Communications Research Centre Canada
An Agency of Industry Canada

Centre de recherches sur les communications Canada
Un organisme d'Industrie Canada

Flow Attributes for Use in Traffic Characterization

Annie De Montigny-Leboeuf

CRC Technical Note No. CRC-TN-2005-003
Ottawa, December 2005

Canada

CRC

Communications Research Centre Canada
An Agency of Industry Canada

Centre de recherches sur les communications Canada
Un organisme d'Industrie Canada

Flow Attributes for Use in Traffic Characterization

Annie De Montigny-Leboeuf

CRC Technical Note No. CRC-TN-2005-003
Ottawa, December 2005

Canada

CRC

Flow Attributes For Use In Traffic Characterization

Annie De Montigny-Leboeuf

CRC Technical Note No. CRC-TN-2005-003
Ottawa, December 2005

Acknowledgements

This work was funded by the Communications Research Centre Canada (CRC) and conducted within the CRC laboratory dedicated to research in network security. The author would like to thank the Royal Military College of Canada (RMC), more specifically Scott Knight and his team, for making the HTTP Reverse Tunnel tool available to us to test against. I would like to thank my co-workers, Frédéric Massicotte and Tim Symchych, for their ideas, support and valuable feedback. I am also grateful to the people of the Information Networks and Systems team (DINFO) at CRC for their help in collecting data for evaluation of the approach.

Abstract

Attackers disguise their activities in order to evade detection and circumvent network security measures. The work presented in this document builds upon earlier work on traffic profiling to reveal the nature of a flow based on its behaviour. An important step, which is the focus of the document, consists of identifying relevant and discriminative flow attributes for use in traffic characterization. We have developed a number of indicators that portray essential communication dynamics, based solely on information that can be gathered from monitoring packet headers. The indicators are lightweight and the characteristics measured can be interpreted from domain knowledge. A tool is under development at the Communications Research Centre Canada to demonstrate the relevance of the flow attributes in characterizing network traffic. In particular, the tool includes the capability to describe the traffic and recognize a number of ubiquitous protocols. Several of the protocols we experimented with are in essence very similar, but were found to be distinguishable with the indicators presented herein. Preliminary assessment shows us that the derived tool is useful as is, and may lead with further research to a number of applications.

Résumé

Les attaquants déguisent leurs activités afin d'échapper à la détection et contourner les mesures de sécurité de réseau. Le travail présenté dans ce document se fonde sur des travaux de recherche visant à révéler la nature des flots de trafic en fonction de leur comportement. Une étape importante, qui est le point focal de ce document, consiste à identifier les propriétés adéquates et distinctives des flots de trafic dans le but de faire la caractérisation du trafic. Nous avons développé un certain nombre d'indicateurs qui exposent les dynamiques essentielles des communications, et ce en fonction seulement d'informations disponibles dans les en-têtes des paquets. Les indicateurs sont simples et les caractéristiques mesurées sont interprétables par un humain. Un outil est en cours de développement au centre de recherches sur les communications (CRC) pour démontrer la pertinence des attributs dans la caractérisation du trafic de réseau. En particulier, le prototype comprend une fonction permettant de décrire le trafic et de reconnaître divers protocoles omniprésents dans les réseaux. Plusieurs des protocoles avec lesquels nous avons expérimentés sont essentiellement très semblables, mais se sont avérés différentiables grâce aux indicateurs présentés dans ce document. Une évaluation préliminaire nous montre que l'outil est utile tel quel et peut mener, avec davantage de recherche, à un certain nombre d'applications.

Table of contents

List of figures

List of tables

1 Introduction

Thousands of networked applications flow daily over networks used by governments, industry, and private users. Attacks can be hidden within these flows by disguising malicious network traffic to appear to be legitimate. Generally, the TCP or UDP port numbers over which communication is established can be mapped to specific networked applications. For instance TCP port 80 is usually associated with HTTP. However, it must be assumed that intruders will masquerade unauthorized activity by using non-standard ports or standard ports in non-standard ways to avoid detection. Authorized internal users may also attempt to hide activities that violate organizational network security policy. The authors of [1] describe some well illustrated scenarios by which malicious internal or external users can attempt to disguise the traffic.

We note here that it is not necessary to be a skilled hacker to disguise traffic. Peer-to-peer applications are easy to deploy and many include disguise capabilities [2]. There also are inexpensive software programs [3] users can installed on their PC to access services otherwise blocked by the firewalls.

Developing a dependable method for classifying flows is a difficult problem that requires extensive research. We no longer can reliably identify applications based on the port numbers now that a growing number of applications have the ability to disguise their activity through the use of arbitrary ports. While payload analysis is a possible approach, it can be resource intensive if exhaustive payload examination is performed; or easily defeated if only minimal decoding is done. Moreover, payload decoding methodologies can only identify protocols for which we have *a priori* knowledge. On the other hand, behavioural analysis can recognize communication patterns. When the methods do not rely on payload analysis, they remain applicable even in cases where the application layer is unavailable (or not visible).

There is a pressing need for alternatives to correctly identify network activities. The Network Security Research Group at the Communication Research Centre (CRC)[1] is working to identify flows of information that disguise abuses and attacks. Research challenges include uncovering unauthorized activities in high-speed, high-volume networks and within protocols that obscure the details of the information carried. In this work, we address the problem of behavioural traffic characterization. The ultimate goal is to develop techniques that can be implemented in a tool that will help identify the network activities and provide useful insight into the traffic. Such a tool may particularly be useful in recognizing the aftermath of a successful attack, when other methods to detect and prevent intrusions have failed. The more insight we have into network flows, the better we can perceive telltale signs of suspicious activities.

We had set the following criteria as guidelines for our approach:

- **Lightweight measurements**. Develop traffic indicators derived with minimal computational complexity.
- **Interpretable measurements**. The characteristics and their representations must be meaningful.
- **Alternative to payload decoding**. The approach must not rely on having access to the payload of packets.

A survey of related research on traffic classification indicated that most approaches that do not rely on payload analysis take as input basic flow metrics (e.g. average packet size, flow duration, recurring use of addresses/ports). These approaches focus mainly on the classification algorithms. To further increase the distinguishing capabilities of non-payload analysis, we felt it was necessary to go back a step and identify other characteristics of traffic behaviour, and derive appropriate ways to measure these characteristics. Therefore the primary issue we chose to address was the identification of relevant flow attributes by which a great variety of network applications could be distinguished. Our main contribution to the research is the identification of about 40 flow features with discriminative power. Not only are the flow features discriminative, but they reveal the behaviour of the flows and thus can help an analyst investigate unknown flows. These features were constructed following a domain knowledge approach and thus can be interpreted meaningfully. We feel it is very important to incorporate domain knowledge to direct feature selection in order to determine the pertinence of an observed pattern.

To demonstrate the relevance of the flow attributes in characterizing networked applications, we developed criteria to recognize applications and describe the flow activity. While broad classes of traffic can be defined based on our flow features, we find that the discriminative power of those features is strong enough to distinguish among similar traffic flows (e.g. distinguish among e-mail protocols). We have developed a toolset based on these indicators to help identify the network activities, and if traffic is not recognized, then to provide useful insight into the traffic behaviour. We provide this useful insight by means of a short description of the traffic behaviour, aiming to reveal the way a networked application is used.

Because we start with discriminative features, only minimal effort was required to derive the profiles of a number of ubiquitous protocols. This has convinced us that when flow attributes are meaningful and discriminative, lightweight rule sets can be defined to classify flows. As will be presented later in the document, traffic is currently classified using simple rule

[1] The Communications Research Centre is an Agency of Industry Canada and is located in Ottawa, Ontario, Canada <www.crc.ca>.

sets derived from the flow features. Alternative classification algorithms could also be used if these are shown to be more accurate. Developing the classification rules was not the focus of the project at this present stage, therefore the rules provided in this document should only be viewed as a starting point for further research.

The method presented herein does assume that the monitor sees every packet (or at least most packets) in both directions of a communication. For this reason the methodology is typically more appropriate when the monitor is located at the edge of a network as opposed to somewhere in the core of the internet, where packets of a given flow may take various paths not necessarily visible to the monitor.

The remainder of the document is organized as follows. Section 2 provides background on traffic flows and specifies our definition of a flow. Section 3 discusses related work on flow attribute selection and traffic characterization. Section 4 represents the core of the document and describes how we derived the flow attributes. To evaluate the usefulness of the flow attributes identified, we developed lightweight criteria to recognize a number of ubiquitous protocols as presented in Section 5. Sections 6 and 7 discuss results obtained from network traffic generated by a subversive tool. Section 8 describes how the flow features can also be used to not only classify the traffic but also to provide a human interpretable description of the traffic. Section 9 describes the current state of this study and future work, and finally we present conclusions in Section 10.

2 Background

To provide the reader with some background on flows and their measurements, this section briefly describes the notions of flows, expiry mechanisms, and flow attributes.

2.1 Notion of Flow

When capturing packet traffic, certain tools, such as tcpdump, write copies of packets (or perhaps just their headers) to a "trace" file on disk. Since packet trace files contain data for every packet, they can be analysed in different ways, yielding insight into different aspects of traffic behaviour. A disadvantage is that the files can easily be very large, and generally require further data processing to reveal pertinent characteristics of the traffic.

In 1995, the IETF's Realtime Traffic Flow Measurement (RTFM) working group was formed to develop a frame-work for real-time traffic data reduction and measurements [4][5][6]. A flow in the RTFM model can be loosely defined as the set of packets that have in common values of certain fields found in headers of packets. The fields used to aggregate traffic typically specify addresses at various levels of the protocol stack (e.g. IP addresses, IP protocol, TCP/UDP port numbers). Herein, we use the term ***key*** to refer to the set of address attributes used to aggregate packets into a flow.

2.2 Flow Granularity

The key specifies the granularity of the flow. The granularity of interest varies with the type of analysis. If the intent is to monitor the amount of traffic between two hosts, the key can be specified in terms source and destination IP addresses. However, if finer detail is required, for instance to identify individual TCP connections, then a 5-tuple key consisting of (source IP address, destination IP address, source port, destination port, IP protocol identifier) is more appropriate. Herein we specify flows with 5-tuple granularity.

2.3 Flow Expiry

Mechanisms to identify the start and the end of a flow must be defined. There are three primary expiry methods that are appropriate for studying characteristics of individual flows: protocol based, fixed timeout, and adaptive timeout [7]. With protocol based mechanisms, the state of a flow is determined by observing protocol specific messages (e.g. TCP SYN, FIN or RST). With a fixed timeout method, a flow has expired when some fixed period has elapsed since the last packet belonging to that flow was captured. Claffy et al. [8] have explored a large range of timeout values, ranging from 2 seconds to 2048 seconds (by power of two) and found that values between 16 to 128 seconds are appropriate to profile flows without being memory intensive. When varying other parameters, they fixed the timeout value to be 64 seconds. An adaptive timeout strategy as presented in [9] is a little more sophisticated than a fixed timeout method. The timeout is different for each flow and is computed based on the packet rate observed so far within each flow.

2.4 Flow Attributes

Flow attributes are used to describe a flow. The goal in defining flow attributes is to identify not only the relevant characteristics but also the proper way to measure them. In the literature, flow attributes are often called features, or characteristics. They can be values from the fields in headers of packets, counters (total bytes, total packets, etc.) or summary attributes such as means, median, and variance. The Realtime Traffic Flow Measurement (RTFM) working group [6] has recognized the need to include discrete distribution attributes to observe the probability repartition of certain variables. A discrete distribution attribute is represented as an array of "bins" with the values stored as counters between a minimum and maximum, with defined steps between adjacent bins [6]. Our findings indicate that discrete distributions are useful in distinguishing among networked applications, especially with regard to the size of packets. To the best of our knowledge, aside from the work presented in [10], [11] and [12], discrete distribution attributes are seldom included in studies on traffic classification.

2.5 Well known Flow Collectors

Cisco's *NetFlow* [13] collects information on flows travelling through a router or switch, but its flows are unidirectional and a limited number of flow attributes are recorded. Nonetheless, NetFlow is widely deployed [14] and many tools such as *cflowd*[15] and *SiLK*[16] have been developed to analyze the NetFlow data format. *Argus* [17] is another popular tool with its own flow format. *Argus* can monitor flows in real time or from pre-recorded packet traces. Argus flows are bidirectional but also have a limited number of Attributes. *NeTraMet* [18], which was the first implementation of the RTFM architecture, provides a general way for a user to specify flows and to specify which flow attributes to measure [19]. The tool suite of *NeTraMet* comprises components to specify flows and attributes, and to collect and read flow data.

At the moment we use our own in-house program to reconstruct flows from pre-recorded packet traces.

2.6 Flow Definition in this work

As part of this initial study, we examined bidirectional TCP and UDP flows. A flow record is created the first time a TCP/UDP packet is captured between two hosts. In order to treat TCP and UDP protocols similarly, we did not choose to use a protocol-based expiry mechanism. While TCP is a connection oriented protocol, UDP has no mechanisms to convey session state. Because of its simplicity, we chose a fixed timeout approach over an adaptive timeout mechanism to expire flows. The timeout value chosen for preliminary examination is 64 seconds, which is also commonly used in related works [2] [8] [20] [21] [22]. Thus, a flow terminates if 64 seconds has elapsed since the last packet belonging to that flow was captured. In practice many communications can have an inactive period that last longer than 64 seconds. When this happens, such communications are broken down into multiple flows, all sharing the same key, and the analysis is conducted on each "active flow" separately.

Flows are identified using a 5-tuple key, defined by the IP protocol (i.e. TCP or UDP), the IP addresses of the Originator and the Responder, and the two TCP/UDP port numbers involved. The Originator is defined as the sender of the first captured packet associated to a 5-tuple key.

It is worth mentioning that fragmentation at the IP layer does pose a problem when reconstructing the flows, since all but the first fragment lack the TCP/UDP port information [8]. Note that there may also be a "cold start" problem when the monitor starts capturing traffic from an existing traffic stream. In this case, the monitor generally cannot determine which host initiated the connection. Therefore the roles specified in the key (originator vs responders) may not reflect reality. We do not address either of these two problems at the moment.

3 Related Work

There is a pressing need for new methods to correctly identify network activities. Research challenges include recognizing traffic in high-speed, high-volume networks and within protocols that obscure the details of the information carried. Some novel approaches are being proposed to recognize the traffic based on its behaviour [1] [20] [21] [22] [23]. However the classification methods proposed take as input basic flow features (e.g. average packet size, flow duration, recurring use of addresses/ports). While such approaches have the advantage of using information that current flow collectors provide, we argue that it is necessary to continue to identify other flow features to characterize traffic.

In this work we explore discriminative flow features that portray essential communication dynamics, based solely on information that can be gathered from monitoring packet headers. We now review related work that has focused on identifying flow attributes and that has inspired this research.

3.1 Flow attributes survey

Dunigan et al. proposed in [10] and [11] an approach to traffic profiling based on a multivariate analysis. Each packet in a flow is "binned" (sorted) according to three parameters: the size of the packet, the delay between the packet and the previous one, and the direction. They used discrete distribution values for the first two parameters (size and inter-arrival time), and the third parameter (direction) is evaluated based on the direction of the packet and the direction of the packet preceding it, producing four possible values for the direction. The binning process yields a finite number of bins (e.g. 10 bins for the size by 10 bins for the delay by 4 bins for the direction would produce 10x10x4=400 bins). The value in each bin is treated as a random variable and represents the percentage of packets in that flow that fell into that bin.

Instead of doing a case by case study of different networked applications, they preferred a more systematic way of separating flows based on Principal Component Analysis (PCA). PCA is used to find the three variables that show the greatest variation among all flow types. Each flow can then be reduced to three attributes: the three most important variables for flow separation. The profile for each flow type can be viewed as a 3-D density function, estimated from samples of flows of the given type. To classify an unknown flow, the three principal components are computed for that flow. The probability that the unknown flow belongs to a given flow type is assessed by comparing the result of PCA against each of the known flow profile. The researchers reported that the classification error rates were high when flows from two different datasets were compared (i.e. when profiles were created using one data set and then compared to flows from the other data set). They suggested that optimizing binning delimiters and density smoothing could improve classification performance.

Lee and Stolfo [24] did an extensive study of traffic using the DARPA data [25], and identified 41 attributes of interest to Network Intrusion Detection Systems (NIDS) technologies. They provided a pre-processed copy of the DARPA 1999 KDD Cup contest [26] dataset, in which each flow record is summarized using these 41 attributes. To derive the 41 features, Lee and Stolfo started with nine basic attributes derived from Bro [27], and applied data mining programs to the connection records to compute the frequent sequential patterns, which were in turn analyzed to construct additional flow attributes. The resulting 41 attributes can be divided into three categories. Nine of the attributes are intrinsic TCP/IP network connection attributes as summarized in Table 1, thirteen are content-based attributes (e.g. "su root" command attempts), and 19 are statistical features based on history of past connections.

