An apparatus for P2P classification in Netflow traces

Andrew M Gossett, Ioannis Papapanagiotou and Michael Devetsikiotis
Electrical and Computer Engineering, North Carolina State University, Raleigh, USA
Emails: amgosset@ncsu.edu, ipapapa@ncsu.edu and mdevets@ncsu.edu

***Abstract*—Application classification from Netflow traces is a challenging process due to the lack of payload information. It is even becoming more challenging when the applications running in the network tend to hide under well known ports, encrypt packets and are distributed. In this paper, we propose an in-the-box apparatus for Netflow classification of Bit Torrent applications. The apparatus includes several device optimizations and requires low processing power. It was developed by reverse engineering the Bit Torrent protocol and by identifying connection patterns. The accuracy of the algorithm reaches high values specifically for Bit Torrent peers that use the DHT protocol.**

I. Introduction

In order to provide Quality of Service (QoS), to overprovision lines and to plan resources optimally, it is of vital importance to know the application that the end user is running [1]. However, several applications tend to hide under ports that IANA [2] has allocated for other purposes. Moreover, pattern recognition and behavior profiling has become a necessity in the discovery of malicious traffic, and Denial of Service attacks. Therefore the Internet Service Providers (ISPs) have incorporated out of the box processor draining devices such as Deep Packet Inspection (DPI) or Intrusion Detection Systems (IDS) [3].

Most researchers have paid attention to application profiling through content identification and behavior pattern matching. The accuracy of classification is most of the times very high [4]. However, packet identification faces limitations due to storage and processing overhead. In addition, payload classification may fail for encrypted payload. New applications signature patterns will need to have vigorous string and regular expression search and optimization.

On the other hand, flow based techniques use only header information. Therefore, flow classification accuracy is always worse than packet based classification. Yet, one of the main advantages of storing Netflow traces is that information are saved per flow and not per packet, decreasing significantly the storage and processing overhead, as well as the cost of implementation. Due to these advantages, Netflows are the most widely collected traces. In order to identify the application from Netflow traces several approaches have been proposed. Port based flow classification has been regarded as not very efficient way for classification [5]. Other techniques use advanced machine learning algorithms to compute patterns from a training dataset [6]. However, such approaches require the existence of a "good" training dataset and require a lot of resources to run, limiting the application to be out of the box (require a separate device for analyzing the traces).

In this work we mainly focus on investigating a simple, effective and modular P2P classification algorithm. The P2P file sharing filter algorithm is designed based on the principles of DHT (Distributed Hash Table) for the Bit Torrent protocol [7]. We dealt with challenges, such as hiding under known ports and encryption, by reverse engineering the protocol and by proposing an effective way to identify communication patterns. Since one of the major challenges of network operators is the limited resources on the network devices, we incorporated several optimizations into configurable arguments.

The paper is structured as follows. In the next section a general overview of the developed algorithm is given. In Section III, a detailed explanation along with the configurable arguments is provided. In Section IV, the algorithm is evaluated for several torrent clients and configurations. Finally, conclusions are drawn based on the performance of the algorithm.

II. Algorithm Overview

In the Bit Torrent protocol a client will obtain a .torrent file. This file contains the required pieces of the file(s) to be downloaded along with information on the tracker(s). A tracker is a web-server that acts as a centralized communication point between clients transferring torrents. The tracker maintains a list of the current peers that can be contacted; it does not store the data being transferred by the Bit Torrent protocol. Once the client receives a list of potential peers and their open ports from the tracker, it can open up TCP sessions and negotiate the transfers.

DHT is a mechanism to perform decentralized file transfers. This eliminates the need of a central tracker that maintains the full list of current peers. Each node will dynamically join the DHT and build routing tables based on special hashes to locate peers who have the desired torrents. This routing table requires constant control methods. Additionally, when a node is searching for peers it will send a burst of queries to several neighbors at the same time. Once the node has found a peer to download the torrent from, it will open up a TCP session. It has been observed that the control packets are sent via UDP and the TCP sessions created are generally opened on the UDP control port of either the requestor, seeder, or both.

The criteria for classifying a particular IP address and port number as BitTorrent traffic is based on the following steps.

