NTCS: A Real Time Flow-based Network Traffic Classification System

Silas Santiago Lopes Pereira, Jorge Luiz de Castro e Silva, José Everardo Bessa Maia
Department of Statistics and Computing
UECE - State University of Ceará
Fortaleza - Ceará - Brazil
Email: {silas, jlcs}@larces.uece.br, jose.maia@uece.br

***Abstract*—This work presents the design and implementation of a real time flow-based network traffic classification system. The classifier monitor acts as a pipeline consisting of three modules: packet capture and preprocessing, flow reassembly, and classification with Machine Learning (ML). The modules are built as concurrent processes with well defined data interfaces between them so that any module can be improved and updated independently. In this pipeline, the flow reassembly function becomes the bottleneck of the performance. In this implementation, was used a efficient method of reassembly which results in a average delivery delay of 0.49 seconds, aproximately. For the classification module, the performances of the K-Nearest Neighbor (KNN), C4.5 Decision Tree, Naive Bayes (NB), Flexible Naive Bayes (FNB) and AdaBoost Ensemble Learning Algorithm are compared in order to validate our approach.**

I. Introduction

This work describes the architecture of a real time Internet traffic classifier monitor for use in corporate networks. It also evaluates different machine learning methods for network traffic classification. The classifier monitor is based on concept of bidirectional flow. This means that the fundamental object to be classified in a determined pattern is the traffic flow, either complete or as subflow. A flow is defined by one or more packets between a host pair with the same quintuple: source and destination IP address, source and destination ports and protocol type (ICMP, TCP, UDP) [1].

The remainder of this paper is organized as follows. Section II overviews the related work about flow reassembly and traffic classification. In section III, we describe the design and implementation of the classifier monitor. Section IV details the data colection used for evaluate the NTCS and describes how the experiments were performed. Section V presents and discusses the performance tests results. Section VI ends with some conclusions and future work.

II. Related Work

Here, we briefly review some important approaches to stream reassembly and traffic classification.

In [2], the authors present an efficient TCP stream reassembly mechanism for real time network traffic processing at high speeds. The mechanism uses the recently-accessed-first principle to reduce the search cost of a connection for each packet arrival. Moreover, to improve the search process, the system keeps established and not established TCP connections in different structures. Experimental results based on network traffic captured in a typical gigabit gateway showed the proposed policy, in comparison of traditional one (RFC 793), was more efficient and could attend the real time property requisite of traffic analysis systems in gigabit networks.

In [3], the author presents a TCP stream reassembly mechanism designed and implemented to an network-based intrusion detection system. The system receives individual packets from network and performs signature detection from the payload. The approach is described as follows: First, the system associates each received packet to its corresponding TCP connection, based on the quadruple composed of source IP address and port, and destination IP address and port. Then, the system checks the packet sequence number and determines if this packet is the next expected packet for the respective connection. If true, the packet is sent for signature detection.

In [4], the machine learning approach applied to traffic classification using only transport-layer statistics is explored. This approach seeks to circumvent the problem that many network applications, such as P2P protocols, make use of dynamic port numbers and content encryption to avoid detection. This becomes inefficient the traditional approaches of port mapping and content analysis. The author evaluate the impact on the performance of data dimensionality, selected attributes, and machine learning used algorithms, which were, respectively, TAN, C4.5, NBTree, RandomForest and Distance Weighted KNN. The application of classification techniques and discriminant selection based on genetic algorithms together can dramatically reduce the learning and modeling time with little variation in the classification process accuracy.

III. The Classifier Monitor

This section details the design and operation of our classifier monitor, and concludes with the presentation of the modules which compose the system.

