Enhancements to Statistical Protocol IDentification (SPID) for Self-Organised QoS in LANs

Christopher Köhnen[1], Christian Überall[1], Florian Adamsky[2], Veselin Rakočević[1], Muttukrishnan Rajarajan[1], Rudolf Jäger[2]

[1]*School of Engineering and Mathematical Sciences*
City University London
Northampton Square, London EC1V 0HB, UK
{*Christopher.Koehnen.1, Christian.Ueberall.1, V.Rakocevic, R.Muttukrishnan*}*@city.ac.uk*

[2]*Department for Information Technology, Electrical Engineering & Mechatronics*
University of Applied Sciences FH Giessen-Friedberg
Wilhelm-Leuschner-Str. 13, D-61169 Friedberg Germany
{*Florian.Adamsky, Rudolf.Jaeger*}*@iem.fh-friedberg.de*

***Abstract*—Since most real-time audio and video applications lack of QoS support, QoS demand of such IP data streams shall be detected and applied automatically. To support QoS in LANs, especially in home environments, a system was developed which enables self-organised QoS for unmanaged networks through host implementations - in contrast to traditional solutions, without network support. It supports per-link reservation and prioritisation and works without a need for application support. One part of this system is an automated traffic identification and classification system, which is subject of this paper. An efficient set of attribute meters, based on the Statistical Protocol IDentification (SPID), was investigated, enhanced and evaluated. We improved the performance, added support for UDP protocols and real-time identification. It was shown that using our implementation efficient near real-time protocol identification on per-flow basis is possible to support self-organised resource reservation.**

***Keywords*-Classification; Statistical Analysis; SPID; Packet Identification; QoS;**

I. Introduction

Today, multimedia services increase dramatically in home and private networks [1]. Television services over the network (IPTV) and voice over ip (VOIP) services have a demand for high bandwidth capacity and very strong quality of service (QoS) needs [2], [3]. To guarantee these, common QoS strategies like IntServ, using RSVP, or DiffServ have to be supported by the network. Since in most home or private networks low cost hardware is used, a support of these techniques cannot be assumed. Beside this, web video applications tunnel their streams using HTTP and are therefore not easily distinguishable from common Internet traffic. This results in less technology acceptance by users, since lacking QoS leads to a lower quality of experience (QoE) level [4]. Especially for IPTV providers the network plane inside the households is unpredictable, as providers only can influence the QoS level until the transfer point to the house. In addition to that is the network a shared medium, in contrast to the traditional TV cable, which leads to new challenges for a traditional service to achieve the accustomed QoE level. One approach to increase QoE in LANs is the QoSiLAN system [5]. It is based on several core technologies and works in three phases. In a first phase physical network discovery algorithms and QoS parameter tests run through, to generate a detailed map about the local network and its available resources. For this purpose, the Mircosoft LLTD protocol [6] was reimplemented extended to efficiently make use of its topology and QoS analysis functionality. In a second phase traffic monitoring, analysis and policing is performed. The research on this part is the subject of this paper. Finally network resources are reserved and prioritised for the monitored flow. Here signalling based on the NSIS protocol's NSLP for QoS Signalling [7] framework is applied to coordinate QoS issues between the network hosts.

In contrast to common QoS strategies QoSiLAN doesn't depend on router or switch support to enable QoS, but makes use of it, if available. The central entity, which performed the mapping and monitoring advises all hosts in the network to shape and DSCP-mark their traffic according to its policies and advises. But only those traffic flows need to be shaped, whose data-paths affect physical links where current reservations apply.

