

**DUBLIN CITY UNIVERSITY**

**SCHOOL OF ELECTRONIC ENGINEERING**

**Video Stream Identification using Netfilter**

**Literature Review**

**Thomas Troy**

ID Number: 13212592

February 2015

MASTER OF ENGINEERING

IN

ELECTRONIC SYSTEMS

Supervised by Dr. Martin Collier

Acknowledgements

I would like to thank my supervisor Dr. Martin Collier for his guidance and commitment to this project. I would also like to thank my partner for her continued patience and support during my studies. Thanks are also due to Accuris Networks for sponsoring my Masters and for their support during my research.

Declaration

I hereby declare that, except where otherwise indicated, this document is entirely my own work and has not been submitted in whole or in part to any other university.

Signed: ...................................................................... Date: ...............................

Abstract

The increased proliferation of video and other high bitrate streaming services on the internet has led to many challenges for network designers. One of these problems is the need for Quality of Service (QoS) for real-time and multimedia protocols. A part of any QoS strategy is efficiently and correctly identifying and classifying the traffic so that the correct QoS profile can be applied. Traditional classification techniques involve looking at “well known” TCP or UDP port numbers, however, increasingly these techniques are being found to be less effective because of new protocols which do not use well defined ports. Additionally, because some protocols attempt to “piggy-back” on the well-known ports to avoid being blocked by firewalls.

This project looks at how video streaming traffic can be correctly identified by using “signatures” for various types of video streams. These signatures can be identified by using various properties of the packets that are distinctive for this type of traffic. This paper looks at the use of properties from both the layer-7 payload and statistical properties from traffic flow using an algorithm known as Statistical Protocol IDentification (SPID).

The targeted platform is the Linux operating system, which uses the Netfilter subsystem for purposes such as firewalling and network address translation. The Netfilter subsystem can be accessed by user space programs to define rules, and also by writing kernel modules in the C programming language which can augment the Netfilter subsystem capabilities.

Table of Contents

[Introduction 1](#_Toc410595235)

[Netfilter 3](#_Toc410595236)

[Overview 5](#_Toc410595237)

[Kernel Layer 6](#_Toc410595238)

[User Layer 9](#_Toc410595239)

[xtables 9](#_Toc410595240)

[nftables 10](#_Toc410595241)

[Netlink 11](#_Toc410595242)

[Classification 12](#_Toc410595243)

[Port Based Classification 13](#_Toc410595244)

[Payload Based Classification 14](#_Toc410595245)

[Payload Based Applications 15](#_Toc410595246)

[Payload Based Research Papers 16](#_Toc410595247)

[Statistical Analysis Based Classification 19](#_Toc410595248)

[Statistical Analysis Based Research Papers 20](#_Toc410595249)

[Summary 23](#_Toc410595250)

[Internet Video 27](#_Toc410595251)

[Video Streaming Techniques and Protocols 27](#_Toc410595252)

[Real Time Messaging Protocol 27](#_Toc410595253)

[Real-Time Transport Protocol 29](#_Toc410595254)

[Peer-to-Peer TV 31](#_Toc410595255)

[HTTP Video Streaming 31](#_Toc410595256)

[Conclusions and Project Outline 34](#_Toc410595257)

Table of Figures

[Figure 1: Netfilter Architecture 6](#_Toc410595258)

[Figure 2: Netfilter Hooks 7](#_Toc410595259)

[Figure 3: RTMP Handshake 28](#_Toc410595260)

[Figure 4: RTP Header 30](file:///C:\Users\ttroy\Dropbox\College\Project\litrature%20survey\Video%20Stream%20Identification%20using%20Netfilter%20-%20Literature%20Review%20-%20Thom%20Troy.docx#_Toc410595261)

Glossary

ABI Abstract Binary Interface

API Application Programming Interface

CBR Constant Bit Rate

DASH Dynamic Adaptive Streaming over HTTP

DPI Deep Packet Inspection

FTP File Transfer Protocol

IANA Internet Assigned Numbers Authority

IP Internet Protocol

ITU-T International Telecommunications Union

JSON JavaScript Object Notation

MPEG Motion Picture Experts Group

MS-SSTR Microsoft Smooth Streaming Protocol

NAT Network Address Translation

QoS Quality of Service

RTCP Real Time Control Protocol

RTMP Real Time Messaging Protocol

RTP Real Time Transport Protocol

RTSP Real Time Streaming Protocol

SSH Secure Shell

SIP Session Initiation Protocol

SRTP Secure Real Time Transport Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

VBR Variable Bit Rate

VoIP Voice over Internet Protocol

XML Extensible Markup Language

# Introduction

A recent survey by Cisco [1] has predicted that 79% of consumer internet traffic will be video by 2018. This increase in video usage is introducing many challenges for network designers, where they have to manage the increase in traffic and cater for the Quality of Service (QoS) needs of real time and streaming data. This means that network designers need to be able to correctly classify and match video and other data on ingress routers so that the appropriate QoS and traffic management profiles can be applied.

The traditional methods for traffic classification were based on matching the port number to a list of well-known ports, typically from the Internet Assigned Numbers Authority (IANA) list of assigned ports (for example, port 80 is HTTP) [2]. However, this method of classification has limitations as applications are increasingly using techniques to bypass firewalls [3]. These techniques include methods such as traffic being tunnelled in other protocols and misappropriating the well-known IANA ports for unintended protocols. The studies in [4] and [5] show that between 30% - 70% of traffic cannot be correctly classified using port based classification

As a result of the limitations with traditional classification rules, more advanced techniques such as Deep Packet Inspection (DPI) have been used. The International Telecommunications Union (ITU-T) has defined recommendations for DPI in next generation networks in the recommendation ITU-T Y.2770 [6]. This defines DPI as follows:

“*Analysis, according to the layered protocol architecture, of:*

* *payload and/or packet properties deeper than protocol layer 2, 3 or 4 (L2/L3/L4) header information, and*
* *other packet properties*

*in order to identify the application unambiguously*”.

DPI can work around the limitation of port and IP based classification by inspecting the L7 payload of a packet and comparing it against well-known signature formats. However, this method has implications both in terms of privacy and processing power.

Newer techniques involve looking at the various statistical attributes of a “flow” of traffic, such as packet size and inter-arrival times, and using this to classify the packet.

This paper proposes a hybrid approach to packet classification based on the SPID algorithm [7] [8]. This will allow video streaming classification signatures to choose relevant attributes and create a protocol signature or fingerprint based on these attribute meters. During classification, observed packet payload and flow statistics will be examined and compared with the fingerprint using a Kullback-Leibler divergence.

In Linux, the firewalling technology is supplied by the Netfilter framework [9] [10]. Netfilter is located in the Linux kernel IP layer and provides a stateful filtering firewall. The Netfilter framework provides a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. The registered callback is then called for every packet that traverses the registered hook in the network stack.

Kernel modules that listen for Netfilter hooks [11] are typically written in the C programming language, and may be manipulated using the user space programs iptables [12] or nftables [13], which is available for kernel versions greater than v3.13.

This paper proposes to use the Netfilter framework to implement a firewall extension which will implement the SPID algorithm. The firewall extension can then be used for live classification and identification of network traffic.

# Netfilter

One of the core uses for Netfilter is as a firewall on the Linux operating system. Before looking at the details of the Netfilter framework, I present some details on the evolution of firewalls and show how Netfilter can be used during each of these evolutions.

As specified in [14] [15], firewall technology can be briefly classified into 4 main categories:

*Packet Filter Firewalls*

Packet filter firewalls (stateless firewalls) are the first generation of firewalls. Packets are examined against a set of rules based on the addressing information of the packet to determine if it may pass through the firewall or not. These rules can typically be based on a subset of:

* Physical Network Interface
* Source IP Address
* Destination IP Address
* Transport Layer Protocol
* Source Port Number
* Destination Port Number

Using Netfilter with the *x*tables frameworks it is possible to build a packet filter framework to examine the IP address and port of a packet. Nftables extends these matching rules to allow a more extensive range of matches.

*Circuit Level Firewalls*

Circuit level firewalls (stateful firewalls) are the second generation of firewall technology. These firewalls validate whether a packet is a connection request, part of an existing connection, or belongs to a virtual circuit. They will maintain a table of valid connections and only forward a packet if it belongs to a valid connection. The following information may be stored about connections:

* A Unique Session ID for Tracking
* Connection State
* Packet Sequence Information
* Source IP Address
* Destination IP Address
* Input Physical Network Interface
* Output Physical Network Interface.

The connection tracking layer of Netfilter, known as conntrack [16] can be used to track the connection state of a firewall and this can also be controlled by the *x*tables and nftables user layer utilities.

*Application Layer Firewalls*

Application layer firewalls, as the third generation of firewall technology, perform examinations on the network packets looking for valid application layer data before allowing a packet. These firewalls contain specialised software and proxy services designed for specific application layer protocols, for example a HTTP request may be checked for malicious content.

