INTERNATIONAL JOURNAL OF NETWORK MANAGEMENT
Int. J. Network Mgmt 2013; **23**: 350–381
Published online 30 August 2013 in Wiley Online Library (wileyonlinelibrary.com) DOI: 10.1002/nem.1837

Fine-grained traffic classification based on functional separation

Byungchul Park,[1] Youngjoon Won,[2,*†] JaeYoon Chung,[3] Myung-sup Kim[4] and James Won-Ki Hong[3]

[1]*INRIA, Sophia Antipolis, France*
[2]*Hanyang University, Seoul, Korea*
[3]*POSTECH, Pohang, Korea*
[4]*Korea University, Jochiwon, Korea*

SUMMARY

Current efforts to classify Internet traffic highlight accuracy. Previous studies have focused on the detection of major applications such as P2P and streaming applications. However, these applications can generate various types of traffic which are often considered as minor and ignorant traffic portions. As network applications become more complex, the price paid for not concentrating on minor traffic classes is in reduction of accuracy and completeness. In this context, we propose a fine-grained traffic classification scheme and its detailed method, called functional separation. Our proposal can detect, according to functionalities, different types of traffic generated by a single application and should increase completeness by reducing the amount of undetected traffic. We verify our method with real-world traffic. Our performance comparison against existing DPI-based classification frameworks shows that the fine-grained classification scheme achieves consistently higher accuracyand completeness. Copyright © 2013 John Wiley & Sons, Ltd.

Received 29 April 2012; Revised 24 April 2013; Accepted 19 June 2013

1. INTRODUCTION

Network traffic classification is an essential measure for understanding network status. Its results are widely used for network management purposes including network planning, usage reporting and customer charging. It plays an important role in user behavior analysis, a popular research topic for academia and industry in recent years. Traffic classification can benefit service providers to comprehend customer behavior so that greater satisfaction can be achieved through the provision ofpersonalized services.

In the early days of the Internet, traffic classification was facilitated by the understanding of prevailing network protocols, which in turn relied on well-known TCP/UDP port numbers. This port-mapping strategy can no longer ensure accuracy as new traffic characteristics such as unfamiliar traffic composition, volume and application trends continue to evolve [1].

Today, the critical objective for Internet traffic classification is to achieve a high level of accuracy and completeness. It is desirable to maximize both of these factors, and many variants have been introduced to improve accuracy, completeness and efficiency. However, a lack of ground truth dataset can raise accuracy questions. Many studies have aimed at capturing different levels of classification, with some having a coarse goal (e.g. classifying traffic protocol/application type) and others having a more detailed goal (e.g. identifying exact application names) [2]. It is often unfair to cross-compare different classification methods in terms of accuracy. To address these issues, it is important to investigate how we can provide meaningfulinformation based on limited classification results, rather than struggle to improve classification accuracy by a small factor.

*Correspondence to: Youngjoon Won, Department of Information System, Hanyang University, Seoul, Korea.
†E-mail: youngjoon@hanyang.ac.kr

In this paper, we propose a fine-grained traffic classification scheme using functional separation. This reduces the amount of undetected traffic and increases the completeness of traffic classification correspondingly. Functional separation helps to identify different types of traffic generated by a single application according to functionalities. The classification filters are then extracted from the respective traffic groups in preparation for fine-grained traffic classification, which further classifies diverse traffic groups arising from the functional separation.

The paper is organized as follows. Section 2 presents previous research on traffic classification. Section 3 explains the proposed fine-grained traffic classification scheme. Section 4 describes the functional separation process. Section 5 presents the evaluation results undertaken using a real-world traffic dataset. Finally, Section 6 provides concluding remarks and possible future work. The Appendix explains the validity of unsupervised Machine-learning algorithms with respect to fine-grained traffic classification.

2. RELATED WORK

In this section, we describe various traffic classification approaches and categorize them according to their levels of classification requirement and analysis capability.

2.1. Application traffic classification approaches

2.1.1. Port-based approach

Most traditional traffic classification methods rely on the inspection of transport layer port numbers. Traffic application can often be inferred from a packet's target port number, as registered in the Internet Assigned Number Authority (IANA) port list [3].

The port-based approach is limited in terms of classification accuracy, as many applications use port numbers not registered in the IANA port lists, and many P2P applications allocate multiple port numbers dynamically. Consequently, it can be difficult to match a certain port number with an application. Moore *et al*. [1] assert that the accuracy of port-based identification is not higher than 50–70%.