II. Related Work

A. Port-Based Identification

The TCP/UDP-port-based packet identification is the simplest method to classify traffic. By using this method the port numbers of the packets are inspected and mapped to the IANA's list of well-known ports. Moore et al. [8] showed that approximately only 70% of the traffic can be identified correctly that way. One of the problems, when using port-based packet identification, is that many applications like P2P may not have a registered port, which makes it completely impossible to detect them. Additionally some ports are ambiguous. One example is port 888, which is used by the CD Database Protocol (CCDP), the AccessBuilder [9] and the RAID controller 3DM and 3DM2 protocols. Another problem with port-based classifiers is the increasing number of applications that use the HTTP's port 80 to bypass firewalls. Also security attacks are not associated with an specific port and can therefore not be detected using port mappings. Overall is an identification based on the port numbers not a reliable method anymore.

B. *Deep-Packet-Inspection*

The Deep Packet Inspection (DPI) is more reliable technique to inspect packets on application level, which is the layer 7 in the OSI reference model. In layer 7, a packet can be scanned for application specific signatures. One disadvantage of this approach is the human intervention to analyse the application and to create an unique application signature [10]. This analysis entails a lot of effort, even if a RFC or whitepaper exists. If no application documentation exists, it is very difficult to reverse-engineer the targeted application's protocol. Another drawback emerges, when the application protocol doesn't carry application significant values. In this case there is no chance to identify the application. DPI also causes significant privacy issues, since it also scans the payload of plain text protocols like e.g. SMTP, HTTP, POP3, etc., which may carry unencrypted private data. This may also lead to false positive results.

C. *Machine-Learning Techniques*

Machine-learning algorithms are powerful for traffic classification as shown in [11] and can be categorised as supervised, unsupervised and semi-supervised learning [5]. Supervised learning means that labelled training data are required to classify flows. In contrast, unsupervised learning algorithms work with unlabeled data. They don't classify, but cluster the flow into groups. The third variant, the semi-supervised machine learning algorithms makes use of both techniques. It needs a mixture of a small amount of labelled training data and a large amount of unlabelled training data for identification and learning. Many machine learning algorithms need full-flow statistics, which makes it difficult to identify flows in real-time [10]. Therefore these approaches are appropriate to us, because our goal is to detect a particular flow and classify it in near real-time. Since most of these algorithms have a high complexity, the computing costs are too high for embedded real-time calculation.

III. Statistical Protocol IDentification (SPID)

A. *Algorithm Details*

The SPID algorithm was deployed by Erik Hjelmvik and Wolfgang John and is a statistical approach to identify applications and protocols [12]. There is no need to search for unique application signatures and in addition real-time detection after the 10th packet is possible. Both features define the advantages of this kind of technique. The SPID algorithm works in three steps as presented in figure 1. At first all packets must be grouped together as bi-directional flows. In the case of TCP a flow starts with a three-way-handshake. A flow will be identified by source IP and port, destination IP and port and transport protocol (5-Tupel). But only packets carrying a payload are significant. This means that e.g. all TCP-ACK packets are not relevant for the statistics.

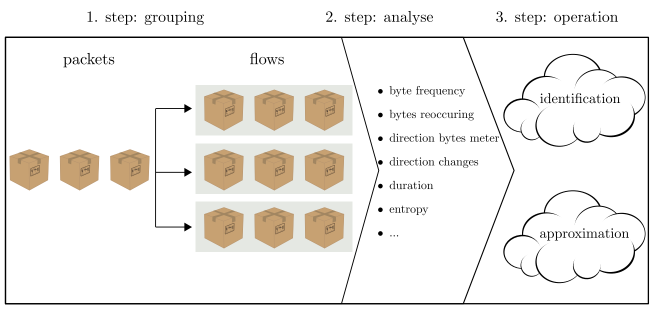

Figure 1. The SPID algorithm works in three phases.