1) If a certain number of flows with the same source IP and source port number are created in a specific interval, then this source IP and source port is a BitTorrent file sharing participant. The value of this interval is based on the Bit Torrent protocol and will be analyzed in the next section.
2) There are non-peer-to-peer applications that accept several connections on the same port (such as a web server). Therefore, we need to include the limitation that a valid BitTorrent port number must be above the well-known IANA range [3] of TCP/UDP port numbers. Note that users can set their BitTorrent control port to any number, but by default these are usually in the Dynamic or Private Port ranges.
3) Finally, if a source or destination port is on the commonly used BitTorrent ports 6881 through 6889 [1], then we can classify the particular IP address and port number as a BitTorrent participant.

Analyzing the packet pattern of the UDP control packets, we can identify a particular IP address and port number that is participating in a BitTorrent file transfer. Since one of the IPs involved in the TCP session will keep the port number used by the UDP control mechanism, we can also identify the negotiated TCP session as a BitTorrent file transfer. To do this, we need to track all flows created for each IP address and port number combination. If we have identified a source IP and source port as a BitTorrent file transfer participant, we can identify related TCP sessions as BitTorrent file transfers even if the port numbers have changed. As long as the IP addresses in the flow and at least one of the port numbers matches the identified BitTorrent file transfer participant and one of its flows, then we can conclude that the TCP session is a result of the UDP control message and therefore must be a BitTorrent file transfer. Therefore, we can introduce the forth heuristics as follows:

- If a flow's source IP and source port match an identified BitTorrent file sharing participant, then the flow can be classified as BitTorrent file sharing
- If the source IP matches a BitTorrent file sharing participant and the destination IP and destination port match a flow previously seen by the BitTorrent file sharing participant, then the flow can be classified as BitTorrent file sharing.
- If a flow's destination IP and destination port match an identified BitTorrent file sharing participant, then the flow can be classified as BitTorrent file sharing.
- If the destination IP matches a BitTorrent file sharing participant and the source IP and source port match a flow previously seen by the BitTorrent file sharing participant, then the flow can be classified as BitTorrent file sharing.

Not all BitTorrent clients use the DHT. Instead, they rely completely on the tracker for peer information. For these clients, no UDP control traffic will be present. However, the client will still attempt to open several TCP connections to peers on a particular source port number in a small interval of time. For this reason, both TCP and UDP flows can be compared to the proposed heuristics to determine if an IP address and port number combination is a BitTorrent file transfer participant. The algorithm is based on the BitTorrent protocol and DHT implementation. However, many point-to-point file sharing protocols have similar characteristics as describe by the proposed heuristics. For this reason, we will label classified IP address and TCP/UDP port number combinations as point-to-point file sharing participants instead of BitTorrent file sharing.

III. Implementation

The algorithm classifies applications based on Netflow data. The algorithm maintains the state of all identified point-to-point file sharing participants (P2P participant), while processing each of the trace files. Therefore, the time interval in which a particular IP address and port combination remain classified as a P2P participant is dependent on the number of trace files and the time interval.

For a flow to be analyzed, four important tables are maintained:

- The Point to Point Profiler table (P2PP),
- The Pending Further Investigation table (PFIT),
- The classified table, and
- The classified lookup table.

The **P2PP** is a hash table of P2PP entries whose key is a flow's source IP address and source port number. The entry maintains a list of all flows that contain this source IP and source port. Each entry also maintains a hash table for fast lookups of flows in its flow list. The P2PP entry's flow lookup table is important for three reasons.

The first is to prevent redundant flows from being checked against the heuristics. For our implementation, a flow is defined by nine parameters, protocol number, source and destination IP, source and destination port, start timestamp, packets, bytes, duration of flow. However, according to Cisco Netflow Format [8], flows can contain additional information such as TCP flags, input and output interface indexes, etc. Therefore, there can be "redundant" flows that must not be used in the classification. For example, in a TCP connection there will be a different flow for each of the steps within the three-way handshake (SYN, SYN+ACK, and ACK). The heuristic looks for multiple connections to different user, therefore the different flows between the same two hosts should not be used. Note that the "redundant" flows are still added to the P2PP entry's flow list to account for packet, byte, and flow statistics.

Second, the **lookup table** provides a quick reference when examining the PFIT (explained later in this section). Finally, it allows for the flow list to be empty while still maintaining the state of each flow within the P2PP entry. This conserves memory when processing a large number of related flows. A P2PP entry is identified as a P2P participant as shown in Fig. 1.

