An Effective Similarity Metric for Application Traffic Classification

Jae Yoon Chung, Byungchul Park, Young J. Won
Dept. of Computer Science and Engineering, POSTECH
Pohang, Korea
{dejavu94, fates, yjwon}
@postech.ac.kr

John Strassner
Division of IT Convergence Engineering, POSTECH
Pohang, Korea
johns@postech.ac.kr

James W. Hong
Dept. of Computer Science and Engineering, POSTECH
Division of IT Convergence Engineering, POSTECH
Pohang, Korea
jwkhong@postech.ac.kr

Abstract—**Application level traffic classification is one of the major issues in network monitoring and traffic engineering. In our previous study, we proposed a new traffic classification method that utilizes a flow similarity function based on Cosine Similarity. This paper compares the classification accuracy of three similarity metrics, Jaccard Similarity, Cosine Similarity, and Gaussian Radius Based Function, to select appropriate similarity metrics for application traffic classification. This paper also defines a new two-stage traffic classification algorithm that can guarantee high classification accuracy even under an asymmetric routing environment, with reasonable complexity.**

Keywords—Application level traffic classification, traffic monitoring, similarity function

I. Introduction

Accurate classification of Internet traffic is one of the major issues in network management. The classification statistics can be utilized for various objectives, such as network planning, QoS management, traffic engineering, and security.

The Internet evolves continuously and rapidly in terms of complexity, speed, and its application trends. Many new types of peer-to-peer (P2P) and multimedia applications have emerged on the Internet, and the traffic characteristics of these applications are very different from traditional network application traffic. Many applications are incorporating various strategies, such as data encryption, random port allocation, and adopting proprietary protocols to avoid detection and filtering. The dynamic nature of Internet traffic adversely affects the accuracy of traffic classification.

In our previous work [1], we proposed a new traffic classification method that utilizes a flow similarity function based on Cosine Similarity. Our work was inspired by document classification, which is widely used in Natural Language Processing (NLP) [14]. We proved that the flow similarity approach has good performance in terms of accuracy.

In this paper, we compared three different similarity metrics, Jaccard Similarity, Cosine Similarity, and Gaussian Radius Based Function (Euclidean distance), as an extension of our previous work. We mainly focus on the accuracy and complexity of each similarity metric, and we analyze characteristics of each metric to provide guideline for selecting similarity metrics for traffic classification. We also define a two-stage classification algorithm for application level, as opposed to protocol level, traffic classification. This algorithm solves a classification problem that occurs when the routing path is asymmetric.

II. Related Work

Port-based traffic classification is used to filter out specific target traffic, such as instant messenger and P2P applications, because it is the simplest method of filtering. However, today's internet applications use dynamic port allocation and port masquerading techniques to avoid port filtering, and the accuracy of port-based traffic classification, even those using well-known ports, is less than 70% [2]. For this reason, alternative methodologies for traffic classification have been developed. We summarize these previous studies into three categories: payload-based, behavior-based, and machine learning based.

Payload-based classification approaches [3][4][5][6][7] inspect packet payloads to find targeted application signatures. This approach assumes that most application traffic contains signaling packets, and the signatures in signaling packets are unique. By identifying these signatures, we can classify which flow is generated by which application. Payload-based approaches show reliable accuracy if the signatures are exactly and uniquely extracted. However, payload-based approaches require exhaustive signature generation, and cannot be used with encrypted packets. In our previous study [1], we converted payloads into a vector representation, and then calculated the similarity between payload vectors to classify them. This approach does not require complex processing.

Behavior-based traffic classification approaches [8][9] have been developed that analyze host interactions to distinguish application types. These approaches focused on the connection patterns used by different applications (e.g., P2P and HTTP). For example, BLINC profiles host behavior using destination IP addresses and port allocation patterns. However, the accuracy of behavior-based classification is still questionable. In addition, behavior-based classifications cannot be applied to application level traffic classification because most applications cannot be distinguished using only host connection patterns.

Recent studies apply machine learning (ML) algorithms to classify network traffic [10][11]. Most ML-based approaches use statistical ML algorithms that require training before they can be used. However, once trained, most ML-based approaches provide high classification accuracy. However, ML-based classification cannot distinguish application level traffic, due to the same problems that the behavior-based approach has. In addition, this approach typically has high CPU overhead and requires a large amount of memory.

Therefore, we focus on payload-based classification, especially payload similarity calculation, because packet payloads can reflect differences between application traffic. Packet payloads contain detailed information that makes it possible to classify application traffic. Our method does not need deep level packet inspection or exhaustive signature extraction. In addition, its complexity is also reasonable compared to other payload-based approaches.