When a flow reaches a minimum number of 10 packets it becomes subject of the actual analysis. Plain-text protocols can be identified after the 10th packet, but binary or HTTP-tunnelled protocols need up to 20 packets for reliable detection as pointed out in section IV. c. By making use of several statistical methods, samples are taken from each flow, e.g. byte frequency, entropy, direction changes and many more. After this step, it branches into identification or approximation. Approximation is the learning part, in which new protocols can be included or old ones improved. The learning part works by making use of the empirical law of large numbers. The identification is the heart of the SPID algorithm. It compares the probabilities of the trained flows with the observed flows by calculating the Kullback-Leibler divergence [13], as shown in equation 1. It is the fingerprint that is compared for protocol identification

$$D(P||Q) = KL(P,Q) = \sum_{x \in X} P(x) \log_2 \frac{P(x)}{Q(x)} \qquad (1)$$

the Kullback-Leibler divergence is a logarithmic measure of the relation between the relative frequency of the observed (P) to the trained flows (Q), summed for each attribute measure. The result is then compared to the list of known protocols. The protocol with the nearest divergence ($D(P||Q)$) is then identified. The distance represents the probability. The threshold to identify a flow as unknown was experimentally determined, as presented in figure 4.

B. Implementation

We reimplemented the SPID algorithm with some differences to Hjelmvik to gain the performance and to support a wider range of protocols, as explained in the following. To ensure a general compatibility even on small scale embedded hardware, we implemented it using the C++ programming language and the libPcap library for portability. New attribute meters were implemented to support UDP and near real-time identification. We have optimised the fingerprint database to keep as small as possible and flexibly exchangeable. In our case the fingerprint database with 17 protocols has a size of 389KB, whereas Hjelmvik's consumes 9.8MB for 12 protocols, using a XML-format. This was achieved by choosing a binary format and defining a variable array size for each attribute meter.

Our focus was on streaming protocols like RTP, MPEG-TS, MMS or RTMP as well as on progressive downloads like flash videos and WMV/OGG streams which are tunnelled through HTTP. We chose 12 protocol attribute meters, which actually worked out very well and with good performance for our purposes.

C. Protocol Attribute Meters

The protocol attribute meters are the statistical measurements of this approach.

byte-frequency: This measurement operates on the first TCP packet of each direction and counts the frequency of a byte in the payload [12]. Encrypted or compressed data appear in an even distribution, whereas data in plain-text show an uneven distribution. See figure 2 for an example.

byte-frequency of the first 32 bytes: Most UDP protocols contain little clear-text information and have only a small header, therefore the byte-frequency of the whole payload results in a high divergence value. To avoid this, we took only the first 32 bytes of the first packet and count the frequency.

direction-changes: It measures how often the protocol or the application communication changes the direction. Interactive protocols like Telnet, SSH or FTP often have direction changes, whereas streaming protocols don't.

direction bytes meter: Percentage of amount of data, which was sent from the client to the server and vice versa. Through this measurement a distinction is possible between:

- upload and download balanced (e.g. RTP for VoIP, ...)
- almost only download (e.g. HTTP, POP3, ...)
- almost only upload (e.g. SMTP, IRC, ...)