Extension modules for Netfilter can be used to add application layer firewalls. One such extension is the payload based classifier L7-Filter [17].

*Dynamic Packet Filter Firewalls*

Dynamic packet filter firewalls are the fourth generation of firewall technology and are capable of dynamically making adjustments to their security rules. They work by associating all UDP traffic with a virtual connection, therefore emulating TCP's connection based semantics.

A simple example of a Dynamic Packet Filter may be a firewall in which incoming packets are only allowed to traverse the firewall if an outgoing packet was first seen going to the same IP address and port. Dynamic packet filter firewalls can be used to stop targeted attacks [18]. Utilities such as Fail2Ban [19] can monitor SSH access logs and update the iptables firewall rules to block an IP address if there are too many failed authentication attempts from that IP address.

## Overview

The Netfilter project [9] began in 1998 by Paul “Rusty” Russell and was merged into v2.3.x of the Linux kernel. Netfilter is located in the Linux kernel IP layer and has various options for packet filtering, Network Address Translation (NAT), connection tracking, and port translation.

The Netfilter framework provides a set of hooks inside the Linux kernel that allows kernel modules to register callback functions with the network stack. The registered callback is then called for every packet that traverses the registered hook in the network stack.

Netfilter provides a number of user space modules that can be used to control the firewall in Linux. From Linux kernel 2.4.x, the iptables [12] package was included. This also contains the packages ip6tables to control IPv6 packet filter rules. From kernel v3.13, the nftables [13] package has been merged into the mainline code. Nftables is designed to be a replacement for the *x*tables set of components including iptables, ip6tables, arptables, and ebtables. Nftables provides a compatibility layer to allow the running of iptables rules. More detailed descriptions of both iptables and nftables are provided below.

The Netfilter subsystem provides the ability to track and list connections with the conntrack set of tools (libnetfilter\_conntrack, conntrack-tools) [16]. This allows the kernel to keep track of logical network connections and provide a relationship between all packets on that connection. Logical connections can include TCP and UDP based flows. In the case of TCP, a flow is determined by the semantics of TCP. In the case of UDP, a connection is considered established if there is bi-directional UDP traffic between the same pair of IP addresses and ports. This conntrack facility could be used to examine packets in a flow of traffic for statistical based classification techniques.

Figure 1 [20] below shows an overview of the Netfilter framework as presented by one of the Netfilter authors, Jan Engelhardt. It shows the structure of Netfilter and the various components.

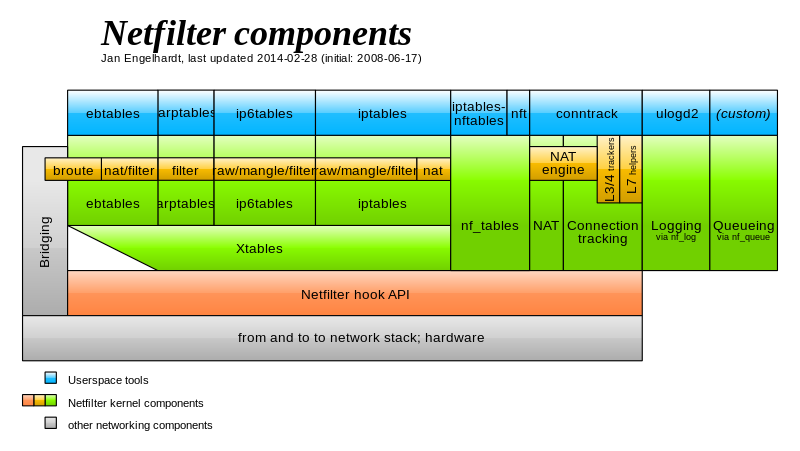


Figure 1: Netfilter Architecture

## Kernel Layer

As mentioned above, the Netfilter framework provides a number of hooks into the Linux kernel networking stack to register callbacks. Figure 2 below shows the path that a packet can take when traversing the Netfilter kernel hooks [11]. These hooks allow registered kernel modules to receive a callback whenever a packet traverses that hook in the network stack.

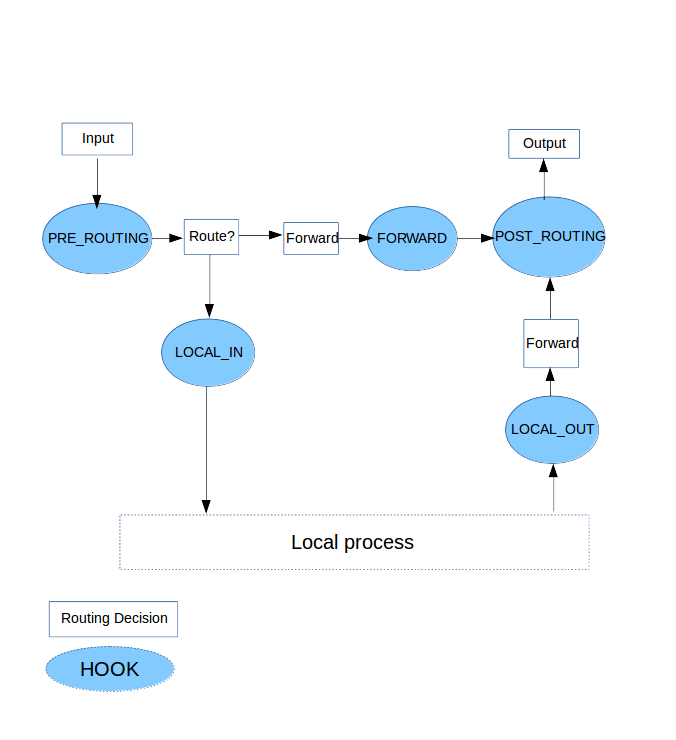


Figure 2: Netfilter Hooks

The supplied hooks are described in Table 1 below:

|  |  |
| --- | --- |
| PRE\_ROUTING | Captures all packets that arrive from a network device driver before Layer 3 processing |
| LOCAL\_IN | Captures all packets that are addressed and destined for this host after layer 3 processing |
| FORWARD | Captures all packets that are identified as a transit packet that is not addressed to this host |
| LOCAL\_OUT | Captures all packets that originated from this host before layer 3 processing |
| POST\_ROUTING | Captures all packets that are sent to a network device driver after layer 3 processing. |

Table 1: Netfilter Hooks

Using the supplied hooks it is possible to model three kinds of traffic flows [16]:

* Traffic going through the firewall, i.e. traffic not destined for the local machine
* Incoming traffic to the firewall
* Outgoing traffic from the firewall

When registering a callback for a given hook, the function prototype is defined as *nf\_hookfn* and is in the structure *nf\_hook\_ops* in the header file netfilter.h. This structure contains information about the callback including; the hook to call, the priority, and the network family.

Callbacks can return several different values, which can be interpreted by the framework as follows:

* ACCEPT: Continue processing the packet
* DROP: Discard the packet
* QUEUE: Pass the packet to the *nf\_queue* facility, and allow a user space process to handle the packet.
* STOLEN: Hold the packet so that it does not continue to traverse through the stack. This is usually used to collect de-fragmented packets.
* REPEAT: Forces the packet to re-enter the hook.

If required for capturing details about a flow of traffic, connection tracking can be enabled using the *nf\_conntrack* utility. For IPv4, this is supplied using the *nf\_conntrack\_ipv4*, and for IPv6 using *nf\_conntract\_ipv6*. When performing connection tracking, the following states are defined for a connection:

* NEW: The connection is starting.
* ESTABLISHED: The connection is established and the firewall has seen two way communication.
* RELATED: An expected connection. For example, in FTP passive mode, the control traffic and data traffic may be marked as related.
* INVALID: A special state for packets that do not follow the expected behaviour of a connection.

## User Layer

There are a number of modules and utilities that play an important role in the Netfilter architecture by providing user space utilities that can be used to manipulate hooks into the Netfilter framework. The 2 main types of the user layer programs are the *x*tables family and the newer nftables.

### xtables

The xtables (iptables, ip6tables, ebtables, and arptables) [12] family of tools allow system administrators to define tables containing chains of rules for the treatment of packets. Each table in the framework is its own hook, and each table is run in a specific order with respect to the other tables.

Chains of rules are named with predefined titles, which are similar to the Netfilter hooks defined above. The chain titles include:

* PREROUTING: Called before a routing decision is made
* INPUT: Called if the packet is going to be delivered to the local host
* OUTPUT: Called if the packet is sent from the local host
* FORWARD: Called if the packet is not for local delivery
* POSTROUTING: Called after a routing decision has been made

Table rules can also include a target or verdict to define what action to take on the packet. Packets will traverse the chain of rules in order and will continue to traverse all rules unless a target or verdict calls ACCEPT, DROP, RETURN or CONTINUE

Extensions can be written for iptables which allow a program to receive a copy of a packet and match it against an iptables rule. The use of extension matching is controlled by the “-match” switch in an iptables rule. Writing Netfilter Modules [10] shows a trivial example of writing such an xtables extension for matching against an IP address.