III. METHODOLOGY

In the previous work [1], we defined the *word* of a payload as follows.

- **Definition 1:** We define *word* as payload data within an *i*-bytes sliding window, where the position of the sliding window can be 1, 2 … *n*-*i*+1 for an n bit-length payload. The size of whole representative words is 2^{8*i}, and the length of the word is *i*.

If the word length *i* is too short, the word cannot reflect the sequence of the byte patterns in the payload. In this case, we cannot recognize the differences among permutations of byte patterns, such as '0x01 0x02 0x03' and '0x03 0x01 0x02'. If the word length is too long, the number of whole representative words increases exponentially.

We could determine the term-frequency vector [13] by counting words within a payload. This vector is called the *payload vector*, which is the same as the document vector in Natural Language Processing research.

- **Definition 2:** When w_i is the count of the i-th word that appears repeatedly in a payload, the *payload vector* is

$$\textit{Payload Vector} = [w_1\, w_2\, \dots\, w_n]^T. \quad (1)$$

where *n* is the size of whole representative words.

We set the sliding window size (*word* size) to 2-byte because it is the simplest case for representing the order of content in payloads. When the word size is 2byte, the size of all representative words is 2^{16}=65536. Thus, the payload vector has 65536 dimensions.

In the remainder of this section, we present three similarity metrics: Jaccard Similarity; Cosine Similarity; and Gaussian radial basis function (RBF). We also define application vectors and how two extract the application vectors. Finally, we explain an algorithm of application level traffic classification.

A. Similarity Metrics

We compared payload vectors using three similarity matrices: Jaccard Similarity, Cosine Similarity, and Gaussian radial basis function. The values of all of these metrics range from 0 (two items being compared, X and Y, are completely independent) to 1 (X and Y are identical).

Jaccard Similarity $\mathrm{J(X,Y)}$ uses word sets from the comparison instances to evaluate similarity. J(X, Y) is defined as the size of the intersection of the word sets divided by the size of the union of the sample sets X and Y:

$$\mathrm{J(X,Y)} = \frac{|\mathrm{X} \cap \mathrm{Y}|}{|\mathrm{X} \cup \mathrm{Y}|}. \quad (2)$$

Cosine Similarity $\mathrm{C(X,Y)}$ measures the similarity between two vectors X and Y of n dimensions by fining the cosine angle between them:

$$\mathrm{C(X,Y)} = \frac{\mathrm{X} \cdot \mathrm{Y}}{|\mathrm{X}||\mathrm{Y}|}. \quad (3)$$

The Gaussian radial basis function $\mathrm{RBF(X,Y)}$ represents similarity between two vectors X and Y as:

$$\mathrm{RBF(X,Y)} = \exp(-\beta\|\mathrm{X}-\mathrm{Y}\|^2), \quad (4)$$

where $\beta > 0$.

To fairly compare the metrics, we do not use any sophisticated techniques, such as weighting functions, since these techniques are not applicable to all similarity functions.

B. Application Vector Heuristics

The vectors that represent typical packets of applications are essential, because the center (basis) of each cluster is needed for application level classification. We define the term *application vectors* to represent packets that are generated by target applications. The ideal condition of the application vector is different for each similarity measure. For Jaccard Similarity, the number of common words among application vectors should be small. In the case of Cosine Similarity, the ideal condition is that the vectors are orthogonal. In the RBF case, the distance between the application vectors should be large.

In addition, after grouping the flows according to similarity, the result is better if the number of groups is small and the number of flows within a group is large. This reflects the case where the application generates flows that contain signatures that are easily recognized, and the flows are quite similar.

To extract application vectors automatically, we developed an application vector generator. Algorithm 1 shows the pseudocode for extracting application vectors. The application vector generator reads packets from the target application trace and divides the packets into several types of clusters without any pre-processing. Within the cluster, packets are converted into payload vectors and merged.

Because the first few packets (signal packets) have very distinctive characteristics, or signatures, that distinguish each flow [12], the application vector generator needs to only consider these signal packets to extract application vectors. To identify signal packets efficiently, a *registered flow table* is used. This is a hash table that uses the 5-tuple (source address, source port, destination address, destination port, protocol) as a key. By matching a packet to its corresponding entry in the registered flow table, we can count the number of packets per flow. More importantly, we can identify signal packets, and ignore meaningless data packets. This dramatically decreases the processing overhead to extract application vectors.