The **PFIT** is a hash table of flows. Each flow whose corresponding P2PP entry is not yet identified as a P2P participant is placed in the PFIT. When classifying only clients

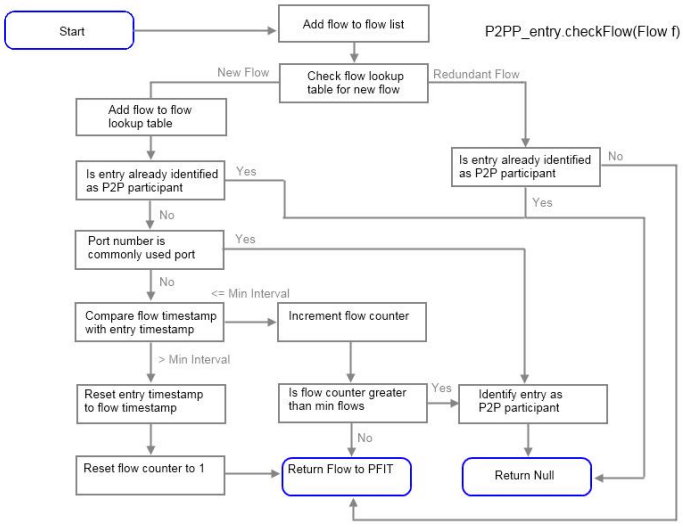

Fig. 1. P2PP flow chart for determining if entry is a P2P participant

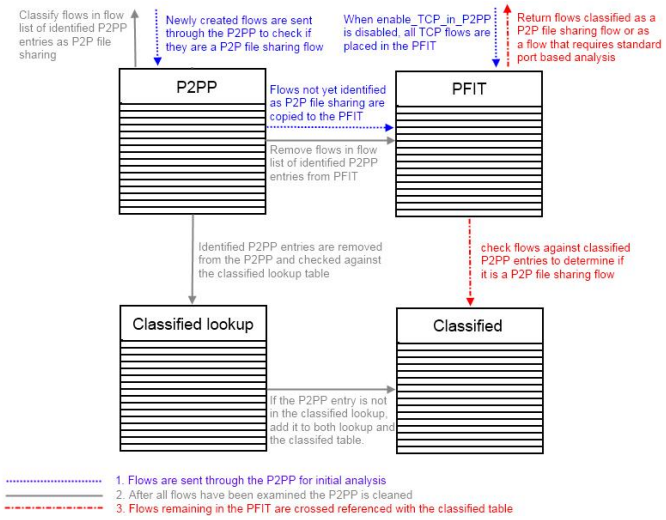

Fig. 2. Overview of the interactions between the P2PP, PFIT, classified, and classified lookup tables.

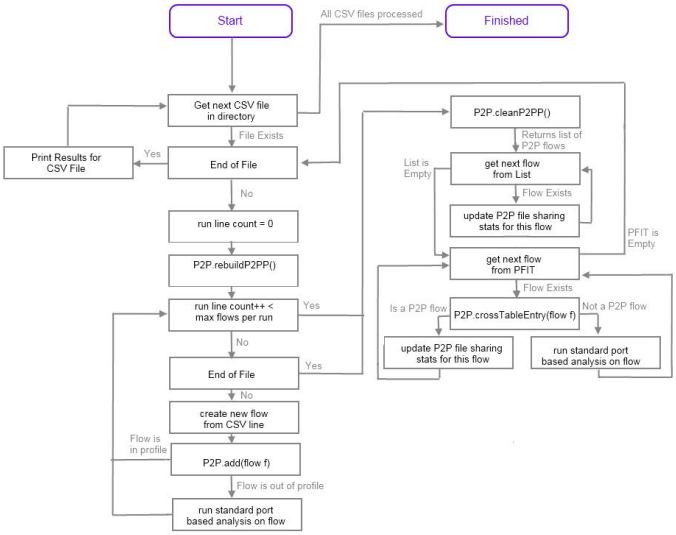

Fig. 3. Complete alogirthm overview for Flow based classification of network traffic.

that use the DHT, all TCP flows are originally placed in the PFIT and then later cross referenced with the classified table. Additionally, all flows that are returned from the P2PP are also placed in the PFIT table (as shown in Fig. 1).