An example of an iptables rule on the INPUT chain, to DROP a packet from the IP address 10.1.1.1 is below

iptables -A INPUT -s "10.1.1.1" -j DROP

An example of a rule to allow all incoming SSH connections (TCP port 22) on a specific interface is below:

iptables -A INPUT -i eth0 -p tcp --dport 22 -m state --state NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -o eth0 -p tcp --sport 22 -m state --state ESTABLISHED -j ACCEPT

### nftables

The nftables [9] project aims to replace the existing *x*tables framework. It provides a new packet filtering framework, a new user space utility and also a compatibility layer for ip/ip6tables. It is built on the Netfilter framework and uses the existing hooks, connection tracking and queueing component.

Nftables adds a pseudo-state (virtual) machine [21] into kernel space. The user space utility can interpret rules provided by a user and compile it into machine bytecode that can be used to make a decision on how a packet should be handled. The benefits of this new bytecode system include:

* A reduction in complexity by simplifying the kernel ABI.
* New matching types can be added to the bytecode program.
* Faster lookups through the use of performant data structures.

As specified in the nftables HOW-TO [22], some of the main differences with *x*tables include:

* A new syntax.
* Tables and chains are fully configurable.
* There is no distinction between matches and targets.
* Multiple actions are allowed in a single rule.

As with *x*tables rules can be organised into chains, some possible base chain types are:

* filter: Used to filter packets
* route: Used to reroute packets
* nat: Used to perform network address translation.

Other custom chains can be added using the *nft* utility. The possible hooks you can use when configuring a chain are the same as those provided by Netfilter and *x*tables.

An example ruleset for a simple firewall is provided below. These show the ruleset as exported to JSON format.

table firewall {

chain incoming {

type filter hook input priority 0;

# established/related connections

ct state established,related accept

# invalid connections

ct state invalid drop

# loopback interface

iifname lo accept

# open tcp ports: sshd (22)

tcp dport {ssh} accept

# everything else

reject

}

}

### Netlink

Unlike iptables or nftables, netlink [23] [24] is not a user space program. Instead it is an interface into the Linux kernel and provides inter-process communication between kernel and user space processes. The Netfilter framework can enable communication between kernel and user space using netlink. There are libraries available which can provide programmable interfaces into this facility. An example of such a library is *libnetfilter\_queue*.

An iptables or nftables target can be configured to send a packet to a numbered NFQUEUE target. A user space program is then expected to listen on that queue using the *lignetfilter\_queue* library and handle the processing of the packet. If no user space program is configured to listen on that queue the packet is dropped.

Other Netfilter modules also rely on the netlink interface, such as the nftables framework uses netlink and *libnftnl* for interacting with the rulesets in the kernel.

# Classification

Network traffic classification can be seen as a process which identifies the application or protocol of network traffic and can potentially mark a packet based on network management policies. Classification can be performed in a number of ways including; port based classification, payload based classification, and statistical analysis based classification.

The comparison and analysis of classification techniques can involve many different techniques and quality metrics. However, many research papers use the following common principles:

* A data set is prepared which will be used for the testing. This data set may be a specialised data set prepared by the researcher, or a data set from a live network such as a university campus firewall or an internet service provider.
* The data set is analysed to classify the traffic using well known classification technique(s) or tool(s). These can use a mixture of techniques, for example; a known payload based classification tool with a high accuracy (e.g. nDPI), and port based classification for data not supported by the payload based classifier. The result of this initial classification can be known as the “ground truth” and is used as the definitive set of protocol definitions for the testing.
* The classification technique(s) are integrated into a test bed and the ground truth data set is analysed by each classifier. This can be done by either loading the captured traces (e.g. in pcap format) or by using a network simulation tool to replay the traffic.
* The results of the classification are compared against the ground truth results and common metrics [25] [7] [26] for predictive accuracy include:
  + True Positives (TP) Percentage of packets correctly classified as belonging to this protocol i.e. 100 – FN = TP%
  + False Positives (FP) Percentage of packets of another protocol classified as belonging to this protocol (for example, it may represent the percentage of RTP traffic classified as RTCP)
  + False Negatives (FN) Percentage of packets incorrectly classified as not belonging to this protocol.
  + True Negatives (TN) Percentage of packets of other classes correctly classified as not belonging to this protocol. i.e. 100 – FP = TN%
* Other importation measurements include precision and recall [25] [7]. They are represented in the formulas below:

When classifying traffic, it is important to look at what unit is being used for the classification measurement. This can typically be byte, packet, or flow. This choice can have a large bearing on how the results are portrayed. If using byte accuracy, the results are portrayed as a percentage of the number of bytes correctly classified. For packet accuracy, the results are a percentage of the number of packets correctly classified. Flow accuracy results are a percentage of the number of flows correctly classified.

Similar to the Netfilter connection tracking module, in traffic classification a flow would typically be considered the 5-tuple pair consisting of

* Source IP address
* Destination IP address
* Source port
* Destination port
* Transport layer protocol

A flow will typically consider packets going in both directions. Flows can be used in all classification methods but are particularly important for statistical analysis, where the relationships between different packets in a flow are considered.

## Port Based Classification

Port based classification (L3 / L4 based classification) is the traditional method for firewall classification and involves matching various attributes from the packet based on the L4 header and below. This can include matching the network interface, IP address, transport layer protocol (TCP/UDP) and most importantly the transport layer port.

Typically applications and protocols used well known ports and published these in the IANA list of assigned ports [2]. This list of ports is publicly available and, in some cases can be defined as part of an operating system. For instance, in various Linux operating systems the file /etc/services can associate a list of port and transport layer protocols with the well-known name for a protocol.

Over time this classification technique has become less reliable because of techniques used to bypass firewalls [3] including application layer tunnelling. There has also been an increase in protocols which do not use well defined ports, for example protocols such as SIP can be used to negotiate ports for RTP voice / video traffic. These RTP ports are often in the ephemeral range. As a result of the above, the usefulness of port based classification is decreasing and the studies by Moore et al. [4] and Madhukar et al. [5] show that between 30% - 70% of internet traffic cannot be correctly classified using this technique.

While port based classification may not be suitable for detailed classification, it may have a role to play in helping to improve the performance of payload or statistical based classification. Some payload based classification applications can have more than 100 available patterns and, to improve performance a payload based classifier could store port based information as hints to help choose the order in which to test various patterns. For example, if port 80 was detected, the pattern for HTTP or HTTP related / tunnelled protocols could be tested first.

## Payload Based Classification

Payload based classification (also known as deep packet inspection) involves using signature analysis to understand and classify applications. A signature is a unique pattern that is associated with the protocol or applications. The methods for generating a signature can involve manually analysing a protocol to define a pattern that can be matched against the L7 payload of a packet. Alternatively newer techniques can use statistical methods to examine the byte pattern and frequency in the L7 payload.

Payload based classification can be a powerful tool to help detect the application or protocol in use; however, it can be bypassed by using encryption to hide the details of the actual payload. As more and more internet traffic is using encryption, the usefulness of payload based classification may decrease. One of the other major drawbacks to traditional payload based classification is the time and effort required to create and maintain payload patterns. With the increasing proliferation of new protocols available on the internet, maintaining a full set of protocol signatures can be difficult and even a small change in a protocol structure can cause the protocol to not match a pattern. As a result of this, new versions of a protocol can be incorrectly classified.

There are a number of both commercial and open source DPI applications available that use various techniques for L7 protocol analysis and identification. Below I present a number of these applications as well as studies into their effectiveness and the effectiveness of other DPI techniques that are currently being researched.

### Payload Based Applications

Some of the common open source DPI applications include L7 Filter [17] [15], nDPI [27] [28], libprotoident [29] [30], and HiPPIE [31] [25]. These tools use various pattern based identification techniques to match a protocol signature against a database of known signatures.

In terms of commercial applications, there are many DPI implementations including iPoque PACE (formally OpenDPI) [32] and Cisco Network Based Application Recognition (NBAR) [33]. As a result of the commercial and closed source nature of these tools no further analysis of their implementation will be presented here.

**L7-Filter**

L7-Filter [17] is a traffic classifier based on the Linux Netfilter framework that can classify packets based on the application layer data. The classification is performed by comparing the L7-Payload of a packet against a regular expression that acts as a packet signature. It includes patterns for over 100 protocols including HTTP, SIP, RTP, RTSP, and RTSP tunnelled within HTTP. Current work on L7 filter seems to have stopped with the last official release and public source control changes in October 2013.

**nDPI**

nDPI [27] is an ntop maintained DPI engine that was based on the OpenDPI library. As described by Deri et al. [28] it was designed “*to be used by applications that need to detect the application protocol of communication flow*”. Protocols are known by a unique numeric protocol ID and a symbolic protocol name. Pattern matching is configured in source code and must be recompiled with each change.

