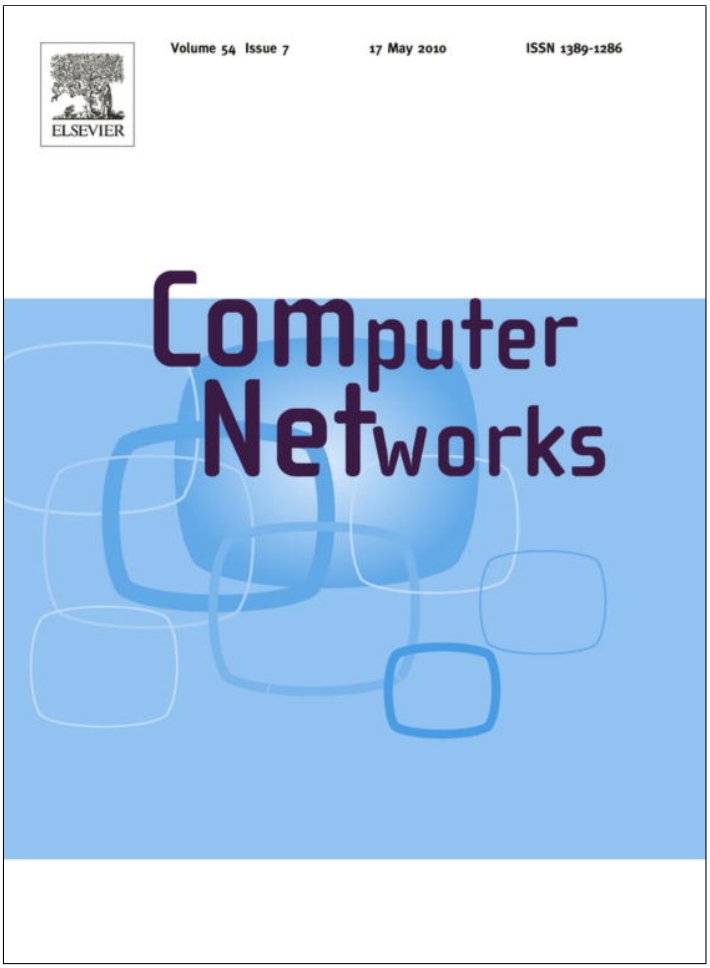

Computer Networks 54 (2010) 1055–1068

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A novel self-learning architecture for p2p traffic classification in high speed networks

Ram Keralapura [a,*], Antonio Nucci [a], Chen-Nee Chuah [b]

[a] Narus Inc., 570 Maude Ct., Sunnyvale, CA 94085, United States
[b] University of California, One Shields Ave, Davis, CA 95616, United States

ARTICLE INFO

Article history:
Received 29 May 2009
Accepted 18 October 2009
Available online 29 October 2009
Responsible Editor: Y.C. Hu

Keywords:
Traffic classifier
p2p Traffic detection
Peer-to-peer networks
Self-learning system

ABSTRACT

The popularity of a new generation of smart peer-to-peer applications has resulted in several new challenges for accurately classifying network traffic. In this paper, we propose a novel two-stage p2p traffic classifier, called *Self-Learning Traffic Classifier* (SLTC), that can accurately identify p2p traffic in high speed networks. The first stage classifies p2p traffic from the rest of the network traffic, and the second stage automatically extracts application payload signatures to accurately identify the p2p application that generated the p2p flow. For the first stage, we propose a fast, light-weight algorithm called *Time Correlation Metric* (TCM), that exploits the temporal correlation of flows to clearly separate peer-to-peer (p2p) traffic from the rest of the traffic. Using real network traces from tier-1 ISPs that are located in different continents, we show that the detection rate of TCM is consistently above 95% while always keeping the false positives at 0%. For the second stage, we use the LASER signature extraction algorithm [20] to accurately identify signatures of several known and unknown p2p protocols with very small false positive rate (<1%). Using our prototype on tier-1 ISP traces, we demonstrate that SLTC automatically learns signatures for more than 95% of both known and unknown traffic within 3 min.

1. Introduction

Managing large networks involves several critical aspects like traffic engineering, network planning and provisioning, security, billing, and fault management. The ability of a network operator to accurately classify traffic into different applications (both known and unknown) directly determines the success of many of the above network management tasks. For example, identifying non-profitable peer-to-peer (p2p) traffic could help an Internet Service Provider (ISP) in providing better quality of service to other revenue-generating delay/loss sensitive applications. Hence it is imperative to develop traffic classification techniques that are fast, accurate, robust, and scalable in order to meet current and future needs of ISPs.

Over the past few years, peer-to-peer (p2p) networks have become extremely popular for many different applications like file (audio, video, and data) sharing, live video streaming, IP-TV, and VoIP services, among several others. As a direct consequence of this, traffic from p2p networks constitute the majority of today's Internet traffic. In fact, many studies (like [5,11]) show that over 60% of the Internet traffic today comprises of p2p traffic. Hence, accurately identifying p2p flows is an important task for network operators.

Traditionally, ISPs have used *port numbers* to effectively identify and classify network traffic. For example, TCP port 80 is HTTP traffic, TCP port 1214 is Kazaa p2p traffic, TCP port 6346 is Gnutella p2p traffic, and so on. This approach is extremely easy to implement and introduces very little overhead on the traffic classifier. However, in order to circumvent detection, p2p networks have started using non-standard ports for communication [11,15,21,19]. In other

* Corresponding author. Tel.: +1 530 219 9674; fax: +1 408 215 4301.
E-mail address: rkeralapura@narus.com (R. Keralapura).

1389-1286/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2009.10.009

words, p2p networks can choose random ports or standard ports used by other applications (like TCP port 80) to send their traffic. These strategies at the application-level have made port number based traffic classification inaccurate and hence ineffective [11,19].

To address the above problems, techniques that rely on application payload signatures were developed [13,21,19]. This approach directly compares the stored signatures to the packets from applications to accurately classify them. Although this approach is fast, accurate, robust, and reusable in different contexts (firewalls, routers, NATs, etc.), it faces the problem of *scalability*: (i) Keeping up with the number of applications that come up everyday is impractical. For example, several new p2p protocols are introduced every year. (ii) Reverse engineering these applications to find accurate signatures is not trivial, and hence keeping an up-to-date list of signatures becomes an herculean task for engineers.

Given the shortcomings of port- and signature-based approaches for detecting p2p traffic, the research community started developing techniques that are less dependent on individual applications, but focused on capturing and extracting commonalities in the behavior of p2p applications based on layer-3/layer-4 information. We refer to these as *pattern classification* techniques. Some approaches like [24,25,18,23,13] examine the connection patterns, and classify traffic into different p2p applications using machine learning and/or clustering algorithms; others like [16] look at specific attributes of flows to group them into different applications. Although pattern classification techniques seem to be very promising (and thus deserves a strong attention from the research community), we believe that there are several open questions about their applicability in the real world. First, due to their dependence on statistical techniques that need multiple flows and multiple packets from each flow, the time required to detect and report the discovery of an application, is much longer when compared to application payload signature matching techniques. Second, most of these techniques are incapable of differentiating individual applications behaving in a similar fashion at the macroscopic level. For instance, many of these techniques detect p2p traffic but cannot identify individual protocols like eDonkey, BitTorrent, or Gnutella. Third, these techniques are not as accurate and reliable as signature-based techniques since they are heavily dependent on the point of observation and network conditions (e.g., traffic asymmetry). Fourth, even though pattern classification appears to be less resource consuming compared to the signature matching approaches (since it requires to monitor only layer-3/layer-4 data), it is in fact not true. Pattern classification requires maintaining considerably larger number of states in memory for processing, and thus severely limits their effectiveness in operating at very high speeds.

In this work, we address the problem of identifying traffic originating from known and unknown p2p networks. In particular, we focus on: (i) *real-time* identification of p2p traffic in *large* networks (for example, tier-1 and tier-2 ISPs) by monitoring the traffic at the *network edge*, and (ii) p2p networks that use superpeer technology (edonkey, gnutella, etc.). In this paper, we propose a novel two-stage p2p traffic classifier called *Self-Learning Traffic Classifier* (SLTC) that brings together the benefits of signature matching (speed, accuracy, and reusability) and pattern classification (scalability) techniques. In the first stage, SLTC separates p2p traffic from the rest of the traffic by exploiting the general behavior of p2p protocols, and in the second stage, it automatically extracts payload signatures to identify specific p2p protocols. Note the second stage is required to separate p2p *protocols* (like edonkey, gnutella, kazaa, etc.) from each other. SLTC populates the extracted signatures into a signature database that will be used to classify all future flows. Thus all flows for which SLTC has a signature in the database bypass the expensive pattern classification step. SLTC can automatically *learn* (i.e., classify and extract signatures) both known and unknown p2p applications in a matter of minutes.