A. Architecture

The monitor works as a three-stage pipeline, with a collect and preprocessing module, a flow reassembly module, and an attribute extraction and classification module. For the purpose of pipeline, the time is divided in intervals of 30s. This value was chosen arbitrarily. Once the monitor starts, three parallel processes are in execution on each interval: the packet capture,

the flow reassembly of the previous interval packet capture, and the flow classification for the collection occurred in two delay intervals. Another parallel process is responsible for closing old connections periodically, in order to reduce the use of memory and processing during reassembly process. This approach allows the classifier monitor reach a response time of $(30 + \alpha)$ seconds, where α is the necessary time to performs the reassembly of the captured data in a given interval. In summary, the monitor works with a *quantum* of 30s of traffic and with an average delay of α seconds in the flow reassembly, feature extraction, and classification. The average found value achieved in the current implementation was $\alpha = 0.49$ seconds.

The Figure 1 exhibits the capturing and processing environment of the monitoring and classification system. Basically, we assume that the traffic is mirrored by a network border router to a network interface monitored by the system. The system periodically performs the processing and categorization of captured data, and presents the obtained information from monitoring and classification process. The Figure 2 exhibits the layered structure of implemented classifier monitor, whose tasks modules are online traffic collection from a network point, preprocessing for flow reassembly, extraction and selection of statistical attributes, flow labeling since payload analysis or port-based method (only during training step), training with a supervised machine learning technique and classification, using the ML model built from training data.

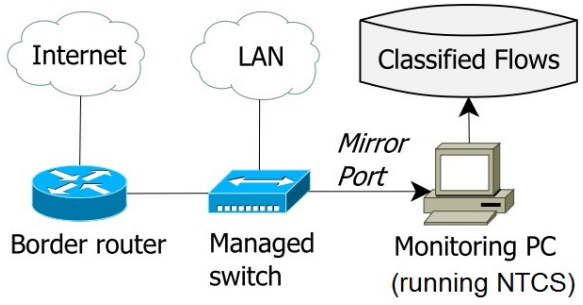

Fig. 1. Traffic Capture Environment .

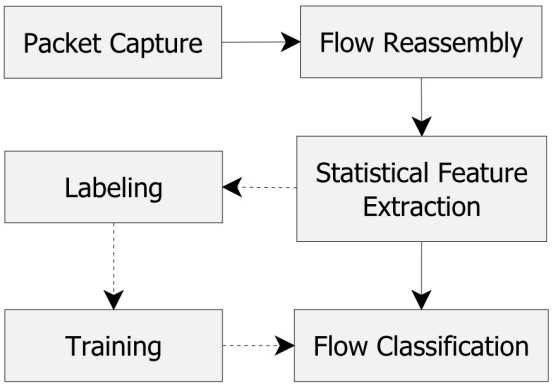

Fig. 2. Block Diagram of Classifier Monitor.

B. Flow Reassembly

Recently-accessed-first Principle: In a high-speed network, there may be hundreds of thousands of simultaneous connections, so that the reassembly system, which maintains a structure for storing connection records, searches in this structure the corresponding record for each collected packet. The search becomes expensive and needs to be optimized as the number of connections increases [2]. The idea of recently-accessed-first principle is to bring the most recently accessed connection records to top of connection record list.

This work uses the same concept of TCP stream presented in [5], and the recently-access-first principle presented in [2] to optimize the reassembly step of our software-based solution. Differently of [2], which uses two hash tables for connection management, we use a simple list to store connections. The adopted reassembly policy was based on mechanism proposed in [3] for TCP session reassembly. The applied reassembly approach was validated experimentally with the *Tcptrace* and *Tcpflow* tools.

C. Flow Labeling

Labeling is a necessary step for training and subsequent evaluation of classifiers. Although the utilization of port-based method [6] to traffic flow labeling can introduce errors due its increasing ineffectiveness since flows can be incorrectly labeled, the existence of some inaccurate values in data sets is a common machine learning problem, and a good ML scheme must have the capability to deal with this situation [7]. Although this labeling method was implemented in the first prototyping of NTCS, other sophisticated labeling techniques can be subsequently incorporated.