For example, if the application vector generator reads a packet from a traffic trace that is generated by BitTorrent, the generator converts the packet into a payload vector and compares the payload vector with application vectors that have been previously extracted. If the payload vector is not similar to any pre-extracted application vectors, the generator adds the payload vector into the application vector list, and the vector added would be a basis for a cluster. If the payload vector is similar to one of the pre-extracted application vectors, the generator updates the center (basis) of the cluster by merging both the payload vector and the pre-extracted application vector. According to the similarity, the application vector contains a priority attribute as metadata that helps flow level classification (this will be explained in the next subsection). After doing all of the above steps, the signal packets of an application's packets are classified into clusters, which represent different type of flows. Finally, a small number of application vectors emerge as dominant flows, and the application can now be classified with high confidence.

We extract the application vectors of each target application using three similarity metrics: Jaccard Similariry, Cosine Similarity, and RBF. The target applications are as follows; FileGuri [17], ClubBox [18], Melon [19], BigFile [20], eMule [21], and BitTorrent [22]. The target application is selected according to user popularity of our campus.

Algorithm 1. Extract Application Vectors

```
procedure GetAppVectors(application_vectors)
//Initial value of application_vectors is empty

while read_packet(pkt) is available
   for av in application_vectors
      sim = similarity(payload2vector(pkt), av.vector)
      if is_similar(sim)
         if is_signal_pkt(pkt)
            //registered flow table look up
            merge(av, pkt)
         end if
      else
         if is_signal_pkt(pkt)
            //registered flow table look up
            application_vectors.append(pkt)
         end if
      end if
end while
end procedure
```

C. *Two-stage Traffic Classification*

Fig. 1 shows the pseudocode for our application traffic classification using similarity metrics. The proposed algorithm consists of two stages; the first stage is packet level clustering, and the second stage is flow level classification.

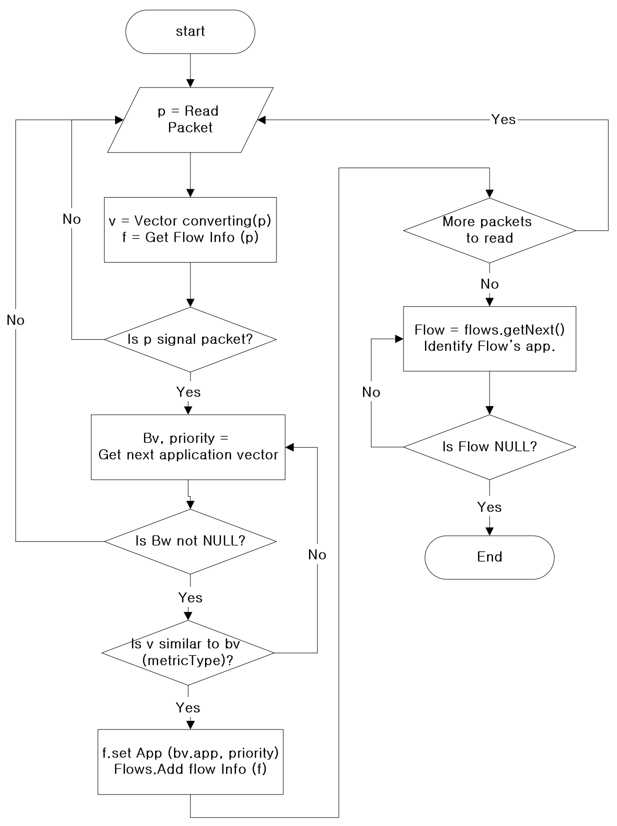


Fig. 1. Flow chart of application traffic classification using similarity metrics.

- In the first stage, we convert packets that are involved in the target traffic into payload vectors, and the flow information is extracted from the packets. By looking up the number of packets in the flow using the *registered flow table*, we can determine whether packet *p* is a signal packet or a data packet. We only need to consider first five packets per flow [3], which reduces the processing overhead. If a packet *p* is a signal packet, we calculate the similarity value (Jaccard, Cosine, RBF) between *p* and known application vectors. We also obtain the priority of the cluster. Except for registered table look up, we classify packets without any consideration about flow information, which need not keep the order of packets in flows. In other words, packet level clustering is not affected by the permutation of packets and small packet losses.

- In the second stage, we rearrange the packets according to flows using the cluster id, similarity value, and priority. We merge the packet level clustering information that is obtained from the first stage and perform flow level classification. In this step, mis-clustered packets that are caused by protocol ambiguities, such as HTTP for Web and HTTP for P2P, are ignored by more distinctly classified packets whose priority should be higher than that of the ambiguous packets. We divide the priority into four classes; Highest, Higher, Lower, and Lowest. In a flow, for example, if two packets are classified into BitTorrent in Highest priority class and three packets are classified into FileGuri in Higher priority class, the overall flow is classified into a BitTorrent flow. In the other words, the clustering result of the lower priority class cannot dominate the clustering result of the higher priority class. The only exception is when a flow is not identified using one class priority clustering. For example, if two packets are both in Highest priority class, and one is BitTorrent and another is FileGuri, we can refer to the classification result of the lower priority class to correct the ambiguous higher priority classification.