To the best of our knowledge, this is the first work to propose a multi-stage, self-learning, real-time p2p traffic classification system that can be used in high speed networks with minimum manual intervention. Our main contributions are:

- We propose a two-stage SLTC system that can quickly learn known and unknown p2p applications and classify them in real-time. For the first stage, we propose a new pattern classification algorithm, called *Time Correlation Metric* (TCM) that explores the *temporal* correlation of incoming and outgoing p2p flows. TCM first identifies p2p nodes, and subsequently classifies flows to/from these nodes as p2p flows. We show how this new concept clearly outperforms previous metrics in (i) discovering p2p nodes with an accuracy well above 95% with 0% false positive, and (ii) distinguishing p2p nodes as either peers or superpeers (Section 5).
- We comprehensively explore the feasibility of SLTC with many tier-1 ISP packet traces. Our experiments show that SLTC can learn over 95% of all p2p traffic in less than 3 min. Once SLTC learns about an application, future flows that belong to the application are directly classified using application payload signatures and hence do not go through the two stages in SLTC. Furthermore, the 90-percentile detection lag (i.e., the total time from detecting the first packet of a p2p flow to extracting a signature for the application corresponding to the flow) is less than 60 s (Section 7).

A preliminary version of this work appeared in [17]. However, our current work improves and extends that work with several additional materials: (i) We explain in detail the deployment scenario of our classifier and the high-level architecture of SLTC. (ii) We describe the architecture of each component of SLTC in detail, and describe all the modules in every component. (iii) We provide more details about the signature extraction module and present some new results.

2. Data description

We collect and analyze four packet traces captured from two tier-1 ISP networks (say ISP-A and ISP-B) that

Table 1
Details of the four data traces used in this work (E-Edonkey; G-Gnutella; B-BitTorrent; S-Skype; K-Kazaa).

Name	Network	Duration	Flows	E/G/B/S/K Flows ($\times 10^3$)
Trace-1	ISP-A	591.209 s	7.26×10^6	84.2/31.4/26.4/7.5/0.6
Trace-2	ISP-B	1067.927 s	10.21×10^6	6.3/1.3/70.6/2.2/0.3
Trace-3	ISP-B	1083.837 s	11.61×10^6	5.6/1.9/68.5/3.1/0.4
Trace-4	ISP-B	1183.637 s	11.35×10^6	6.1/1.4/75.4/1.7/0.3

are located in different continents (Table 1). For both the ISPs, we captured all packets (with no sampling) between the ISP and one of its customers (a tier-2 ISP).

Validating our classification algorithm needs *ground truth*. In other words, we should be able to classify the traffic in these traces using an alternative method (other than the algorithms in SLTC), so that we can compare the results from SLTC algorithms. To accomplish this, we built a *L7 protocol analyzer* (L7PA) based on the application payload signatures publicy available at L7-Filter [7] and used this to verify the accuracy of SLTC algorithms. L7PA has signatures for 25 different applications that include both p2p (BitTorrent, Gnutella, EDonkey, Skype, and KaZaa) and non-p2p (HTTP, SMTP, POP3, DNS, etc.) protocols. Table 1 also shows the number of flows for different p2p protocols identified by L7PA in each of the four traces. We use these flows as the ground truth in the rest of this paper.

3. Related work and challenges

3.1. Peer-to-peer traffic classification

Given the shortcomings of the basic approaches (i.e., port-based and signature-based traffic classifications), there has been a lot of effort in developing p2p traffic classification techniques that rely just on the layer-3 and layer-4 information. Some approaches (like [24,25,18,23,13]) examine the connection patterns at layer-3 and classify traffic into different applications using machine learning techniques and/or clustering algorithms, while others (like [15,16]) look at specific attributes of flows to group them into different applications. For example, the authors in [15] propose many flow-based heuristics to identify p2p nodes: (i) p2p nodes use both TCP and UDP protocols as their transport layer protocol, (ii) p2p node are characterized by both incoming and outgoing connections, and (iii) the ratio of the source IP to source port ratio for all incoming flows into a p2p node approaches 1. However, as we show later in this section, these heuristics lead to false positives[1] and false negatives.

[1] The authors in [15] acknowledge that the heuristics can lead to false-positives and provide refinements to the basic heuristics to minimize false-positives. However, we find that in the case of ISP networks there could also be a lot of false negatives which the authors do not account for. Please see [8] for more details.

Although the above approaches result in high detection rates, their main limitation is that they are not feasible for *real-time* classification in high speed ISP networks for three reasons. First, most of the above techniques rely on time consuming algorithms (like statistical clustering, machine learning, etc.) that have to be applied to every flow seen by the classifier; thus keeping up with the traffic rate becomes extremely difficult. Second, most of the above techniques are designed to identify high-level application classes, but cannot identify individual applications, an important requirement for network operators to prioritize traffic. Third, none of the above techniques can effectively identify new (i.e., currently unknown) applications that come up in the future.

Our approach in this work addresses the above issues and is geared towards real-time identification of both known and unknown p2p applications in high speed ISP networks where asymmetric routing is commonly used. Our p2p traffic detection algorithm is simple, fast, accurate, and resistant to data obscured by asymmetric routing.

3.2. Peer-to-peer networks

In this work, p2p traffic refers to the traffic originating from: (i) *unstructured* p2p networks where different peers join and leave the network as and when they please, (ii) dynamic p2p networks that are used to exchange files, music, video, and other forms of information, and (iii) p2p networks that use superpeer technology to manage their network. Examples of such p2p networks include eDonkey [2], Gnutella [4], KaZaa [6], BitTorrent [1], Skype [9], etc.

Unstructured p2p networks are inherently distributed in nature providing an infrastructure to all users to exchange files, music, video, and other information with each other without relying on any centralized servers. Many popular p2p networks have several million users at anytime, and hence a *completely distributed* approach to finding and exchanging information leads to network meltdown. Most of the successful p2p networks that exist today adopt the strategy of constructing *hybrid* networks, where the p2p network elects a few nodes as *leaders* for a group of nodes based on the nodes' computing/network resources. These leaders are usually referred to as *superpeers* or *ultrapeers*.

Superpeers are typically connected to several other superpeers and the main objective is to ensure that these superpeers (and hence the peers connected to them) are connected to the rest of the network. We can think of this architecture of p2p networks as a two-level hierarchy. The first level contains all the superpeers connected to several other superpeers in the same level. The second level contains peers connected to one or more superpeers in the first level. Note that these peers at the second level may or may not be connected to other peers in the same level. This architecture ensures that when peers join or leave a network, the impact on the network (in terms of connectivity of other peers) is minimal. However the impact is higher when superpeers leave the network. Hence nodes that have significantly higher uptimes are chosen to be superpeers.

Although the actual functionality of a superpeer varies depending on the particular p2p application, in general, a superpeer acts as a gateway to the rest of the network for the group of peers that are connected to it.

3.3. Challenges to p2p traffic detection

Although p2p networks are application layer networks built on top of the IP layer, traffic from these networks behave very similar to the rest of the Internet traffic and is virtually indistinguishable. Hence, most strategies proposed in the past for classifying p2p traffic based on only layer-3/layer-4 information rely on first detecting *nodes* that are running p2p applications, and then identifying p2p traffic based on the these p2p nodes. In this subsection, we present the most obvious metrics (that are feasible to be used for p2p traffic classification)[2] for identifying p2p nodes, and show why these metrics fail to accomplish their objective in the context of our problem definition (i.e., real-time superpeer-based p2p traffic classification at network edges).

A common strategy adopted by most p2p networks to get around the connectivity problem introduced by firewalls is to use both TCP and UDP protocols on any of the open ports. Furthermore, to optimize their performance, p2p nodes typically use both TCP and UDP protocols for control, signaling, and/or data flows. For example, a Skype peer connects to its superpeer usig both TCP and UDP [10]. Another characteristic that distinguishes a p2p node from a node that does not run any p2p applications is the p2p node's ability to act as both a client and a server. Several heuristics have been proposed to take advantage of these properties of p2p nodes [15].

However, there are several problems while using the above heuristics to detect p2p nodes: (i) **False positives:** Several other protocols in the Internet, like DNS, gaming, streaming, IRC, etc., also exhibit these properties. In other words, these non-p2p applications also use both TCP and UDP protocols to communicate between node pairs. As an example consider Fig. 1, where several nodes running DNS protocol in Trace-1 also have both TCP and UDP connections. Also, nodes running these non-p2p applications can both accept and open connections to other nodes. Fig. 2 once again uses DNS nodes as an example to show that nodes running non-p2p applications can have both incoming and outgoing connections. Note that we saw similar results for a number of other protocols like SMTP, gaming, etc. Hence the above heuristics could lead to a lot of false positives [15,12]. (ii) **False negatives:** All p2p nodes (or p2p node pairs) do not always satisfy the above heuristics. For example, not all p2p node pairs use both TCP and UDP protocols to talk to each other. Several p2p protocols use TCP port 80 (a port most likely to be open in almost every firewall) as a way to bypass firewalls and hence may not use both TCP and UDP protocols. Also, several p2p nodes may not be observed (from the perspective

[2] Several other techniques have been proposed (like [24,25,18,23,13,16]) in the literature. However we believe that these techniques are infeasible for real-time p2p traffic classification in high speed networks.