It includes tools to classify pcap network traces, and to integrate nDPI into the nProbe framework to analyse NetFlow traffic. While not part of the core nDPI platform there is also a plugin to allow nDPI to work with the Linux Netfilter framework.

nDPI allows for extendible protocol pattern matching, including both packet string matching and identification of metadata strings in a protocol for greater clarity, for example the HTTP Host header or an SSL certificate server name. It includes over 100 protocols, such as RTMP, PPLive, SopCast, MPEG.

**Libprotoident**

The libprotoident website [29] describes it as a “*library that performs application layer protocol identification for flows*”. Libprotoident only uses the first 4 bytes of the payload sent in each direction, the size of the first payload packet, and the port number in what it describes as a “*lightweight packet inspection*” technique [30].

Libprotoident has support for over 250 protocols including RTSP, RTMP and SIP. It contains tools to analyse trace files and can give statistics for live traces using the lpi\_live tool. Currently there is no mechanism for using the libprotoident protocol library with the Linux Netfilter framework.

**HiPPIE**

The Hi-Performance Protocol Identification Engine (HiPPIE) [31] [25] is a Linux kernel module to analyse network traffic and determine the protocol for a given session. It has support for the Linux Netfilter frameworks and includes signatures for approximately 30 protocols including HTTP, SIP, PPStream, and RTSP. It appears to no longer be under active development.

### Payload Based Research Papers

There have been a number of research papers into payload based classification techniques. Below I present a number of these papers which include papers on technical details and papers analysing the performance and correctness of various techniques and tools.

The paper by Deri et al [28] covers the reasoning behind, design, and implementation of the nDPI classification tool. The authors state that a DPI library should have the following characteristics:

* High-reliability protocol detection.
* Extensibility for new protocols.
* An open source license for multi-application use and ability to embed in an operating systems kernel.
* The ability to extract basic network metrics.

The paper by Alcock and Nelson [30] describes the “*lightweight packet inspection*” technique used by libprotoident. This technique only inspects the first four bytes of payload, packet size, and port number. This is justified by the authors because almost all application signatures use at most 32 bytes of payload. And by only using the first four bytes it was possible to maintain the privacy of users. Comparisons were made between libprotoident and the following packet classification tools; OpenDPI, PACE, L7-Filter, and Nmap, using datasets from the University of Auckland and a New Zealand ISP. The ground truth for the accuracy was determined by the PACE inspection tool and the results of the analysis seems to show that libprotoident has a better accuracy than the port based classification in Nmap and a similar accuracy to the L7-Filter tool.

The paper by Marpaung et al. [15] looks at the use of regular expressions for pattern matching in an application layer classifier. Their system was based on the L7-Filter tool built on top of the Linux Netfilter / iptables framework. They studied the Real Time Messaging Protocol (RTMP) [34] by examining the specification from Adobe and also reverse engineering various open source projects. The results from the limited test data showed that the RTMP regular expression had a high true positive rate and, when tested against random data, there was only a small chance of a false positive.

The study by Finsterbush et al. [25] gives an analysis and comparisons between some open source classification tools including OpenDPI, nDPI, libprotoident, HiPPIE, and L7-Filter. The paper gives an overview of the classifiers and test bed. It also provides results showing the classification success rate, memory usage, and CPU usage of each tool. The authors noted that for unencrypted traffic most classifiers had high TP rates and low FP rates, with libprotoident having the highest TP rates and OpenDPI / nDPI having a high TP rate and very low FP rate, because of this the authors note that OpenDPI / nDPI may be a good tool to use while generating a ground truth data set. As well as the above most traffic was classified within the first 6 to 8 packets of a flow. In terms of CPU usage, L7-Filter is the worst of the classifiers because of the high overhead with regular expressions. nDPI comes out as the classifier which uses the lowest CPU usage but it does have high per flow memory usage.

The paper by Wang et al. [35] defines an algorithm for real-time video stream key frame identification that can be used for QoS. The principle behind the algorithm is that a video stream may use layered coding techniques that includes key frames and non-key frames. In order to increase user perceived quality in congested networks, key frames are marked with higher priority than non-key frames. The algorithm uses a L7 filter to initially mark packets as video packets and then a key frame identifier to further mark packets which contain key frames.

As noted in the paper by Finsterbush et al. [25], the regular expression pattern matching in L7-Filter causes it to be considerably more CPU intensive and slower than other DPI programs. This can also be true of other designs which use string matching or regular expressions filters and can cause throughput to suffer if live inspection is happening. As a result of this, there have been a number of research papers into how to improve the performance of pattern matching in DPI applications. These include, but are not limited to, the use of efficient parallelized design in multicore servers [36], and fast string matching techniques such as parallel bloom filters [37] [38].

The paper by Finamore et al. [26] describes a packet classification technique for UDP traffic which generates signatures using a Chi-Square test on the L7 payload. Called KISS, this technique “*allows application protocol format to emerge while ignoring protocol synchronisation and semantic rules*”. This is different from the previous payload based techniques because instead of having to manually create the payload signature, the authors examine the statistics of the bytes in the payload to automatically generate a signature from training traffic. This can speed up the generation and updating of signatures because they can be automatically derived instead of reverse-engineered. However, as with other payload based DPI techniques, it has the limitation that it is only able to classify unencrypted data. UDP was used for this research because the first bytes of a UDP packet typically contain an application layer protocol header which can identify the packet. The authors considered traces which included RTP, DNS, eMule, Skype, and P2P-TV applications such as PPLive, Joost, SopCast and Tvants. The results of the classification showed that a 99% true positive (TP) percentage can be achieved whilst still maintaining a low false positive (FP) percentage.

## Statistical Analysis Based Classification

The classification of traffic using payload based techniques can have problems classifying traffic if it is encrypted and may also have legal and privacy concerns for users. As a result of these limitations, there has been research into classification techniques using statistical analysis of the traffic flow. Network traffic classification based on statistical analysis (also known as behavioural based classification) involves attempting to classify traffic flow based on the characteristics of the connection. These characteristics of the flow can include information such as packet size, inter-arrival time, flow duration, and amount of bytes exchanged. Some statistical techniques may also attempt to classify traffic based on the connection patterns of multiple flows on a host [39].

Many of the techniques for statistical analysis rely on machine learning, in particular supervised machine learning. Supervised machine learning typically involves a training phase, where a training dataset is provided in order to build a classification model, and a testing or classifying phase, where new data is classified based on the properties learned in the training phase. For packet based classifiers, this can result in the need for two ground truth traffic sets, one to use for training the algorithm and a second set for testing the effectiveness of the algorithm.

Statistical analysis based techniques can have advantages over payload based techniques because they can work on encrypted data. However, the disadvantages and limitations of these techniques include:

* They can be computational expensive and require more memory.
* The quality of the training data used can greatly affect the outcome of the classification. In order to obtain a good training dataset, it may be necessary to use an alternative classification technique to obtain ground truth datasets.
* With some algorithms, classification can take a large number of packets to complete. For short lived traffic there may not be enough packets to complete classification. Some algorithms may not be suitable for live classification due to a requirement for the complete traffic flow [40] [41].
* Techniques that use the inter-arrival time of packets may create a classification database that is not transferable to other networks.

There is currently no open source implementation of statistical analysis based traffic classifiers that have moved past the proof of concept stage. Even the released proof of concept open source tools are limited in functionality and are only available for offline classification of captured packets. Below I present information on some current research papers which give information on statistical analysis techniques.

### Statistical Analysis Based Research Papers

The paper by Moore & Zuev [40] looks at using Bayesian analysis techniques for internet traffic classification. For their training data, they limit the algorithm to only “semantically complete TCP connections”, where semantically completed means that both the setup (3-way handshake) and teardown are included in the testing data. The authors use up to 248 flow based discriminators which include duration, TCP port, payload size, and packet inter-arrival time. The full set of discriminators is available in a separate paper [42]. Instead of classifying the traffic into individual application or protocols, the authors use a set of 10 classification categories including categories such as BULK, MAIL, WWW, P2P, and MULTIMEDIA. The authors showed that Naive Bayes kernel estimation with Fast Correlation-Based Filter on the discriminators has the most potential. For WWW and MAIL traffic, it can have a >95% accuracy. For MULTIMEDIA traffic, it can be approximately 80%, however, for P2P traffic, the accuracy was only approximately 28%. The authors noted that the correct selection of discriminators could result in a large improvement in the results of the classification. This classification technique is not suited to live classification because it requires the full network flow in order to classify the data.

The paper by Aule, Moore & Gull [41] carries on from the work in [40] but attempts to use Bayesian neural networks to try to remove some of the effect of interdependence and redundancy among certain features of each flow. The data set and classification categories were the same as used in [40]. The results show that this method is superior to the Naive Bayes method and can classify with an accuracy of up to 99% for a classification data that is from the same day as the training set and 95% for data that is from a day 8 months later than the training set. As with the algorithm in [40], this research is not suitable for live classification.