Table 1. *Lee and Stolfo: Intrinsic Features of Network Connection Records*

Features	Description	Value type
duration	Length (number of seconds) of the connection s	Continuous
protocol_type	Type of the protocol, e.g., TCP, UDP, etc.	Discrete
service	Network service on the destination, e.g., HTTP, TELNET, etc.	Discrete
src_bytes	Number of data bytes from source to destination	Continuous
dst_bytes	Number of data bytes from destination to source	Continuous
flag	Normal or error status of the connection	Discrete
land	1 - connection is from/to the same host/port; 0 - otherwise	Discrete
wrong_fragment	Number of "wrong" fragments	Continuous
urgent	Number of urgent packets	Continuous

source: This table is taken from W. Lee, and S.J. Stolfo [24]

Extracting all of the 41 attributes from a traffic trace can be resource intensive since the monitor has to review each packet's payload and keep the history of numerous preceding connections. Since the 1999 KDD Cup contest dataset already has the tedious and time-consuming pre-processing step done, it has been used as the basis for most of the recent research on data mining IDSs.

Paxson and Zhang [28] developed a general purpose algorithm to identify keystroke-interactive connections by testing packet size, timing, and directionality against preset criteria. They noted that keystroke packets tend to be very small, carrying 20 bytes or less of data. They also noted that humans transmit (type in) keystrokes relatively slowly and thus the delay between two consecutive keystroke-packets is likely to be in the range of [10, 2000] milliseconds. From these observations they derived: 1) an indicator that quantifies how often the inter-arrival time between two consecutive small packets falls in the range [10, 2000] milliseconds, 2) an indicator to characterize the relative proportion of small packets, and 3) an indicator that quantifies how clustered together small packets are.

Thus, in the approach adopted by Paxson and Zhang to identify interactive flows, there are three flow attributes taking continuous values between zero and one. The closer to one a value is, the stronger the indication of interactivity. A connection is marked as interactive when all three computed values are greater than certain thresholds.

***Table 2.** Paxson and Zhang: Flow Attributes for detecting keystroke-interactive traffic*

Attributes	Description	Value type
β	Proportion of small packets (where small packets have 20 bytes or less of data)	Continuous
γ	Indicator of the frequency of consecutive small packets	Continuous
α	Proportion of keystroke interarrivals (where keystroke interarrivals are delays between consecutive small packets that fall in the range of [10, 2000] msec)	Continuous

Paxson and Zhang also developed a set of special purpose algorithms for identifying specific interactive protocols (SSH, RLOGIN, TELNET, FTP, NAPSTER, and GNUTELLA) [28]. The SSH detection algorithm includes an indicator based on packet size. All other special-purpose algorithms rely on access to the payloads of packet. Even so, certain applications looked alike. In particular, they noted that the FTP detection signature, based on status information found in the payload of packets, could not distinguish between FTP and the SMTP mail transfer protocol. Herein we note that the two protocols behave differently and may be distinguished.

Ilvesmaki et al. [29] proposed to divide traffic into three classes of interest to researchers working in the field of Quality of Service (QoS). The classes they defined can be summarized as follow:

> **Class 1-** *Interactive traffic*, regrouping applications that send packets at very short intervals for lengthy periods of time and usually have a human user at both ends of the communication (e.g. VoIP).
>
> **Class 2-** *Transactional traffic*, regrouping applications that occasionally send packets at very short intervals, but have moderate intervals between bursts. These applications typically have a human user at one end and requesting information from a server at the other end (e.g. SSH, HTTP).
>
> **Class 3-** *Supportive traffic*, traffic important to network functionality and reliable functioning of the applications but whose prioritization does not affect the perceived overall quality of the original application (e.g. DNS).

They examined the distribution of packet inter-arrival times, packet lengths, and flow inter-arrival times. They did not provide classification rules, but similar to Paxson and Zhang, they felt that the inter-arrival time of packets could be used to classify the traffic into two distinct classes (interactive or non-interactive). However, they reported that the division into three classes remained difficult and needed further research.

Lastly, in parallel to our work, Hernández-Campos et al. [30] derived a set of statistics (features) per TCP connection to group connections into statistical traffic clusters. They too have worked on identifying characteristics based on the behaviour of the traffic that does not rely on port numbers nor on payload analysis. They aimed to regroup traffic based on the network usage as opposed to characterizing protocols individually. Therefore a pursued goal was to produce clusters that would regroup "file transfer traffic", "streaming media traffic", "interactive traffic" for example. As with our methodology, before categorizing the traffic, each connection is summarized by a set of features. The novelty in their approach is in the use of a unit of data which is different from a "packet". The unit, called the Application-Data Unit (ADU), may contain several packets. Alternating patterns in the advances of the TCP *Sequence Number* and TCP *Acknowledgement Number* of packets mark the boundaries of an ADU. Instead of modelling the patterns of packet exchanges within a TCP connection, they model the patterns of ADU exchanges. In our study, we had observed that patterns in the direction dynamics of packets stand out more clearly when we remove from the sequence of packets those that contain no payload[2]. Hernández-Campos et al.'s approach also permits observation of these interesting directional patterns and is more robust to packet disordering; although constructing the ADUs requires more computational effort than discarding empty packets from the analysis. Each TCP connection is represented as an n-dimensional vector $(c_1,...c_n)$ where n is the number of ADU exchanges, called *epochs*. An epoch is a triplet of the form $c_i=(a_i, b_i, t_i)$ in which a_i is the number of bytes transmitted by the initiator of the connection within the current ADU, b_i is the number of bytes in the other direction, and t_i is the idle time between the current ADU and the next ADU in the sequence. Twenty-six features selected as a first cut at the problem of dividing traffic are reported in Table 3.

[2] Over an established TCP connection, these TCP packets are normally transmitted to simply acknowledge having received data up to a given *Acknowledgement Number*.

***Table 3.** Hernández-Campos et al.: 26 Flow Attributes used for separating traffic*

Attributes	Description	Value type
n	Number of epochs	continuous
a_{tot}, b_{tot}	Total bytes in each direction	continuous
$a_{max}, b_{max}, t_{max}$	Maximum bytes and seconds per epoch	continuous
a_{min}, b_{min}	Minimum bytes per epoch	continuous
$a_{\mu}, b_{\mu}, a_{\sigma}, b_{\sigma}$	Mean bytes per epoch and standard deviation	continuous
a_{1q}, b_{1q}	First quartile	continuous
a_{2q}, b_{2q}	Second quartile	continuous
a_{3q}, b_{3q}	Third quartile	continuous
a_{vs}, b_{vs}	Total variation ($X_{vs} = \sum_{j=2}^{n} \lvert x_j - x_{j-1} \rvert$)	continuous
a_h, b_h	homogeneity ($X_h = (x_{max} + 1)/(x_{min} + 1)$)	continuous
a_p, b_p	Lag-1 autocorrelation	continuous
$\rho_1(a_{1...n}, b_{1...n})$	Spearman's rank correlation	continuous
$\rho_2(b_{1...n-1}, a_{2...n})$	Spearman's rank correlation with lag 1	continuous

source: This table is taken from F. Hernández-Campos et al. [30]

After normalizing the features that vary over several orders of magnitudes and addressing other practical issues dealing with connections with fewer *epochs*, Hernández-Campos et al. compared different clustering methods to classify the traffic. Results presented in [30] indicate that, without a priori knowledge of the protocols in use on a network, meaningful classes of traffic may be obtain through clustering techniques when input vectors carry important communication characteristics.

3.2 Principles of our Approach

The methodology can be viewed as a three step process in which the output of a previous step becomes the input for the next. The process is outlined below and illustrated in Figure 1.

Step 1) *Packets are grouped into flows.* Each flow is identified by a 5-tuple defined by the IP protocol, IP addresses of the Originator and the Responder, and the two TCP/UDP ports numbers involved. Minimal information per packet is recorded.

Step 2) *Characteristics (attributes) are measured on each flow.* About forty communication characteristics are measured for each flow. The output is a set of flow records containing the flow attributes.

Step 3) *Flows are Recognized and Described.* Based on the characteristics obtained during step 2, we tag each flow with two properties: the application recognized (if any) and a flow description based on the traffic behaviour.

The analysis is currently done off line. Our main contribution to the research is the identification of about 40 flow features by which different types of applications can be distinguished. These features were constructed following a domain knowledge approach and thus can be interpreted meaningfully. These attributes are different from those of Lee and Stolfo [24] which were specific to the field of Network Intrusion Detection Systems (NIDS). The process of developing the flow features was greatly inspired by the work of Paxson and Zhang [28] for detecting interactivity (human control). Herein, we go a step further by deriving flow features that can be used to successfully discriminate among a variety of networked applications, whether they are interactive or machine-driven. We are able to capture distinctive communication characteristics such as conversation, transaction and data transfer; and derive signatures from a number of observable patterns.

Hernández-Campos et al. [30] also aimed to identify communication patterns and distinguish networking applications. Most of the attributes they derived are captured by our indicators through some other means. We believe our indicators reveal more insight into the traffic behaviour than those listed in Table 3. The three attributes that are present in their work and not in ours are those concerning correlation. These may be examined in future work.

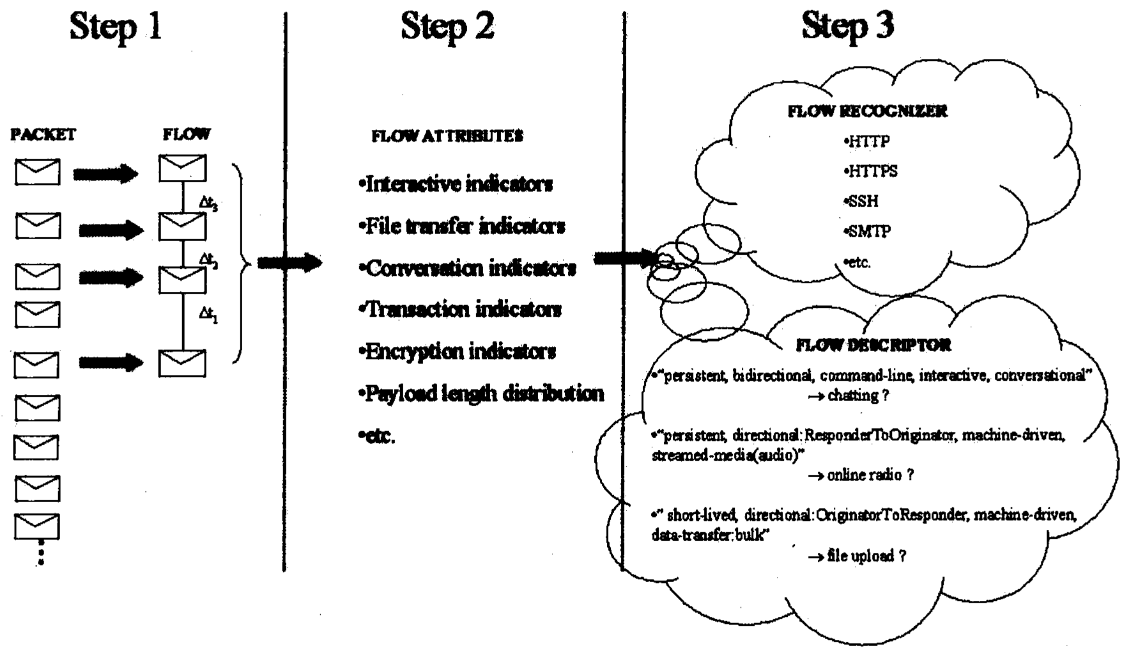

Step 1
Step 2
Step 3
PACKET
FLOW
Δt3
Δt2
Δt1
FLOW ATTRIBUTES
•Interactive indicators
•File transfer indicators
•Conversation indicators
•Transaction indicators
•Encryption indicators
•Payload length distribution
•etc.
FLOW RECOGNIZER
•HTTP
•HTTPS
•SSH
•SMTP
•etc.
FLOW DESCRIPTOR
•"persistent, bidirectional, command-line, interactive, conversational" → chatting ?
•"persistent, directional:ResponderToOriginator, machine-driven, streamed-media(audio)" → online radio ?
•" short-lived, directional:OriginatorToResponder, machine-driven, data-transfer:bulk" → file upload ?

Figure 1. Overview of the three step analysis included in the proof of concept tool

We now enumerate some highlights of our approach:

- The focus has been placed on identifying *characteristics* of traffic that portray distinctive communication patterns. We have identified about 40 flow attributes to help divide traffic into human-recognizable categories.
- The flow attributes identified may be used to define classes of traffic at different level of granularity. Some flow attributes are appropriate for defining signature-like patterns to recognize specific applications. We refer to such flow attributes as "give-away" features. Other attributes can help classify the traffic at a coarser grain by grouping similar services based on their behaviour.
- The flow attributes can serve as a starting point for different traffic characterization studies not necessarily related to network security. The classes derived from the flow attributes may be defined differently depending on the intent being pursued (e.g. traffic accounting, QoS).
- We completely avoid relying on port numbers or payload analysis. This provides an alternative method to more conventional traffic categorization techniques provided by current networking tools.
- We only examine communication patterns found at the network and transport layers, requiring minimal information per packet to be retained.
- Patterns are identified at the 5-tuple flow granularity of TCP/UDP communications. Therefore even sporadic or ephemeral malicious activities may be identified without the requirement of waiting until multiple connections can be examined.
- We show with step 3 that simple rule sets can be defined based on the measured characteristics to distinguish among various networked applications, ranging from human controlled applications to machine driven data transfer. Other researchers may choose to replace these simple rule sets by more sophisticated classification techniques.
- Even when a flow cannot be classified based on the categories defined, we provide a short description of the traffic to help a network analyst understand the kind of activities that took place.

4 Description of the Flow Attributes

4.1 Pre-processing

The traffic analysis process starts with a *tcpdump* data file from which we extract the packet level information. Very little information per packet is required for the analysis. More precisely, during the pre-processing phase (i.e. step 1), each packet is summarized using only four attributes: the direction (Originator to Responder or vice-versa), the time of arrival (in microseconds), the total length of the IP datagram in bytes, and the length in bytes of the payload (data appended to the transport layer). The packet summaries are grouped by flows. Each flow is identified by a 5-tuple key described earlier.

4.2 Deriving the Flow Attributes

Deriving the flow attributes did require considerable knowledge and experience with network protocols. On top of intuition acquired from analysing large amount of data collected from real world environments, we have studied protocols within the TCP/IP suite and other published research, such as those discussed in the related work section and the bibliography, for important features that characterize networked applications. We also experimented with network traffic within our testbed for further insight on session dynamics.

The attributes we derived are summarized in Table 4 and Table 5. The following subsections will describe each attribute with greater detail. Table 4 lists the attributes that are measured over the entire flow. The first three attributes (key, BeginTime, EndTime) are simply used to identify and sort the flows. Table 5 gives the attributes that are specific to each direction, and thus are measured in each direction separately.