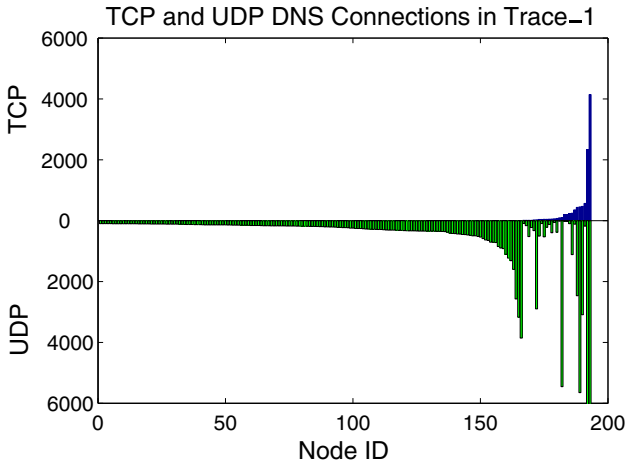

Fig. 1. False positives: Nodes in Trace-1 (with at least 100 connections) running DNS use both TCP and UDP to communicate with other nodes.

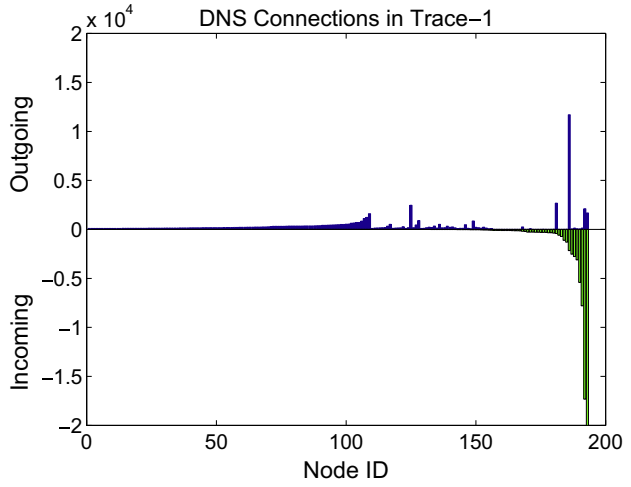

Fig. 2. False positives: Number of incoming and outgoing connections in nodes (with at least 100 connections) running DNS in Trace-1.

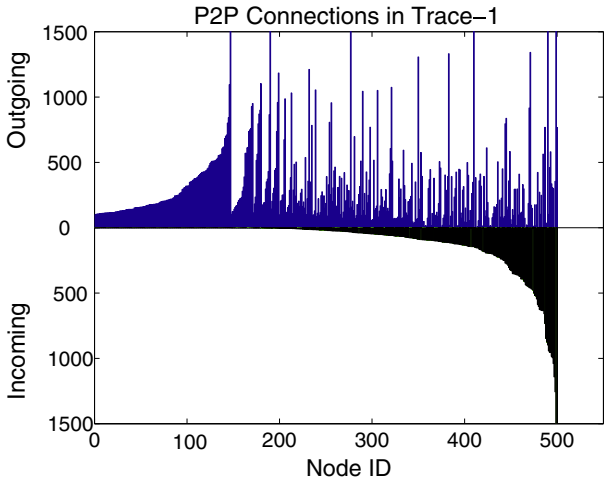

Fig. 3. Number of incoming and outgoing connections in nodes (with at least 100 connections) running p2p applications in Trace-1.

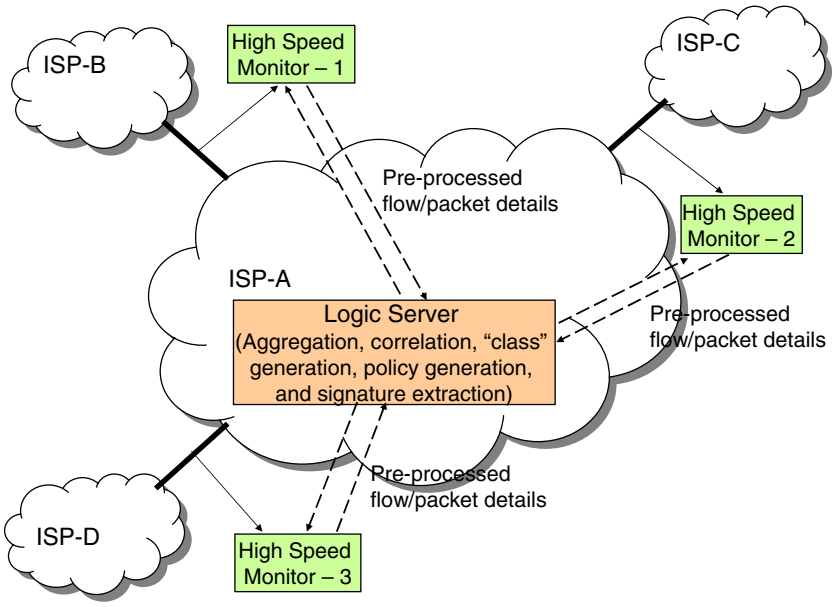

Fig. 4. Self-Learning Traffic Classifier (SLTC) system: typical network deployment.

of the monitoring point) to act as both server and client[3] (see Fig. 3). Thus there could be a lot of false negatives while using these heuristics as well.

We propose a novel light-weight approach to easily detect p2p nodes for the problem defined in Section 1. We present the intuition and algorithm for our approach in Section 5.

4. Self-learning traffic classifier: architecture and deployment

In this section, we first describe a typical network deployment scenario and then provide a high-level architectural overview of the *Self-Learning Traffic Classifier* (SLTC).

SLTC is a 2-tiered system comprised of a distributed collection tier and a centralized processing tier as shown in Fig. 4. Data is collected directly off-the-wire using high speed *passive* listeners, called *High Speed Monitors* (HSM). These monitors passively observe network traffic on different links, and try to classify traffic using application payload signatures. If a monitor can successfully identify a transiting flow using known signatures, then the flow is marked as "known", and no further analysis is necessary to classify it. However, if a flow cannot be successfully classified, then HSM forwards layer-3/layer-4 information of the flow to the centralized server, called *Logic Server* (LS). Since the logic server has a network-wide view of the traffic activity, it is in a better position to carry out a more reliable processing of such information. Using this architecture problems like asymmetry in routing, very common in large ISPs, are easily overcome.

A detailed description of the SLTC architecture is shown in Fig. 5. For easy of illustration we show only one HSM connected to the central LS, although we envision that in a real network deployment many HSMs are required to collect information from major network aggregation points.

Each HSM is comprised of the following functional blocks: (i) *L7 monitor and signature database*: Traffic traversing links monitored by HSMs passes through this signature matching component where a fast pattern matching algorithm tries to classify traffic. Any traffic that is successfully classified is marked as "known" and sent out of the monitor; other flows are marked as "unknown" and sent to other components for further analysis. (ii) *L4 monitor*: This component extracts and subsequently forwards packet- and flow- header information to the logic server for further analysis. (iii) *Full Packet Capture (FPC) and policy database*: As soon as the pattern classifier in LS classifies a flow as belonging to a particular application, a *policy* (explained later in this section) is generated and stored in the policy database. Based on these policies FPC forwards raw packet information to the signature extraction component in LS.

The logic server receives input from several HSMs, and performs the following: (i) Aggregates and correlates inputs from different passive monitors (*Aggregator*), (ii) classifies traffic into application classes like multimedia, p2p, etc. (*Pattern Classifier* or PC), (iii) generates and enforces collection policies in the HSMs (*Policy Generator*), and (iv) extracts signatures of unclassified applications and populates the signature database (*Signature Extractor* or SE).

The pattern classifier (PC) is one of the most important components that classifies traffic into *application classes*

[3] Even if all p2p nodes act as both clients and servers, observing all these connections depends on the location/s of the monitoring equipment. In several cases, the location of the monitoring equipment could force us to believe that p2p nodes are acting only as servers or only as clients. Hence we believe it is reasonable to assume that not all the connections can be observed to make accurate conclusions here.