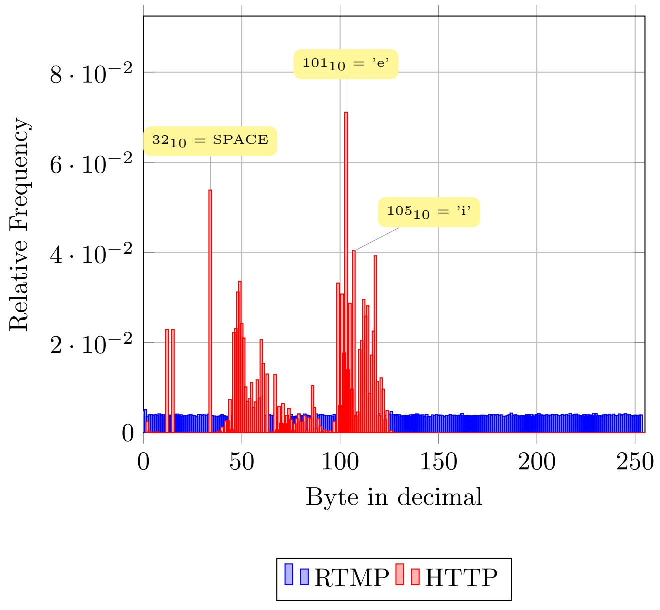

Figure 2. Byte Frequency Histogram

entropy: The entropy is a measurement for the amount of random information within a system [14]. The maximal entropy of 256 possible bytes is $H(I)_{max} = 8$. It is applied to the first packet in each direction. E.g. plain-text protocols with natural language in it, have low entropy values, while those with encrypted or compressed data have high ones.

first 4 bytes hash-function: The first four bytes of the first packets in each direction are very characteristic for most protocols [15]. For the first four bytes a hash-value is calculated. To simplify matters we make use of a cross-sum. An sample hash function for HTTP is shown in table I.

```
1: 47 45 54 20   G E T /   = 166
2: 48 54 54 20   H T T P   = 206
```

Table I
EXAMPLE HASH FUNCTION FOR THE HTTP

action reaction first 3 byte hash meter: It generates a hash-function of the first 3 bytes of each packet that wasn't sent in the same direction as the previous one. The idea behind this measurement is to get a connection between a request and a response, especially for command based protocols like HTTP, FTP or POP3.

byte pairs reoccurring: This measurement identifies bytes that occur more than once within the first 16 bytes of the first packet. E.g. it identifies the "SS" in the SSH-banner or the "TT" in the HTTP GET and POST request.

unicode frequency: It scans for alphabetical unicode strings in the first five packets and saves the byte frequency for it. Since some protocols like WMV streams or MMS [16] use unicode strings in their protocols.

first 8 bytes equality meter: It checks how often the first 8 bytes are equal, since this is significant for protocols that have a fixed header in every packet, e.g. the RTP [17] or MPEG-TS [18].

first bit positions meter: Especially UDP protocols have a small header, which consists of fields and flags on bit level. This fact makes it hard for the byte frequency meter to detect the particular protocol. This attribute meter counts the frequency of a single bit in connection with its offset. The first 128 bits will be viewed and counted.

first payload size: Payload length of the first packet in a flow [19]. In most cases the first packet contains information for initialisation of a session. The first packet size of a HTTP session is only between 120 and 1000 bytes. POP3 for example is between 10 and 100 bytes.

Implementation experiences revealed, that some attribute meters are not very accurate and adulterated the results. For this reason the duration of flows, port numbers, packet size and the inter-arrival-times were not included for the SPID calculation. Instead, we included additional attribute meters to enhance the results. These were the number of direction changes, the first payload size, the entropy and the unicode frequency.

IV. Evaluation

A. Datasets and Implementation Setup

An important part of the research presented here was the creation and procuration of datasets. On one hand we created datasets under laboratory conditions, and on the other hand we used freely available datasets[1,2,3] and extracted the specific protocols. On creation, we paid extra attention to capture training data from application with as different versions and implementations as possible. E.g. for HTTP we used various browsers on different operating systems to get a wider spectrum of payload data. In order to do this, we automated as much as possible using Perl scripts. To capture network traffic we used the packet analysers Tcpdump[4] and Wireshark[5]. Overall we gathered 3135 pure flows taking more than 1.5GB for the 17 protocols we tested.

B. Evaluation Methods

Evaluations are performed by the formula for the Recall, the Precision and the F-Measure to analyse the accuracy and stability of this approach [20], as presented in equations 2 to 4. It is necessary to know the true-positives (TP), which represent all the flows that are identified correctly. The false-positives (FP), represent the not or wrong identified flows. The false-negatives (FN), represent the other protocols that were wrongly identified. The F-Measure is the weighted combination between recall and precision.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F - Measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{4}$$

C. Results

The figure 3 contains the final results created using 30 trained session each.