***Table 4.** Attributes measured over the whole flow*

Attributes	Description	Value type
Key	A 5-tuple indicating the Originator IP address, the Responder IP address, the IP Protocol (i.e. TCP or UDP), the source port of the Originator, and the source port of the Responder.	String
BeginTime	Arrival time of the 1st packet as provided by libpcap.	String
EndTime	Arrival time of the last packet as provided by libpcap.	String
Duration	Completion time of the flow in microseconds.	Continuous
FirstNonEmptyPacketSize	Payload length of the first non-empty packet.	Continuous
FirstFewNonEmptyPacketDirections	An array of 10 discrete values for the directions (-1 or 1) of the first 10 non-empty packets. 1: Originator to Responder, -1: Responder to Originator Array is initialized with values equal to 0 in case fewer than 10 packets contain data.	Array of 10 Discrete values
DatabyteRatioOrigToResp	Total amount of payload data transmitted by the Originator over the Total amount of payload data transmitted by the Responder (initialized to -1 for flows with no data transmitted by the Responder).	Continuous
InterarrivalDistribution	A discrete distribution of inter-packet delays represented by an array of 9 continuous binned values. The value in each bin is between 0 and 1 and represents the relative proportion of packets that fell into that bin.	Array of 9 Continuous values.
Conversation Indicators:		
$\alpha_{conversation}$	The number of non-empty packets that belong to a *conversation* over the total of non-empty packets.	Continuous
$\beta_{conversation}$	The number of non-empty packets that belong to a *sustained conversation* over the total of non-empty packets that belong to a conversation.	Continuous
$\gamma_{conversation}$	The proportion of *conversation* packets that are transmitted by the originator.	Continuous
Transaction Indicator:		
$\alpha_{transaction}$	Indicator of how often "ping pong" exchanges are seen in a flow.	Continuous

For each direction of the flow, we measure the following attributes:

Table 5. Attributes measured for each direction of the flow

Attributes	**Description**	**Value type**
InterarrivalDistribution	A discrete distribution represented by an array of 9 continuous values. The array contains the binned values for inter-packet delays in the considered direction. The value in each bin is between 0 and 1 and represents the relative proportion of packets that fell into that bin.	Array of 9 Continuous values.
PayloadDistribution	A discrete distribution of packet payload length represented by an array of 23 continuous values. The array contains the binned values for payload lengths per packet. Again, the value in each bin is between 0 and 1.	Array of 23 Continuous values.
ByteCount	Total amount of byte transferred (including bytes found in the network and transport headers).	Continuous
DatabyteCount	Total amount of byte transferred as payload.	Continuous
PacketCount	Total number of packets.	Continuous
DatapacketCount	Total number of non-empty packets.	Continuous
Encryption Indicators:		
$\alpha_{cipherblock}$	Estimated popular GCD among the packet payload lengths.	Continuous
$\beta_{cipherblock}$	Ratio of non-empty packet-payload lengths that are divisible by $\alpha_{cipherblock}$.	Continuous
Keystroke Interactive Indicators:		
$\alpha_{key\_interactive}$	Indicator of interactive inter-packet departure (for keystroke packets).	Continuous
$\beta_{key\_interactive}$	Indicator of interactivity based on the proportion of small packets.	Continuous
$\gamma_{key\_interactive}$	Indicator of consecutive small packets.	Continuous
$\delta_{key\_interactive}$	Indicator of piggyback packing.	Continuous
$\varepsilon_{key\_interactive}$	Indicator of irregularity between inter-arrival of consecutive small packets.	Continuous
Command-line Interactive Indicators:		
$\alpha_{cmd\_interactive}$	Indicator of interactive inter-packet departure (for command-line packets).	Continuous
$\beta_{cmd\_interactive}$	Indicator of interactivity based on the proportion of small packets.	Continuous
$\gamma_{cmd\_interactive}$	Indicator of consecutive small packets.	Continuous
$\delta_{cmd\_interactive}$	Indicator of piggyback packing.	Continuous
$\varepsilon_{cmd\_interactive}$	Indicator of irregularity between inter-arrival of consecutive small packets.	Continuous
File transfer Indicators:		
α_{file}	Indicator of inter-packet departure during a file transfer.	Continuous
β_{file}	Indicator of file transfer based on the proportion of big packets.	Continuous
γ_{file}	Indicator of consecutive big packets.	Continuous
BitrateMean	The bit rate is measured every 5-second. BitrateMean gives the average of the measurements.	Continuous
$\alpha_{constantbitrate}$	Indicator of how close to the mean the 5-second bit rate measurements are.	Continuous
PacketrateMean	The packet rate is measured every 5-second. PacketrateMean gives the average of the measurements.	Continuous
$\alpha_{constantpacketrate}$	Indicator of how close to the mean the 5-second packet rate measurements are.	Continuous
PayloadMedian	The "middle" packet payload length.	Continuous
$\alpha_{constantpayload}$	Indicator of how close to the median the payloads per packet are.	Continuous

The remainder of Section 4, which is the core of the document, describes the attributes in detail with the reasoning behind each one. When appropriate, we disclose some threshold values we find suitable for separating traffic types, for instance for distinguishing between interactive and non-interactive flows. These proposed thresholds were based on experiment conducted within the testbed. They are by no means absolute thresholds, but do give an indication of the range in which values are expected to fall.

4.2.1 Discrete Distribution Attributes for payload and inter-packet delay

Discrete distributions are used for inter-packet delay and packet-payload length. Distribution values are especially important for packet payload length. Application protocol overhead may imply that non-empty packets are always greater or equal in length to a given minimum (due to header length). Application negotiation mechanisms may also exhibit a high frequency of packets of special sizes. Moreover, certain applications may have a preferential packet size, and completely avoid sending packets of lengths within a given range. For instance, as noted in [29], the HTTP-protocol is characterized by many short and long packets. Such characteristics are not effectively reflected by means and variances which may be sensitive to outliers. Discrete distribution attributes are therefore preferred in this work.

The bin delimiters for payload length have been empirically selected to capture payload sizes that are seen frequently or have special significance. Determining useful bin delimiters required domain knowledge and several cycles of refinement. We currently use the following bin delimiters for payload length (in bytes):

[0-1[, [1-2[, [2-3[, [3-5[, [5-10[, [10-20[, [20-40[, [40-50[, [50-100[, [100-180[, [180-236[, [236-269[, [269-350[, [350-450[, [450-516[, [516-549[, [549-650[, [650-1000[, [1000-1380[, [1380-1381[, [1381-1432[, [1432-1473[, [1473-inf[.

When monitoring TCP and UDP packets on an Ethernet network, where the Maximum Transmission Unit (MTU) is 1500 bytes, the last bin (1473 bytes and over) should always be empty (the network and transport headers take a minimum of 28 bytes). Our findings indicate that, when distinguishing among application protocols, the distribution of packet payload size is one of the most critical flow attributes. The distribution depends on the application protocol and is largely independent of the size of the network-dependent MTU.

The inter-packet delays are distributed according to bins ranging from 0 to 64 second-delays. The 64 second-delay upper bound is due to the fixed timeout expiry. We currently use the following bin delimiters for inter-packet delay (in seconds):

[0-0.000001[, [0.000001-0.0001[, [0.0001-0.001[, [0.001-0.01[, [0.01-0.1[, [0.1-1.0[, [1.0-10.0[, [10.0-64.0[, [64.0-inf[.

The last bin should always be empty because of the fixed timeout flow expiry.

Figure 2 illustrates how packets are being sorted into the discrete bins for payload size and inter-packet delay. In particular, the fictive payload distribution example illustrated in the Figure would indicate that 45% of the packets carried no data, and another 45% of the packets were relatively big, not full size packets but big packets.

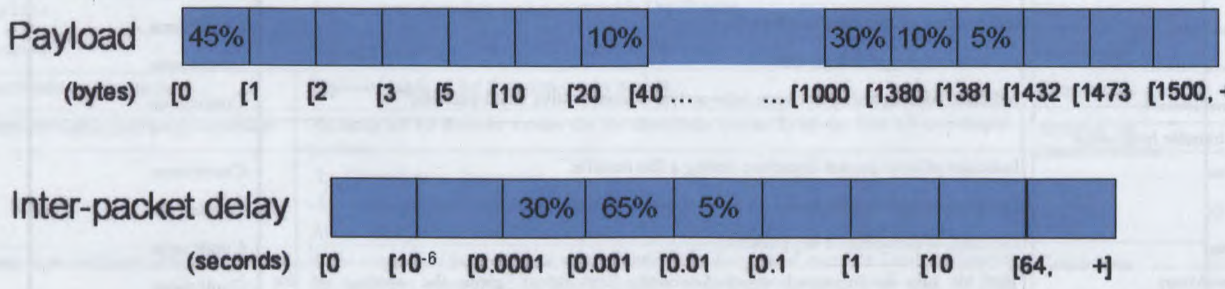


Figure 2. Conceptual illustration of discrete distribution

4.2.2 Characterizing regularity in payload and transmission rate

While a discrete distribution attribute for inter-arrival delays can help identify applications with a fixed inter-departure time, it may fail to identify applications that adjust their transmission rate to accommodate a target rate. When they examined RealAudio traffic, Lan et al. [31] observed a constant bit rate when measured at medium time scales (tens of seconds), but a bursty on/off source behaviour at small time scale (single seconds). When studying the implication of multiplexing audio and video, Kuang et al. [32] observed that although RealVideo can be compressed as Variable-Bit-Rate (VBR) at the application layer, it is often streamed as Constant-Bit Rate (CBR) at the network layer. This observation was for measurements at large timescale (minutes). They noted that at smaller time scales (seconds), the packet transmission was showing on/off patterns due to the interleaving of audio and video.

In this work we chose to estimate bit rate and packet rate over 5-second intervals. A five second interval appeared to be long enough to smooth the perceived transmission rate of streaming applications, and small enough to prevent false positive identification. Herein we qualify an application as having a target rate, if a large number of the measurements are close to the mean. Let R be the total number of measurements (bit rate or packet rate estimated over 5-second intervals), and r be the number of these measurements that are within one standard deviation from the mean, we compute the ratio *r/R* as an indicator of regularity. When the ratio is high (close to 1), the mean value is a usually useful in describing the flow. Obviously, the mean and the ratio *r/R* may only be measured for flows that last longer than 5 seconds[3].

We derive a similar measurement to identify flows that transmit packets of a fixed length (streaming audio is an example). We use the median instead of the mean so that the attribute value represents a packet length effectively transmitted (as opposed to a calculated length).

4.2.3 Characterizing interactivity: command-line versus keystrokes

In this work we define an interactive flow as being driven, and not just initiated, by human interaction. In contrast with Ilvesmakï et al. [29], the definition we use classifies a HTTP flow as machine-driven. The human interaction may be visible across multiple HTTP flows (e.g. by examining flow inter-arrival times), but within a given flow, the information transfer is an automated mechanism. We test interactivity on each direction. We note here that even if only one side is controlled by a human, our mechanisms may falsely identify the connection as being interactive in both directions. This typically happens when the application allows the echo-mode to be turned on. With the echo-mode on, commands typed by the client side (human driven) are echoed verbatim by the server side (machine-driven), and thus both directions may appear as interactive.

With the exception of a few minor differences, the interactive indicators we use are essentially those of Paxson and Zhang [28]. We however derive two distinct classes of human-driven packet transmission: *keystroke* transmission and *command-line* transmission. *Command-line* transmissions are larger in size and are separated by longer delays than keystrokes. The distinction between *command-line* and *keystroke* interactivity helps refine the classification process a step further. FTP command for instance can be distinguished from interactive SSH and TELNET sessions; and it is foreseen that chat sessions will be classed differently depending on the "flavour" (MSN Messenger for example is of the *command-line* type).

For keystroke interactive indicators, a small packet is defined as a non-empty packet when carrying 60 bytes or less (Paxson and Zhang used 20 bytes or less). We augmented the threshold to 60 bytes after observing that with current ciphers, packets may be as big as 52 bytes, although encapsulating a single encrypted character. We also modified the range of inter-arrival delays between keystrokes. We use d_{min}=25ms and d_{max}=3000ms for the delimiters.

For command-line interactive indicators, a small packet is defined as a non-empty packet carrying 200 bytes or less. The range of inter-packet delays for command-line packets is defined as [d_{min}=250, d_{max}=30000] ms. The boundaries are arbitrary (10 times greater than those of keystrokes), and modifying the range does not appreciably change the detection performance.

For reference purposes, we denote the three indicators derived by Paxson and Zhang by the same three Greek letters used in their paper [28], namely α, β, and γ. In the remainder of the document, whenever we derive multiple indicators of a given property, we use the notation $\alpha_{property}$, $\beta_{property}$, $\gamma_{property}$, $\delta_{property}$, etc. to refer respectively to the first indicator, the second, the third and so on.

For each direction of the flow, let Ω be the set of delays between consecutive small packets and $\Delta = \{ \omega \in \Omega$, such that $d_{min} \leq \omega \leq d_{max} \}$, the indicator of interactive inter-packet departure is defined as:

$$\alpha_{interactive} = \frac{\text{number of elements in } \Delta}{\text{number of elements in } \Omega} . \tag{1}$$

Let S be the number of small packets, let N be the number of non-empty packets, let G be the number of gaps[4] between small packets, the indicator of interactivity based on the proportion of small packets is:

$$\beta_{interactive} = \frac{S}{N}, \tag{2}$$

and the indicator of consecutive small packets is

$$\gamma_{interactive} = \frac{S - G - 1}{N} . \tag{3}$$

A value close to one for $\gamma_{interactive}$ indicates that there are many small packets and these packets are grouped together, while smaller values[5] suggest that small packets are all spread throughout the connection.

[3] As any other time-dependent attribute, the target rate indicator is sensitive to network conditions. Under congestion conditions, the regularity in the transmission rate may not be perceived if the monitor is located far away from the transmitting end.

[4] A gap occurs whenever two small non-empty packets are separated by at least one packet (big or empty).

[5] Note that $\gamma_{interactive}$ as defined here may even be negative if packets transmitted are either big or empty, in which case S and G are equal to zero and thus $\gamma_{interactive}$= -1/N.

On top of the first three indicators from Paxson and Zhang, we further define two new indicators to penalize certain types of non-interactive applications that may still get high scores for the first three indicators.

The fourth indicator gives the proportion of small non-empty packets with respect to the total number of small packets (including empty packets). The goal with this heuristic is to penalize machine-driven applications that transmit a lot of small packets, which may however be dominated by empty control segments (i.e. TCP ACK packets without piggyback data). The reasoning is that a node transmitting a high number of empty packets is likely to have a passive role in a communication. Thus let E be the number of empty packets, we define:

$$\delta_{\text{interactive}} = \frac{S}{S+E} \tag{4}$$

as an indicator of piggyback packing.

Lastly, we define a fifth indicator measuring irregularity in the transmission rate of consecutive small packets. This heuristic penalizes automated transmission processes that send packets at a regular, but slow rate, and thus may get a high score for $\alpha_{\text{interactive}}$. Here we qualify the transmission rate as being regular, if a large number of inter-arrival delays are close to the mean.

Let μ and σ be respectively the mean and standard deviation of the delays between consecutive small packets; let $\Lambda = \{ \omega \in \Omega$, such that $\omega \in [\mu-\sigma,\mu+\sigma] \}$, then the indicator of irregularity between inter-arrival times of consecutive small packets is:

$$\varepsilon_{\text{interactive}} = 1 - \frac{\text{number of elements in } \Lambda}{\text{number of elements in } \Omega}\,. \tag{5}$$

For all five indicators, the closer to one a value is, the stronger the indication of interactivity is. Based on traffic collection experiments within our testbed, we mark a direction of a flow as interactive if $\alpha_{\text{interactive}}$, $\beta_{\text{interactive}}$, $\gamma_{\text{interactive}}$, $\delta_{\text{interactive}}$, and $\varepsilon_{\text{interactive}}$, are greater or equal to 0.3, 0.6, 0.6, 0.3, and 0.3 respectively. The first three criteria are in general sufficient to recognize interactive flows, and the last two are merely there to reduce the number of false positives. Separately, the indicators are also useful in distinguishing among applications whether they are interactive or not. For instance, episodes of consecutive small packets were found to be extremely rare in the case of HTTP, which translates into values smaller or equal to zero for the indicators of consecutive small packets ($\gamma_{\text{keystroke\_interactive}}$ and $\gamma_{\text{cnd\_interactive}}$).

4.2.4 Characterizing data transfer

From the interactive indicators, we derive file transfer indicators. In general, a file transfer flow contains episodes of consecutive big packets transmitted within a short delay. In order to include video and audio streams in the category of data transfer, we have set a very lax threshold for the size of a big packet. A big packet is defined as carrying 225 or more bytes. A short inter-packet delay is 50ms or less.

For each direction of the flow, let B be the number of big packets, let N be the number of non-empty packets, let G' be the number of gaps between big packets. Furthermore, let Ω' be the set of delays between consecutive big packets and $\Delta' = \{ \omega \in \Omega'$, such that $\omega \in [0, d_{max}] \}$, then the indicator of inter-packet departure during a file transfer is:

$$\alpha_{\text{file}} = \frac{\text{number of elements in } \Delta'}{\text{number of elements in } \Omega'}\,. \tag{6}$$

The indicator of file transfer based on the proportion of big packets is defined as:

$$\beta_{\text{file}} = \frac{B}{N}, \tag{7}$$

and lastly, the indicator of consecutive big packets is

$$\gamma_{\text{file}} = \frac{B-G'-1}{N}\,. \tag{8}$$

For all three indicators, a value close to one is a strong indication of data transferring. We find that thresholds of 0.5, 0.5, and 0.2 work well in practice. We therefore mark the direction of a flow as a file transfer if α_{file}, β_{file}, γ_{file}, are greater or equal to 0.5, 0.5, and 0.2 respectively.