D. Traffic Classification

The presented approach uses real traces in the evaluation of Naive Bayes [8], Kernel Naive Bayes [9], C4.5 Decision Tree [10] and K-Nearest Neighbor [11] methods for Internet traffic classification using statistical information derived from packet headers. The Weka [12] (Waikato Environment for Knowledge Analysis) is an Open Source tool implemented in JAVA, and contains a collection of machine learning algorithms for data mining problems. The NTCS utilizes the Weka libraries for the training and evaluation of the machine learning methods. We utilize the IKVM [13] software to enable Java and .NET interoperability. This tool allows to generate the Weka *dlls*, which were imported to C# code. Thus, it is possible to use the weka classifiers in the classification module of our classifier monitor.

IV. Methodology

This section starts with a presentation of the network traffic data used to evaluate the NTCS, and finishes with the description of the statistical features.

TABLE I
CHARACTERISTICS OF THE TRACE T_1

Characteristic	$\mathbf{T_1}$	$\mathbf{T_2}$
Packets Number	614282	1579921
Capture Size	565.62MB	1.88GB
Capture Duration	3516.79s	1355.83s
Average packet Size	920.78 Bytes	1195.16 Bytes
Average capture Rate	1.28 Mbps	11.14 Mbps

A. Data Collection and Experiments

The performance of packet capture and reassembly modules was evaluated for capacity verification, under variable load conditions. The classifier monitor was performed with a Core i5 computer with CPU 2.30 GHz and 4GB of memory. The trace-driven simulation allows flexibility in the evaluation of distinct classifiers and reassembly approaches. This is because the different executions of our system for the same packets trace always generate the same flows set. Without this determinism, it would be extremely difficult to reproduce the same results of an online packet capture, given the possibility of delay and packet loss, for example. For confidently evaluate the online monitor, we used traffic traces collected in a host connected to the broadband Ethernet 100Mbps. Each flow in a reassembly process is configured with a 60 seconds timeout, in order to avoid the storage of old or idle connections, which consume memory and processing resources unnecessarily. This means that TCP flows whose duration is greater than this value are finished by the collector process periodically. We compare the time complexities and the number of recontructed flows of our flow reassebly module and the external tools Tcptrace, TcpFlow, TcpRecon and Wireshark. Using Weka resources, we use 10-fold cross validation for accuracy evaluation of the aforementioned classification models. The used traffic traces characteristics, referred as T_1 and T_2, reflect the communication between a host at State University of Ceará (UECE) and the Internet.

The identified application categories in the current traces were: *Www* (*World Wide Web*), *Https* (*Http protocol over TLS/SSL*), *Ftp* (*File Transfer Protocol*), *Xvttp* (*Xvttp Protocol*) and *Isakmp* (*Isakmp Protocol*). The most representative categories in T_1 traffic trace are Www and Ftp applications. In T_2 traffic trace, the Https and Isakmp application have a greater number of flow instances. In our study, the classification and training steps are performed at the end of packet capture simulation and reassembly. This methodology is necessary to evaluate the modules of our classifier monitor.

B. Statistical Features

In order to evaluate the classification process, we considerate the following features: *elapsed time between the first and last packets*, *number of packets*, *number of bytes*, *the number of all packets with at least a byte of TCP data payload*, *the number of all packets seen with the PUSH bit set in the TCP header*, and *the median and the variance of the number of bytes in IP packet*. Since each attribute is computed for both directions of flow (uplink and downlink), each flow instance has 14 statistical discriminators plus the class label. There was not a proper selection of attributes in this study. We chose some of the most often found attributes in previous published work [1] which could be calculated from the data contained in the header of packets without examining their payload.