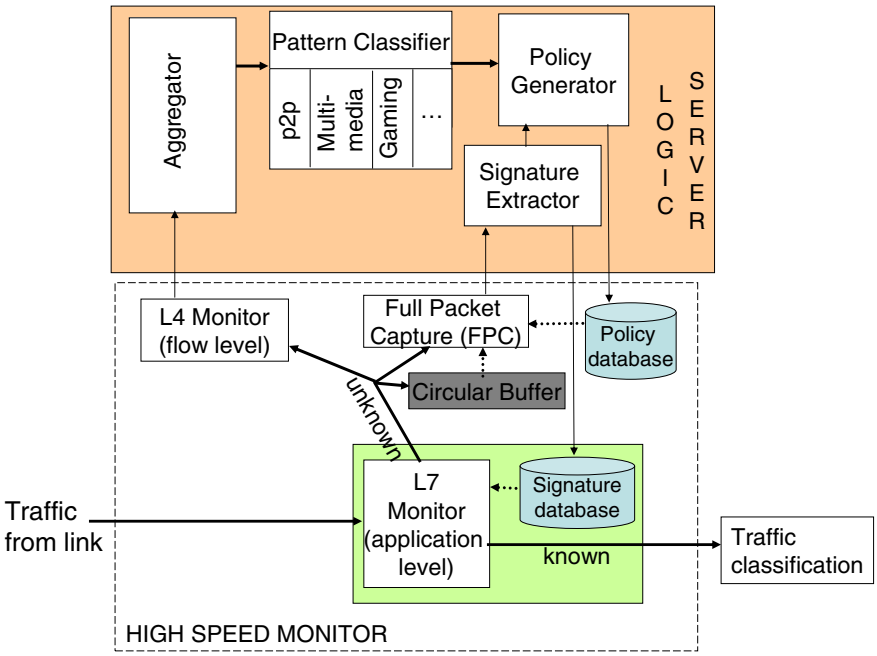

Fig. 5. Self-Learning Traffic Classifier (SLTC) system: design and architecture.

using only layer-3/layer-4 information. An application class represents a group of applications exhibiting the same generic properties that can be used to distinguish them from applications in another class. For example, all real-time voice traffic can be grouped into the same high-level class since all of them exhibit similar delay and/or loss properties. PC comprises of several disparate modules (Fig. 5), each of them focusing on identifying a specific application class. These *configurable* modules provide the flexibility to define new classes that are combinations of existing high-level classes. These modules can either be connected to each other in a serial fashion or parallel fashion. For example, connecting the p2p module and Voice Multimedia module in a serial fashion will result in a new high-level class where all the traffic identified are p2p flows that carry voice traffic. We will elaborate more about the p2p pattern classifier in Section 5.

The policy generator generates policies based on the output of the PC and distributes them to all the HSMs. Now the question is "what is a policy?". We define a policy as a *logical rule* that is used by the FPC component to filter specific data streams of interest from all traffic traversing the link being monitored. A policy is specified as a vector with several fields, e.g., the vector {dst ip, dst port, num of packets in a flow, num of bytes in each packet} could be a policy generated for all flows that belong to the p2p application class. Note that each policy or set of policies are specific to a particular application class. Finally, the policy generation process in SLTC is iterative, i.e., if a particular policy generated by the policy generator does not lead to a successful signature extraction then the SE component and the policy generator communicate with each other until a signature is successfully identified or the maximum allowed number of iterations is reached.

The signature extractor (SE) extracts signatures for different applications that belong to a particular application class. We will explain this component in detail in Section 6. In the rest of this work, we will focus mainly on two of the most important components of SLTC, *Pattern Classifier (PC)* for the p2p application class and the *Signature Extractor (SE)*.

5. Peer-to-peer pattern classifier

5.1. p2p Traffic classification: intuition and approach

Our approach to identifying p2p traffic class relies on the following observations in *hybrid* p2p networks (i.e., p2p networks that use superpeer technology). When a peer (or host) joins a p2p network, it typically connects to one or more servers and/or superpeers. If the peer connects to a server (as in the case of a few hybrid p2p networks) at the start, then the server provides the peer with the superpeer contact information. The peer will eventually contact the superpeer to let the superpeer know of its arrival. Fig. 6 depicts the above process. When a new peer, **Peer A**, joins the p2p network, it talks to a superpeer, **Superpeer S**, in its host cache (i.e., a table containing all the neighboring peers). The information about the superpeer could already be in **Peer A**'s host cache due to the past activity of the peer in the network, or it could be obtained from a centralized database by first connecting to a central server. Either ways, **Peer A** ultimately connects to **Superpeer S**, and sends the information that can be used by other peers to contact **Peer A**. As soon as **Superpeer S** receives this information, it forwards the information to other peers (like **Peer B**), and superpeers that are connected to it. This process of disseminating the peer contact information is critical in p2p networks for two reasons: (i)

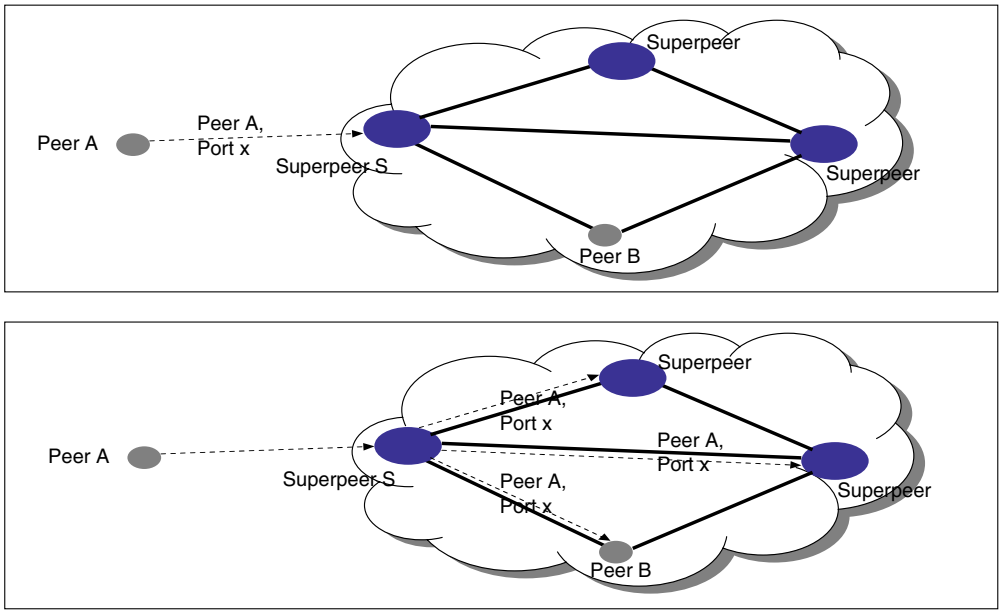

Fig. 6. A peer joining (or searching for information) a p2p network.

Fault tolerance: A typical p2p network experiences a lot of churn (i.e., peers joining and leaving the network). In Fig. 6, if **Superpeer S** decides to leave the network, then **Peer A** loses connectivity to the rest of the network. If other superpeers know about **Peer A**, then they can take over the responsibility from **Superpeer S**, thus providing all the required services to **Peer A**. (ii) File download/upload: Peers in the network need **Peer A**'s contact information to upload/download files.

Based on the above observations, when a new node arrives into the p2p network, a superpeer accepts a connection from the node, and subsequently opens a connection to one or more other nodes in the network. From the perspective of a superpeer, *an incoming connection is closely followed in time by one or more outgoing connections*. Hence our hypothesis is that by observing connections coming into and leaving a node in close succession we can accurately identify superpeers in p2p networks. We call this as the *time correlation metric* (TCM), i.e., a metric that captures the temporal correlation between the incoming and outgoing connections in p2p protocols.

There are several reasons why we believe that TCM is ideal in the context of our problem (i.e., for identifying superpeer-based p2p traffic in high speed networks in real-time by monitoring peering links):

- **Churn in p2p networks.** p2p Networks experience a lot of churn [22] and support constant searches. Since TCM aims to exploit the p2p network behavior during these commonly occurring events, we believe that TCM can be very successful.
- **Location of HSM.** HSM monitors all bidirectional traffic on a peering link. Given that peers in p2p networks typically connect to random superpeers, there is a very high probability that several incoming and outgoing connections from the *same* superpeer crosses a peering link multiple times. Notice that, unlike other approaches, in TCM, *we do not need to observe all the flows, but only require to monitor a few flows for every superpeer*. Once again, this results in an advantage for TCM.

Another important point that we wish to draw the reader's attention to is that monitoring network traffic on peering links also eliminates *false positives* to a large extent. For instance, DNS nodes could also exhibit the TCM property as described earlier. Consider a hierarchical DNS system where higher level DNS servers are located outside a network. A recursive DNS query to a DNS server in this system could result in the server opening new connections to other DNS servers. However, the natural association of network borders and the DNS servers in a hierarchial system, ensures that the HSM (sitting on a peering link at the network edge) *either captures the incoming or the outgoing connections but not both*. The same is true for other applications like smtp, pop3, etc.