The classified table is a hash table that contains a list of P2PP entries that have been identified as P2P participants. Each P2PP entry in the list shares the same source IP address but will have different source ports. The classified lookup table is a hash table of all P2PP entries currently in the classified table. It is used to quickly determine if a newly identified P2PP entry should be added to the classified table without searching each list in the classified table.

Several configurable attributes are defined within the algorithm implementation. Table 1 lists and describes each constant.

A P2P file sharing application should use port numbers above the well-known IANA range 0-1023, therefore the default minimum port was chosen to be 1023. The minimum flows and minimum interval are values that require the most analysis. The DHT protocol routing algorithm [7] defines 'buckets' of nodes in which queries may be sent to. The maximum size of these buckets is eight. By observation, we found that queries sent to neighboring nodes were in burst of five to seven. For this reason, we choose the default minimum flows to be five. The minimum interval was chosen based on empirical data presented in the next section.

The "max flows per run" was incorporated as an optimization. When processing large trace files, the P2PP and PFIT tables would grow too large, leading to saturation of memory resources. The default of 750,000 flows was set as a performance limitation of the testing environment. Faster machines can likely handle greater than 1,000,000 flows without much of a performance impact.

An overview for the interactions between the P2PP, PFIT, classified, and classified lookup tables is shown in Fig. 2. The complete steps for classifying network traffic based on Netflow data are shown in Fig. 3.

Note in Fig. 3 the following processes in function format: P2P.rebuildP2PP(), P2P.add(flow f), P2P.cleanP2PP(), and P2P.crossTableEntry(flow f).

A. P2PP Rebuild process

The algorithm maintains the state of all identified P2P participant while processing each of the trace files. However, after each trace file is processed or the maximum allowed flows per run is exceeded, the PFIT and P2PP tables are cleared. Therefore, to maintain the state of all identified P2P participants, the P2PP is re-populated at the beginning of each run with classified P2PP entries in the classified table. An important thing to note is that all P2PP entries in the classified table have no flows in their flow list. They only have the

TABLE I
CONFIGURABLE ATTRIBUTES

Constant	Default Value	Description
min port	1023	Minimum port value for a flow to be considered in profile to be a P2P participant.
min flows	5 flows	Minimum number of flows that must be seen within min interval for a P2PP entry to be considered a P2PP participant. Once a P2PP entry is identified as a P2PP participant, all associated flows will be classified as P2P file sharing flows for the duration of the algorithm
min interval	200 msecs	Small interval of time in which the min flows must be seen to classify a P2PP entry as a P2PP participant.
enable TCP in P2PP	true	Flag to indicate if TCP flows can be used to create new P2PP entries. When disabled, no P2P file sharing applications will be identified for clients that do not use the DHT.
max flows per run	750,000	Maximum number of flows processed before the PFIT and P2PP are cleaned. Higher numbers will have potentially higher number of flows identified as p2p file sharing applications. However, higher numbers also degrade performance as more memory and collisions occur when accessing the PFIT and P2PP hash tables. Note for small CSV files (less flows than max flows per run) this variable will not come into play.

appropriate keys for each flow within their flow lookup table. This conserves space as the flows (potentially 100,000's) can take up a lot of memory. After classifying the flow, there is no reason to store the flow data. We only need to know that the flow existed. The flow lookup table will contain the flow key to indicate this.

B. Add Flow process

The algorithm attempts to add a flow to a particular P2PP entry and/or PFIT table. However, to be classified as P2P file sharing flow, the flow must be UDP or TCP and both the source and destination port number must be above the well known IANA range of 1024. The flow is first checked against this criterion to determine if it is within profile. If so, then the following occurs:

If the flow is UDP it will be added to the P2PP. If the flow is TCP and TCP flows are not allowed in the P2PP, then the flow is added to the PFIT. Otherwise, the TCP flow will also be added to the P2PP. When a flow is placed in the P2PP, the P2PP table will be checked for an entry with the flows source IP and source port. If none exists, a new entry will be created. Finally, the flow will be added to the P2PP entry via $P2PP\_entry.checkFlow(flowf)$ as shown in Fig. 1. If the corresponding P2PP entry is not identified as a P2P file sharing participant, then the flow will also be added to the PFIT.