4.2.5 Characterizing directional dynamics

A main observation when we examined directional behaviour of the communications is that patterns are more easily identified when discarding empty packets (i.e. carrying no payload) from the sequence of analysed packets. We recognize however that most of the applications considered in this study run over TCP. With TCP, empty packets transmitted over an established connection normally have the ACK field set to 1 and are transmitted to acknowledge the receipt of data. Since the UDP header has no acknowledgement field, the ACK mechanism must be handled at the application layer if the service requires such mechanism. Thus UDP packets should always contain some data, except possibly in rare cases where an application may transmit empty packets to prevent the session from timing out. From the traces collected on our organization's network, there has not been evidence of any common UDP applications transmitting empty packets. Perhaps in order to observe patterns in the directional dynamic of UDP applications, we may have to derive a threshold by which packets carrying less data would be discarded from the directional analysis. We leave the UDP case for future work when a greater variety of UDP applications will be examined.

We describe below how we capture directional dynamics observable during the beginning of TCP flows, and then later on we describe indicators of conversation and transaction that can be perceived throughout the life of a flow.

We observed that monitoring the direction of the first few non-empty packets was very useful. Option negotiation and authentication processes appear to be standardised. This was not as apparent for connection termination processes, where a greater variability was observed among different implementations of a given application. The chance of recognizing a pattern of initial direction dynamics is greatest within the first four or first five non-empty packets. Nonetheless, we maintain an array containing the direction of the first ten non-empty packets as reported in Table 4 for the attribute named "FirstFewNonEmptyPacketDirections". Figure 3 gives an example with the Simple Mail Transport Protocol (SMTP). While it is the client that initiates the connection, the first non-empty packet is sent by the server. From this point, SMTP behaves like a Command-Response protocol driven by the client commands.

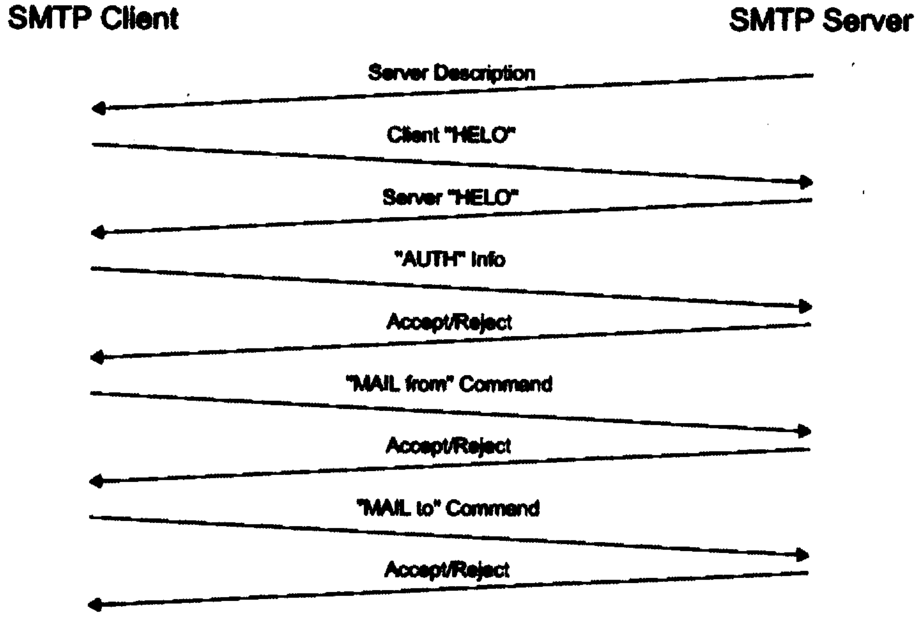


Figure 3. Illustration of directional patterns based on non-empty packets

Knowing the size and direction of the very first non-empty packet is particularly insightful. The size of the first non-empty packet tends to be constant among flows a given network application. We record the payload length of this packet in the attribute named "FirstNonEmptyPacketSize". The direction of this packet determines which, between the two end-points, is the first to "speak up" once the session is established. We noted differences among certain protocols with respect to this characteristic. In particular, it is the client of a RLOGIN session that transmits the first non-empty packet, while in the case of TELNET it is the server. The information is contained in the first element of "FirstFewNonEmptyPacketDirections".

Another flow attribute that is quite useful when characterizing traffic is the total amount of data transferred in each direction. For instance, successful SMTP connections transfer more than 300 bytes due to a more-or-less fixed overhead [33]. Comparing the amount transferred by each side is a good indicator of the directionality of the traffic. Bulk transfers tend to be highly asymmetrical. FTP data connections for instance are unidirectional, where non-empty packets flow in one direction only. For a TELNET connection, the ratio between bytes sent by the computer-side and bytes sent by the user is about 20:1 [33]. On the other hand, the proportion of bytes transferred in each direction of a "chat" session may be similar. The attribute comparing the amount of data transferred between the two end-points is named "DatabyteRatioOrigToResp" in Table 4.

4.2.5.1 Conversations and transactions

Communications may be built from many episodes, where an episode is a communication sequence marked by the occurrence of an incident or the presence of a distinctive feature. To capture the directionality dynamics of the whole connection we developed heuristics to quantify *transaction* and *conversation* episodes. Both heuristics are computed after empty packets (i.e. packets carrying no payload) have been removed. Thus TCP ACK packets have no effect on the two types of indicators we derive. Packets are treated as a sequence of positive and negative values (the sign of each value indicates the direction of the packet) and the idea is to characterize the changes in sign. Whether they are interactive or machine-driven, applications often exhibit differences with respect to the *transaction* and *conversation* indicators.

4.2.5.2 Characterizing conversations

To derive the conversational heuristics (conversation indicators), we need to introduce the notion of *conversation* episodes:

***Conversation* episode**

A *conversation* episode in this work contains consecutive (back to back) packets in one direction followed by consecutive packets in the other direction.

We define a second notion named *sustained conversation.* To qualify as a *sustained conversation* episode, a sequence of packets must contain at least three "exchanges". More explicitly,

***Sustained conversation* episode**

A *sustained conversation* episode must contain consecutive packets in one direction, followed by consecutive packets in the opposite, and followed again by consecutive packets in the first direction (e.g. A->B, B->A, A->B).

The conversation indicators are then based on packet counts. We define three heuristics that take their values between 0 and 1. Let M be the total number of non-empty packets in a flow, let C be the number of non-empty packets associated with a conversation, and ζ be number of non-empty packets associated with a sustained conversation. We define $\alpha_{\text{conversation}}$ as the number of non-empty packets that belong to a conversation over the total number of non-empty packets:

$$\alpha_{\text{conversation}} = \frac{C}{M}. \tag{9}$$

We define $\beta_{\text{conversation}}$ as the number of non-empty packets that belong to a sustained conversation over the total of non-empty packets that belong to a conversation:

$$\beta_{\text{conversation}} = \frac{\zeta}{C}. \tag{10}$$

Let O be the number of non-empty packets associated with a conversation and transmitted by the Originator, we define an indicator of symmetry in a conversational flow, $\gamma_{\text{conversation}}$ as the proportion of conversation packets that are transmitted by the originator:

$$\gamma_{\text{conversation}} = \frac{O}{C}. \tag{11}$$

The indicators $\alpha_{\text{conversation}}$, $\beta_{\text{conversation}}$, and $\gamma_{\text{conversation}}$ are initialized to zero and are only computed if the denominators are nonzero. A value close to 0.5 for the last indicator denotes conversational symmetry, and a value close to 1 for the first two indicators is a strong indication of conversational behaviour. In practice we find that when the first two indicators ($\alpha_{\text{conversation}}$ and $\beta_{\text{conversation}}$) are greater or equal to 0.4, then it is likely that the flow has conversational episodes.

4.2.5.3 Unique versus multiple transactions

As mentioned earlier, data transfers are characterized by asymmetry in the total bytes transferred in each direction. We define here an additional indicator, which attempts to differentiate between "one shot" transfers, and transactions involving multiple negotiations. FTP data is an example of the first category and SMTP is an example of the other. SMTP involves multiple exchanges between the client and the server, in the form of automated Command/Response mechanisms. As with the conversation indicators, we examine a sequence of signed values, where each sign gives the direction of a packet in the flow. Again, only non-empty packets are retained. When looking at the sequence of signed values for various types of applications involving multiple transactions, it appeared that a lot of "ping-pong" exchanges take place, where one packet is followed by a packet coming in the opposite direction. We quantify this phenomenon by comparing the number of changes in sign effectively seen, with the maximum number of times a change of sign can occur, given the number of positive and negative values in that sequence.

More precisely, let ρ and η be respectively the number of positive and negative values, the maximum number of time a change in sign can occur, denoted by τ, is

$$\tau = \begin{cases} 2\rho - 1 & \textit{if } \rho = \eta \\ 2\min(\rho,\ \eta) & \textit{otherwise} \end{cases} \tag{12}$$

and let δ be the number of sign changes observed, then

$$\alpha_{\text{transaction}} = \frac{\delta}{\tau} \qquad (13)$$

is an indicator of how often "ping pong" exchanges are seen in a flow. τ is equal to 0 when the flow is unidirectional and thus $\alpha_{\text{transaction}}$ may not be defined for all flows. $\alpha_{\text{transaction}}$ is initialised to zero by default. When $\alpha_{\text{transaction}}$ is non-zero, a value close to 1 is a strong indicator of multiple transaction exchanges.

While on the subject of Command/Response mechanisms, it is interesting to note that an essential invariant of such interactions is that endpoints do not transmit data concurrently. In their paper, Hernåndez-Campos et al. refers to this category of applications as "sequential connections" [30]. By looking at the *Sequence Number* and *Acknowledgement number* of TCP packets, Hernåndez-Campos et al. are in better position to recognize concurrent data transmission. While they report that patterns of concurrent data exchanges are not commonly found in TCP connections on the Internet today, they can occur in certain protocols such as BitTorrent, HTTP, and NNTP [30] at some point of the communication. We anticipate that flows dominated by concurrent data transmissions would get a low score for both the indicators of transaction and conversation, since the direction of packets is expected to be irregular. Obviously, both of these indicators can also get an artificially lower value if many packets arrive out of order at the monitor.

4.2.6 Characterizing block ciphers

Finally, we developed a heuristic that attempts to determine whether packets are encrypted or not based on the following assumptions:

- Encryption schemes used by attackers are likely to utilize block ciphers. The Advanced Encryption Standard (AES) block cipher algorithm for instance is used in backdoor programs [34] and network utility tools [35].
- With block ciphers, lengths of encrypted packets typically have a greatest common divisor (GCD) different than one [28].
- Depending on the transport layers and the applications, it is possible that not all packets within a flow will be encrypted using a cipher block.

The algorithm follows an iterative process. At each step,

- the array of input is broken into two parts for pair wise GCD calculation, and
- the array to be examined in the following step will contain the GCD values that are greater than 1.

The process is interrupted if, at a given step, the count of GCDs that are greater than 1 is smaller than the count of GCDs equalled to 1.

The calculation is done for each direction separately, the output gives two values:

$\alpha_{\text{cipherblock}}$ gives the estimated popular GCD among payload lengths of packets.

$\beta_{\text{cipherblock}}$ gives the ratio of non-empty packet-payloads that are divisible by $\alpha_{\text{cipherblock}}$.

If the GCD calculation process got interrupted due to too many pair-wise GCD equal to one, then the value for $\alpha_{\text{cipherblock}}$ is equal to 1 and the value for $\beta_{\text{cipherblock}}$ is set to 0.

Currently, the initial array of input that is passed to the algorithm contains the payload lengths in bytes of each non-empty packet. It is worth mentioning that while this heuristic can recognize SSH and popular encrypted utility tools such as *aes-netcat* [35], it fails to identify applications for which one or more unencrypted application headers are appended to the application data. SSL is an example of the latter case. Therefore even when a SSL flow utilizes a block cipher encryption scheme, the flow will not be marked as such by our heuristics. Fortunately, the SSL protocol can be distinguished based on other flow attributes.

To demonstrate the relevance of the flow attributes in characterizing networked applications, we developed simple rule sets to recognize applications and describe the flow activity as discussed in the next sections.

5 Special-purpose Recognizers

We developed rule sets to distinguish among flows of various networked applications. Because we start with discriminative flow attributes, lightweight rule sets can be defined to classify flows. We call a *Recognizer* a set of rules associated with a given networked application. As part of this initial study, we concentrated on a limited number of applications and protocols. We have developed *Recognizers* for FTPdata, HTTP, HTTPS, IMAP, POP, SMTP, FTPcontrol, RLOGIN, SSH, TELNET, MSNchat, and TCPAudio. The *Recognizers* take as input a file containing flow records, in which each flow record is summarized based on the attributes listed in Table 4 and Table 5.

The flow features selected in this first round are listed in Table 6 for each *Recognizer*. To distinguish between the protocols listed above, we found that very few criteria were required. The selection of criteria was done based on knowledge of the protocols and the thresholds have been chosen by analysing samples of flows collected from our testbed. Since the traffic traces were collected in a controlled environment and for the purpose of this experiment, we did not have to worry about hidden malicious activities within our trace. As reported in [22] a major difficulty encountered when building profiles is to find a "clean" reference dataset. If the dataset is taken from a real-world network, selecting the network traffic for each class based on port numbers may not yield reliable profiles. Herein we produced traffic within the testbed for each of the 12 service types listed above. The initial study included an average of fifty flows per applications. While this does not qualify for an exhaustive reference dataset, it was considered a sufficient proof-of concept of the usefulness of the flow features in distinguishing among applications. One can expect however that from such a small dataset, the profiles may fail to represent all possible variants and implementations of a given networked application. Therefore the goal pursued when constructing the profiles was not to derive the most accurate set of rules but rather to identify distinguishing patterns using our flow attributes.

Recognizers are typically based on "give-away" features. For instance FTPdata is strictly unidirectional in terms of payload carried in non-empty packets; it is the Originator of a RLOGIN session that transmits the first non-empty packet, while in the case of TELNET it is the Responder; SMTP Responders avoid sending packets of length between 1 and 5 bytes and over 350 bytes. Such give-away features may not necessarily provide much insight about the communication. In fact, when defining *Recognizers*, we even refrain from using criteria based on interactivity for applications that may easily be scripted (using *expect* [36], for instance).

Deriving the profiles does require human decision when identifying patterns, whether the dataset is small or not. When a pattern can be interpreted, a human analyst can decide whether the pattern is likely to be observed on other datasets. We also recognize that since the communications are within a local network, the applications have a very high throughput due to the low Round-Trip-Time (RTT) and hence packet inter-arrival time may differ significantly from other datasets. For this reason we have used very loose thresholds for rules based on delays.

Some of the protocols for which we built *Recognizers* are in essence very close in terms of purposes (e.g. POP and IMAP for retrieving e-mails, HTTP and HTTPS for web browsing), but are found to be distinguishable with the indicators presented herein.

Table 6 provides a list of the flow features currently used in the rule sets composing the profiles of the different applications. Indicators (e.g. for conversation, interactivity, etc.) are constructed to get high scores when a given characteristic is strongly present. Low scores are also useful in many circumstances; therefore a check mark in Table 6 does not necessarily indicate that the criterion is based on a high value. We note herein that there is room to investigate new flow attributes constructed from those of Table 6. In particular, certain flow attributes have better discriminative power when combined with others. This is the case for instance with the data packet count in a given direction. While less significant on its own, the ratio of DatapacketCount over the PacketCount was found to be useful in many of the *Recognizers*. For TCP flows, this ratio corresponds to the relative proportion of times the packets are transmitted in order to send data (as opposed to acknowledge having received data). For some *Recognizers*, we also compare the count of data packet transmitted in each direction.

The actual rule sets used in each profile are provided in the Appendix.