The TCM algorithm based on the above approach is shown in Fig. 7. The algorithm is characterized by three parameters:

- **TCM time threshold** (T_{th})**.** This parameter represents the maximum time difference between incoming and outgoing connections in superpeers. A large value for this parameter means that we will group together unrelated incoming and outgoing flows. A very small value implies that we do not group together even the flows that are correlated. Hence choosing an optimal value of this metric is critical to the effectiveness of TCM.
- **TCM pattern threshold** (P_{th})**.** As we explained earlier (in Fig. 6), *every* incoming connection to the superpeer from a new peer (or a search query from the existing peer) results in *several* outgoing connections from the superpeer. P_{th} represents the number of outgoing connections that should be temporally correlated with an incoming

TCM(curTime, newFlow(sIP, dIP, sPort, dPort, L4Prot))

1: $OutgoingFlows(sIP{:}sPort).\text{add}(curTime, newFlow)$
2: $IncomingFlows(dIP{:}dPort).\text{add}(curTime, newFlow)$
3: **for all** (F such that $F \in IncomingFlows(sIP{:}sPort)$) **do**
4: **if** (($consideredDstIP$ does not contain dIP) and ($curTime - F.time) < T_{th}$) **then**
5: $consideredDstIP.\text{add}(dIP)$
6: $pattern(sIP{:}sPort) \leftarrow pattern(sIP{:}sPort) + 1$
7: **if** ($pattern(sIP{:}sPort) > P_{th}$) **then**
8: $repetition(sIP{:}sPort) \leftarrow repetition(sIP{:}sPort) + 1$
9: **if** ($repetition(sIP{:}sPort) > R_{th}$) **then**
10: $P2PNodePortPair.\text{add}(sIP, sPort)$

Fig. 7. TCM algorithm.

connection to assume that a TCM pattern has occurred. A very high value of this parameter could lead to a lot of false negatives (i.e., p2p superpeers not identified as superpeers), where as a small value could lead to false positives (non-p2p nodes identified as superpeers).

- **TCM repetition threshold** (R_{th})**.** If a superpeer is observed for a long period of time, then the TCM pattern (i.e., one incoming connection resulting in several outgoing connections) should occur several times. We use R_{th} as a parameter to specify the number of times that we should see the TCM pattern before declaring a node to be a superpeer in a p2p network. Note that a very small value could imply that non-p2p nodes could be included in the superpeer set by pure coincidence. However, a large value could once again lead to false negatives.

5.2. Reducing false negatives

An assumption that we make in the TCM heuristic is that when a new peer establishes a connection with the superpeer, the superpeer opens *new* connections to other existing superpeers and/or peers to convey the information about the new peer. However, in reality this might not always be true. That is, the superpeer may convey the information about the new peer using *existing* connections to other superpeers/peers.

Fig. 8 shows the total data rate of p2p connections that lasted for more than 10 min in Trace-4.[4] We can see that there are several connections whose overall data rate is very small, i.e., just 1–2 bytes per second. These connections last more than 10 min, but only exchange 1000–2000 bytes of data. This suggests that these are long lasting *control* connections that carry small control data. Thus, ignoring communications on long lasting connections results in several false negatives in TCM. Hence, in our TCM algorithm, instead of always looking for *new* outgoing connections, in addition, we also look at existing connections carrying small control packets. Furthermore, we consider small outgoing control packets on existing connections only if: (i) the connection lasts for a long time, and (ii) the average packet size of the connection is also small (<150 bytes). This heuristic eliminates the possibility of considering small packets from non-control flows. In all our experiments, we incorporate these changes to the algorithm in Fig. 7.

6. Signature extractor (SE)

The signature extraction component resides in the logic server (LS) and the goal of this component is to *automatically* extract signatures for different p2p applications. Note that this component is critical for self-learning mode of SLTC for two reasons: (i) It helps in differentiating p2p applications from each other, and (ii) it helps to bypass the expensive pattern classification step for known flows. The input to the SE component is a set of packets from the flows belonging to p2p superpeers as identified by

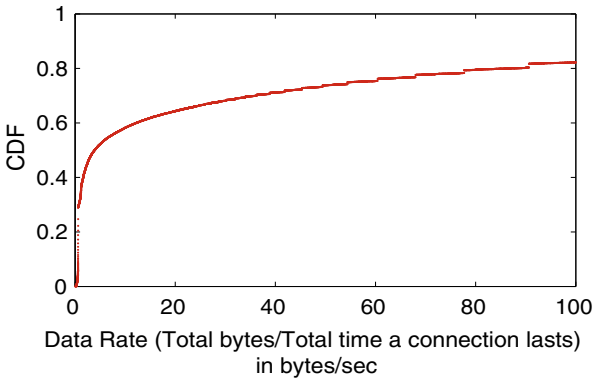

Fig. 8. Overall data rate of p2p flows in Trace-4 that last more than 10 min.

[4] The graph was generated using the ground truth from L7PA and zooms in on a particular range of values on the *x*-axis to show the region of interest.

the TCM algorithm. In other words, the TCM algorithm identifies certain nodes as p2p superpeers and conveys this information to the high speed monitors (HSMs). As soon as the HSMs receive this information, they forward some/all packets (including the packet payloads) from/to the superpeer. Using this information the SE component extracts a signature for the p2p application that the superpeer is running.

6.1. Challenges to signature extraction

Today's p2p applications try to disguise themselves using complex polymorphic flow behavior (i.e., signatures appear in different packets in every flow and different position in each packet). In other words, applications change their behavior by using floating signatures that appear in different places in different flows, thus making it very hard to extract their signature and identify them. Also, the number of p2p applications in the Internet has increased exponentially making it impractical to extract signatures manually.

In the context of SLTC, another challenging issue is that we do not know the application to which each flow belongs to. For instance, using TCM we can only determine whether a node is running a p2p application or not, but not the particular p2p application that it is running. This makes it harder to extract a unique signature for each application. To address this, in this work, we assume that all flows destined to a particular node on a particular port, and using a given layer-4 protocol as running the same p2p application.

6.2. Algorithm for signature extraction

We use the LASER algorithm to extract signatures from packet payloads [20]. LASER uses the least common subsequence (LCS) algorithm that is popularly used in DNA sequence matching. The authors in [20] have shown that LASER is very efficient and accurate in extracting signatures for unknown applications. Note that the LASER algorithm resides in the central logic server.

As soon as TCM identifies a node as a superpeer, all HSMs start forwarding packets for flows to/from the node (on the specific port identified by TCM) to the SE algorithm. For each node identified as a superpeer, HSM forwards flows from at least C distinct peers that communicate with the superpeer. This ensures that the input to the SE algorithm is statistically not biased, thus resulting in a more generic signature. A large value of C outputs a more accurate signature but results in a large delay. However, a small value of C results in inaccurate signatures. Using packet payloads from these flows as input, the LASER algorithm extracts and refines the signature for the application until a stable signature is obtained.[5] Once a signature is obtained, it is sent to all the HSMs. The HSMs store this signature in their database (unless the signature already exists) and use it to instantly classify any future flow (without sending the flow through the TCM and SE components).

[5] In our SE component, we set $C = 5$. However, we continue forwarding flows to the SE component from more peers until a stable signature is obtained. For more details about the LASER algorithm, please refer to [20].

7. System evaluation

To evaluate the proposed SLTC architecture, we built a prototype of SLTC (both HSM and LS components). We replayed all the four traces (described in Section 2) and used them as input to the system. In Section 7.1, we show how we select the various TCM parameters. In Sections 7.2 and 7.3, we use the *ground truth* generated by the L7 protocol analyzer (L7PA) to evaluate the *accuracy* of TCM and SE algorithms. In Section 7.4, we evaluate the performance of the end-to-end SLTC system by exploring all (known and unknown) flows in the four traces.

7.1. TCM parameter tuning

As mentioned earlier in Section 5, the TCM algorithm depends on three different parameters – T_{th}, P_{th}, and R_{th}. The choice of values for these parameters directly influence the accuracy and efficiency of TCM. In this section, we tune the values of these parameters using our L7PA such that the output of TCM results in high detection rate and low false positive rate.

We use Trace-1 for all the parameter tuning experiments, however, the results from the other three traces were very similar. To eliminate the dependence of our parameter values on the length of the trace (i.e., the trace time interval), we first split the trace into multiple segments each of which is 180 s long. We compute the detection and false positive rates for each of these segments. The results in Figs. 9–11 show the average detection and false positive rates for all the 180-s segments in Trace-1.