Packet level clustering result in the first stage is a function of each packet in order to allow for variations among the same kinds of flows. Flow level clustering in the second stage collects the scattered clustering information and extracts the complete classification result.

In the previous study, the flow similarity scoring and comparison approach has difficulties in the asymmetric routing case. In the flow similarity scoring approach, packets are compared sequentially with only the corresponding packet in the other flow, which is not suitable to asymmetric routing. However, the approach described in this paper can be applied in asymmetric routing cases because our approach does not collect all results of signal packets in flows, but instead uses the one or two most likely packet classification results. In the asymmetric routing case, because of the limited condition for collecting all packets in flows, our algorithm does not need to collect all packets, which also saves on storage and processing time.

IV. Evaluation

To validate the proposed approach, we classified some of our campus network traffic traces. The traces were collected from one of the two Internet junctions at POSTECH, a university with a user population of about 4000. To avoid packet loss, a monitoring probe equipped with an optical tap, DAG 4.3GE card [16], was used to monitor a 1-Gbps Ethernet link.

A. *Classification Accuracy*

We compared the classification results that are obtained using three similarity metrics. TABLE I shows the classification results including the traffic size and the accuracy. We categorized target applications into *fixed-port applications* (FileGuri, ClubBox, Melon, BigFile) and *untraceable-port applications* (eMule, BitTorrent). Because of the size of captured traffic, comparing classified results by hand is impossible. We decided to extract exact flows of the target application using active port numbers as much as possible. To extract *Fixed-port applications* traffic from the backbone trace, which is used as a ground-truth, the first thing we have to perform is monitoring the active port numbers of a target application at the backbone link. We compared fixed-ports of target applications with active flows, and selected heavy active flows as a ground-truth traffic. In *untraceable-port application* cases, however, selecting exact flows of target application by active port numbers monitoring is almost impossible because we do not know their port numbers. For these reasons, we had to use the Traffic Measurement Agent (TMA) to extract *untraceable-port applications* traffic. TMA is a Windows based client program and is deployed at target endpoints to determine the exact flow information at the origin of the traffic. The traffic, extracted using TMA's flow information, is considered as an arbitrary traffic of the target application, which is the same situation that we perform extracting traffic of the target application at the backbone link.

1) *Accuracy Analysis of Fixed-port Applications*

In TABLE I, FileGuri is one of the most popular Korean P2P file sharing services, and ClubBox and BigFile are web-hard services. Web-hard traffic is classified better than P2P traffic. P2P sends SYN or Hello packets to construct an overlay network. These small packets are not easy to classify, because most small packets are short-lived, and consist of only the TCP and IP header. Otherwise, web-hard services guarantee more stable connections between their server and user clients. Web-hard services required hand-shaking that is not interrupted when the services are provided.

Because the traffic of FileGuri is large, more than 2 GB, the application vector of FileGuri is extracted better than others, and the accuracy of classification result is also about 95%.

ClubBox is clustered with 100% accuracy except for Cosine Similarity. The reason for the low accuracy of Cosine Similarity is the flow size of ClubBox as well as the characteristics of Cosine Similarity. The web-hard file sharing traffic consists of a small number of flows whose volume is very large. For this reason, even though we missed only 2 flows whose sizes are 372 MB and 156 MB using Cosine Similarity, the classification accuracy is significantly decreased.

Melon is a Korean music streaming application. We can detect signal packets of Melon using payload similarity metrics with more than 99% accuracy (RBF is 94%).

2) *Accuracy Analysis of Untraceable-port Applications*

It is difficult to measure the exact accuracy of eMule and BitTorrent backbone traffic, since both do not use static ports. Using TMA, we collected about 2 GB of eMule traffic. The packets generated by eMule contain a very short signature that is not enough to distinguish packets using word appearance frequency. This is also shown in the signature-based classification approaches because of the short length of signatures. However, the classification accuracy of eMule traffic is about 50% without any exception handling. Because of the short length signature of eMule, non-signature data significantly affects the similarity calculation. The approximate 50% classification accuracy is due to the effect of these non-signature data. To generate a large quantity of eMule traffic, we downloaded more than five files in the searched result with a single query. For this reason, our eMule trace consisted of traffic with a limited number of nodes, and their session information in application payload, such as node hash value that should not be a signature, was extracted as a signature. These mis-extracted signatures on eMule traffic affect its traffic classification.