The papers by Crotti et al. [43] [44] present a flow based method which looks at the packet size, inter-arrival time, and arrival order. The authors construct a “protocol fingerprint” which is used on a simple classification algorithm based on “normalized thresholds”. The authors use a smoothing filter on the probability density function to counter the effect of noisy factors in the definition of the protocol fingerprint. They tested for HTTP, SMTP, and POP3 protocols and, after tuning of configuration parameters, were able to obtain TP percentages of greater than 90% with FP percentages of less than 7%.

The paper by Dusi et al. [45] builds on the work from [43] [44] and other work by the same authors to create a method for detecting application layer tunnels with statistical fingerprinting. The authors look at POP3, SMTP, CHAT and P2P traffic tunnelled over either HTTP or SSH. In the case of HTTP using a one class algorithm, the authors were able to archive a 98% TP result for valid HTTP traffic and a 100% TP ration for POP3, SMTP and CHAT, however, P2P only achieved an 88% TP percentage. In the case of SSH, the one class algorithm had a 99% TP for SSH and SCP but between approximately 80 – 90% TP for the tunnelled protocols. Adapting the SSH analysis to use a multi-class classification, which increases the amount of traffic seen by the classifier, the authors were able to increase the TP percentage of the tunnelled traffic to 99%, without decreasing the ability to detect legitimate traffic. This shows that statistical analysis can be used to help identify protocols even when the tunnel is encrypted.

In the paper by Wright et al. [46], the authors use statistical analysis to look at encrypted VoIP traffic and attempt to identify the language in use. This shows a more targeted approach to statistical analysis where knowing the type of traffic can cause the patterns in the traffic to reveal information about it. The authors show that when a variable bit rate codec (in this case Speex) is in use for encrypted VoIP RTP (SRTP) traffic, it can be possible to analyse the frequency of the packet size changes due to the bit rate to determine the language of the conversation. This can determine some languages with greater than 90% accuracy using a variant of a chi-squared classifier. The authors noted that using a non-length preserving encryption technique of a constant bit rate codec can significantly reduce the accuracy of their algorithm.

In the paper by Hjelmvik and John [8], they describe a Statistical Protocol IDentification algorithm (SPID). This algorithm can look at both the statistical flow based data and application layer data features, such as byte frequencies and offsets for common byte values. This would make SPID a hybrid technique of both payload based identification and statistical flow analysis. The algorithm creates a protocol model containing a set of attribute fingerprints. Once generated from training data, the protocol models are stored in XML files for later use during classification. The tested SPID algorithm uses over 30 attribute meters. Observed sessions are compared against the generated protocol models by calculating the Kullback-Leibler (K-L) divergence or relative entropy between the probability distributions of the observed sessions and the protocol model. The K-L divergence can be derived using the following formula, where P is the observed probability distribution, and Q is the protocol models’ probability distribution:

These K-L distributions can range from 0 (identical) to infinity and a lower K-L divergence results in a better match. A K-L threshold is set and if the calculated divergence is lower than this threshold, the observed session is considered part of the protocol. The paper by Hjemvik concentrates on the use of SPID on TCP traffic and preliminary results [7] showed good results with recall values of greater than 98% for BitTorrent, HTTP, and SSH. However, the eDonkey protocol had a lower recall of 77%. A proof of concept version of the SPID algorithm was developed in C# for offline classification of pcap file.

The paper by Kohnen et al. [47] looks at enhancements to the SPID algorithm. The authors re-implemented the SPID algorithm in C++ and, instead of the original XML based protocol models, the authors used a binary format to reduce the space requirements. The focus of the paper was on streaming protocols such as RTP, MPEG-TS, and RTMP over UDP. The authors used 12 attribute meters including byte-frequency from the first UDP packet in each direction, byte-frequency of the first 32 bytes of a packet, direction changes of the communication data flow, direction bytes meter, byte entropy, and a hash function of the first 4 bytes. Using the same metrics as the paper in [7], the authors were able to get 100% recall on MPEG-TS, RTP, and WMV. Other streaming protocols were able to obtain over 95% recall. However it should be noted that HTTP and BitTorrent received a lower recall value than the original SPID implementation. The authors showed that after 10 packets many protocols can be identified but for some protocols this can increase to 20 packets.

The paper by Buyukkayhan et al. [48] uses the SPID algorithm to differentiate between voice (RTP) and data traffic. The authors used only 4 attribute meters including a ByteFrequencyMeter, a PacketSizeChangeMeter, a First2BytesEqualityMeter, and a First4BytesChangeMeter. Comparing the results of the original SPID algorithm with their SPID-E algorithm, the authors were able to show that RTP recall improved from 85% to 100%, however, this also resulted in a decrease to the recall percentage of both SIP and HTTP. It should be noted that only a limited number of training and validation sessions were used in this paper.

The paper by Nguyen and Armitage [49] is a survey paper looking at the classification techniques using machine learning. The authors do not perform any independent analysis of the various techniques. However, they do present an overview of machine learning techniques and include details of many of the important criteria and challenges for those techniques.

## Summary

Port based classification can no longer classify data to satisfy the security and QoS requirements for network management except in small scale and controlled networks. However, it may still have a place in helping with the efficiency of lookups in other classification techniques.

Payload based classification can be very effective for classifying unencrypted traffic, but manually creating payload signatures is time consuming and pattern matching can be computationally expensive. Techniques such as KISS remove the need to manually create payload signatures and can be effective at classification.

Statistical based classification seems to work around the limitations of payload based classification in regards to encryption, however, it can create network specific signatures, and, in general statistical based classification can have a higher FP rate.

Solutions such as SPID, which use a hybrid approach of both examining the flow and looking at statistics of the payload, seem to offer a promising mix of both methods. For unencrypted protocols, the use of statistics on the payload should provide for high precision and recall during matching. The use of flow statistics can help with the classification of encrypted traffic.

I propose to use the SPID algorithm to generate application signatures for video streaming traffic. A number of discriminators (attribute meters) will be decided, this will be the maximum number of attribute meters that a protocol can use. During training and classification an operator can choose to only use a subset of the available attribute meters for classification of that protocol. This subset would be the attributes which best suit the requirements for a protocol.

As an example for protocol specific attribute meters, we can look at the difference between RTMP and RTMPS, where RTMPS is RTMP over TLS. As described in the coming chapter on Internet Video, RTMP includes a connection handshake where three chunks (packets) of a specified format are sent in a specific order between hosts in a flow. For RTMP, you could use the following non-exhaustive list of meters:

* Byte Frequency for full packet
* Byte Offsets
* Packet Size
* Packet sequencing in direction on the flow

For RTMPS, the Byte Frequency and Byte Offset attributes would be less important because of the use of encryption, however the packet size and flow sequencing may still be of use when choosing the payload. Here, an RTMPS signature may include the following:

* Packet Size
* Packet sequencing in direction on the flow
* Byte frequency for the first 4 bytes from the first 4 packets. This could be used to detect the TLS handshake.

As a result of the similarities of some protocols, I propose that a protocol classification is not marked as permanent until after a specific number of packets, which I will call the “defining limit”. Papers such as [25], [26], and [47], have shown that classification can be determined within 10 packets for many protocols; however, the exact limit should be configurable. Before the defining limit, a protocol below the K-L divergence threshold can be chosen as the proposed or temporary classification. If another protocol achieves a lower K-L divergence it can take the place of the original protocol. After the defining limit, the protocol with the best match below the threshold can be marked as permanent.

If a protocol has not been chosen before the defining limit, the first protocol to indicate a K-L divergence below the threshold should be marked as the chosen protocol. An upper limit known as a “cut-off limit” should also be chosen. If the number of packets is above the “cut-off limit” then the flow will be marked as unknown and no further attempt will be taken to classify the flow. This is to avoid extra overhead on the firewall. The exact figure for this cut-off limit is still to be determined using testing however as a starting point I propose to initially use at least 100 packets.

The defining limit may help to avoid false positives, but it could increase the classification time and the computational requirements of the algorithm. The reason behind this is to have the algorithm look at all protocols until the limit, instead of classifying when a single protocol goes below the K-L threshold.

After the defining limit has been passed, if a protocol has not been classified, I propose to use a port based approach to help sort the order in which classification is performed. Pseudocode showing an example algorithm for the above would be:

classifer(port=80, protocol=TCP, flow, payload)

{

List<protocol> protocols = getOrder(port, protocol)

foreach(p in protocols)

{

Bool matched = protocol->doClassification(port,protocol,flow, payload)

if (matched)

Return protocol->id

}

}

In the above, the getOrder function for port 80 and protocol TCP could return a list in an order as follows:

1. RTMPT (RTMP Tunnelled over HTTP)
2. HTTP-MPEG
3. HTTP-WebM
4. HTTP
5. …..

Where RTMPT is RTMP tunnelled over HTTP, HTTP-MPEG is MPEG streamed over HTTP, HTTP-WebM is the WebM codec streamed over HTTP. If a match is not found within the ordered protocols all other protocols would be tried in their pre-defined order. The standard pre-defined order may be network specific based on the protocols in use on that network. In the event that two protocols are very similar or tunnelled (e.g. HTTP and HTTP-MPEG), the tunnelled or more specific of the protocols should be attempted first. This rule should help to avoid false positives with the outer protocol in a tunnel.