Table 6. *Attributes as currently used by the Recognizers and Traffic Descriptor*

Attributes / Recognizers & Descriptor	'imap'	'pop'	'smtp'	'ssh'	'telnet'	'rlogin'	'ftpcmd'	'ftpdata'	'http'	'https'	'chatmsn'	'tcpaudio-stream'	Descriptor
Duration	√	√	√				√		√	√		√	√
FirstNonEmptyPacketSize	√	√	√		√				√	√		√	
FirstFewNonEmptyPacketDirections	√	√	√	√	√	√	√		√	√	√	√	
DatabyteRatioOrigToResp	√	√	√	√	√		√	√	√		√	√	√
InterarrivalDistribution													
Conversation Indicators						√				√	√		√
Transaction Indicator					√		√						√
Originator.InterarrivalDistribution													
Originator.PayloadDistribution	√	√	√	√	√	√	√	√	√	√	√	√	
Originator.ByteCount													√
Originator.DatabyteCount								√	√				
Originator.PacketCount	√	√	√					√					√
Originator.DatapacketCount	√	√	√		√		√	√		√			
Originator.EncryptionIndicators				√									
Originator.KeystrokeInteractiveIndicators									√				√
Originator.CommandlineInteractiveIndicators									√		√		√
Originator.FileTransferIndicators													√
Originator.BitrateMean													
Originator.$\alpha_{constantbitrate}$	√	√	√						√	√			√
Originator.PacketrateMean													
Originator.$\alpha_{constantpacketrate}$	√	√	√						√	√			√
Originator.PayloadMedian													
Originator.$\alpha_{constantpayload}$													√
Responder.InterarrivalDistribution													
Responder.PayloadDistribution	√	√	√	√			√	√		√	√	√	
Responder.ByteCount													√
Responder.DatabyteCount			√										
Responder.PacketCount	√	√		√	√	√	√	√					√
Responder.DatapacketCount	√	√	√	√	√	√	√	√					
Responder.EncryptionIndicators				√									
Responder.KeystrokeInteractiveIndicators									√				√
Responder.CommandlineInteractiveIndicators									√		√		√
Responder.FileTransferIndicators							√						√
Responder.BitrateMean													
Responder.$\alpha_{constantbitrate}$	√	√	√						√	√		√	√
Responder.PacketrateMean													
Responder.$\alpha_{constantpacketrate}$	√	√	√						√	√			√
Responder.PayloadMedian													
Responder.$\alpha_{constantpayload}$													√

6 Evaluation Experiment

For the purpose of a first evaluation early in the research process, we obtained a traffic trace that met our needs. The traffic trace contains traffic "to" and "from" the Internet and also contains a significant amount of internal LAN to LAN traffic. Over 20 million packets were captured from almost 2 thousands different IP addresses, for an average bandwidth utilization of 16 Mbits per second. Statistics about that trace can be found in Table 7.

Table 7. *Summary of the network traffic trace used for the preliminary evaluation*

Start Time:	February 10, 2005 13:38:38	Average bits/s:	16,017,040.06
Time Saved:	February 10, 2005 16:13:00	Max bits/s:	63,394,608.00
Duration:	2:34:21	Only the first 100 Bytes of each packet were saved on disk.	

Group	Statistics	Bytes	Packets	B/sec	P/sec	% of B	% of P
General	Total Bytes	18,372,330,066	-	1,983,651.27	-	100.00%	-
General	Total Packets	-	21,393,474	-	2,309.84	-	100.00%
General	Total Broadcast	26,768,909	183,990	2,890.23	19.865	0.15%	0.86%
General	Total Multicast	4,130,273	54,635	445.944	5.899	0.02%	0.26%
General	Dropped Packets	-	0	-	0	-	0.00%
Counts	Physical Addresses Seen	923	923	923	923	923	923
Counts	IP Addresses Seen	1,854	1,854	1,854	1,854	1,854	1,854
Counts	Protocols Seen	198	198	198	198	198	198
Size Distrib.	<= 64	-	2,539,406	-	274.178	-	11.87%
Size Distrib.	65-127	-	4,985,037	-	538.232	-	23.30%
Size Distrib.	128-255	-	547,283	-	59.09	-	2.56%
Size Distrib.	256-511	-	446,776	-	48.238	-	2.09%
Size Distrib.	512-1023	-	1,879,708	-	202.951	-	8.79%
Size Distrib.	1024-1517	-	3,826,915	-	413.19	-	17.89%
Size Distrib.	>= 1518	-	7,168,349	-	773.963	-	33.51%

Source: EtherPeek Analysis tool, from WildPackets (http://www.wildpackets.com/)

A large number of failed connection attempts was observed on the test dataset. In an effort to eliminate those failed connections, we kept only flows with more than three packets in each direction.

To facilitate the estimation of misclassifications, we separated the traffic into files based on port numbers commonly associated with the protocols considered. Other TCP/UDP traffic was directed to a separate file, which we further sorted based on four categories:

- netservices (LDAP, file sharing, printing, etc.),
- remotepc (rdesktop, timbuktu, pcanywhere, etc.),
- dns, and
- other TCP/UDP.

Note that we had no reference dataset for any of the last four categories of traffic. Since it is an objective to be able to classify new applications, these last traces can be quite insightful. In particular, as we will later see, a profile for the *remotepc* traffic can easily be derived from the profile of ftp data.

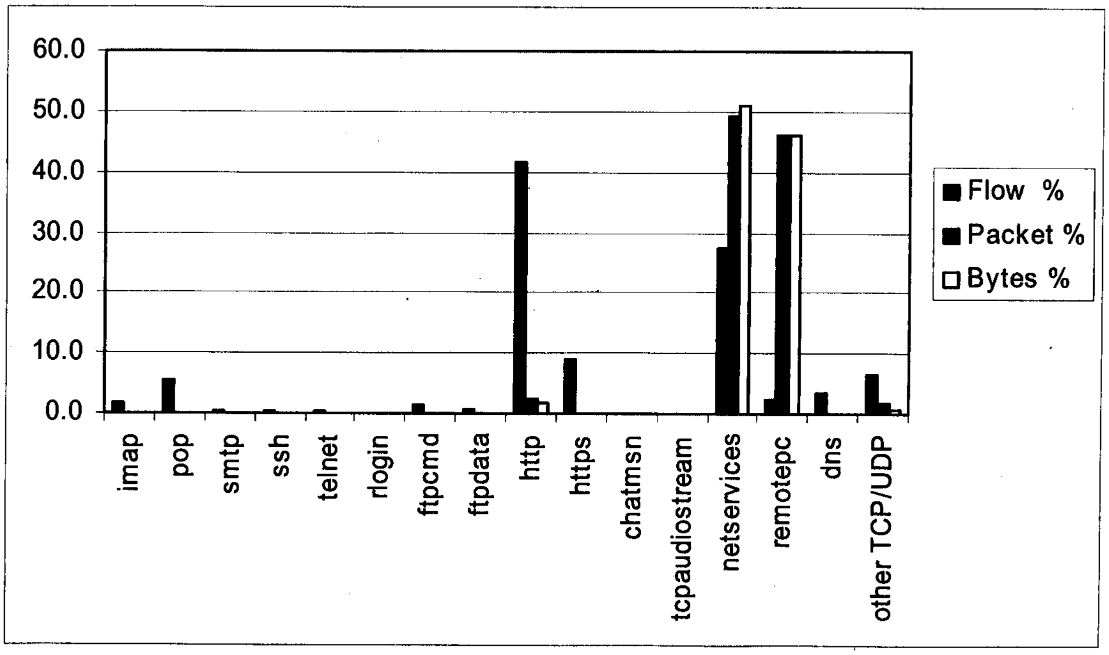


Figure 4. Proportion of flows, packets and bytes per application.

The sorted traffic traces are not evenly represented. The count of HTTP flows was much larger than the flow count for other protocols. In particular, there was no rlogin session and no MSN chat sessions running on the default port. There was one TCPAudio stream, but the beginning of the session was missing and the roles of "Originator" and "Responder" were reversed. That flow was not recognized as TCPAudio stream by our program. Figure 4 gives a high level view of the proportion of flows, packets and bytes per application. The statistics shown in Figure 4 are for flows with a minimum of three packets per direction. Therefore the actual flow count for the DNS traffic is much greater than what is artificially displayed here. For other applications, the flows discarded for the analysis are largely dominated by failed connection attempts. Had there been numerous UDP based streaming media flows (such as video conferencing flows) a greater number of legitimate sessions would have been mistakenly discarded for the analysis. This is because it is common for UDP based video conferencing applications to open unidirectional channels in each direction for carrying voice and video packets separately.

The evaluation process consisted of running all *Recognizers* over each trace and counting the flows that were correctly and incorrectly classified. The results are shown in Table 8 and Table 9. Table 8 contains the results obtained on the traces consisting of traffic the *Recognizers* were expected to distinguish. Table 9 reports the number of errors on other traffic.

Both tables are read line by line, where each line corresponds to a different traffic trace. The first column gives the category of traffic contained in the trace. The second column of Table 8 gives the count of flows correctly identified, the other three columns give the count of flows that the corresponding behavioural *Recognizer* failed to *uniquely* identify. For the '*pop*' trace for instance, 427 out of the 429 flows were marked as '*pop*'; however 51 flows marked as '*pop*' were also marked as '*imap*'. Therefore in the case of the POP protocol, we consider the count of flows correctly identified as being 376 instead of 427. In the columns reporting flows that were falsely marked, we also give the "labels" these flows were marked with.

Aside from the HTTP traffic trace for which the percentage of flows reported as "missed" was as high as 30% and the TCP Audio trace where the one TCP Audio flow was not recognized, the tool correctly identified 80% to 100% of the flows it was programmed to recognize. Although these results indicate that the profiles need to be refined, they are much more encouraging than those reported in [10] when performing the test on a new dataset.

Moreover, since the traffic traces were separated based on port numbers which we know cannot be reliable, what we have reported as "errors" (i.e. flows incorrectly identified) may in fact be associated to suspicious flows. Further examination of the traffic traces indicated that the majority of the flows in the HTTP traffic trace that were missed by the *HTTPRecognizer* had an unusually high proportion of very large packets flowing in the Client-to-Server direction. While the first bytes of many of these large packets indicated that they were HTTP *GET Requests*, it is hard to tell whether these were legitimate requests since most of the payload portion was stripped off at capture time. The author would rather mark these flows as being suspicious, especially since as noted in [2] several peer-to-peer protocols use HTTP requests and responses to transfer files.

Table 8. *Estimating Errors on traces consisting of traffic the tool is programmed to recognized*

Traces \ Errors	Flows Correctly identified	Ambiguous Flows (overlapping profiles)	Flows Falsely Identified	Unrecognized Flows
'imap' (140 flows)	113 flows (81%)	0	2 flow (1%): 1 'pop', 1 'telnet'	25 flows (18%)
'pop' (429 flows)	376 flows (88%)	51 flows (12%) imap/pop	0	2 flows (0.5%)
'smtp' (30 flows)	26 flows (87%)	0	0	4 flows (13%)
'ssh' (24 flows)	19 flows (79%)	0	1 flow (5%): 'telnet'	4 flows (16%)
'telnet' (15 flows)	15 flows (100%)	0	0	0
'rlogin' (0 flow)	-	-	-	-
'ftpcmd' (109 flows)	101 flows (93%)	0	0	8 flows (7%)
'ftpdata' (47 flows)	39 flows (83%)	0	0	8 flows (17%)
'http' (3249 flows)	2093 flows (64%)	69 flows (2%): 'http/https'	68 flows (2%): 'ftpdata' 33 flows (1%): 'https'	986 flows (30%)
'https' (700 flows)	643 flows (92%)	5 flows (0.7%): 'http/https'	2 flows (0.3%): 'http'	50 (7%)
'chatmsn' (0 flow)	-	-	-	-
'tcpaudiostream' (1 flow)	0	0	0	1 (100%) due to cold start.

On traces consisting of other traffic (Table 9), the false positive rate remained low with respect to the total number of flows included in those traces (3096 flows). The *Recognizers* have falsely marked 5% or less of the flows. However, when examining applications separately, some *Recognizers* frequently misclassified certain types of flows. For instance 30% of the DNS flows were marked as '*imap*'. The *FTPDataRecognizer* also falsely identified 72% of the '*remotepc*' flows. Further analysis of the '*remotepc*' trace indicated that the misclassification count goes to zero by adding two new rules to the *FTPDataRecognizer*. One rule is on the Duration, acknowledging that FTP data transfers are completed relatively quickly; and the other rule imposes that FTP data transfers cannot be interactive. These changes also slightly reduce the number of HTTP flows falsely marked as '*ftpdata*' in Table 8 (the count goes to 59 flows instead of 68), while not affecting the accuracy on the '*ftpdata*' trace. The results presented in Table 8 and Table 9 do not reflect the changes made to the *FTPDataRecognizer* rule sets. The reader can refer to Table 6 and the Appendix to see which flow attributes have been used to produce this first round of results. It is worth mentioning that the two flows marked as '*chatmsn*' among the "other TCP/UDP" category turned out to be Yahoo Messenger sessions over TCP/5101.

Table 9. *Count of misclassified flows in traces consisting of other traffic*

Traces \ Recognizers	'imap'	'pop'	'smtp'	'ssh'	'telnet'	'rlogin'	'ftpcmd'	'ftpdata'	'http'	'https'	'chatmsn'	'tcpaudio-stream'
'netservices' (2136 flows)	40	0	0	0	0	0	1	4	58	42	0	0
'remotepc' (188 flows)	0	0	0	0	0	0	0	135	0	0	0	0
'dns' (267 flows)	85	0	0	0	0	0	10	0	0	0	0	0
'other TCP/UDP' (504 flows)	15	2	0	7	0	0	5	3	8	80	2	0
Total (3096 flows)	140 (5%)	2	0	7	0	0	16	142 (5%)	66	122 (4%)	2	0

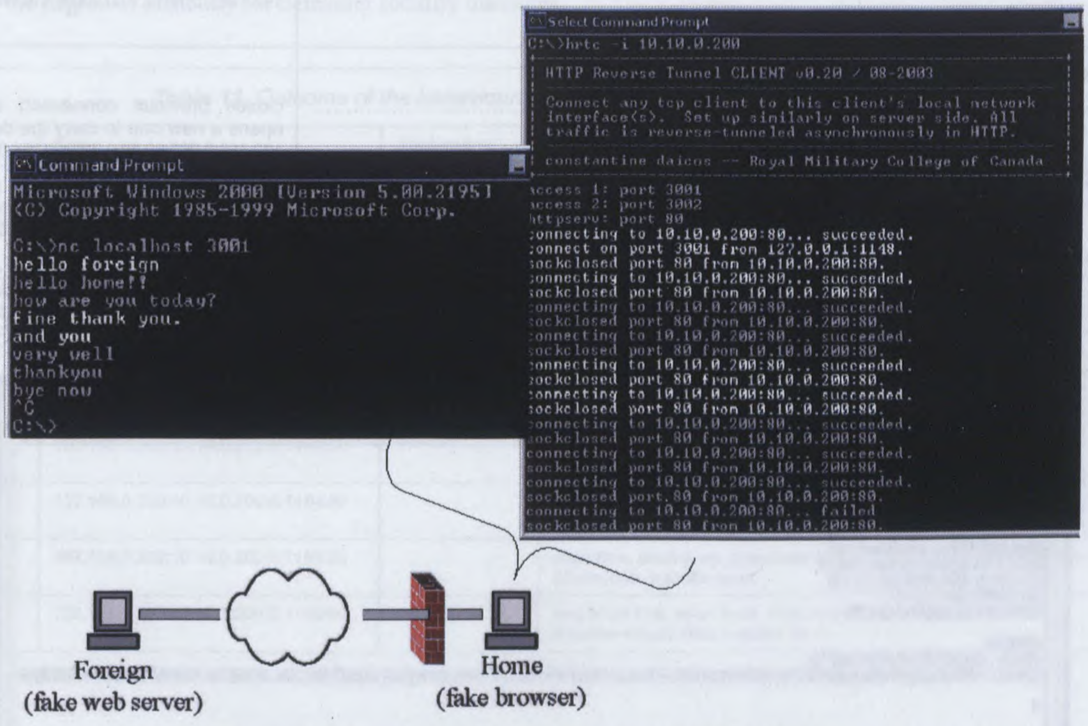
We plan to refine some of the profiles based on this preliminary experiment, and follow up with similar evaluation experiments in the near future.

7 Evaluation Experiment using a Subversive Tool

We have also tested our proof of concept program against a subversive tool called HTTP Reverse Tunnel (HRT) [37], which was developed at the Royal Military College of Canada to demonstrate that any networking activity disguised as HTTP can be tunnelled through proxies and firewalls. A first experiment with HRT used as a chat session showed that the HTTP disguise failed because of the difficulty of imitating all behavioural characteristics of the given protocol.

We briefly describe this primarily test with HRT used as a chat session. HRT is a Windows 2000/XP implementation of an HTTP reverse tunnel that serves as a transport medium for tunneling arbitrary data over HTTP. The "Quick-Start Guide" that is provided with the tool describes how it can be used with *netcat* to allow for an interactive covert channels.