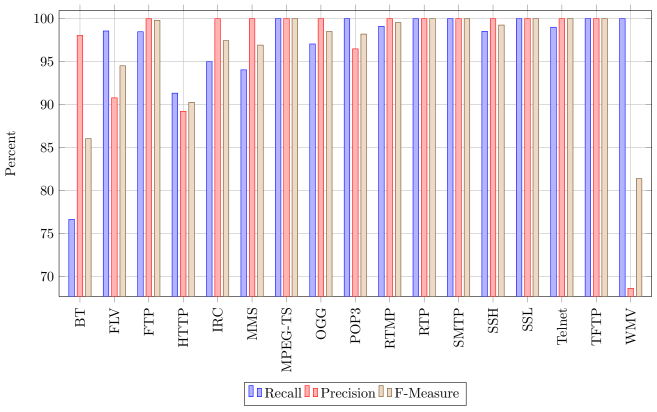

Figure 3. Protocol Identification Results

For a more detailed view, see table II, which presents the underlying data.

Protocol	#	Val.	Rec.	Prec.	F-Meas.
BT	30	197	76.65%	98.05%	86.04%
FLV	30	70	98.57%	90.70%	94.52%
FTP	30	461	97.62%	100.00%	99.80%
HTTP	30	127	91.34%	88.55%	89.92%
IRC	30	100	95.00%	100.00%	98.51%
MMS	30	252	94.05%	100.00%	96.93%
MPEG-TS	30	40	100.00%	100.00%	100.00%
OGG	30	122	97.06%	100.00%	98.51%
POP3	30	55	100.00%	96.49%	98.21%
RTMP	30	112	99.11%	100.00%	99.55%
RTP/UDP	30	69	100.00%	100.00%	100.00%
SMTP	30	464	100.00%	100.00%	100.00%
SSH	30	136	98.53%	100.00%	99.26%
SSL	30	41	100.00%	100.00%	100.00%
Telnet	30	812	99.01%	100.00%	99.50%
TFTP	30	42	97.62%	100.00%	98.80%
WMV	30	35	100.00%	68.63%	81.40%

Table II
Validation Results

[1] http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html
[2] http://www.pcapr.net
[3] http://www.openpacket.org/
[4] http://www.tcpdump.org/
[5] http://www.wireshark.org/

It's shown that this implementation achieves good results for most of all the tested protocols. The HTTP identification is less good, since it is a more loosely described protocol [12], e.g. it is difficult to differentiate between a flash video and a website, which is compressed with gzip or which loads lots of images using a single TCP connection. Also a lower precision was archived for WMV streams, since most WMV data is not only progressively tunnelled through HTTP, but also streamed using the MMS protocol. That's why they are often mixed up by the classificator. The BitTorrent protocol makes use of the Message Stream Encryption (MSE) [21], to complicate identification through several obfuscation techniques and encryption. The AccumulatedDirectionBytesMeter [15] is a special attribute meter to identify BitTorrent. We do not make use of this meter, since it blurs significantly the results for the other protocols. Figure 4 shows the F-Measure depending on the number of trained flows.

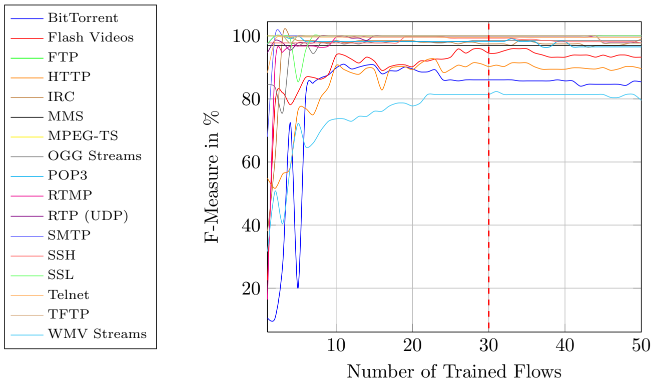

Figure 4. The F-Measure Depending on the Number of Trained Flows

There one can see that with the number of trained flows the stability of the identification becomes better. The graph shows that after the 20th packet the F-Measure for all protocols reaches the 80 percent marker, which is a critical value for a stable protocol identification. For plain-text protocols like IRC, Telnet, SMTP and so on, is a stable identification possible after the 10th packet.