# Internet Video

Internet video is increasingly becoming one of the dominant types of IP traffic available on the internet. As part of their Visual Networking Index [1], Cisco has predicted that IP video traffic will account for 79% of IP traffic by 2018. Cisco defines a number of types of video traffic and they can be divided into the following:

* Short form: Video clips generally less than 7 minutes in length (e.g. YouTube [50])
* Long form: Video content generally greater than 7 minutes (e.g. Netflix [51])
* Video Calling: Video messaging or calling
* Live Internet TV: Peer-to-Peer TV or live television streamed over the internet
* Ambient Video: Web enabled cameras such as security cameras, or other persistent cameras.

The Cisco VNI report also looks at the symmetry of Internet video and concludes that most forms on internet video do not have large upstream components, and it is likely that residential upstream traffic will remain asymmetric. The main suppliers of symmetric video traffic include P2P content providers such as SopCast [52], and high end video communications such as high definition PC-to-PC video calling.

## Video Streaming Techniques and Protocols

As mentioned above, there are a number of different types of video streaming categories and these categories can include various different protocols for playback control, streaming video encapsulation, and video encoding. Below I present a number of the major techniques and protocols in use.

### Real Time Messaging Protocol

Real Time Messaging Protocol (RTMP) [34] is a protocol for transmission of audio and video over a reliable stream transport, such as TCP. Originally developed by Macromedia as a proprietary protocol, it is now available as an open specification.

The protocol includes an RTMP chunk stream which provides guaranteed timestamp-oriented, end-to-end delivery of all messages. RTMP includes a handshake at connection initialisation and this handshake consists of three static-sized chunks. Each chunk has a specific format and the handshake process is performed as shown in Figure 3 [34].

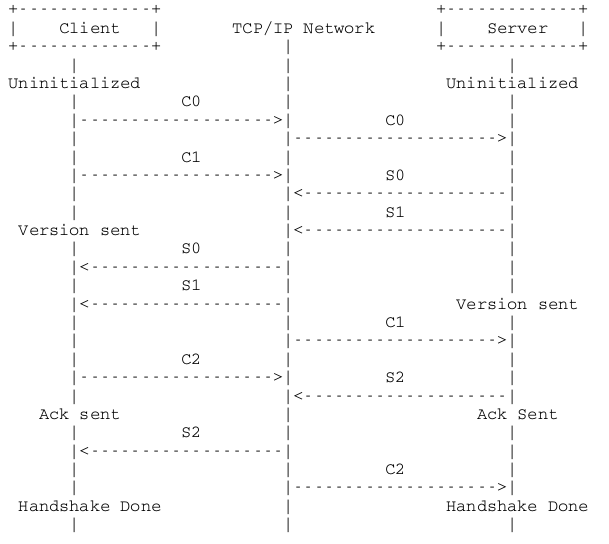


Figure 3: RTMP Handshake

After connection setup via the above handshake, the connection multiplexes one or more chunk streams, which each carry messages of one type from one message stream. A chunk consists of a basic header which can include a stream ID (in the cs id field), chunk type (in the fmt field), and the chunk message header which has four types and is determined by the fmt field in the basic header. The chunk header includes information such as message stream id, message length, timestamp, and message type id. The message type id defines where the message contains multimedia data or control / command messages. Command message are encoded using the Action Message Format.

There are a number of variants of RTMP to provide additional functionality. These include:

* RTMPS – RTMP over TLS/SSL
* RTMPE – Encrypted RTMP.
* RTMPT – RTMP tunnelled within HTTP.

Marpung et al. [15] show an example process for reverse engineering the RTMP protocol to generate regular expression syntax for the L7-Filter protocol. The generated regular expression is in the current implementation in the L7-Filter source code and is below:

^\x03.+\x14.+\x02.+\x07.(connect)?.+(app)?

This shows that RTMP can be a good candidate for payload based classification because of the predictable header of the chunk and the handshake at the start of every connection. This handshake could possibly provide good predictable measurements that a flow based statistical method could use as part of the signature.

### Real-Time Transport Protocol

The Real-Time Transport Protocol (RTP) [53] provides end-to-end delivery services for audio and video traffic over IP networks. RTP provides application level framing and the protocol includes payload identification, sequence numbering and time-stamping. RTP is transport layer agnostic and can be used with any transport layer protocol, however it typically used with UDP. It does not provide any means to control the signalling of a multimedia session and this control is typically provided by a signalling protocol such as SIP with Session Description Protocol (SDP), Real Time Streaming Protocol (RTSP), or H.323. Real-Time Transport Control Protocol (RTCP) is a complementary protocol to RTP to provide data relating to the transmission statistics and QoS of packet delivery.

The basic RTP header has the format as shown in Figure 4 [53]:

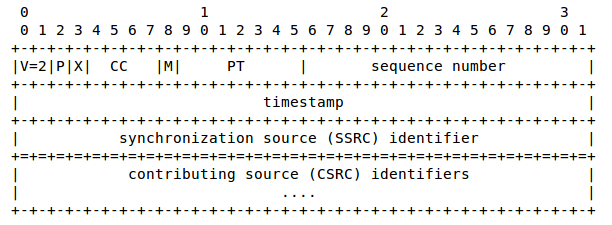


Figure 4: RTP Header

The first 12 octets are present in every RTP packet while the list of CSRC identifiers is optional. Some of the header fields are presented below:

* Version (V): 2 bits representing the RTP version. RFC 3550 defines the version as 2.
* Payload Type (PT): 7 bits identifying the format of the RTP payload. A set of default mappings are defined in RFC 3551 [54].
* Marker (M): 1 bit that allows profile specific events to be marked in the packet stream, for example, frame boundaries.
* Sequence Number: 16 bits that increment by one for each RTP data packet sent. Allows the receiver to reorder packets and detect packet loss.
* Timestamp: 32 bits reflecting the sampling instant of the first octet in the RTP data packet

An RTP header may include extension headers, but the RFC notes that this is intended only for limited usage. If an extension header is included, the X bit must be marked in the packet header.

The payload formats for specific video encoding techniques are often specified in IETF RFCs. For example RFC 6184 [55] defines the RTP Payload Format for H.264 video. There are also draft RFCs for the VP8 [56] and VP9 [57] codecs.

Secure RTP (SRTP) [58] defines an extension payload to RTP, that can allow for encrypted payloads. In SRTP, the default RTP header is still unencrypted on the wire.

The authors of [26] and the protocol definition in L7-Filter [17] note that due to the short and compact size of the RTP header, it can be difficult to generate a pattern to match RTP traffic. For example, while testing the KISS [26] classifier, during the classification phase a single difference in the version field caused the classifier to miss all RTP packets from a particular endpoint.

### Peer-to-Peer TV

There are a number of available P2P-TV protocols available such as SopCast [59], PPTV (previously PPLive) [60], PPStream [61]. Many of these protocols originate from China and are proprietary in nature. As a result of the proprietary nature of these protocols, it can be difficult to generate patterns to classify these protocols. Despite this difficulty, nDPI includes patterns for all three mentioned protocols, L7-Filter includes patterns for PPTV, and libprotoident includes patterns for PPTV and PPStream. These patterns seem to have been obtained by manually examining packet traces and reverse engineering the signature.

From a brief examination of the patterns available (via the source code of nDPI and libprotoident), these protocols appear to be binary protocols which include fixed packet lengths. nDPI includes definitions for both TCP and UDP while libprotoident only includes definitions for UDP traffic.

There are a number of papers [62] [52] which classify the client behaviour, bytes exchange and connection patterns of the various protocols, however there is limited data available on the design of the actual protocol.

The authors of the KISS classifier specifically state that the classification of P2P-TV was a goal in the design of their classifier. The use of statistical methods to analyse the payload removes the need for manual reverse engineering of the protocol and the results of the paper showed good classification results.

### HTTP Video Streaming

HTTP video streaming could be considered the streaming of video content embedded in web pages, which includes sites such as YouTube. In the past, HTTP streaming services used proprietary containers such as Microsoft Sliverlight [63] or Adobe Flash [64] to stream video traffic, however many sites are now looking to HTML5 [65] as the future of web streaming [66].

In a paper by Rao et al. [67], the video streaming techniques of YouTube and Netflix are examined to look for the network characteristics in use. The authors observed that streaming video from both companies, irrespective of the container, follows a pattern of a buffering phase, where initially video is sent as fast as the connection allowed, and a steady state phase, where the download rate is slightly larger than the video encoding rate.