To classify BitTorrent traffic, we also captured BitTorrent traffic at an end host. The well-known signatures of BitTorrent, "BitTorrent protocol", portray a distinctive difference among payload vectors. Because of the clear signatures of BitTorrent traffic, the classification accuracy is almost 100.00%, except for the RBF metric. RBF is affected by the distance of non-signatures, but Jaccard Similarity just counts the fragments of signature and Cosine Similarity also is overwhelmed by a small number of signatures. The transferred data in some BitTorrent flows are completely different, which shows large distance and critically affects the low accuracy of RBF.

We also classified BitTorrent traffic on the backbone network, and the Cosine Similarity metric suffers from false positive classification (Fig. 2). The traffic volume of BitTorrent is the highest with Cosine Similarity, while those of Jaccard Similarity and RBF are almost the same. Comparing the traffic volume of Cosine Similarity with the traffic volume of both Jaccard Similarity and RBF, the classification result of Cosine Similarity is the worst because of its high false positive rate. The reason for false positives is because Cosine Similarity is very sensitive to the value on each axis. For example, let's consider two different types of vectors, v1 = (100, 1, 2) and v2 = (1, 0, 0). Because of high value on first axis, the dot product of v1 and v2 is almost the same as v1. This means that while v1 and v2 are very different, their Cosine Similarity is almost 1 (i.e., identical) when one of v1 and v2 has a much higher value on the same axis. If an application vector has a high frequency of a certain word w, a packet that contains only one word w is classified to the same cluster with the application vector.

Surprisingly, Cosine Similarity shows the worst accuracy (TABLE I). Most Natural Language Processing (NLP) approaches use Cosine Similarity for document classification. In NLP, the most important thing is *key* words, which implies words in documents are written very intentionally. If we excluded *stop words* in the documents, Cosine Similarity would show more accurate classification results. In the payloads, however, filtering out *stop words* is much more difficult because in many cases, some byte patterns are used as both signatures and garbage data (e.g. 0x00 is used to fill unused space in the payload and also frequently used as a hand shake message or a test message). In addition, if we remove signatures in the payload, the data is almost random. For this reason, some arbitrary word acts as a signature, which significantly reduces the accuracy of the similarity calculation.

Jaccard Similarity shows the best accuracy, and can also deliver relatively good results for *unknown traffic*. In our payload vector, which is a kind of term-frequency vector, the signature is broken into small fragments. Eventually, Jaccard Similarity counts the number of signature fragments. However, Jaccard Similarity is too low when both packets contain the same signature because the proportion of signature is still small.

Fig. 3 is the traffic volume histograms of three similarity metrics. Jaccard Similarity does not use its entire range, which is from 0 to 1. Cosine Similarity uses a large portion of its range, and has several peaks. RBF uses the largest portion of its range, but has smaller peaks, distributed from 0 to 1. Both Jaccard and Cosine Similarity almost follow a Gaussian distribution, while RBF is much more evenly distributed.

RBF seems to be distributed uniformly rather than Gaussian, which means the distance between vectors is almost the same. This is caused by the distribution of distance among vectors which is extremely scattered. Only 0.5% of the shortest distance, from 0 to 10000, is about 90% (Fig. 4b). The whole range of distance is 0 to 2130097. The tuning of RBF for payload similarity comparison using β (which is the only parameter of RBF) has difficulties in this type of distribution.

TABLE I
CLASSIFICATION ACCURACY COMPARISON.
FIXED-PORT APPLICATION: FILEGURI, CLUBBOX, MELON, BIGFILE.
UNTRACEABLE-PORT APPLICATION EMULE, BITTORRENT.

Application	Metric	Classified (MB)	Total (MB)	Accuracy
FileGuri	Jaccard	2069.644	2195.551	94.27%
	Cosine	2069.644		94.27%
	RBF	2067.915		94.19%
ClubBox	Jaccard	1073.425	1073.450	100.00%
	Cosine	544.029		50.68%
	RBF	1073.425		100.00%
Melon	Jaccard	178.454	178.487	99.98%
	Cosine	178.374		99.94%
	RBF	168.482		94.39%
BigFile	Jaccard	0.557	0.561	99.20%
	Cosine	0.537		95.59%
	RBF	0.558		99.39%
eMule	Jaccard	1164.291	2296.639	50.70%
	Cosine	1147.812		49.98%
	RBF	1136.559		49.49%
BitTorrent	Jaccard	834.607	834.632	100.00%
	Cosine	834.622		100.00%
	RBF	605.187		72.51%