Fig. 9 shows the detection and false positive rates as a function of the time threshold (T_{th}) for different values of P_{th} and $R_{th} = 2$. We can see that large values of T_{th} results in higher detection rates, but also results in higher false positive rates. Also, the maximum detection rate decreases as the value of P_{th} increases. In other words, for P_{th} values between 2 and 4, the maximum detection rate reaches 100%. However, for $P_{th} = 5$, the maximum detection rate falls below 100%, showing that we will be unable to detect all the superpeers using TCM if we set a large value for P_{th}. Finally, from Fig. 9, we can clearly see that there is no region in the graph where the detection rate is 100% and the false positive rate is 0%. In other words, we cannot find any parameter values that result in optimal detection and false positive rates.

Figs. 10 and 11 are similar to Fig. 9, but for $R_{th} = 3$ and $R_{th} = 4$ respectively. From Fig. 11, we can see that for $R_{th} = 4$, the maximum detection rate is always less than 100% irrespective of the values of T_{th} and P_{th}. Hence increasing the value of R_{th} beyond 3 will not result in optimal detection rate. Finally, from Fig. 10, we can see that the optimal detection and false positive rates can be obtained when $T_{th} \in (2, 3]$, $P_{th} = 4$, and $R_{th} = 3$. Hence, in the

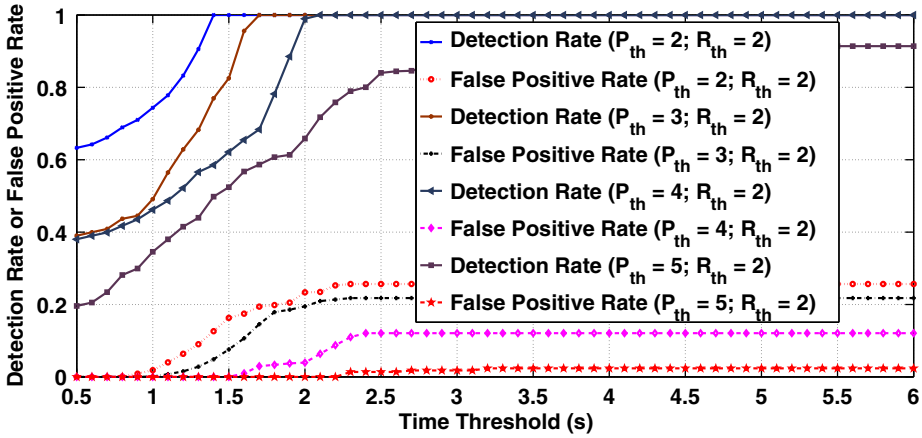

Fig. 9. Detection rates and false positive rates for $R_{th} = 2$.

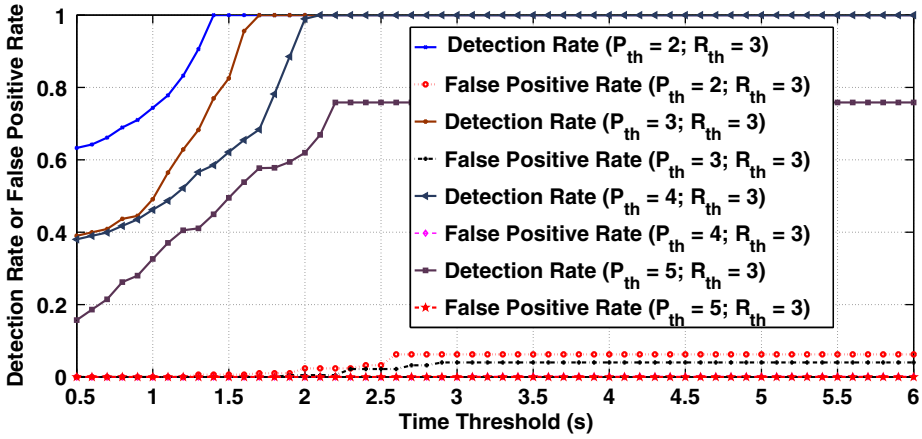

Fig. 10. Detection rates and false positive rates for $R_{th} = 3$.

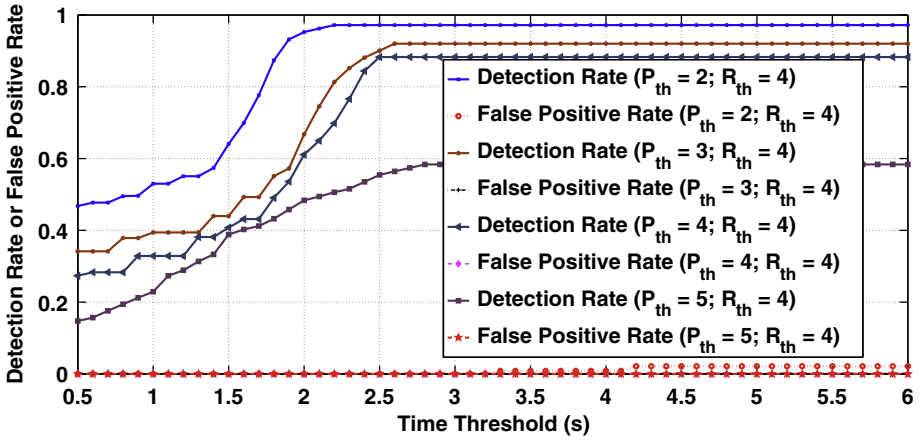

Fig. 11. Detection rates and false positive rates for $R_{th} = 4$.

experiments in the rest of this paper we use $T_{th} = 2.5$ s, $P_{th} = 4$, and $R_{th} = 3$.[6]

[6] The ideal parameter values vary with the deployment scenario. In our case, we are monitoring peering links between a tier-1 and tier-2 ISP. The ideal parameter values will change with the location of the monitoring point.

7.2. Peer-to-peer traffic classifier

Although our L7PA has signatures for several p2p and non-p2p protocols, the list is surely not exhaustive. Hence the output of L7PA contains several flows that are just marked as "unclassified" TCP/UDP traffic. We represent the set of nodes identified by L7PA that belong to p2p

networks (i.e., Gnutella, eDonkey, BitTorrent, Skype, and Kazaa) by P, the set of nodes belonging to non-p2p applications (like HTTP, DNS, SMTP, IMAP, POP3, etc.) by N, and the unclassified nodes by O. In order to eliminate ambiguity of a node belonging to multiple sets, we ensure that if a node belongs to P then it does not belong to the other two sets. Similarly, if a node belongs to O, then it is not included in N. Hence all the three sets are *mutually exclusive* of each other.

Let T represent the total time of a data trace. We represent T as a sum of several *time intervals* of fixed length z. We represent the total number of such intervals in any trace file by n. Now, let P_i denote the set of p2p nodes identified by L7PA in the i^{th} time interval where, $i \in 1 \ldots n$. Similarly, N_i and O_i represent the set of nodes in the non-p2p and unclassified set identified in the i^{th} interval. Note that the sets P_i, N_i, and O_i constitute the ground truth in the i^{th} interval. Similarly, let D_i represent the set of p2p nodes detected by our TCM algorithm in the i^{th} interval.

We use three metrics to evaluate the accuracy of TCM: *detection rate* (DR), *false positive rate* (FP), and *false negative rate* (FN). We define the detection rate in any interval, DR_i, and overall detection rate until (and including) the interval i, DR_i^{full}, as:

$$DR_i = \frac{|D_i \bigcap P_i|}{|P_i|}; \quad DR_i^{full} = \frac{\left|\left\{\bigcup_{k=1..i} D_k\right\} \bigcap \left\{\bigcup_{k=1..i} P_k\right\}\right|}{\left|\bigcup_{k=1..i} P_k\right|}. \tag{1}$$

Similarly, we define false positive and false negative rates as:

$$FP_i = \frac{|D_i \bigcap N_i|}{|N_i|}; \quad FP_i^{full} = \frac{\left|\left\{\bigcup_{k=1..i} D_k\right\} \bigcap \left\{\bigcup_{k=1..i} N_k\right\}\right|}{\left|\bigcup_{k=1..i} N_k\right|}, \tag{2}$$

$$FN_i = \frac{|P_i - D_i|}{|P_i|}; \quad FN_i^{full} = \frac{\left|\left\{\bigcup_{k=1..i} P_k\right\} - \left\{\bigcup_{k=1..i} D_k\right\}\right|}{\left|\bigcup_{k=1..i} P_k\right|}. \tag{3}$$

We evaluate the accuracy of our TCM algorithm by directly replaying all flows in the four traces into the TCM component in the LS. Once the TCM algorithm identifies the <*superpeer, port*> pairs in the trace, we flag all the other nodes that connect to <*superpeer, port*> pairs as a p2p node. We use this information in Eqs. (1)–(3).

The top graph in Fig. 12 shows the overall detection rate $\left(DR_i^{full}\right)$ as a function of time for all the four traces. The detection rate is computed every 5 s (i.e., time interval, $z = 5$). In other words, we accumulate all our findings in every 5-s interval and use them to update the detection rate at the end of the interval. We can see that the overall detection rate is between 40% and 60% after the first time interval, but it increases beyond 90% within 3 min in all of the traces. The overall detection rate reaches close to 100% in most of the traces within 10 min.

