Section 5.1). Most misclassification errors for the Dominant heuristic happen when a connection is assigned to a cluster consisting of two (or more) applications and does not belong to the application that predominates. For all protocols the use of the Cluster+Port improves the quality of the classification because the port number helps determinate the application. Even with Cluster+Port there are some misclassifcations. These misclassifications happen when a connection in the test set follows a behavior that was not present in the training set or when a connection is assigned to a cluster with more than one application and when the server port does not help.

This table shows that our classifier, based on the sizes of the first three data packets achieves a very high accuracy and that it recognizes SSLv2 and SSLv3 for more than 80% of the connections with Cluster+Port. With the Dominant heuristic, some SSL connections are assigned to clusters which contain SSL and another application that predominates. The reason for the 2.3% of false positives for SSLv2 is that, some SSLv3 connections are classified as SSLv2 (hence the only 81% true positives for SSLv3). This is not unexpected because behaviors of SSLv2 and SSLv3 are similar in some cases. However, we can easily limit the impact of this misclassification by inspecting packets from connections classified as SSLv2 to decide on the real SSL version that is used.

6.2   Recognition of encrypted Applications

We evaluate the recognition of encrypted applications on two different test sets. First, we extract real HTTPS and POP3S connections from the P6-2006 trace: we filter SSL traffic directed to known web and mail servers from the university and obtain more than

Heuristice	Dominant		Cluster+Port	
Application	True Positives	False Positives	True Positives	False Positives
bittorent	74.65%	0.01%	97.30%	0.23%
edonkey	94.76%	2.89%	95.08%	0.18%
ftp	91.00%	0.04%	97.95%	0.04%
http	96.50%	2.96%	98.95%	0.00%
msn	95.36%	0.90%	100.00%	0.00%
nntp	94.40%	0.34%	99.15%	0.00%
pop3	96.65%	2.67%	99.25%	0.00%
smtp	86.35%	0.42%	98.85%	0.00%
ssh	97.73%	0.00%	96.10%	0.00%
sslv2	82.07%	2.20%	94.71%	2.30%
sslv3	67.75%	0.33%	81.20%	0.27%

Table 2. Application detection, including SSL(P6-2006 Trace)

5000 connections for both applications. Then, to evaluate our method against other applications, we manually encrypt 500 connections of FTP, Bittorent and Edonkey traffic and apply our classifier on resulting connections. We perform this classification using the same model we used in section 6.1 and therefore each connection was given a label among the 11 applications in our model.

Table 3 presents the proportion of connections correctly labeled. This table shows that applications that often use SSL (HTTP and POP3) are very well recognized when they are encrypted. The last three rows of this table evaluate our mechanism for applications that cannot be detected with port-based methods and are usually recognized based on signatures. Our classifier accurately classifies these applications when they are encrypted with more than 85% accuracy for the Cluster+port heuristic. We tested this classification without the modification of the payload size (i.e. without subtracting 21 bytes to each encrypted payload). The results where not as good but we were still able to achieve more than 80% accuracy for the Cluster+Port heuristic.

Real Applications	Dominant	Cluster+Port	Manually Encrypted	Dominant	Cluster+Port
http	99.95%	99.95%	ftp	90.58%	92.67%
pop3	98.45%	98.45%	bittorent	77.87%	86.48%
			edonkey	94.56%	96.57%

Table 3. Detection of Encrypted Applications (HTTP and POP from P6-2006 trace, FTP, Bittorent, Edonkey manually encrypted)

7 Conclusion

The contributions of this paper are two-fold. First, a **characterization of SSL usage on two campus networks**. Our analysis shows that the usage of SSL is growing and that the number of applications using SSL is increasing. Second, a **mechanism to recognize the underlying application in SSL encrypted connections based on the size of the first packets in the connections**. We show that our method achieves more than 85% accuracy. The implementation and the data used in this study are available at: http://rp.lip6.fr/ bernaill/earlyclassif.html.

In future work, we plan to extend our method to other encryption mechanisms such as SSH and IPsec. For both these protocols, the isolation of connections and the determination of the first application packets will be more challenging. Besides, the latest SSL implementations include options for compressing data and sending empty segments. These options would affect our detection mechanism and we plan to extend it to take them into account.

References

1. Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., Faloutsos, M.: Is p2p dying or just hiding? In: Globecom. (2004)
2. Paxson, V.: Bro: a system for detecting network intruders in real-time. Computer Networks (Amsterdam, Netherlands: 1999) **31** (1999) 2435–2463
3. Snort: http://www.snort.org.
4. Ma, Levchenko, Kreibich, Savage, Voelker: Unexpected means of protocol inference. In: Internet Measurement Confererence. (2006)
5. Song, D.X., Wagner, D., Tian, X.: Timing analysis of keystrokes and timing attacks on ssh. In: Proc. 10th USENIX Security Symposium. (2001)
6. Hintz, A.: Fingerprinting websites using traffic analysis (2002)
7. Roughan, M., Sen, S., Spatscheck, O., Duffield, N.: A statistical signature-based approach to ip traffic classification. In: IMC. (2004)
8. McGregor, A., Hall, M., Lorier, P., Brunskill, J.: Flow clustering using machine learning techniques. In: Passive and Active Measurement. (2004)
9. Zuev, D., Moore, A.: Traffic classification using a statistical approach. In: Passive and Active Measurement. (2005)
10. Moore, A., Zuev, D.: Internet traffic classification using bayesian analysis. In: Sigmetrics. (2005)
11. Erman, J., Arlitt, M., Mahanti, A.: Traffic classification using clustering algorithms. In: MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data, New York, NY, USA, ACM Press (2006) 281–286
12. Bernaille, L., Teixeira, R., Akodkenou, I., Soule, A., Salamatian, K.: Traffic classification on the fly. SIGCOMM Comput. Commun. Rev. **36** (2006) 23–26
13. Bernaille, L., Teixeira, R., Salamatian, K.: Early application identification. In: To appear in Conference on Future Networking Technologies. (2006)
14. Wright, Monrose, Masson: On inferring application protocol behaviors in encrypted network traffic. the Journal of Machine Learning Research Special Topic on Machine Learning for Computer Security (2006)
15. Karagiannis, T., Papagiannaki, D., Faloutsos, M.: Blinc: Multilevel traffic classification in the dark. In: SIGCOMM. (2005)
16. Wright, Monrose, Masson: Using visual motifs to classify encrypted traffic. Workshop on Visualization for Computer Security (2006)
17. SSLv2: http://wp.netscape.com/eng/security/SSL_2.html.
18. SSLv3.0: http://wp.netscape.com/eng/ssl3/draft302.txt.
19. TLS: http://www.ietf.org/rfc/rfc2246.txt.
20. Netcraft: http://www.netcraft.com.