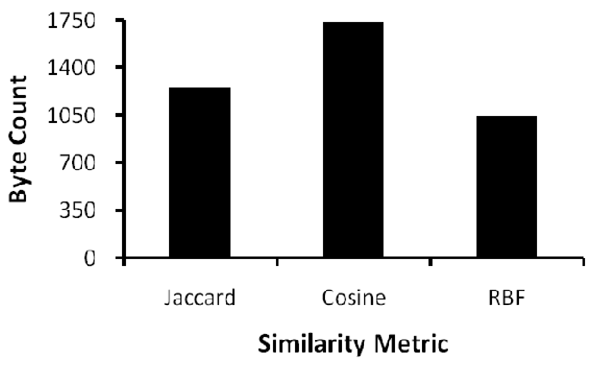
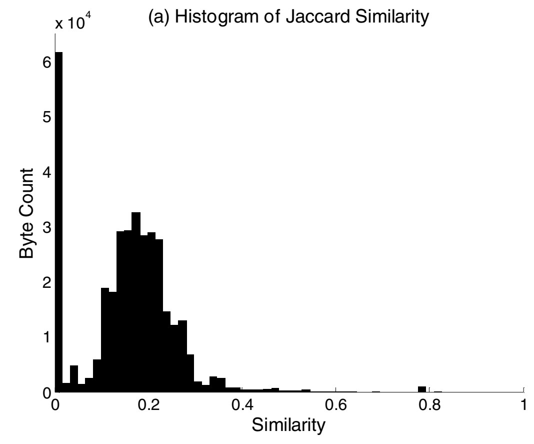
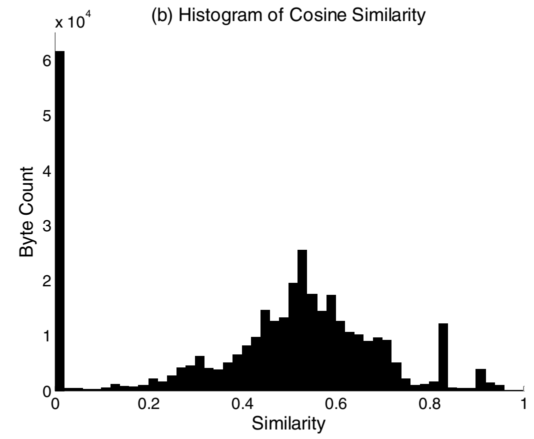


Fig. 2. Classified traffic of BitTorrent on backbone network with three similarity metrics: Jaccard, Cosine, and RBF.





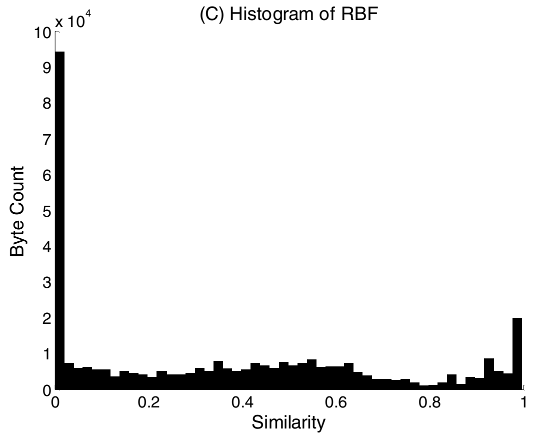
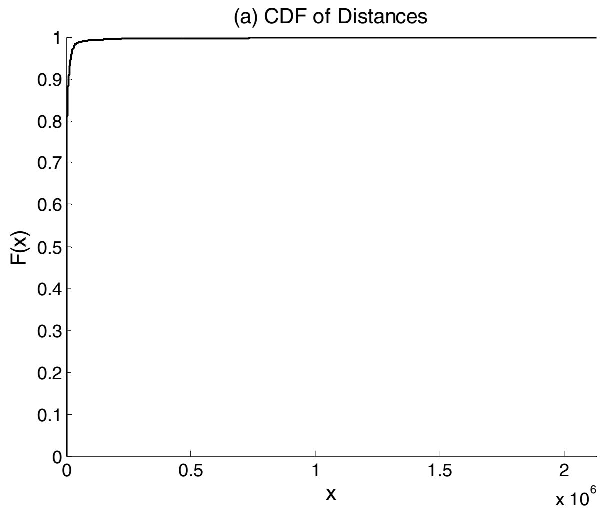


Fig. 3. Traffic volume histogram of three similarity metrics: Jaccard, Cosine, and RBF. (a) Jaccard Similarity: mean is 0.158, s.d. is 0.102; (b) Cosine Similarity: mean is 0.446, s.d. is 0. 0.249; and (c) RBF: mean is 0.369, s.d. is 0.3327.

B. Complexity

The time and space complexity are important parameters of the performance evaluation. We performed experiments with changing the number of packets and application vectors. The machine for experiments has 2.6 GHz single-core Intel processor, and 2 GB main memory.