C. Clean P2PP process

After all flows within a trace file have been examined, or the maximum number of flows per run has been exceeded, then the P2PP will be cleaned. Each entry within the table will be checked, and if it has be identified as a P2P participant, than all flows within its flow list will be removed from the PFIT and the returned flows will be classified as P2P file sharing flows. The P2PP entry will be added to the classified table for future reference if more flows need to be examined. The final P2PP table will be empty.

D. Cross Table Entry process

The final step of the algorithm is to cross reference all flows still remaining within the PFIT after the P2PP has been cleaned with the classified table.

- Search the classified table for a list of p2pp entries with source IP equal to this flow's source IP.
  - if the P2PP entries source port equals this flow's source port, then the flow is a P2P file sharing flow
  - if any of the flows within the P2PP entry's flow lookup table have a destination IP equal to this flow's destination IP and the destination port is equal to this flow's destination port, then the flow is P2P file sharing flow
- Search the classified table for a list of p2pp entries with source IP equal to this flow's destination IP.
  - if the P2PP entries source port equals this flow's destination port, then the flow is a P2P file sharing flow
  - if any of the flows within the P2PP entry's flow lookup table have a destination IP equal to this flow's source IP and the destination port is equal to this flow's source port, then the flow is P2P file sharing flow
- The flow is not a P2P file sharing flow.

The source code is available at

IV. VALIDATION

We tested the algorithm on an isolated lab with several users. A Linux server was setup with three different BitTorrent clients installed:

1) Transmission 1.34 [5]
2) Vuze 4.3.1.4 [6]
3) uTorrent 2.0.1 [7]

Note that Transmission 1.34 is not DHT aware and therefore will not send any UDP control packets during the BitTorrent file transfer. Vuze and uTorrent clients are DHT aware.

Two additional programs were installed to export and collect Netflow statistics: Flow-tools [8] and Softflowd [9]. Softflowd can be used to monitor the Network Interface Card (NIC) on the server and create flows. These flows were then exported to the server's loopback address and stored in standard Netflow Flow-tools format. Later, the Netflow flows were exported to plain text CSV (Comma separated value) files and processed.

Two tests were performed for each client. In the first test, the server sent and received only BitTorrent traffic. In the

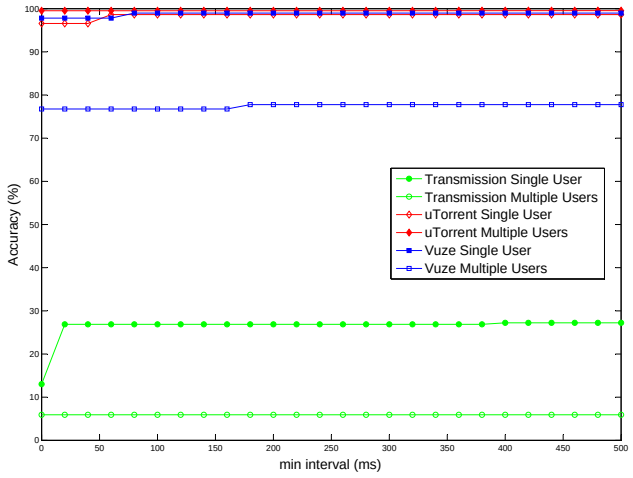

Fig. 4. Accuracy of each BitTorrent client (DHT and non-DHT) for both tests with varying the "minimum interval".

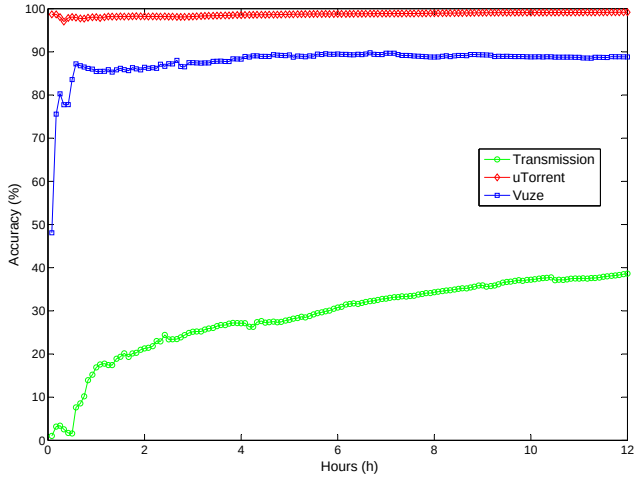

Fig. 5. Flow Accuracy with TCP enabled in the P2PP as a function of flow correlation time.

second test, the server was connected to a hub so additional network traffic would be present in the Netflow data. This included streaming music over HTTP traffic, FTP downloading sessions, and Play Station online gaming (a form of P2P traffic using fixed port numbers: TCP 80, 443, 5223 and UDP 3478, 3479, 3658).