HRT consists of two parts, one that acts as a client (to simulate a web browser) and one that act as a server (to simulate a web server). Assuming that the client side has been successfully installed on a host behind a protected network, upon execution of the HRT program on the client host, the client establishes an HTTP tunnel with the server and data from either direction can be transmitted within that opened connection. For this first preliminary experiment, we used HRT coupled with *netcat* as described in the "Quick-Start Guide" to create a chat session between two hosts and we captured the traffic involved. Figure 5 depicts the scenario where a host named *foreign* is running the server side of HRT and a host named *home* is running the client side. The host named *home* has the IP address 192.168.0.200 and the host *foreign*, located on a different network, was given the IP address 10.10.0.200. The two Command Prompt windows shown in the Figure were opened on the client side. One window was used to launch HRT and the other to use netcat on the opened channel to send and receive the data. The corresponding Command Prompt windows on the server side were omitted from the Figure for clarity. As the left Command Prompt window indicates, both hosts took turn to chat. The details of the commands are given in Table 10. The tool does a really good job at disguising the activity. TCP flows are always established by the client side as it is expected with legitimate web browsing activities, and the data carried appears as legitimate HTTP requests and responses as shown in Figure 6 for one of the TCP connection produced during this experiment. By emulating legitimate web browser behaviour, this tunnel pierces most contemporary network perimeter defences like firewall, IDS, and HTTP caching proxies [37].



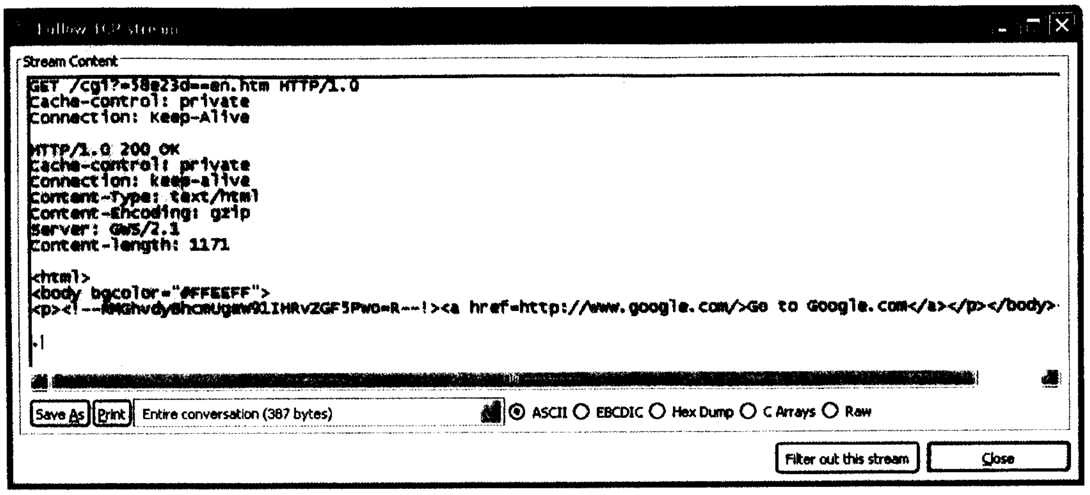
```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\>nc localhost 3001
hello foreign
hello home!!
how are you today?
fine thank you.
and you
very well
thankyou
bye now
^C
C:\>
```

```
C:\>hrtc -i 10.10.0.200
+-----------------------------------------------------------+
| HTTP Reverse Tunnel CLIENT v0.20 / 08-2003                |
|-----------------------------------------------------------|
| Connect any tcp client to this client's local network     |
| interface(s).  Set up similarly on server side. All       |
| traffic is reverse-tunneled asynchronously in HTTP.       |
|-----------------------------------------------------------|
| constantine daicos -- Royal Military College of Canada    |
+-----------------------------------------------------------+
access 1: port 3001
access 2: port 3002
httpserv: port 80
connecting to 10.10.0.200:80... succeeded.
connect on port 3001 from 127.0.0.1:1148.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port 80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... succeeded.
sockclosed port.80 from 10.10.0.200:80.
connecting to 10.10.0.200:80... failed
sockclosed port 80 from 10.10.0.200:80.
```

Figure 5. HRT coupled with netcat to create a chat session

***Table 10.** Commands and Traffic flows produced using HRT as a chat session*

Commands	**Traffic Flows**
<u>On Foreign (10.10.0.200):</u>	
Open cmd shell and run the server: C:\>hrts	
Open cmd shell and connect netcat to default local port on hrts: C:\>nc localhost 2001	
<u>On Home(192.168.0.200):</u>	
Open cmd shell and run the client: C:\>hrtc -i 10.10.0.200	192.168.0.200:10.10.0.200:6:1147:80
Open cmd shell and connect netcat to default local port on hrtc: C:\>nc localhost 3001	
In netcat shell, type "hello foreign" and hit enter:...	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1149:80
<u>On Foreign(10.10.0.200):</u>	
In netcat shell, type "hello home!!" and hit enter:...	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1150:80
In netcat shell, type "how are you today?" and hit enter:	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1151:80
<u>On Home(192.168.0.200):</u>	
In netcat shell, type "fine thank you." and hit enter:...	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1152:80
In netcat shell, type "and you." and hit enter:...	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1153:80
<u>On Foreign(10.10.0.200):</u>	
In netcat shell, type "very well" and hit enter:..	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1154:80
In netcat shell, type "tankyou" and hit enter:	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1155:80
In netcat shell, type "bye now" and hit enter:	closes previous connection and opens a new one to carry the data: 192.168.0.200:10.10.0.200:6:1156:80



```
GET /cgi?=58e23d==en.htm HTTP/1.0
Cache-control: private
Connection: Keep-Alive

HTTP/1.0 200 OK
Cache-control: private
Connection: keep-alive
Content-Type: text/html
Content-Encoding: gzip
Server: GWS/2.1
Content-length: 1171

<html>
<body bgcolor="#FFEEFF">
<p><!--RMGhvdyBhcmUgaW91IHRvZGF5Pwo=R--!><a href=http://www.google.com/>Go to Google.com</a></p></body>

.1
```

Entire conversation (387 bytes) [✓ ASCII] EBCDIC Hex Dump C Arrays Raw

Figure 6. Data disguise example as produced by HRT
Source: Ethereal, www.ethereal.com

During the experiment, each line typed in the *netcat* Command Prompt was transmitted in a new TCP connection. This had the effect of hiding from our analysis tool the interactivity since the indicators are measured on each flow separately. However, although each flow was meant to appear as a legitimate HTTP connection established on port TCP/80, none of the flows were recognized as HTTP. In the context of network security, these flows would have been marked suspicious. As shown in Table 11 in the column entitled "Application" these flows were not recognized as any other application neither. The last column entitled "Description Summary" provides a description of the flows as perceived by our tool. How we produce this description will be briefly discussed in section 8.

The HTTP disguise failed in this case because of two attributes, both measuring characteristics of the packet payload size. Namely, the first attribute is *FirstNonEmptyPacketSize*, and the second is *Originator.PayloadDistribution*. The reader can refer to Table 4 and Table 5 of Section 4.2 for a short description of these attributes. Moreover, aside from one flow marked as bidirectional, the directionality of the flows was correctly identified as per the last column of Table 11. While the Originator of each flow is the host named *home* (IP 192.168.0.200), the Description Summary reveals the direction in which the data was actually flowing.

It is expected that when HRT is used for file transferring then the disguise will have a better chance to succeed. However for tunneling chat sessions, some padding would be required to mimic the packet length distribution of HTTP traffic. The HTTP tunnel will be a nice tool to test against and we plan to conduct a more thorough set of tests covering a variety of tunneling scenarios.

As a general note on subversive tools, we acknowledge that detecting network misuse and abuse is much more difficult when the attacker deliberately attempts to circumvent detection. The traffic recognition approach adopted here is expected to work well for benign-evasive activities such as the use of port-agile peer-to-peer applications in order to bypass firewall rules. However the profiling mechanisms can be defeated by ingenious intruders, though it requires more effort, time, and more sophisticated intrusion software such as HRT. To defeat our classification technique, packets could be padded to fixed or varying lengths, transmission of packets could be regulated or randomized, and "dummy" packets could even be transmitted to further obscure the dynamics of inter-packet departure. While ideally any detection algorithms should be conceived to be resistant to evasive attackers, ensuring such robustness can sometimes be exceedingly difficult. We feel that there is utility in elevating the degree of difficulty for defeating security measures.

Table 11. *Outcome of the behavioural analysis tool on the HRT chat sessions*

Flow	Key	Application	DescriptionSummary
1	192.168.0.200:10.10.0.200:6:1147:80		short flow, short-lived, directional:OriginatorToResponder, machine-driven, supportive
2	192.168.0.200:10.10.0.200:6:1149:80		short flow, short-lived, directional:OriginatorToResponder, machine-driven, streamed-media:supportive
3	192.168.0.200:10.10.0.200:6:1150:80		short flow, short-lived, directional:ResponderToOriginator, machine-driven, streamed-media:data-transfer:spurt
4	192.168.0.200:10.10.0.200:6:1151:80		short flow, short-lived, directional:ResponderToOriginator, machine-driven, data-transfer:spurt
5	192.168.0.200:10.10.0.200:6:1152:80		short flow, short-lived, bidirectional, machine-driven, supportive
6	192.168.0.200:10.10.0.200:6:1153:80		short flow, short-lived, directional:OriginatorToResponder, machine-driven, supportive
7	192.168.0.200:10.10.0.200:6:1154:80		short flow, short-lived, directional:ResponderToOriginator, machine-driven, data-transfer:spurt
8	192.168.0.200:10.10.0.200:6:1155:80		short flow, short-lived, directional:ResponderToOriginator, machine-driven, data-transfer:spurt
9	192.168.0.200:10.10.0.200:6:1156:80		very short flow, short-lived, directional:ResponderToOriginator, machine-driven, data-transfer:spurt

8 Flow Descriptor

On top of allowing classification of network traffic, the flow attributes we derived may be used to describe the network activity. We have developed a flow *Descriptor* that gives a human-readable description of each flow. As with the special-purpose *Recognizers*, the flow *Descriptor* takes as input a file containing flow records. The output gives a *FlowDescription* for each flow record. A *FlowDescription* is a structure with six attributes as summarized in Figure 7. The *FlowDescription* is based on directionality, length (in terms of packet count and of duration), control (human-driven or machine-driven), transmission rate, and possible purpose.

The output of the *Descriptor* is quite useful for a security analyst as it provides the practitioner with insight about the nature of the communication. A given application may receive different descriptions depending on its use. The *Descriptor* has also proved to be useful when investigating certain flows running on unprivileged ports. For instance, a traffic trace examined recently contained a significant number of flows on high ports (TCP/6881- TCP/6886) that were described as "long flow, persistent, directional:OriginatorToResponder, machine-driven, data-transfer:bulk". These ports are now known to be associated with BitTorrent [38], a peer-to-peer file sharing technology quickly growing in popularity [39].

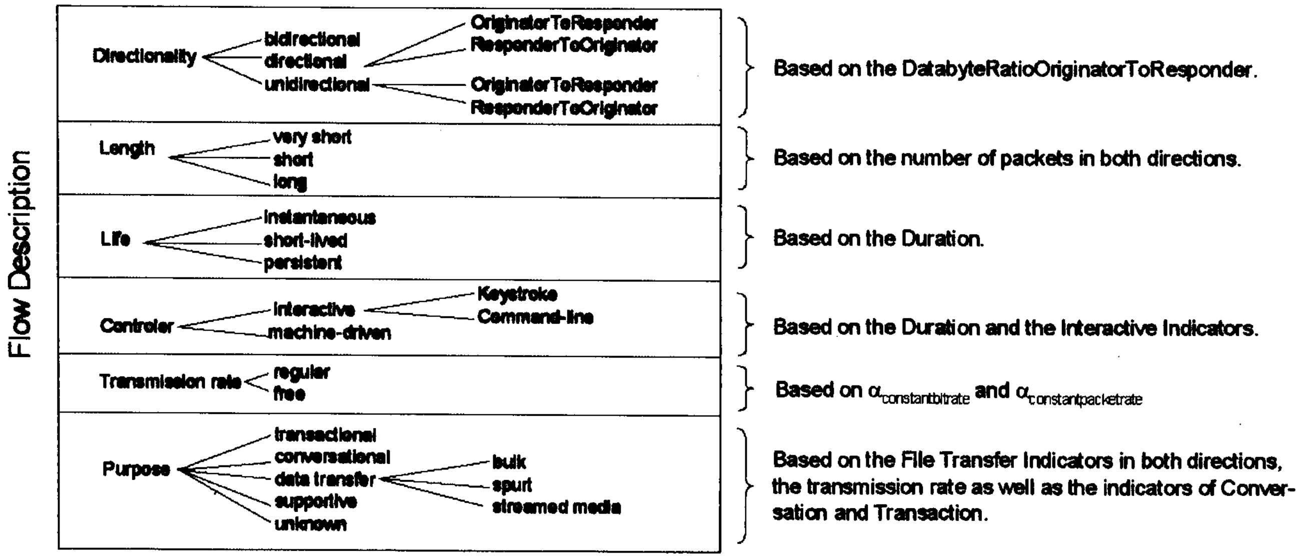


Figure 7. Flow Description

9 Current State and Future work

The characterization process is currently done off-line, and in three steps. The first step is programmed in JAVA and consists of analysing a packet trace to group the packets into flow records. The next step is currently done with MATLAB [40] and consists of extracting the essential flow attributes. The output is a file of flow records in which all flow records are summarized with the same set of flow attributes. Finally, the last step, also programmed in MATLAB, consists in processing the file containing the flow attributes to deduce information about the nature of each flow. During this step we use both the special-purpose *Recognizers* and the flow *Descriptor* to tag each flow with two properties: the application recognized (if any) and the FlowDescription.

We are in the initial stages of our study where the focus has been placed on identifying flow attributes that are useful in characterizing network traffic. We have identified flow attributes that can differentiate between specific networked applications and also between categories of networked applications (interactive remote control, file transfers, streaming media, and chat). While we examined applications running on either TCP or UDP, the majority of the services considered are TCP based. Much remains to be done, in particular we plan to:

- Based on the preliminary evaluation of Section 6, refine the special-purpose *Recognizers* developed so far and reassess the accuracy on different real world traffic traces.
- Examine a greater number of applications not yet considered (e.g. VoIP, gaming traffic). This may involve developing new flow attributes to capture other important traffic characteristics.
- Examine the traffic produced by a number of subversive tools such as those discussed in [37].
- Identify the most important subset of flow attributes for discriminating between applications. While we are still in the exploratory stage of flow characterization, future work may require reducing the number of measurements of each flow to accelerate the analysis without loosing significant accuracy. The trade-off between performance and accuracy will need to be determined.
- Add the capability of doing cross-examination of flows to determine patterns of applications than span over multiple connections. On the other hand, it may also be interesting to examine portions of a flow through sliding windows in order to characterize "episodes" within that flow.

- Examine the possibility of estimating the flow attributes in near real-time as opposed to measuring them over a connection's total lifetime. Selecting attributes that can be updated in a streaming (recursive) fashion [22] [41] would allow approximation of measurements in near real-time and eliminate the need for storing data per packet.
- Examine the possibility of using alternative classification methods to derive profiles. The profiles described in Section 5 were derived manually. It would be interesting to see how machine-learning techniques could be used in step 3 to classify traffic based on the flow features obtained from step 2.
- Rebuild the tool as a stand-alone program written in C or in JAVA (as opposed to MATLAB), or perhaps include the capability into an existing tool if applicable.

There also are a few technical problems we would like to address. As mentioned in Section 2.6, problems due to the "cold start" of the monitor and the presence of fragments are overlooked at the moment. A future version may include some precautions when constructing the flows with regards to these two shortcomings. We also have encountered a number of difficulties during the validation process that we hope to address for future evaluation experiment. In particular, even when much (if not all) of the payload is available in each packet, it is still extremely painful and difficult to identify the true nature of a flow to validate our results. On the other hand, we note that it is exactly because manual investigation of flows is so cumbersome that we put effort into developing analysis tools like the one presented herein. As mentioned in Section 6, we also faced another shortcoming during the preliminary evaluation when attempting to eliminate failed connections from our analysis. We kept only flows with a minimum of three packets in each direction. Had there been several unidirectional flows in the traffic trace, this practice would have mistakenly discarded many flows from our analysis.

This work on traffic recognition is an ongoing project. We are currently doing outreach activities to other researchers in the field to identify avenues of collaboration. This will help further develop the concepts and to see the problem from different perspectives. We also have recently initiated a related project with Virtual Private Networks (VPNs) for which one of the goals is to determine how the encryption layer alters the characteristics measured by the flow attributes described herein. The study will focus on traffic produced by commonly available business class VPN equipment.

10 Conclusions

We have presented in this document a number of indicators that reveal intrinsic characteristics by which we can recognize particular networked applications or groups of networked applications. The flow attributes are supported by interpretation with domain knowledge and thus allows separation of traffic into flows of human-recognizable categories. When flow attributes are meaningful and discriminative, lightweight rule sets based on these attributes can be defined to classify flows. In this work, much of the effort so far has been concentrated on identifying significant flow attributes. We have also experimented with special-purpose *Recognizers* that can pinpoint particular applications. Our methodology can be used not only to "tag" a flow as belonging to a given category, but is also useful at providing a human-interpretable description of what the flow is about. This provides a starting point when investigating suspicious flows.