**HTML5**

The proposed HTML5 standard has added a <video> tag which allows websites to embed a video within a HTML page. The HTML5 standard does not specify the use of any particular video codec and instead leaves the choice of video codec as an implementation decision for browsers and web developers. Some of the primary desktop browsers currently support subsets of the following codecs and format containers, H.264, Theora in an Ogg container, and VP8 / VP9 in a WebM container. An instance of the video tag may include multiple “src” attributes each specifying the address with which to access a different video codec.

The HTML5 specification also defines a standardised JavaScript API to allow the access and control of the video, for example, play, pause, and seek. While this API can be used to control the playback in the browser, HTML5 does not mandate any method for the actual streaming of the video content and there a number of techniques available. The paper by Daoust et al [68] gives an overview of these techniques, which include HTTP Progressive Deliver, HTTP Streaming, and HTTP Adaptive Streaming.

HTTP Progressive Delivery is the simplest way to deliver video. It involves using regular HTTP to serve the video content. HTTP Streaming involves using HTTP progressive delivery with an additional method to let users jump to a specific position in the video at any time. HTTP Adaptive Streaming allows the bitrate to be changed based on network conditions; this can include the use of techniques such as MS Smooth Streaming Protocol (MS-SSTR) [69] or Dynamic Adaptive Streaming over HTTP (DASH) [70].

An extension to the HTML5 specification known as Encrypted Media Extensions has been created to enable the HTML5 video tag to play digital rights management encrypted content. The extension is currently in use by content providers such as Netflix to enable streaming of copyright protected videos such as TV shows and movies.

# Conclusions and Project Outline

There are many different methods and protocols used for sending video streams over the internet. These techniques can include UDP based RTP, and TCP based adaptive streaming over HTTP. As a result of this variation in protocols, any classification method should be adaptable and able to look at different transport layer protocols.

While many video streaming techniques do not specify encryption as a requirement, it is often an optional component of these techniques (e.g. HTML5 Encrypted Media Extensions, and SRTP). Increasingly companies are moving to having the optional encryption elements of streaming protocols enabled by default. This can lead to problems for payload based classification techniques. The overhead from the encryption of the data may also affect the flow attributes of the traffic, such as packet size. With this in mind, I believe that protocols which support transport via encrypted means should be included twice in a classifier, once for the unencrypted variant and once for the encrypted variant of the protocol.

I propose that the use of the Statistical Protocols IDentification (SPID) algorithm may be a valid technique in detecting internet streaming of video. The SPID algorithm can be configured to use statistical values from the data flow and application layer data. Research into the algorithm has shown positive results for classification of both encrypted and unencrypted data. The paper by Kohnen et al. [47] looked at using the SPID algorithm for classification of RTP, RTMP and MPEG-TS and resulted in positive results. I believe it can be further enhanced to include other video streaming techniques.

The current implementation of the SPID algorithm is written in C# and can only be used for offline classification of pcap network traces. I plan to implement the algorithm in the C and / or C++ programming language and integrate this into the Netfilter framework in the Linux operating system in order to perform live classification of internet traffic. The iptables firewall could be used to divert packets to user space where a process can use the *libnetfilter\_queue* mechanism to receive the packet and classify it via the SPID algorithm. This live classification could be used to either mark a classified packet with a DiffServ code in the DS Field [71] of the IP header. Alternatively it could be used to log the path of the packet through the firewall.

A separate training program will need to be implemented that can take in packet captures of training data and produce the classification database. I aim to create this program in either C or C++ and store the classification database in a format that is easily accessible in any programming language. This may take the form of the XML structures outlined in the original research or a custom format.

In the current research and implementation of the SPID algorithm, all protocols use the same discriminators or attribute meters for their classification. I would like to make an addition to the implementation, in which a maximum set of discriminators are passed to the algorithm and each individual protocol can be configured to only use the discriminators that it deems necessary for a valid match. As an example of this technique, taking into account that protocols that support encryption should be included in both encrypted and unencrypted variants, would be the RTP protocol. The classification of RTP may be possible by examining packet size, the application layer byte statistics for the RTP header, and the application payload header (e.g. H.264). SRTP would be unable to use all the application layer bytes because the payload is partially encrypted. As a result of this, it might rely on only the first 4 bytes of the payload, the packet size, and the inter-arrival time.

An important aspect of the SPID algorithm is the training of the classification database in order to obtain the probability distributions for the observed sessions from the training program. In order to obtain this training data, I plan to look at capturing traffic using the packet capture tool tcpdump. A program may be configured to automatically start a packet capture and then start to play a video from a video streaming server. The resulting pcap may then be saved and later used as feed in data into the training program. Some captures may also be taken by manually starting the video stream, for example, if the proposed video is RTP from a VoIP device. A video application such as VLC [72] may be used for the playback because it supports various streaming technologies and can be used not only as a video client to accept a stream but also as a video server to stream video. The video streams to be tested could take a number of forms but at a minimum I plan to include RTP, RTMP, RTSP and a HTTP based video stream.

During the classification phase, network traffic including video must be sent through the firewall in order to correctly verify that the implementation can classify packets and flows into the correct protocol. For this, a similar method as the one outlined for the collection of the training data may be used. This could involve a video stream being automatically started and a capture taken to show the stream as it flows through the firewall. A log could also be used to show the “flow details” and the classification result. This would then be manually matched against the capture to confirm if the packet was correctly classified.

References

[1] Cisco, (2014, February). Cisco Visual Networking index: Global Mobile data Traffic Forecast Update, 2013-2018. Avaliable: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white\_paper\_c11-520862.pdf. (Accessed: December 2015).

[2] Touch, J., Kojo, M., Lear, E., Mankin, A., Ono, K., Stiemerling, M., & Eggert, L. (2013). Service Name and Transport Protocol Port Number Registry. The Internet Assigned Numbers Authority (IANA).

[3] Hill, J. (2000, March). Bypassing firewalls: tools and techniques. In 12th Annual FIRST conference, chicago, IL, uSA.

[4] Moore, A. W., & Papagiannaki, K. (2005). Toward the accurate identification of network applications. In Passive and Active Network Measurement (pp. 41-54). Springer Berlin Heidelberg.

[5] Madhukar, A., & Williamson, C. (2006, September). A longitudinal study of P2P traffic classification. In Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on (pp. 179-188). IEEE.

[6] Recommendation ITU-T Y.2770 (11/2012), Requirements for deep packet inspection in next generation networks

[7] Hjelmvik, E., & John, W. (2009, May). Statistical protocol identification with spid: Preliminary results. In Swedish National Computer Networking Workshop.

[8] Hjelmvik, E. (2008). The spid algorithm-statistical protocol identification. Gävle, Sweden.

[9] Welte, H., & Ayuso, P. N. (2014) The netfilter.org project [Online]. Available: http://www.netfilter.org. (Accessed: December 2014)

[10] Engelhardt, J., & Nicolas B. (2012, July). Writing Netfilter Modules. Available: http://inai.de/documents/Netfilter\_Modules.pdf. (Accessed: December 2014)

[11] Ye, X., & Zeng, B. (2010, August). Method of handling the contents of TCP packets on router via netfilter. In Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on (Vol. 1, pp. 258-261). IEEE.

[12] Welte, H., & Ayuso, P. (2014) The netfilter.org "iptables" project [Online]. Available: http://www.netfilter.org/projects/iptables/index.html (Accessed: December 2014)

[13] Welte, H., & Ayuso, P. (2014) The netfilter.org "nftables" project [Online]. Available: http://www.netfilter.org/projects/nftables/index.html. (Accessed: December 2014)

[14] Douglas Blansit, B. (2009). Firewalls: basic principles and some implications. Journal of Electronic Resources in Medical Libraries, 6(3), 260-269.

[15] Marpaung, J. A., Bhakti, M., & Yazid, S. (2013, December). A Study on Application Layer Classification for Firewalls Using Regular Expression Matching. In Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on (pp. 417-422). IEEE.

[16] Ayuso, P. (2006). Netfilter’s connection tracking system. LOGIN: The USENIX magazine, 31(3).

[17] Clearfoundation. (2013). L7-filter [Online]. Available: http://l7-filter.clearfoundation.com/. (Accessed: January 2015)

[18] Arafat, M. Y., Alam, M. M., & Ahmed, F. A Realistic Approach and Mitigation Techniques for Amplifying DDOS Attack on DNS.

[19] Fail2Ban. (2014). Fail2Ban [Online]. Avaliable: http://www.fail2ban.org/ (Accessed: January 2015)

[20] Engelhardt, Jan. (2014) Netfilter Components [Online]. Avaliable: http://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Netfilter-components.svg/800px-Netfilter-components.svg.png. (Accessed: January 2015)

[21] Märdian, Lukas M. "What’s New in the Linux Network Stack?." intervals 2 (2014): 3.

[22] Lebond, E. (2014). nftables HOW-TO [Online]. Available: http://wiki.nftables.org/wiki-nftables/index.php/Main\_Page. (Accessed: December 2014)

[23] Horman, N. (2004). Understanding and programming with netlink sockets. Available: http://people.redhat.com/nhorman/papers/netlink.pdf. (Accessed: January 2015)

[24] Neira‐Ayuso, P., Gasca, R. M., & Lefevre, L. (2010). Communicating between the kernel and user‐space in Linux using Netlink sockets. Software: Practice and Experience, 40(9), 797-810.