When calculating the algorithm's accuracy, we defined all traffic destined to or from the server's IP address with TCP/UDP port number greater than the IANA well-known range as P2P file sharing traffic. Additionally, we included HTTP traffic on TCP/UDP port 80 as contact to the tracker will be HTTP data.

We analyzed the Netflow data for test 1 and test 2 varying several parameters. We first varied the minimum interval defined in heuristic 1. We found that accuracy quickly levels off after 200 milliseconds for all tested clients and for running time equal to 1h, as shown in Fig. 4. We chose to plot Transmission as an example of a non-DHT client, and Vuze and uTorrent as examples of DHT clients. Our algorithm was performing with great accuracy for both tests for the DHT aware clients. While the accuracy of the non-DHT torrent client was low, this was observed to be affected by the "learning" time, as shown in Fig. 5.

More specifically the classification of a particular flow is also dependent on the amount of past data present. This is directly proportional to the number of flows that previously existed. As shown in Fig. 3, the classified table and P2PP are maintained between each run. If a large number of files representing a long interval of time are processed together, the probability of identifying a P2P file sharing application is increased and therefore increases the accuracy of later dependent flows. In our test, a new Netflow file was created for every 5 minutes of data. In Fig. 5, the flow accuracy with TCP enabled in the P2PP is presented as a function of flow correlation time (Time difference between the first and last flow processed.).

Notice from Fig. 5 that the non-DHT aware Transmission client accuracy continues to increase as the correlation interval increases whereas the DHT aware clients quickly level off at a much smaller interval. This seems reasonable as the DHT aware clients have a dynamic routing table and are more likely to create new connections not previously seen by the algorithm. However, the non-DHT aware client is limited to the list of clients maintained by the tracker and is therefore more likely to have several flows to the same destination. As the history of identified IP address and port number pairs increases with correlation time, we see the accuracy of the algorithm increasing. Note that the DHT capable clients had an accuracy exceeding $> 90\%$.

In Fig. 6, the ROC curve is presented comparing the sensitivity, or True Positive Rate (TPR), vs. (1 - specificity), or False Positive Rate (FPR), through methodically varying each combination of the "min interval" and the "min flows". The diagonal divides the ROC space. Points above the diagonal represent good classification results, points below the line poor results. The Transmission single user, for some configuration, has high FPR. On the other hand, as more flows are added, the classifier looks to have a more randomized behavior (closer to the red line), therefore a conclusion cannot be made from the accuracy graph (Transmission Multiple Users in Fig. 4). However, such a behavior is normal since Transmission Multiple Users had low accuracy values. On the other hand, the DHT clients had high values of TPR, which shows that the accuracy graph for those BitTorrent graph is accurate. Similar behavior was observed when the TCP argument of the P2PP was disabled.

The accuracy of the algorithm was also compared against the standard port-based analysis (Std.1 and Std.2 in Fig.7).