Figure 5 shows the F-Measure with 30 trained flows depending on the number of inspected packets with payload. As shown in this figure, the F-Measure has risen rapidly after the 13th packet for OGG vorbis and WMV streams. Only after this point there is a distinction possible between those two. The F-Measure evens out after the 20th packet. This is early enough to initiate actions, e.g. to prioritise the particular flow. For other protocols like Telnet, SSH or TFTP a stable identification after the 10th packet is possible. Figure 6 shows the F-Measure for 30 trained flows and 20 inspected packets depending on the threshold.

This threshold value is compared with the sum of the values of the attribute meters. If it is equal or lower, the

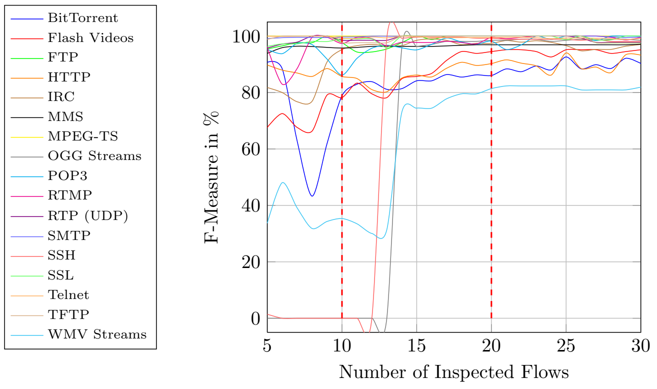

Figure 5. F-Measure Depending on the Number of Inspected Packets

protocol is known. If it is higher than the threshold, the flow is marked as unknown. It also helps to avoid the detection of false-positively identified flows. The optimal value was experimentally determined as 13, using the results presented in figure 6. At this point an optimal identification with a good precision is possible and unknown protocols are correctly identified as unknown.

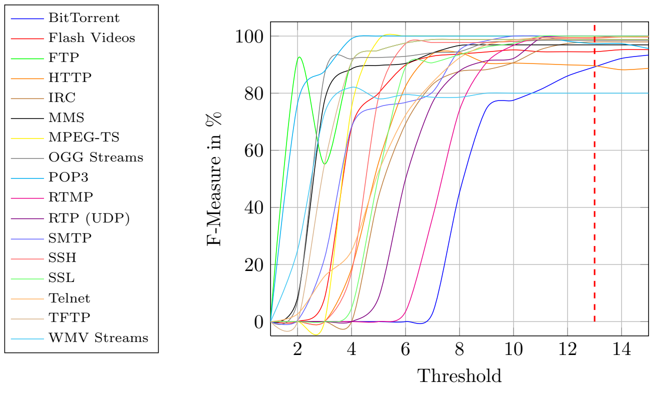

Figure 6. F-Measure Depending on the Threshold

D. *Classification*

The currently implemented classificator only distinguishes between media and best-effort traffic. Media traffic is marked for expedited forwarding, whereas best-effort traffic is not marked. This allows for prioritisation of real-time and progressive download flows.

V. Conclusion

In order to enable self-organised, application independent QoS in LANs, we presented an optimised implementation of the SPID algorithm with a particular number of indicators that allow for identification of protocols and applications in near real-time. We archived very good results for plain-text protocols like Telnet, SMTP POP3, IRC, etc., but even if the payload is not human-readable good results have been archived. Another advantage of our approach is the flexibility of the level of granularity. For our purpose it was important to look deeper into the HTTP to also identify protocols which are tunnelled, like progressive video downloads. To include new protocols for identification it is sufficient to add at least 30 data set samples. This is surely easier than looking for an unique application signature. The math behind this algorithm is trivial, thereby a real-time recognition even on low cost hardware is possible.