We have developed a proof of concept tool that analyses pre-recorded traffic traces. The tool first extracts for each flow the important flow attributes. Then based on simple rule sets the tool attempts to recognize as whether a flow belongs to one the commonly used protocols and at the same time provides a description of the flow behaviour. Preliminary assessments with the proof-of-concept tool are quite encouraging. With further research and development, this tool can become very handy in many security-related tasks. It could be used to document and label traffic traces produced in the context of experiments or forensic analysis; allow an analyst to search for flows matching a given profile (a particular network service or flow description); identify suspect flows that may be in violation of the security policy; alert security personnel; and provide contextual information on flows surrounding a security event.

The majority of currently available tools that classify traffic rely on well-known port numbers to identify the networked applications. Some of these monitoring tools also use protocol-aware mechanisms based on payload decoding. The author believes that cloaking attacks to appear as innocuous applications will greatly accelerate. Mechanisms that infer the true nature of information flows based on traffic behaviour can be use to increase the level of confidence in the monitoring tools.

The pressing need for alternatives to correctly identify network activities has attracted attention in the research community. However, a large proportion of related work focus on the classification algorithms and take as input basic flow features, such as those provided by NetFlow[13] and *Argus* [17]. We argued that it is necessary to continue investigate discriminative characteristics and ways to measure these characteristics so that the nature of the communications can be revealed. We hope that our findings will assist other researchers in their search for new attributes to increase the discriminating capabilities of their classifiers.

References

[1] J. P. Early, C. E. Brodley, C. Rosenberg, "Behavioral Authentication of Server Flows", in Proc. of the 19th Annual Computer Security Applications Conference, December 2003.

[2] T. Karagiannis, A.Broido, M. Faloutsos, and K.C. claffy. "Transport layer identification of P2P traffic". In IMC, 2004.

[3] iNetPrivacy Software: AntiFirewall^{TM} (~35$), http://www.antifirewall.com.

[4] Realtime Traffic Flow Measurement Working Group (RTFM), in proceeding of the Thirty-fourth Internet Engineering Task Force (IETF), Dallas, Texas; December 1995. http://www.ietf.org/proceedings/95dec/ops/rtfm.html

[5] N. Brownlee, C. Mills, G. Ruth, "Traffic Flow Measurement: Architecture", RFC 2722, category Informal, October 1999.

[6] S. Handelman, S. Stibler, N. Brownlee, G. Ruth, "RTFM: New Attributes for Traffic Flow Measurement", RFC 2724, category Experimental., October 1999

[7] K. Keys, D. Moore, R. Koga, E. Lagache, M. Tesch, and k. claffy, "The Architecture of CoralReef: an Internet Traffic Monitoring Software Suite," in PAM2001 -- A workshop on Passive and Active Measurements, Apr 2001.

[8] K. Claffy, H-W. Braun, G. Polyzos, "A Parameterizable Methodology for Internet Traffic Flow Profiling," IEEE JSAC, 13(8), pp. 1481-1494 Oct.1995.

[9] Ryu, D. Cheney, H. Braun, Internet Flow Characterization: Adaptive Timeout Strategy and Statistical Modeling, Passive and Active Measurement Workshop, Amsterdam, April 2001.

[10] S. Abdulrahman "Network Intrusion Detection Using Flow Characterization", project description, http://www.cs.utk.edu/~abdulrah/project/paper.html

[11] T. Dunigan, G. Ostrouchov, "Flow Characterization for Intrusion Detection", ORNL/TM-2001/115, November 2000, available at http://www.csm.ornl.gov/~dunigan/pubs.html

[12] A. Soule, K. Salamatian, N. Taft, R. Emilion, and K. Papagiannaki. "Flow Classification by Histograms or How to Go on Safari in the Internet. In Proceedings of ACM Sigmetrics", New York, NY, June 2004.

[13] Cisco, "NetFlow", Cisco's flow measurement technology, http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html

[14] C. Gates, M. Collins, M. Duggan, A. Kompanek and M. Thomas, "More Netflow Tools: For Performance and Security", Proceedings of USENIX LISA, Atlanta, November 2004.

[15] CAIDA, "cflowd", traffic flow analysis tool, http://www.caida.org/tools/measurement/cflowd

[16] CERT/NetSA, "SiLK", NetFlow Analysis Suite, http://sourceforge.net/projects/silktools/

[17] QoSient, "Argus", network Audit Record Generation and Utilization System, http://www.qosient.com/argus

[18] Nevil Brownlee, "NeTraMet", an open-source implementation of the RTFM architecture, http://www2.auckland.ac.nz/net/NeTraMet/

[19] N. Brownlee, "Using NeTraMet for Production Traffic Measurement", in Integrated Management Strategies for the New Millenium, 14-18 May 2001.

[20] T. Karagiannis, K. Papagiannaki and M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark," in Proc. of ACM SIGCOMM, August 2005.

[21] K. Xu, Z. Zhang, and S. Bhattacharya. "Profiling Internet Backbone Traffic: Behavior Models and Applications". In SIGCOMM, 2005.

[22] M. Roughan, S. Sen, O. Spatscheck, N. G. Duffield. "Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification". In IMC 2004.

[23] A. W. Moore and D. Zuev. "Internet Traffic Classification Using Bayesian Analysis Techniques". In ACM SIGMETRICS, 2005.

[24] W. Lee, S.J. Stolfo, "A Framework for Constructing Features and Models for Intrusion Detection Systems," ACM Transactions on Information and System Security, Vol. 3 No. 4, November, 2000.

[25] DARPA Intrusion Detection Evaluation, Lincoln Laboratory, http://www.ll.mit.edu/IST/ideval/

[26] KDD-cup data set, data set used for The Third International Knowledge Discovery and Data Mining Tools Competition, available at http://kdd.ics.uci.edu/ databases/kddcup99/kddcup99.html

[27] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Computer Networks, 31(23-24), pp. 2435-2463, 14 Dec. 1999.

[28] Y. Zhang and V. Paxson, "Detecting Backdoors", in Proc. of USENIX Security Symposium, August 2000.

[29] M. Ilvesmaki, M Luoma, "On the capabilities of application level traffic measurements to differentiate and classify Internet traffic", in Proc. of SPIE, Internet performance and Control of Network Systems II, Vol.4523, 2001.

[30] F. Hernández-Campos, A. B. Nobel, F. Donelson Smith, K. Jeffay, "Understanding Patterns of TCP Connection Usage with Statistical Clustering", in Proc. of the Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Atlanta, September 2005.

[31] K. Lan, J. Heidemann, "Structural Modeling of RealAudio Traffic". Technical Report ISI-TR-544, USC/Information Sciences Institute, Sept. 2001.
[32] T. Kuang, C. Williamson, "A Measurement Study of RealMedia Audio/Video Streaming Traffic", in Proc. of SPIE ITCOM, pp. 68-79, July 2002.
[33] V. Paxson, "Empirically-Derived Analytic Models of Wide-Area TCP Connections", IEEE/ACM Transactions on Networking, Vol. 2 No. 4, August 1994.
[34] Les Gordon, "SANS Intrusion Detection FAQ, What is the Q Trojan?", http://www.sans.org/resources/idfaq/qtrojan.php
[35] *aes-netcat*, by Mixter, is a patch for netcat 1.10 in order to provide netcat with AES encryption. Available at http://packetstormsecurity.org/UNIX/utilities/
[36] *expect*, by Don Libes, a tool for automating interactive applications such as telnet, ftp, passwd, etc. Available at http://expect.nist.gov/
[37] C. Daicos, G.S. Knight, "Concerning Enterprise Network Vulnerability To Http Tunnelling", IFIP TC11 18th International Conference on Information Security, Athens, May 2003.
[38] *BitTorrent*, a file distribution software, available at http://bittorrent.com
[39] CacheLogic, a market research firm. http://www.cachelogic.com/news/mediacoverage.php
[40] MATLAB, a technical computing language. It is a commercial product by MathWorks http://www.mathworks.com/products/matlab/
[41] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss. "Fast, small-space algorithms for approximate histogram maintenance". In Proceeding of the ACM Symposium on Theory of Computing, Montreal, 2002.

Bibliography

[1] W. Richard Stevens, "TCP/IP Illustrated", Volume 1: The protocols, Addison-Wesley, 1994.
[2] J. Postel, RFC 791: "Internet Protocol", status: standard, September 1981, available at http://www.ietf.org/rfc/.
[3] J. Postel, RFC 792: "Internet Control Message Protocol", status: standard, September 1981, available at http://www.ietf.org/rfc/.
[4] J. Postel, RFC 793: "Transmission Control Protocol", status: proposed standard, September 1981, available at http://www.ietf.org/rfc/.
[5] J. Postel, RFC 768: "User Datagram Protocol", status: standard, August 1980, available at http://www.ietf.org/rfc/.
[6] J. Klensin, RFC 2821: "Simple Mail Transfer Protocol", status: proposed standard, April 2001, available at http://www.ietf.org/rfc/.
[7] M. Crispin, RFC 3501, "Internet Message Access Protocol", version 4rev1, status: proposed standard, March 2003, available at http://www.ietf.org/rfc/.
[8] J. Myers, M. Rose, RFC 1939, "Post Office Protocol", version 3, status: standard, May 1996, available at http://www.ietf.org/rfc/.
[9] P. Chown, RFC 3268, "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", status: proposed standard, June 2002, available at http://www.ietf.org/rfc/.
[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, RFC 2616, "Hypertext Transfer Protocol – HTTP/1.1", status: draft standard, June 1999, available at http://www.ietf.org/rfc/.
[11] R. Khare, S. Lawrence, RFC 2817, "Upgrading to TLS within HTTP/1.1", status: proposed standard, May 2000, available at http://www.ietf.org/rfc/.
[12] E. Rescorla, RFC 2818, "HTTP Over TLS", status: informal, May 2000, available at http://www.ietf.org/rfc/.
[13] W. Lee, S. J. Stolfo, K. W. Mok. "Mining in a data-flow environment: Experience in network intrusion detection". In Proc. of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-99), August 1999.
[14] W. Lee, S. J. Stolfo. "Data mining approaches for intrusion detection", in Proc. of the 7th USENIX Security Symposium, 1998.
[15] W. Lee, S. J. Stolfo. "Combining Knowledge Discovery and Knowledge Engineering to Build IDSs". In Proc. of the Workshop on Recent Advances in Intrusion Detection (RAID), September 1999.
[16] M. Mellia, A. Carpani, R. Lo Cigno, "Measuring IP and TCP behavior on Edge Nodes", IEEE Globecom, Taipei (TW), Nov. 2002.
[17] Y. Zhang and V. Paxson, "Detecting Stepping Stones", In Proc. of the 9th USENIX Security Symposium, August 2000.
[18] X. Wang, D. Reeves, S. F. Wu, "Inter-Packet Delay Based Correlation for Tracing Encrypted Connections Through Stepping Stones", in Proc. of European Symposium on Research in Computer Security (ESORICS 2002), Aug 2002.

Appendix: Recognition Criteria

The appendix contains the rule sets currently used in the profiles of the protocols found in Table 6. To satisfy a profile, all of the specified tests must succeed. Note that the lines corresponding to the rules are not exactly written as is in the proof of concept tool. They have been transcribed in a form that is easy to follow. As mentioned in Section 5, the profiles were built from examining small samples of flows per applications. Therefore the profiles may fail to represent all possible variants and implementations of a given network service. The goal pursued when constructing the profiles was not to derive the most accurate set of rules but rather to identify distinguishing patterns using our flow attributes. Even so, the rules below did perform reasonably well at recognizing network activities during preliminary experiments described in Sections 6 and 7. Because they constitute a starting point for deriving profiles, we chose to provide them in appendix. Also note that the rules below apply to the first flow seen with a given 5-tuple key (IP addresses, IP protocol, TCP/UDP ports). When processing subsequent flows associated with the same key, the rules actually implemented differ slightly, in some cases, from those provided below.

1 HTTP web browsing

Test_duration:

```
Duration > 50000 µsec
```

Test_transmissionrate:

Originator.$\alpha_{constantbitrate}$ <0.5 && Originator.$\alpha_{constantpacketrate}$ <0.5 &&
Responder.$\alpha_{constantbitrate}$ <0.5 && Responder.$\alpha_{constantpacketrate}$ <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

```
Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([180-650[)>0.8  &&
Originator.PayloadDistribution([1-100[)+Originator.PayloadDistribution( [1380-inf[)==0
```

i.e. At least 80% of the packets transmitted by the Originator fall in the bins corresponding to 0 byte and [180-650[bytes of payload; and the Originator of the connection never sends packets with 1 to 100 bytes of payload nor does it send packets containing more than 1380 bytes of payload.

Test_databyteratio:

```
0.005<DatabyteRatioOrigToResp<4
```

i.e. The server sends less than 200 times the data it receives, and the client never sends more than 4 times the data it receives. Note that we allow the client to send more data than the server in case the page is in cache.

Test_requestdatabyte:

```
Originator.DatabyteCount < 21000
```

i.e. The client sends less than 21000 bytes in total (which could correspond to 60 HTTP GET requests containing an average of 350 bytes each).

Test_firstnonemptypacketsize:

```
120 < FirstNonEmptyPacketSize < 1000
```

i.e. The first non-empty packet of the session, which is a HTTP GET, contains at least 120 bytes of data (small URL and only essential HTTP fields) and at most 1000 bytes of data (long URL and many HTTP fields).

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:2)=[1, -1]
```

i.e. The first non-empty packet is sent by the Originator (client) and the second is sent by the Responder (server).

Test_noconsecutivesmallpackets:

Originator.$\gamma_{key\_interactive} \leq 0$ && Originator.$\gamma_{cmd\_interactive} \leq 0$ &&
Responder.$\gamma_{key\_interactive} \leq 0$ && Responder.$\gamma_{cmd\_interactive} \leq 0$

2 HTTPS web browsing

Test_duration:

```
Duration > 50000 µsec
```

Test_transmissionrate:

Originator.$\alpha_{constantbitrate}$ <0.5 && Originator.$\alpha_{constantpacketrate}$ <0.5 &&
Responder.$\alpha_{constantbitrate}$ <0.5 && Responder.$\alpha_{constantpacketrate}$ <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

```
Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([50-180[) > 0.6  &&
Originator.PayloadDistribution([1-5[)+Originator.PayloadDistribution( [1000-inf[)==0 &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([20-100[)+
Responder.PayloadDistribution([549-inf[) > 0.6
```

i.e. At least 60% of the packets transmitted by the Originator fall in the bins corresponding to 0 byte and [50-180[bytes of payload; and the Originator of the connection never sends packets with 1 to 5 bytes of payload nor does it send packets containing more than 1000 bytes of payload. A large proportion (60%) of the packets transmitted by the Responder falls in the bins corresponding to 0 byte, [20-100[bytes and over 549 bytes of payload.

Test_datapacketcount:

```
Originator.datapacketcount<10
```

i.e. The Originator sends very few non-empty packets. This rule failed to recognize 8 flows out of 700 in the Evaluation Experiment described in Section 6. The flows missed contained 60 or less non-empty packets in the direction of the Originator-to-Responder.

Test_firstnonemptypacketsize:

```
90 < FirstNonEmptyPacketSize < 250
```

i.e. The first non-empty packet of direct SSL connections (a *SSL Client Helo* packet) is typically small (contains very few cipher specifications). When tunnelling through a HTTP proxy, the size of the first non-empty packet (a *CONNECT* packet) is also expected to be within the specified boundaries. As is, this rule failed to recognize 4 flows out of 700 in the Evaluation Experiment described in Section 6. The flows missed had a first non-empty packet of size between 250 bytes and 300 bytes.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:2)=[1, -1]
```

i.e. The first non-empty packet is sent by the Originator (client) and the second is sent by the Responder (server).

Test_conversation:

(Originator.$\alpha_{conversation}$ > 0.25 && Originator.datapacketcount $\geq$ 5) || Originator.datapacketcount < 5

i.e. Based on the 50 flows composing the training data set, 95% of the https connection had a score higher than 0.25 for the first conversation metric. This score is higher than with http. The flows composing the remaining 5% had less than 5 data packets in the Originator to Responder direction. Therefore the conversation rule does not discard a flow containing too few data-packets.