The bottom graph in Fig. 12 is similar to the top graph, but shows the detection rate (DR_i) in every interval. We can once again notice that the detection rate in any interval

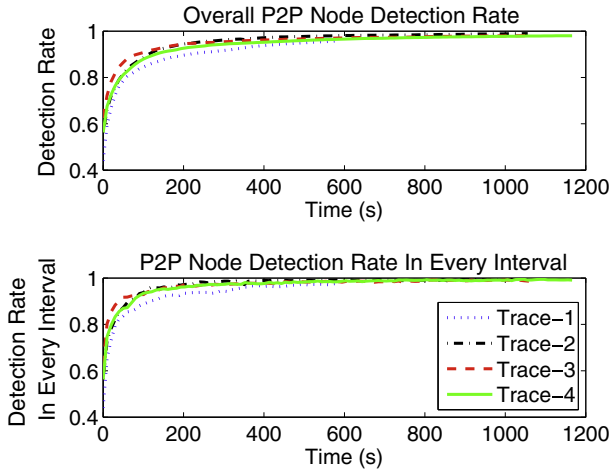

Fig. 12. Overall and per-interval detection rates of TCM vs. time.

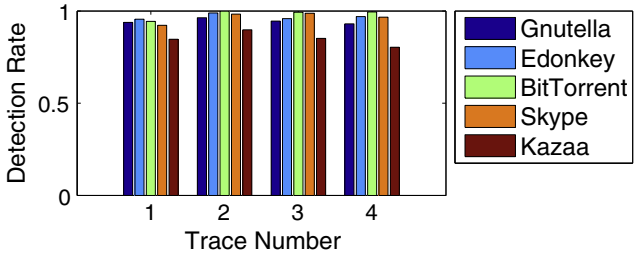

Fig. 13. Overall TCM detection rate for different p2p applications.

reaches 95% within 3 min and remains close to 100% after 7 min. The main take-away point here is that after the first few minutes, the detection rate in any interval remains very close to 100%, showing that TCM can identify all p2p nodes in the network very effectively.

Fig. 13 shows the overall detection rate of five popular p2p applications in the Internet for which we have the ground truth in our L7PA. We can see that the overall detection rate of most of the applications in all the four traces is over 95%. Note that Fig. 13 shows the overall detection rate after full trace replay. However, the detection rate in every interval for all the applications is much higher (over 99%) after the first few intervals.

Based on our four traces, we find that some of the applications (like Kazaa) are not very popular. We found only a few Kazaa flows in all our traces (see Table 1). Given that the TCM parameters are tuned to capture superpeers that are significantly active, we miss a few nodes in these applications that are not popular; hence the detection rates are smaller compared to the other applications. However, from Fig. 12, we can clearly see that the p2p detection rate is still very high.

Although our detection rates are very high, the accuracy of our TCM algorithm depends on the false positive and false negative rates as well. **The false positive rates in all our above experiments were consistently** *zero* **for all the four traces.** In other words, using TCM algorithm, we did not identify any nodes that belonged to the non-p2p set as a p2p node. The false negative rate, by definition in Eq. (3), is the complement of the detection rate, i.e.,

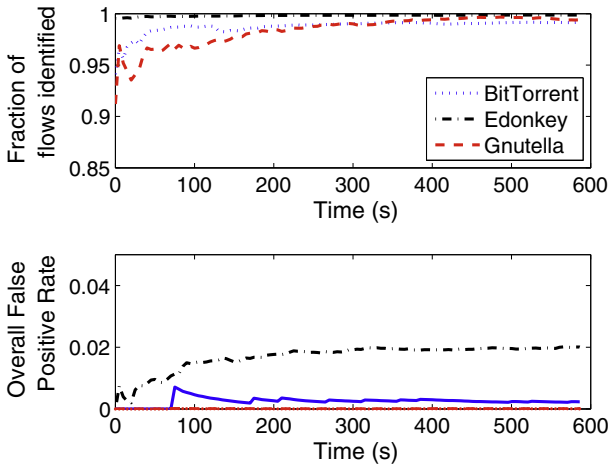

Fig. 14. Top graph shows the fraction of flows identified and the bottom graph shows the false positive rate of the SE algorithm in Trace-1.

$DR_i = 1 - FN_i$ and $DR_i^{full} = 1 - FN_i^{full}$. Hence, we do not explicitly plot the results for the false negative rate.

7.3. Signature extractor

To evaluate the accuracy of the SE algorithm, we first identified different p2p applications in the traces using L7PA, and then passed the flows corresponding to these applications to the SE component running the LASER algorithm. The signatures extracted by the LASER algorithm were identical to the signatures used in L7PA for all p2p applications.

We evaluate the accuracy of our signature extraction algorithm using three metrics: (i) *Fraction of flows* that are accurately identified by the extracted signature, and (ii) *false positive rate*. The top graph in Fig. 14 shows the fraction of flows identified for the three p2p protocols in Trace-1. We can clearly see that for all the three protocols that we tested, as time progresses the detection rate is well beyond 99%. The main take away point is that the signatures that we extract using our simple algorithm in Section 6 is very accurate. Also, the detection rate for most of the applications is lower at the beginning of the trace when the SE algorithm is still learning the signature for the application. The bottom graph of Fig. 14 shows the false positive (i.e., identifying a different application as a particular p2p application) rate of the SE algorithm. We can clearly see that for both BitTorrent and Gnutella, the false positive rate is very small (<0.001). However, for eDonkey the false positive rate was higher (≈ 0.02). The main reason for this is that the signature for eDonkey is very small compared to the other protocols and hence few other flows can be mistaken to be eDonkey flows.

We evaluate the *robustness* of these signatures using *flow recall rate*, i.e., the fraction of flows in Trace-*x* that are accurately identified using signatures from Trace-*y*. A high value of flow recall rate implies that the signatures extracted are generic, stable, and reusable. Fig. 15 shows detection rate when using the signatures extracted from Trace-1 in Trace-4. Note that Trace-1 and Trace-4 are collected from two ISPs in different continents. We can clearly see that the detection rate is still very high (>0.995), showing that the signatures that we extract are robust across different networks.

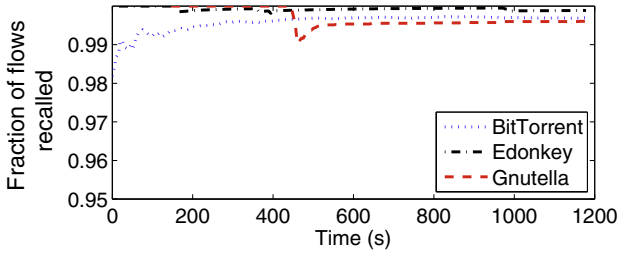

Fig. 15. Recall rate when using signatures extracted from Trace-1 in Trace-4.

7.4. SLTC system

There are two main objectives for SLTC as a system: (i) Learn signatures for *all* known and unknown p2p applications seen by the HSM, and use these signatures to accurately classify future incoming flows, and (ii) learn these signatures as quickly as possible so that the number of "unclassified" flows can be minimized. For experiments in this subsection we replay all flows in the trace files into the HSM, and any flow that cannot be directly classified in the HSM is sent to the TCM component in the LS. The output (i.e., <*ip, port*> pairs of superpeers) from the TCM is used by the HSM to send packets to the SE component. The SE algorithm finds the signatures of p2p applications and populates the signature database in the HSMs.

We demonstrate how well SLTC meets both its objectives using two metrics: (i) *Fraction of flows* that are "unclassified" in every interval. This represents the fraction of p2p flows that SLTC has not learnt about, i.e., the p2p flows sent to LS for classification and signature extraction, and (ii) *time lag*, i.e., the total time taken to extract a signature and populate the database after seeing the first packet of a flow.

The top graph in Fig. 16 shows the fraction of "unclassified" flows in SLTC, i.e., the p2p flows that are sent to LS for

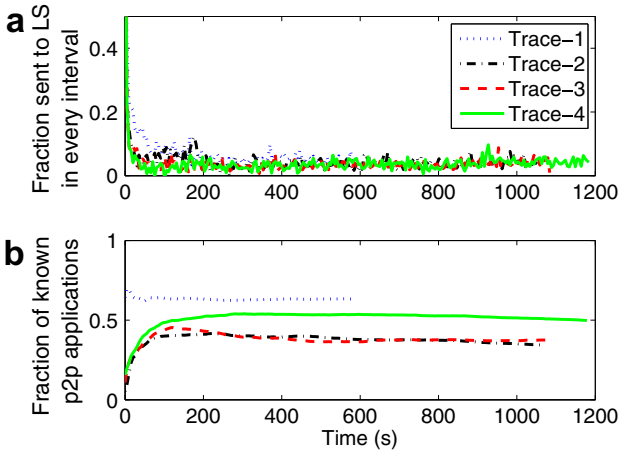

Fig. 16. (a) Classified flows and (b) known applications in classified flows.