The processing time linearly increases with the number of packets in the trace file (Fig. 5). The processing time is measured every 10,000 packets, and the number of application vectors is always 10. The time complexity is *O(n),* where *n* is the number of packets.



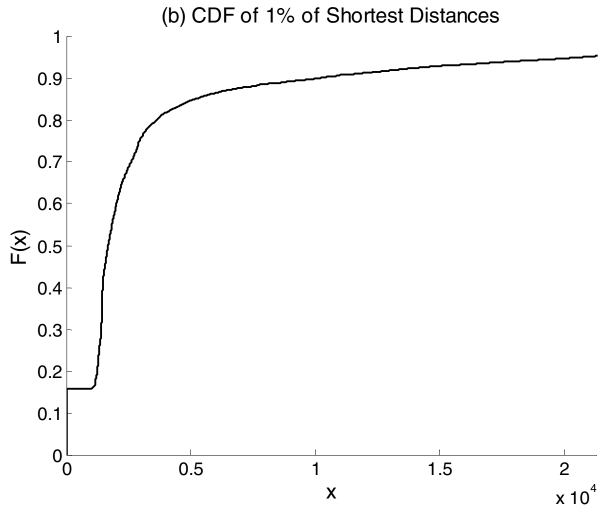


Fig. 4. (a) CDF of distances among vectors and (b) CDF of 1% of the shortest distances where *x* is distance. The maximum distance is 2130097, and distances less than 10000 is about 90%. The mean of distance is 9374.2 and s.d. is 88271.

Fig. 6 shows the processing time over the number of application vectors when the number of packets is 500,000. We also measured the time complexity *O(m),* where *m* is the number of application vectors, and found it to be the same as the complexity of the Boyer–Moore string search algorithm. This is the basis for the fastest known ways to find one string of characters in another [15]. Our method shows compatible complexity with that of signature-comparing-based approach, even though our implementation is not optimized for vector operations. The overall time complexity is *O(mn),* which is close to linear when *m* is much smaller than *n*. In most cases, the number of packets is much larger than the number of applications, even though countless applications exist.

Memory consumption is less than 300 MB for populating the *registered flow table* and packet clustering results while processing 60 GB of traces. Because our method does not process large matrices or compute complex transformations, other sophisticated methods for memory limitation handling are not essential to use our method.

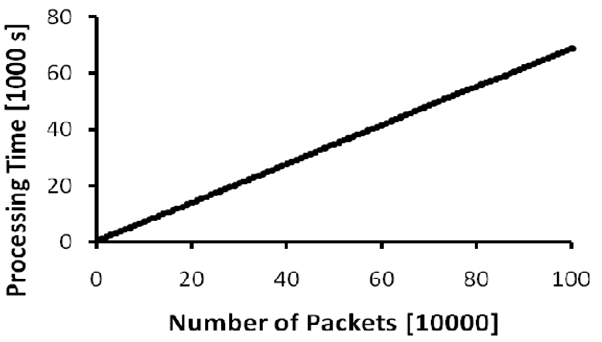


Fig. 5. Processing time over the number of packets with 10-application vector. The time complexity is O(n) where n is the number of packets

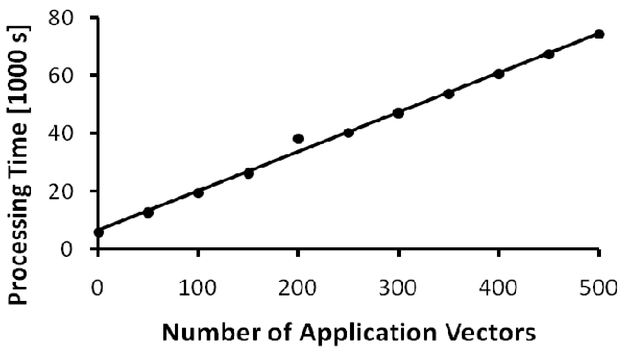


Fig. 6. Processing time over the number of application vectors with 500,000-packet. The time complexity is O(m) where m is the number of application vectors.

V. Conclusion and Future Work

In this paper, we classified application traffic using three similarity metrics: Jaccard Similarity, Cosine Similarity, and RBF. We also compared the classification accuracy and processing time of these three similarity metrics. Jaccard Similarity shows the best performance without using any sophisticated techniques. Jaccard Similarity can be considered as counting fragments of signatures in comparing the similarity between payloads. Cosine Similarity shows the worst performance because of the difficulties introduced by the lack of use of *stop words* filtering, and RBF (Euclidean distance) shows mixed results, mainly due to the difficulty in optimizing the value for the parameter β in its computation.