3 IMAP[6]

Test_duration:

```
Duration > 100000 μsec
```

Test_transmissionrate:

$\text{Originator.}\alpha_{constantbitrate} < 0.5$ && $\text{Originator.}\alpha_{constantpacketrate} < 0.5$ &&

$\text{Responder.}\alpha_{constantbitrate} < 0.5$ && $\text{Responder.}\alpha_{constantpacketrate} < 0.5$

i.e. The transmission rate is more irregular than regular.

Test_payload:

```
Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-180[)>0.8  &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([20-100[)>0.2
```

i.e. At least 80% of the packets transmitted by the Originator fall in the bins corresponding to 0 byte and [5-180[bytes of payload. At least 20% of the packets sent by the Responder fall in the bins corresponding to 0 byte and [20-100[bytes of payload.

Test_databyteratio:

```
DatabyteRatioOrigToResp<1
```

i.e. The server sends more data than the client.

Test_firstnonemptypacketsize:

```
10 < FirstNonEmptyPacketSize < 250
```

i.e. The first non-empty packet, which is sent by the mail server, is typically small ("OK" + optional info such as server version, name, capabilities, etc.).

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:5)=[-1, 1,-1, 1,-1] ||
FirstFewNonEmptyPacketDirections(1:6)=[-1,-1, 1,-1, 1,-1]
```

i.e. 1)Responder describes server,

2)(optional) If client remains quiet, server sends an empty response. From this point, IMAP behaves like a Request/Response protocol driven by the client Requests. For instance, the following sequence may follow:

3)Originator asks for capability

4)Responder responds with capability

5)Originator sends login & password

6)Responder accepts/rejects login

Test_nonemptypacketratio:

```
Originator.datapacketcount/Originator.packetcount > 0.5 &&
Responder.datapacketcount/Responder.packetcount > 0.6
```

i.e At least 50% of the packets sent by the client carry data, and at least 60% of the packets sent by the mail server carry data.

6 As seen from Table 9 of Section 6, many non-IMAP flows are recognized as IMAP based on these criteria. This profile needs to be tightened to reduce the number of false identification.

4 POP

Test_duration:

```
100000 < Duration < 10000000 µsec
```

i.e. In contrast with IMAP, POP terminates the session once mail messages have been downloaded, this typically takes less than 5 seconds (say a maximum of 10 seconds to set a loose threshold).

Test_transmissionrate:

$\texttt{Originator.}\alpha_{constantbitrate} < 0.5$ && $\texttt{Originator.}\alpha_{constantpacketrate} < 0.5$ &&
$\texttt{Responder.}\alpha_{constantbitrate} < 0.5$ && $\texttt{Responder.}\alpha_{constantpacketrate} < 0.5$

i.e. The transmission rate is more irregular than regular.

Test_payload:

```
Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-20[)>0.9  &&
Originator.PayloadDistribution([1-5[)==0  &&
(Responder.PayloadDistribution([0-50[)+Responder.PayloadDistribution([236-269[)+
Responder.PayloadDistribution([516-549[)+Responder.PayloadDistribution([1432-1473[))>0.6
```

i.e. At least 90% of the packets transmitted by the Originator fall in the bins corresponding to 0 byte and [5-20[bytes of payload. The Originator does not transmit packets carrying only 1 to 5 bytes of payload. The majority (60%) of the packets transmitted by the Responder fall into two categories: small packets (0 to 50 bytes of payload) and full length packets. To represent the second category ("full length packets"), we sum over the bins corresponding to [236-269[, [516-549[and [1432-1473[of payloads. These bins are based on typical MTUs (296,576,1492,1500).

Test_databyteratio:

```
DatabyteRatioOrigToResp<0.65
```

i.e. The server sends more data than the client.

Test_firstnonemptypacketsize:

```
10 < FirstNonEmptyPacketSize < 100
```

i.e. The first non-empty packet, which is sent by the mail server, is typically small ("OK" + optional info such as server version, and name). POP server responses tend perhaps to be smaller than IMAP server responses.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:5)=[-1, 1,-1, 1,-1] ||
FirstFewNonEmptyPacketDirections(1:6)=[-1,-1, 1,-1, 1,-1]
```

i.e. Similar to IMAP with regards to the initial directional dynamics.

Test_nonemptypacketratio:

```
Originator.datapacketcount/Originator.packetcount < 0.7 &&
Responder.datapacketcount/Responder.packetcount > 0.5
```

i.e At most 70% of the packets sent by the client carry data, and at least 50% of the packets sent by the mail server carry data.

5 SMTP

Test_duration:

```
100000 < Duration < 10000000 µsec
```

i.e. SMTP terminates the session once mail messages have been transferred, this typically takes less than 5 seconds (say a maximum of 10 seconds to set a loose threshold).

Test_transmissionrate:

Originator.$\alpha_{constantbitrate}$ <0.5 && Originator.$\alpha_{constantpacketrate}$ <0.5 &&
Responder.$\alpha_{constantbitrate}$ <0.5 && Responder.$\alpha_{constantpacketrate}$ <0.5

i.e. The transmission rate is more irregular than regular.

Test_payload:

```
(Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-100[)+
Originator.PayloadDistribution([236-269[)+ Originator.PayloadDistribution([516-549[)+
Originator.PayloadDistribution([1432-1473[)) >0.6  &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([5-100[)>0.8 &&
Responder.PayloadDistribution([1-5[)+Responder.PayloadDistribution([350-inf[)==0
```

i.e. The majority (60%) of the packets transmitted by the Originator fall into three categories: empty ACKS, small packets (5 to 100 bytes of payload) and full length packets. To represent the third category ("full length packets"), we sum over the bins corresponding to [236-269[, [516-549[and [1432-1473[based on typical MTUs (296,576,1492,1500). A large proportion (80%) of the packets transmitted by the Responder fall in the bins corresponding to 0 byte and [5-100[bytes of payload; and the Responder never sends very small packets or very big packets.

Test_databyteratio:

```
DatabyteRatioOrigToResp > 1
```

i.e. The databyte ratio Originator To Responder is greater than one, typically MUCH greater than 1.

Test_firstnonemptypacketsize:

```
20 < FirstNonEmptyPacketSize < 300
```

i.e. SMTP server responses are typically around a hundred bytes. We chose loose boundaries (20 and 300 bytes).

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:5)=[-1, 1,-1, 1,-1]
```

i.e. 1)Responder describes server. From this point, SMTP behaves like a Request/Response protocol driven by the client Requests. For instance, the following sequence may follow:

2)Originator sends Client Helo
3)Responder sends Server Helo
4)Originator sends AUTH command
5)Responder accepts/rejects

Test_nonemptypacketratio:

```
Originator.datapacketcount/Originator.packetcount > 0.5
```

i.e At least 50% of the packets sent by the client carry data.

Test_datapacketcount:

```
4 < Responder.datapacketcount < 15
```

SMTP servers respond with a somewhat fixed number of non-empty packets.

Test_databytecount:

```
300 < Responder.databytecount < 900
```

SMTP servers respond with a somewhat fixed number of data bytes.

6 SSH[7]

Test_payload:

```
Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([10-180[) > 0.8  &&
Originator.PayloadDistribution([1-10[)==0% && Responder.PayloadDistribution([1-10[)==0 &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([10-180[) > 0.5
```

Test_databyteratio:

```
DatabyteRatioOrigToResp<1
```

i.e. The server sends more data than the client.

Test_cipherblock:

mod(Originator.$\alpha_{\text{cipherblock}}$, 4)==0 && mod(Responder.$\alpha_{\text{cipherblock}}$, 4)==0 &&

Originator. $\beta_{\text{cipherblock}}$ > 0.8 && Responder. $\beta_{\text{cipherblock}}$ > 0.8

i.e. At least 80% of the non-empty packets must be divisible by 4.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1)= -1
```

i.e. The first non-empty packet is sent by the Responder (server).

Test_nonemptypacketratio:

```
Responder.datapacketcount/Responder.packetcount > 0.5
```

i.e At least 50% of the packets sent by the server carry data.

7 TELNET

Test_payload:

```
Originator.PayloadDistribution([0-10[) > 0.8  &&  Originator.PayloadDistribution([350-inf[)==0
```

Test_databyteratio:

```
DatabyteRatioOrigToResp<0.2 && Originator.datapacketcount < Responder.datapacketcount
```

i.e. The server sends much more data than the client. The server also sends more non-empty packets.

Test_firstnonemptypacketsize:

```
FirstNonEmptyPacketSize < 30
```

i.e. empirical estimate based on a max of 10 options negotiated.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:2)= [-1, 1]
```

i.e. The first non-empty packet is sent by the Responder (server) and the second is sent by the Originator.

Test_nonemptypacketratio:

```
Responder.datapacketcount/Responder.packetcount > 0.4
```

i.e At least 40% of the packets sent by the server carry data.

Test_transaction:

$\alpha_{\text{transaction}}$ > 0.7

i.e Telnet is mostly transactional.

[7] This profile is specific to *ssh* used as a remote login program. File transfers using *scp* do not match this profile. The only criteria that are satisfied by *scp* traffic are Test_cipherblock and Test_firstnonemptypacketdirections.

8 RLOGIN[8]

Test_payload:

```
Originator.PayloadDistribution([0-5[) > 0.8  &&  Originator.PayloadDistribution([350-inf[)==0
```

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1)= 1
```

i.e. The first non-empty packet is sent by the Originator.

Test_nonemptypacketratio:

```
Responder.datapacketcount/Responder.packetcount > 0.4
```

i.e. At least 40% of the packets sent by the server carry data.

Test_conversation:

$\alpha_{conversation} > 0.01$ && $\beta_{conversation} > 0.4$ && $\gamma_{conversation} > 0.6$

i.e. RLOGIN appears a little like a conversation (compared to SSH,TELNET, and FTPcommand). When conversing, the Originator sends more packets than the Responder.

9 FTPcommand

Test_duration:

```
Duration > 500000 µsec
```

Test_payload:

```
Originator.PayloadDistribution([0-1[)+Originator.PayloadDistribution([5-100[)>0.8  &&
Originator.PayloadDistribution([1-5[)+Originator.PayloadDistribution([350-inf[)==0  &&
Responder.PayloadDistribution([0-1[)+Responder.PayloadDistribution([10-180[)>0.8 &&
(Responder.PayloadDistribution([1-5[)+Responder.PayloadDistribution([350-650[)+
Responder.PayloadDistribution([1000-inf[))==0
```

Note that while the responder also avoids sending big packets, it was not unusual to see packets containing between 650 and 1000 bytes of payload, in particular when transmitting "code 220" for greetings and warnings.

Test_databyteratio:

```
0.1 < DatabyteRatioOrigToResp < 0.5 &&  Originator.datapacketcount < Responder.datapacketcount
```

i.e. The server sends more data and non-empty packets than the client.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1)= -1
```

i.e. The first non-empty packet is sent by the Responder.

Test_nonemptypacketratio:

```
Responder.datapacketcount/Responder.packetcount > 0.6
```

i.e. At least 60% of the ftpcmd packets sent by the server carry data.

Test_transaction:

$\alpha_{transaction} > 0.95$

i.e. FTPcommand is mostly (if not completely) transactional.

Test_noconsecutivebigpackets:

Responder.$\gamma_{file} \leq 0$

i.e While IMAP and FTPcommand are similar with respect to the other criteria, ftp server tend not to transmit consecutive big packets in the FTPcommand connection.

8 Aside from the training data set, the RLOGIN profile has not been tested on RLOGIN traffic.

10 FTPdata[9]

Test_payload:

```
(Originator.PayloadDistribution([1-5[)==0 && Responder.PayloadDistribution([0-1[)==1)  ||
(Responder.PayloadDistribution([1-5[)==0 && Originator.PayloadDistribution([0-1[)==1)
```

Test_databyteratio:

```
DatabyteRatioOrigToResp == 0 || DatabyteRatioOrigToResp == -1
```

i.e. The data is flowing in one direction only. The value is -1 if the transmitting end is the Originator, and the value is 0 if the Responder is the transmitting end. A value of -1 can be associated to two cases: the transfer is an *ACTIVE get* or a *PASSIVE put*, depending on whether the Originator is the FTP server or the FTP client respectively. Similarly, a value of 0 indicates an *ACTIVE put* or a *PASSIVE get*, depending on whether the Originator is the FTP server or the client respectively. The role of the Originator can be determined by examining related flows marked as *FTPcommand.*

Test_databytecount:

```
Originator.DatabyteCount + Responder.DatabyteCount > 0
```

i.e. As a rule of thumb, a FTPdata session involves transferring data... therefore there should be packets carrying data in at least one of the direction.

Test_packetcount:

```
0.3 < (Originator.PacketCount/(Originator.PacketCount+Responder.PacketCount)) < 0.7
```

i.e. The amount of packets transmitted in each direction is similar.

Test_nonemptypacketratio:

```
(Originator.datapacketcount==0 && Responder.datapacketcount/Responder.packetcount > 0.3) ||
(Responder.datapacketcount==0 && Originator.datapacketcount/Originator.packetcount > 0.3)
```

This rule typically holds provided there are more than 5 packets in each direction.

11 CHATMSN

Test_payload:

```
Originator.PayloadDistribution([0-1[) + Originator.PayloadDistribution([100-450[) > 0.8  &&
Originator.PayloadDistribution([1-5[)==0 &&  Responder.PayloadDistribution([1-5[)==0 &&
Responder.PayloadDistribution([0-1[) + Responder.PayloadDistribution([100-450[) > 0.8
```

i.e. At least 80% of the packets fall into the bins corresponding to 0 and [100-450[bytes of payload. And there are no packets carrying only 1 to 5 bytes of payload.

Test_databyteratio:

```
0.1 < DatabyteRatioOrigToResp < 10
```

i.e. This assumes that one of the user may be at most 10 times chattier than the other.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1)= 1
```

i.e. The first non-empty packet is sent by the Originator.

Test_interactive:

$\texttt{Originator.}\alpha_{cmd\_interactive} > 0.3$ && $\texttt{Originator.}\beta_{cmd\_interactive} > 0.6$ && $\texttt{Originator.}\gamma_{cmd\_interactive} > 0.6$ &&
$\texttt{Originator.}\delta_{cmd\_interactive} > 0.3$ && $\texttt{Originator.}\varepsilon_{cmd\_interactive} > 0.3$ &&
$\texttt{Responder.}\alpha_{cmd\_interactive} > 0.3$ && $\texttt{Responder.}\beta_{cmd\_interactive} > 0.6$ && $\texttt{Responder.}\gamma_{cmd\_interactive} > 0.6$ &&
$\texttt{Responder.}\delta_{cmd\_interactive} > 0.3$ && $\texttt{Responder.}\varepsilon_{cmd\_interactive} > 0.3$

i.e. Each direction is command-line interactive.

Test_conversation:

$\alpha_{conversation} > 0.4$ && $\beta_{conversation} > 0.4$ && $(0.35 < \gamma_{conversation} < 0.65)$

i.e the flow must have *conversational* episodes ($\alpha_{conversation}$), it must have *sustained conversation* episodes ($\beta_{conversation}$) and the amount of packets belonging to a conversation must be similar in each direction ($\gamma_{conversation}$).

9 The FTPdata profile will catch FTPdata flows whether the FTP transfer is in mode PASSIVE or ACTIVE. While this set of rules is sufficient to distinguish FTPdata from the other applications of the Appendix, this FTPdata profile falsely recognize flows associated with applications like *pcanywhere* and *rdesktop*. As mentioned in section 6, adding two new rules to the FTPData profile eliminates the ambiguity.

12 TCPaudiostream

Test_tcp:

```
key.ipproto==6
```

i.e. The profile is only valid for audio stream using TCP as the transport protocol.

Test_duration:

```
Duration > 10000000 µsec
```

i.e. Typically last much longer than 10 seconds.

Test_transmissionrate:

Responder.$\alpha_{constantbitrate}$ >0.5

i.e. The server tries to transmit data at a target bit rate.

Test_payload:

```
Originator.PayloadDistribution([0-1[) >0.8  &&
Originator.PayloadDistribution([1-20[) + Originator.PayloadDistribution([1000-inf[) ==0 &&
Responder.PayloadDistribution([650-1381[)>0.6
```

i.e. The client transmits small packets, the server sends relatively large packets.

Test_databyteratio:

```
DatabyteRatioOrigToResp < 0.01
```

i.e. The client sends little data.

Test_firstnonemptypacketsize:

```
100 < FirstNonEmptyPacketSize < 1000
```

i.e. Similar in size than a *HTTP Get*.

Test_firstnonemptypacketdirections:

```
FirstFewNonEmptyPacketDirections(1:2)=[1, -1]
```

i.e. The first non-empty packet is sent by the Originator and the second by the Responder.