[25] Finsterbusch, M., Richter, C., Rocha, E., Muller, J., & Hanssgen, K. (2014). A Survey of Payload-Based Traffic Classification Approaches.

[26] Finamore, A., Mellia, M., Meo, M., & Rossi, D. (2010). KISS: stochastic packet inspection classifier for UDP traffic. Networking, IEEE/ACM Transactions on, 18(5), 1505-1515.

[27] ntop. (2015) nDPI – Ntop [Online]. Available: http://www.ntop.org/products/ndpi/ (Accessed: January 2015)

[28] Deri, L., Martinelli, M., Bujlow, T., & Cardigliano, A. (2014, August). nDPI: Open-source high-speed deep packet inspection. In Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International (pp. 617-622). IEEE.

[29] WAND Network Research Group (2014) libprotoident [Online]. Available: http://research.wand.net.nz/software/libprotoident.php (Accessed: January 2015)

[30] Alcock, S., & Nelson, R. (2012). Libprotoident: Traffic Classification Using Lightweight Packet Inspection. WAND Network Research Group, Tech. Rep.

[31] Ballard, J. (2013) Hi-Performance Protocol Identification Engine/HiPPIE [Online]. Available: http://sourceforge.net/projects/hippie/ (Accessed: January 2015)

[32] iPoque. (2015) PACE: Network Analysis with Layer-7 Deep Packet Inspection [Online]. Available: http://www.ipoque.com/en/products/pace. (Accessed: January 2015)

[33] Cisco (2015). Network Based Application Recognition (NBAR) [Online]. Available: http://www.cisco.com/c/en/us/products/ios-nx-os-software/network-based-application-recognition-nbar/index.html. (Accessed: January 2015)

[34] Parmar, H., & Thornburg, M. (2012, December). Adobe’s Real Time Messaging Protocol. Available: https://www.adobe.com/content/dam/Adobe/en/devnet/rtmp/pdf/rtmp\_specification\_1.0.pdf. (Accessed: January 2015)

[35] Wang, R., Wang, W., Gong, X., Que, X., & Ma, J. (2010, April). A Real-Time Video Stream Key Frame Identification Algorithm for QoS. In Multimedia and Information Technology (MMIT), 2010 Second International Conference on (Vol. 1, pp. 115-118). IEEE.

[36] Guo, D., Bhuyan, L. N., & Liu, B. (2012). An efficient parallelized L7-filter design for multicore servers. Networking, IEEE/ACM Transactions on, 20(5), 1426-1439.

[37] Dharmapurikar, S., Krishnamurthy, P., Sproull, T., & Lockwood, J. (2003, August). Deep packet inspection using parallel bloom filters. In High Performance Interconnects, 2003. Proceedings. 11th Symposium on (pp. 44-51). IEEE.

[38] Yu, F., Chen, Z., Diao, Y., Lakshman, T. V., & Katz, R. H. (2006, December). Fast and memory-efficient regular expression matching for deep packet inspection. In Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems (pp. 93-102). ACM.

[39] Karagiannis, T., Papagiannaki, K., & Faloutsos, M. (2005, August). BLINC: multilevel traffic classification in the dark. In ACM SIGCOMM Computer Communication Review (Vol. 35, No. 4, pp. 229-240). ACM.

[40] Moore, A. W., & Zuev, D. (2005, June). Internet traffic classification using bayesian analysis techniques. In ACM SIGMETRICS Performance Evaluation Review (Vol. 33, No. 1, pp. 50-60). ACM.

[41] Auld, T., Moore, A. W., & Gull, S. F. (2007). Bayesian neural networks for internet traffic classification. Neural Networks, IEEE Transactions on, 18(1), 223-239.

[42] Moore, A., Zuev, D., & Crogan, M. (2005). Discriminators for use in flow-based classification. Queen Mary and Westfield College, Department of Computer Science.

[43] Crotti, M., Dusi, M., Gringoli, F., & Salgarelli, L. (2007). Traffic classification through simple statistical fingerprinting. ACM SIGCOMM Computer Communication Review, 37(1), 5-16.

[44] Crotti, M., Dusi, M., Este, A., Gringoli, F., & Salgarelli, L. Application Protocol Fingerprinting for Traffic Classification.

[45] Dusi, M., Crotti, M., Gringoli, F., & Salgarelli, L. (2009). Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. Computer Networks, 53(1), 81-97.

[46] Wright, C. V., Ballard, L., Monrose, F., & Masson, G. M. (2007, August). Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?. In USENIX Security (Vol. 3, No. 3.6, p. 3).

[47] Kohnen, C., Uberall, C., Adamsky, F., Rakocevic, V., Rajarajan, M., & Jager, R. (2010, August). Enhancements to Statistical Protocol IDentification (SPID) for Self-Organised QoS in LANs. In Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on (pp. 1-6). IEEE.

[48] Buyukkayhan, A. S., Kavak, A., & Yaprak, E. (2013, November). Differentiating voice and data traffic using statistical properties. In Electronics, Computer and Computation (ICECCO), 2013 International Conference on (pp. 76-79). IEEE.

[49] Nguyen, T. T., & Armitage, G. (2008). A survey of techniques for internet traffic classification using machine learning. Communications Surveys & Tutorials, IEEE, 10(4), 56-76.

[50] YouTube, LLC. (2015). YouTube [Online]. Available: https://www.youtube.com/ (Accessed: January 2015)

[51] Netflix, Inc. (2015). Netflix [Online]. Available: http://www.netflix.com/ (Accessed: January 2015)

[52] Horvath, A., Telek, M., Rossi, D., Veglia, P., Ciullo, D., Garcia, M. A., ... & Mellia, M. (2008). Dissecting pplive, sopcast, tvants. submitted to ACM Conext.

[53] Jacobson, V., Frederick, R., Casner, S., & Schulzrinne, H. (2003). RTP: A transport protocol for real-time applications. RFC 3550

[54] Schulzrinne, H., and Casner, S. (2003). RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551

[55] Wang, Y., Even, R., Kristensen, T., and Jesup, R. (2011). RTP Payload Format for H.264 Video. RFC 6184

[56] Westin, P., Lundin, H., Glover, M., Uberti, J. and Galligan, F. (2014). RTP Payload Format for VP8 Video. Draft-ietf-payload-vp8-13

[57] Uberti, J., Holmer, S., Flodman, M. and Lennox, J. (2014). RTP Payload Format for VP9 Video. Draft-uberti-payload-vp9-00

[58] Baugher, M., McGrew, D., Naslund, M., Carrara, E., & Norrman, K. (2004). The secure real-time transport protocol (SRTP). RFC 3771

[59] SopCast. (2007). SopCast - Free P2P Internet Tv [Online]. Available: http://www.sopcast.org/ (Accessed: January 2015)

[60] PPLive Inc. (2014) PPLive [Online]. Available: http://www.pptv.com/ (Accessed: January 2015)

[61] PPStream. (2009). PPStream [Online]. Available: http://www.pps.tv/en/. (Accessed: January 2015)

[62] Borges, A., Gomes, P., Nacif, J., Mantini, R., Almeida, J. M., & Campos, S. (2012). Characterizing SopCast client behavior. Computer Communications, 35(8), 1004-101

[63] Microsoft. (2015). Microsoft Silverlight [Online]. Avaiable: https://www.microsoft.com/silverlight/. (Accessed: January 2015)

[64] Adobe Systems Incorporated. (2008). Adobe Flash Video File Format Specification, 10.1 edition

[65] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E. D., O’Connor, E., Pfeiffer, S. HTML 5, A vocabulary and associated APIs for HTML and XHTML. W3C Recommendation (October 28, 2014).

[66] Leider, R. (2015, January). YouTube now defaults to HTML5 <video> [Online]. Available: <http://youtube-eng.blogspot.jp/2015/01/youtube-now-defaults-to-html5_27.html>. (Accessed: January 2015)

[67] Rao, A., Legout, A., Lim, Y. S., Towsley, D., Barakat, C., & Dabbous, W. (2011, December). Network characteristics of video streaming traffic. In Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies (p. 25). ACM.

[68] Daoust, F., Hoschka, P., Patrikakis, C. Z., Cruz, R. S., Nunes, M. S., & Osborne, D. S. (2010). Towards Video on the Web with HTML5. NEM Summit, 6.

[69] Microsoft Corporation. Smooth Streaming Protocol Specification, v20140502. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-SSTR%5D.pdf (May 15, 2014)

[70] ISO/IEC 23009-1:2012, Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.

[71] Nichols, K., Black, D. L., Blake, S., & Baker, F. (1998). Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers.

[72] VideoLan Organisation. (2014). VideoLAN - VLC Official Site [Online]. Available: http://www.videolan.org. (Accessed: January 2015)