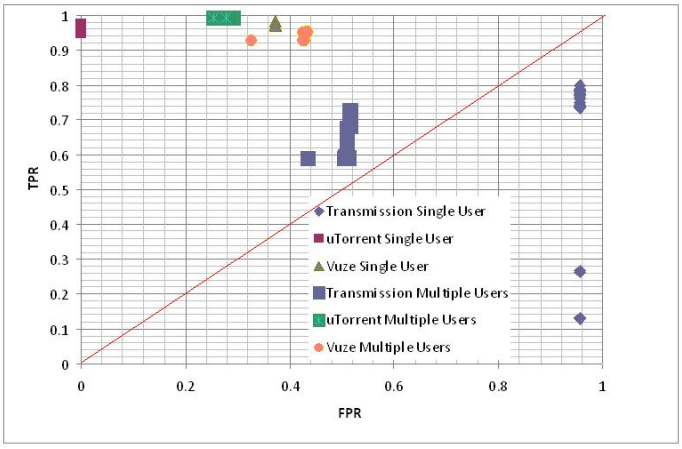

Fig. 6. ROC curve with TCP enabled

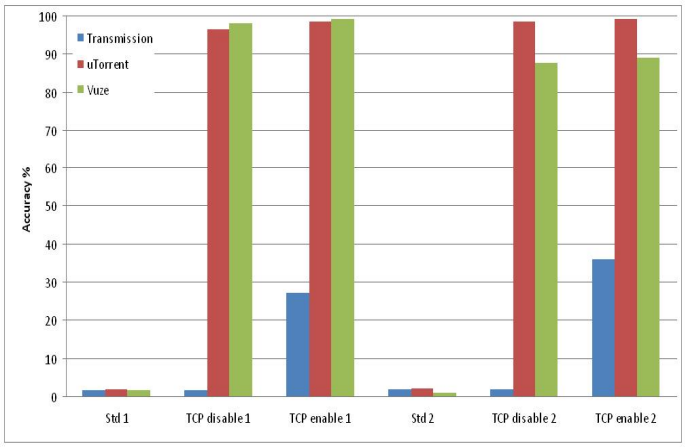

Fig. 7. Comparison of standard port-based classification and proposed heuristic with TCP enabled and TCP disabled in P2PP.

This was done for both tests with TCP enabled and disabled in the P2PP. The Transmission accuracy is identical to the standard port-based analysis, when TCP is disabled in the P2PP. This is because the non-DHT Transmission client does not send any UDP frames and therefore no IP addresses will be identified as a point-to-point participant. However, when TCP flows are allowed in the P2PP table it is observed that the accuracy of the non-DHT Transmission client increases significantly. Moreover, the DHT clients have very high accuracy for both tests.

V. Conclusion

In this paper, we have presented an algorithm that identified Bit Torrent flows based on the characteristics of the Bit Torrent protocol as well as communication pattern matching. The evaluation of the apparatus has shown accuracy $> 95\%$ for the Bit Torrent clients that exchange files through DHT. The algorithm was designed with configurable arguments such that it fits the limited resources on a network backbone. Finally the modular nature of the algorithm makes it easily implemented with other flow classification techniques. Future research includes the addition of other classification modules for other types of applications, with optimum goal to reach high accuracy with low processing power.

Appendix

Modularity: The flows in the PFIT that have not been identified as Bit Torrent flows, can be easily added to another filter for further processing. The filter can be implemented as a hash table and function similar to P2PP.

The work was supported by the Institute for Next Generation IT Systems (ITng).

References

[1] I. Papapanagiotou, M. Devetsikiotis, "Aggregation Design Methodologies for Triple Play Services" IEEE CCNC 2010, pp. 1-5, 9-12 January 2010, Las Vegas, USA.
[2] IANA, Internet Assigned Numbers Authority, http://www.iana.org/assignments/port-numbers (accessed May 2010)
[3] V. Paxson, "Bro: A system for detectin network intruders in real-time", Elsevier Computer Networks, vol.31, pp. 2435-2463, 1999
[4] A.W. Moore, K. Papagiannaki, "Toward the accurate identification fo network applications", Springer Passive and Active Network Measurement pp 41-54, 2005.
[5] T. Karagiannis, K. Papagiannaki, M. Faloutsos "BLINC: multilevel traffic classification in the dark", pp. 229-240, vol. 35, ACM SIGCOMM Computer Communication Review, 2005
[6] TTT. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning", IEEE Communications Surveys & Tutorials, pp. 56-76, vol.10, 2008
[7] The BitTorrent Protocol Specification. Feb 2008. http://www.bittorrent.org, (accessed May 2010)
[8] Cisco IOS NetFlow Version 9 Flow-Record Format. White Paper, Feb 2007.
[9] Transmission, http://www.transmissionbt.com/, (accessed June 2010)
[10] Vuze, http://www.vuze.com/, (accessed June 2010)
[11] uTorrent, http://www.utorrent.com/, (accessed June 2010)
[12] Flow-tools, http://www.splintered.net/sw/flow-tools/, (accessed June 2010)
[13] Softflowd. http://www.mindrot.org/projects/softflowd/, (accessed June 2010)