V. RESULTS AND DISCUSSION

Table II presents some performance metrics of our classifier monitor for the used traces. We observed that the highest achieved throughput for the packet capture and reassembly modules was 3.95 flows per second (fps) forT_1 traffic trace. This means that this is the number of traffic flows are delivered by reassembly process at every second. Although this is a low value, since the mean packet capture throughput of T_1 is only 1.28 Mbps, the reassembly process achieves a throughput of 24997.25 fps at one of packet capture intervals. The average packet capture and reassembly rate, expressed by $Mbits/(T_{CO} + T_{RE})$, was 1014.73 Mbps, where T_{CO} and T_{RE} are the total duration times of packet capture and reassembly, respectively Similarly, the same performance metrics are presented for T_2 traffic trace. We can observe no bottlenecks in the reassembly process, which could support the considerate traffic. Our software-based monitor is effective to work in real time for a corporate network, for example. The average delivery delay α for T_1 and T_2 traffic traces is 0.49s and 7.50s, respectively. This means that this is the average reassembly duration for ever quantum of 30s. We can observe that delivery delay varies greatly between the two compared traces. Although they are from the same packet capture point, the two trace files are essentially different. The throughput collection of the T_2 is much larger than T_1. Furthermore, the traffic load in T_2 is larger than T_1. This means that there are less traffic to process in T_1, and consequently, its delivery delay is lower than the other trace.

TABLE II
PERFORMANCE OF MONITORING AND REASSEMBLY PROCESSES

Metric	T_1	T_2
TCP Connections Number	2969	365
Max Capture & Reassembly Throughput	3.95 fps	2.89 fps
Max Reassembly Throughput	24997.25 fps	128.21 fps
Average Capture & Reassembly Rate	1014.73 Mbps	635.34 Mbps
Average Delivery Delay	0.49s	7.50s
Total Monitor Time	75.67s	310.84s

In Table III, our system is compared with the Tcptrace, TcpFlow, TcpRecon and Wireshark tools. The TcpRecon was modified to use a flow timeout of 60 seconds. We can observe that the number of flows is not the same between the tools, because the divergence of the used traffic flow concept, as explained previously. We can observe that our adopted reassembly approach execution time is lower than the other tools. Our reassembly scheme was implemented in TCP session Reconstruction Tool, replacing the TcpRecon default

policy. In summary, the difference between these two policies is the adopted recently-accessed-first principle and the use of different data structures to hold established and not established TCP connections.

TABLE III
COMPARISON WITH EXTERNAL TOOLS

Approach/Tool	Flows Number	Reassembly Time
Proposed Approach	2969	75.67s
TcpFlow	5894	118.87s
Tcptrace	3044	612.95s
Wireshark	3036	182.69s
TcpRecon	3036	96.97s

Since TcpRecon and our proposed scheme are written in same language and uses the same packet capture libraries, we also compare the performance of these two policies one of each other. The confidence interval estimation of an event population will have greater reliability if the event is executed at least 30 times [14]. We executed and measured the elapsed times of the aforementioned TCP reassembly policies. The policies were evaluated over the already presented datasets. We computed the average execution time and confidence level for each TCP policy. We consider a high confidence level of 95%. The resulting confidence levels for TcpRecon and our adopted reassembly scheme are presented in Table IV. For the T_1 traffic trace, our scheme obtain a time complexity advantage of 20.31 seconds. For the T_2 traffic trace, there is also a reduction of 9.39 seconds with our approach.

TABLE IV
PERFORMANCE COMPARISON OF REASSEMBLY POLICIES

Traffic Trace	TcpRecon	Proposed Policy
T_1	91.85 ± 5.61 seconds	71.54 ± 3.57 seconds
T_2	380.36 ± 14.82 seconds	370.97 ± 7.72 seconds

The Table V presents the main results about the classification process. We can observe that C4.5 Decision Tree was able to categorize on average 87.40% and 89.86% of the traffic correctly for the two traffic traces. The AdaBoost ensemble algorithm, using the DecisionStump classifier, was able to categorize on average 78.17% and 89.58% of the traffic correctly. The KNN technique, with $k = 10$, was able to categorize on average 86.25% and 91.50% of the traffic correctly, against 72.48% and 80.00% for NB classifier. The duration of classification phase was a few seconds, and the results aim to validate the previous phases of the classifier monitor.