Table 2
New p2p application signature – Gigaget.

Protocol/set	Index	String
Gigaget/set-1	0	0 × 29000000
	10	0 × 00000007000000434f4e4e454354
Gigaget/set-2	0	0 × 29000000
	10	0 × 0000000
	14	0 × 000000
	17	0 × 474554

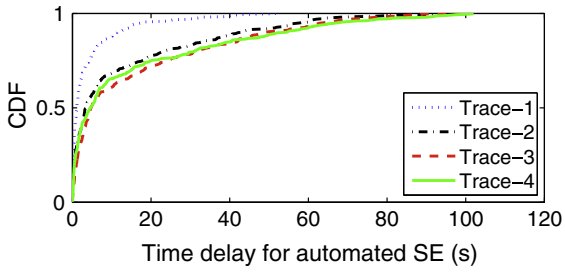

Fig. 17. CDF of the total time lag.

classification, as a function of time. Note that at time 0 we do not have any signatures in the database and hence all flows are sent to the logic server. However, as time goes by, the logic server extracts more and more signatures, thus ensuring that most of the traffic is classified by the HSM. In fact, more than 90% of the p2p traffic is classified within the first minute. We can clearly see that the fraction of traffic entering the logic server decreases over time. Note that the fraction of traffic sent to the logic server steadies at about 5% in all the four traces. This means that our SE component is not able to extract signatures for these flows even after repeated attempts. These flows could be either encrypted or simply do not have a signature. We defer further exploration of this issue as a part of our future work.

The bottom graph in Fig. 16 shows the fraction of the "classified" flows that are "known". In other words, among all the classified flows (i.e., p2p flows for which SLTC learns a signature) from the top graph in Fig. 16, there are some flows that belong to "known" applications and others belong to "unknown" or "new" p2p applications. We use five applications – BitTorrent, Gnutella, eDonkey, Skype, and Kazaa – as known p2p applications and classify the rest as unknown. From the bottom graph in Fig. 16, we can see that more than 40% of the classified p2p applications are unknown for Trace-1 whereas the percentage increases to about 60% for the other traces. Learning unknown or new p2p applications is one of the key features of SLTC and from the bottom graph of Fig. 16 we can clearly see that SLTC has learnt several new p2p applications. In fact, we manually explored the signature of an unknown application and found that it belongs to Gigaget p2p network (Table 2) [3].

Fig. 17 shows the CDF of the total time lag. Total time lag is the sum of the times taken TCM and SE algorithms, and both these times depend on the parameters set in those components. For example, the time delay in TCM is characterized by three parameters T_{th}, P_{th}, and R_{th}. Changing the values of these parameters could result in a tradeoff between accuracy and time lag. Using the default parameter values for TCM and the SE algorithm, in Fig. 17, we can see that classification and signature extraction of over 90% of the flows in all the traces takes less than 1 min, making it feasible to use this architecture in real-time with strict time constraints.

8. Discussion and conclusions

In this paper, we presented a novel self-learning p2p traffic classifier that can learn known and unknown applications with minimum manual intervention. We showed that SLTC can learn over 95% of p2p applications in a few minutes. Although our focus in this paper was on classifying p2p applications, we strongly believe that the SLTC framework can be used for classifying other classes of Internet traffic as well.

We presented a simple, light-weight, and effective algorithm, called TCM, for superpeer-based p2p traffic identification using temporal correlation of flows. We showed that the detection rate is very high with no false positives. One of the main reasons for having no false positives is the fact that the monitoring points are at the network edge. Unlike p2p-protocols, most of the non-p2p protocols in the Internet (like smtp, ftp, pop3, imap, http, etc.) are typically configured such that they are aware of network borders.

Also, in this work, we have used packet traces with no sampling. However, we envision the TCM algorithm to work even if the packets/flows in the traces were sampled. We plan to explore this further as a part of our future work.

References

[1] Bittorrent. <http://www.bittorrent.com/>.
[2] Emule project. <http://www.emule-project.net/>.
[3] Gigaget. <http://www.gigaget.com/>.
[4] Gnutella. <http://www.gnutella.com>.
[5] Ipoque survey. <http://www.ipoque.com/>.
[6] Kazaa. <http://www.kazaa.com/>.
[7] L7 filter. <http://l7-filter.sourceforge.net/>.
[8] Narus technical report. <http://www.narus.com/research/SLTC.pdf>.
[9] Skype. <http://www.skype.com/>.
[10] S. Baset, H. Schulzrinne, An analysis of the skype peer-to-peer internet telephony protocol, in: Technical Report, CS Dept., Columbia University, 2004.
[11] CacheLogic, The True Picture of Filesharing. <http://www.cachelogic.com/home/pages/research/p2p2004.php>.
[12] F. Constantinou, P. Mavrommatis, Identifying known and unknown peer-to-peer traffic, in: IEEE International Symposium on Network Computing and Applications, 2006.
[13] J. Erman, M. Arlitt, A. Mahanti, Traffic classification using clustering algorithms, in: ACM SIGCOMM Workshop on Mining Network Data, 2006.
[13] P. Haffner, S. Sen, O. Spatscheck, D. Wang, ACAS: automated construction of application signatures, in: ACM SIGCOMM Workshop on Mining Network Data, August 2005.
[15] T. Karagiannis, A. Broido, M. Faloutsos, K. Claffy, Transport layer identification of p2p traffic, in: Proceedings of ACM SIGCOMM Internet Measurement Conference, 2004.
[16] T. Karagiannis, K. Papagiannaki, M. Faloutsos, BLINC: multilevel traffic classification in the dark, in Proceedings of ACM SIGCOMM, August 2005.
[17] R. Keralapura, A. Nucci, C.-N. Chuah, Self-learning peer-to-peer traffic classifier, in: ICCCN, 2009.

[18] A. McGregor, M. Hall, P. Lorier, J. Brunskill, Flow clustering using machine learning techniques, in: PAM, April 2004.
[19] A. Moore, K. Papagiannaki, Toward the accurate identification of network applications, in: PAM, March 2005.
[20] B.-C. Park, Y.J. Won, M.-S. Kim, J.W. Hong, Towards automated application signature generation for traffic identification, in: IEEE/IFIP NOMS, 2008.
[21] S. Sen, O. Spatscheck, D. Wang, Accurate, scalable in-network identification of p2p traffic using application signatures, in: WWW2004, May 2004.
[22] D. Stutzbach, R. Rejaie, Understanding churn in peer-to-peer networks, in: IMC, 2006.
[23] N. Williams, S. Zander, G. Armitrage, A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification, in: CCR, October 2006.
[24] S. Zander, T. Nguyen, G. Armitage, Automated traffic classification and application identification using machine learning, in: LCN'05, November 2005.
[25] S. Zander, T. Nguyen, G. Armitage, Self-learning IP traffic classification based on statistical flow characteristics, in: PAM, 2005.

Dr. Ram Keralapura is currently a Senior Member of Technical Staff at Narus, Inc. He received his BE in Electrical Engineering from Bangalore University, India in 1998 and MS in Computer Science from the University of Alabama in Huntsville in 2000. He received his Ph.D. from the Electrical and Computer Engineering Department at UC, Davis in 2006. His current interests are in overalay/p2p networks, managing distributed networks, Internet routing, traffic engineering, security, and cellular networks.

Dr. Antonio Nucci received the Dr.Ing Degree in Electronics Engineering in 1998 and the Ph.D. degree in telecommunications engineering in 2002, both from the Politecnico di Torino, Turin, Italy. In September 2001 he joined the Sprint Advanced Technology Laboratories, Burlingame, CA, where he has been a principal member of technical staff in the IP research group until February 2005. He currently holds a CTO position at Narus Inc. His research interests include traffic measurement, characterization and analysis, performance evaluation, traffic engineering, security and network design. He is a IEEE senior member.

Prof. Chen-Nee Chuah is currently an Associate Professor in the Electrical and Computer Engineering Department at the University of California, Davis. She received her B.S. in Electrical Engineering from Rutgers University, and her M.S. and Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley. Her research interests lie in the area of computer networks and wireless/mobile computing, with emphasis on Internet measurements, network anomaly detection, network management, multimedia, online social networks, and vehicular ad hoc networks. She received the NSF CAREER Award in 2003, and the Outstanding Junior Faculty Award from the UC Davis College of Engineering in 2004. In 2008, she was selected as a Chancellor's Fellow of UC Davis. She has served on the executive/technical program committee of several ACM and IEEE conferences and is currently an Associate Editor for IEEE/ACM Transactions on Networking.