The proposed approach shows competitive results with signature-based classification in both accuracy and complexity aspects. The accuracy of our classification results is about 95% on P2P traffic, and more accurate on web-hard traffic regardless of asymmetric routing environment. In addition, our two-stage application traffic classification algorithm can be applied to asymmetric routing case, which is unsolved in our previous study. The time complexity of our algorithm is linear.

For future work, we plan to analyze the flexibility of our approach by comparing with machine learning based methods. We also plan to extract orthogonal vectors, which are traffic classifiers, from payload vectors to reduce ambiguity among the proposed application vectors.

Acknowledgements

This work was partly supported by the IT R&D program of MKE/IITA [2008-F-016-02, CASFI] and WCU program through the KSEF of MEST, Korea [R31-2008-000-10100-0].

References

[1] J.Y. Chung, B. Park, Y.J. Won, J. Strassner, and J.W. Hong, "Traffic Classification Based on Flow Similarity", 8th IEEE International Workshop on IP Operations & Management, Venice, Italy, Oct. 29-30, 2009.

[2] A.W. Moore, and K. Papagiannaki, "Toward the Accurate Identification of Network Applications", Passive and Active Measurement Conference, Boston, MA, USA, Mar. 31-Apr. 1, 2005.

[3] S. Sen, O. Spatscheck, and D. Wang, "Accurate, Scalable In-Network Identification of P2P Traffic using Application Signatures", International World Wide Web Conference, NY, USA, May 19-21, 2004, pp. 512-521.

[4] P. Haffner, S. Sen, and O. Spatscheck, "ACAS: Automated Construction of Application Signatures", ACM SIGCOMM 2005, Philadelphia, PA, USA, Aug. 21-26, 2005.

[5] B. Park, Y.J. Won, M.S. Kim, and J.W. Hong. "Towards Automated Application Signature Generation for Traffic Identification", IEEE/IFIP Network Operations and Management Symposium (NOMS 2008), Salvador, Brazil, Apr. 7-11, 2008, pp. 160-167.

[6] T.S. Choi, C.H. Kim, S. Yoon, J.S. Park, B.J. Lee, H.H. Kim, H.S. Chung, and T.S. Jeong, "Content-aware Internet Application Traffic Measurement and Analysis", IEEE/IFIP Network Operations and Management Symposium (NOMS 2004), Seoul, Korea, Apr. 23, 2004, vol. 1, pp. 511-524.

[7] F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus, "Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation", International Conference on Communications, Beijing, China, May 19-23, 2008.

[8] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel Traffic Classification in the Dark", ACM SIGCOMM 2005, Philadelphia, PA, USA, Aug. 21-26, 2005.

[9] T. Karagiannis, A. Broido, M. Faloutsos, and Kc claffy, "Transport Layer Identification of P2P Traffic", Internet Measurement Conference, Taormina, Sicily, Italy, Oct. 25-27, 2004, pp. 121-134.

[10] A.W. Moore and D. Zeuv, "Internet Traffic Classification Using Bayesian Analysis Techniques", International Conference on Measurements and Modeling of Computer Systems, Banff, Alberta, Canada, Jun. 6-10, 2005, pp. 50-60.

[11] J. Erman, M. Arlitt, and A. Mahanti, "Traffic Classification Using Clustering Algorithms", SIGCOMM Workshop on Mining Network Data, Pisa, Italy, Sep. 11-15, 2006, pp. 281-286.

[12] K.P. Gummadi, R.J. Dunn, S. Saroiu, S.D. Gribble, H.M. Levy, and J. Zahorjan, "Measurement, Modeling, and Analysis of a Peer-to-Peer Filesharing Workload", ACM Symposium on Operating Systems Review, Dec. 2003, vol. 27, pp. 314-329.

[13] G. Salton, A. Wong, and C.S. Yang, "A Vector Space Model for Automatic Indexing", Communications of the ACM, Nov. 1975, vol. 18, pp. 613-620.

[14] G. Salton and C. Buckley, "Term-weighting Approaches in Automatic Text Retrieval", Information Processing and Management, 1988, vol. 24, No. 5, pp. 513-523.

[15] R.S. Boyer and J.S. Moore, "A Fast String Searching Algorithm", Communications of the ACM, Oct. 1977, vol. 20, Issue 10, pp. 762-772.

[16] Endace, DAG 4.3GE, http://www.endace.com/.

[17] FileGuri, http://www.fileguri.com/.

[18] ClubBox, http://www.clubbox.co.kr/.

[19] Melon, http://www.melon.com/.

[20] BigFile, http://www.bigfile.co.kr.

[21] eMule, http://www.emule-project.net/.

[22] BitTorrent, http://www.bittorrent.com/.