VI. CONCLUSION

This paper presented the architecture, implementation, and performance of an Internet traffic classifier monitor. The monitor is composed of three modules which were implemented as concurrent processes: capture and preprocessing, flow reassembly, and classification. For the T_1 traffic trace, the throughput reassembly module of the current implementation is 24997.25 flows per second. The average delivery delay is 0.49 seconds. For the classification module, the C4.5 algorithm outperforms KNN and AdaBoost classifiers with average accuracy of 87.40% and 89.86% against 72.48% and 80% for the KNN and AdaBoost methods, respectively.

TABLE V
GLOBAL ACCURACY PER TRACE

Classifier	T_1	T_2
C4.5 Decision Tree	87.40%	89.86%
AdaBoost(*DecisionStump*)	78.17%	89.58%
K-Nearest Neighbor	86.25%	91.50%
Naive Bayes	72.48%	80.00%
Flexible Naive Bayes	64.09%	88.76%

Future directions for this research includes to incorporate subflow based classification in NTCS to reduce response time. Second, we aim to verify the performance impact of our classifier monitor at gigabit links, which are becoming increasingly common at computer networks.

REFERENCES

[1] A. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. ACM, 2005, p. 60.

[2] B. XIONG, C. Xiao-su, and C. Ning, "A Real-Time TCP Stream Reassembly Mechanism in High-Speed Network," *JOURNAL OF SOUTHWEST JIAOTONG UNIVERSITY*, vol. 17, no. 3, 2009.

[3] P. Agarwal, "TCP Stream Reassembly and Web based GUI for Sachet IDS," Master's thesis, Indian Institute of Technology Kanpur, Kanpur, India, 2007.

[4] L. Jun, Z. Shunyi, L. Yanqing, and Z. Zailong, "Internet traffic classification using machine learning," in *Second International Conference on Communications and Networking in China, 2007. CHINACOM'07*, 2007, pp. 239–243.

[5] G. Wagener, A. Dulaunoy, and T. Engel, "Towards an estimation of the accuracy of tcp reassembly in network forensics," in *Future Generation Communication and Networking, 2008. FGCN'08. Second International Conference on*, vol. 2. IEEE, 2008, pp. 273–278.

[6] IANA. (2014, May) Internet assigned numbers authority. [Online]. Available: http:/www.iana.org

[7] Y. Wang and S. Yu, "Machine Learned Real-Time Traffic Classifiers," in *Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on*, vol. 3. IEEE, 2009, pp. 449–454.

[8] D. Zuev and A. Moore, "Traffic classification using a statistical approach," *Passive and Active Network Measurement*, pp. 321–324, 2005.

[9] G. John and P. Langley, "Estimating continuous distributions in Bayesian classifiers," in *Proceedings of the eleventh conference on uncertainty in artificial intelligence*, vol. 1. Citeseer, 1995, pp. 338–345.

[10] T. Korting, "C4. 5 algorithm and multivariate decision trees," *Image Processing Division, National Institute for Space Research–INPE Sao Jose dos Campos–SP, Brazil*.

[11] M. J. Islam, Q. M. J. Wu, M. Ahmadi, and M. A. Sid-Ahmed, "Investigating the performance of naive- bayes classifiers and k- nearest neighbor classifiers," *Convergence Information Technology, International Conference on*, vol. 0, pp. 1541–1546, 2007.

[12] E. Frank, M. Hall, and L. Trigg, "Weka 3-Data Mining with Open Source Machine Learning Software in Java," *The University of Waikato*, 2000.

[13] J. Frijters, "Ikvm, an implementation of java for mono and the .net framework [computer software and documentation]," 2004.

[14] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley - Interscience, ISBN:0471503361., New York, NY, April, 1991.