In contrast to Hjelmvik we included more protocols and focused on media protocols, which are worthy to be prioritised. These were tested for their precision and robustness. We took particular care of a small and fast implementation and made a selection of different attribute meters for better performance and more accurate identification. In addition, we added UDP support to the near real-time identification and showed the feasibility. We also showed that SPID is a appropriate approach for identification of real-time protocols for self organised QoS configuration. Future work will focus on automated bandwidth estimation and resource reservation for identified media flows

References

[1] R. Briel. (2009, June) German iptv numbers set to grow. [Online]. Available: http://www.broadbandtvnews.com/2009/06/17/german-iptv-numbers-set-to-grow/

[2] B. Goode, "Voice over internet protocol (voip)," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1495–1517, Sep 2002.

[3] I.-T. F. G. on IPTV standardization, "Classification of iptv services based on network qos requirements," *2nd FG IPTV meeting*, 2006.

[4] K. Kilkki, "Quality of experience in communications ecosystem," *Journal of Universal Computer Science*, vol. 14, no. 5, pp. 615–624, 2008. [Online]. Available: http://www.jucs.org/jucs_14_5/quality_of_experience_in

[5] C. Koehnen, "Qosilan - a novel qos strategy using synergy effects of network topology discovery, traffic analysis and nsis qos signalling," Transition Report, City University London, Tech. Rep., 2009.

[6] M. Corporation, "[ms-lltd]: Link layer topology discovery (lltd) protocol specification," Open Specifications Documentation, 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/cc233983%28PROT.10%29.aspx

[7] A. M. J. Manner, G. Karagiannis, "Nslp for quality-of-service signaling (qos nslp), internet draft (draft-ietf-nsis-qos-nslp-16), work in progress." 2009.

[8] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *IN PAM*, 2005, pp. 41–54.

[9] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification," *IN IMC04*, pp. 135—148, 2004.

[10] Y. Wang and S. Yu, "Move Statistics-Based traffic classifiers online," in *Computer Science and Software Engineering, 2008 International Conference on*, vol. 4, 2008, pp. 721–725.

[11] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the 2008 ACM CoNEXT Conference*. Madrid, Spain: ACM, 2008, pp. 1–12.

[12] E. Hjelmvik and W. John, "Statistical protocol identification with spid: Preliminary results," in *sourceSwedish National Computer Networking Workshop*, 2009. [Online]. Available: http://www.cse.chalmers.se/~johnwolf/publications/sncnw09-hjelmvik_john-CR.pdf

[13] S. Kullback and R. A. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, vol. 22, pp. 49–86, 1951.

[14] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, 1948.

[15] E. Hjelmvik. (2009) Wikipage of SPID. [Online]. Available: http://sourceforge.net/apps/mediawiki/spid/index.php

[16] M. Corporation, "[MS-MMSP]: Microsoft media server (mms) protocol specification," Open Specifications Documentation, 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/cc234711%28PROT.10%29.aspx

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (Standard), Internet Engineering Task Force, Jul. 2003, updated by RFC 5506. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[18] "ISO/IEC 13818-1 information technology - generic coding of moving pictures and associated audio information: Systems," 2002.

[19] A. D. Montigny-Leboeuf, "Flow attributes for use in traffic characterization," December 2005. [Online]. Available: http://www.crc.ca/files/crc/home/research/network/system_apps/network_systems/network_security/publications/ADeMontigny_CRCTN2005003.pdf

[20] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambridge University Press, July 2008. [Online]. Available: http://nlp.stanford.edu/IR-book/information-retrieval-book.html

[21] (2009) Message stream encryption specification. [Online]. Available: http://wiki.vuze.com/w/Message_Stream_Encryption