Nevertheless, the port-based approach is still a popular solution for classifying traffic within the Internet backbone, owing to high traffic volume and limited computing resources for traffic classification. Moreover, owing to its simplicity, the port-based approach is efficient in identifying general trends of application usage.

2.1.2. Payload-based approach

This approach was proposed to resolve uncertainties not addressed by the port-based classification approach [4,5]. Theoretically, a complete protocol parsing would be the most accurate way to classify traffic. However, such protocol parsing may not be viable, as many applications use proprietary protocols to avoid public disclosure, and some applications also incorporate well-known protocols into their application layer. Protocol parsing is also complex, as it requires a great amount of computing resources and is not suitable for real-time analysis for backbone.

Once a set of unique payload signatures is available for an application, the payload-based approach can produce extremely accurate classification. Signatures can be manually extracted by network administrators or security experts. This extraction processmust be preceded by protocol semantic analysis or by empirical pattern recognition inspection of packet payloads. Sen *et al*. [5] generated the signatures of a few P2P applications by analyzing their application layer protocols; using such analysis, they were able to reduce false positives (FPs) and false negatives (FNs) to 5% of the total bytes. As it is also important to find signatures that can feasibly deliver acceptable accuracy given traffic in asymmetric routing environments (e.g.ISP backbone links), researchers have proposed automated methods to extract application signatures in order to ease the manual signature generation process [6–8].

Although the payload-based approach eliminates dependency on fixed port numbers and increases traffic classification accuracy, it still has some drawbacks. Inspecting payload data is computationally

DOI: 10.1002/nem

resource intensive. Furthermore, extracting signaturesfrom encrypted or tunneled traffic can be difficult or impossible. Finally, privacy legislation with respect to payload inspection inhibits the use of the payload-based approach.

2.1.3. Host behavior-based approach

The host behavior-based approach was proposed for the classification of traffic based on social interactions observable even for encrypted payloads [9–11]. For example, Karagiannis *et al*. [9] developed a method called BLINC, which identifies applications or services by comparing a captured profile with predefined host behavior signatures (also described by graphlets). The key advantage of this approach is flexibility, as no additional information on the application (such as port number) is needed. Using a given set of pattern models, the authors claim that BLINC can identify application traffic with more than 90% accuracy. However, classification is limited to identifying application types, as exact application names cannot be classified.

Iliofotou *et al*. [10,12] proposed a graph-based representation of network traffic that captures the network-wide interactions of applications. Traffic dispersion graphs (TDGs) represent an individual IP address as a node and particular communications as edges between nodes. The idea behind BLINC and TDGs is to investigate the communication pattern generated by a host and extract behavioral patterns that may represent distinct activities or applications.

2.1.4. Statistical approach

The statistical approach exploits connection-related statistical information which can extracted by examining only TCP/IP headers as a feature set regardless of techniques for traffic classification. The major strength is that no payload inspection is required. In other words, this may provide a better chance for handling encrypted traffic. The underlying idea is that the traffic at the network layer has unique statistical properties for certain applications. Since machine learning (ML) was first utilized for Internet flow classification in the context of intrusion detection in 1994 [13], many others have used ML algorithms to classify network traffic [14]. Learning algorithms such as supervised, unsupervised, reinforcement and hybrid learning are used to build classifying models [15–21]. They take the input in the form of a dataset of instances. An instance refers to an individual and independent example of the dataset. Each instance is characterized by values of its features that measure different characteristics. The dataset is ultimately presented as a matrix of instances versus features.

2.2. Level of application traffic classification

2.2.1. Application protocol breakdown scheme

Traffic classification is a process of identifying network traffic based on the features that can be passively observed in the traffic. The features and classification results may vary according to specific classification requirements and analysis needs. Early on, traffic classification was performed as a part of traffic characterization work, and was often motivated by the dominance of certain protocols in the network. Several studies [22,23] analyzed the packet and byte distributions regarding transport and application layer protocols. TCP/UDP port numbers map to a well-known TCP/UDP protocol. The protocol breakdown scheme shows a rough estimation of the traffic composition and is still a popular solution at the Internet backbone because of its high traffic volumes and limited computing resources for traffic analysis [7,18,20,23–26].

2.2.2. Traffic clustering scheme

A straightforward classification, relying on IP protocol and port information, cannot provide in-depth classification of traffic within a single protocol where similar traffic types use different protocols. This scheme reflects traffic workload characteristics rather than the protocol composition [15,27,28]. McGregor *et al*. [15] proposed an ML-based classification methodology that can break the traffic down into clusters: bulk transfers, small transactions and multiple transactions. This allows us to understand the major types of traffic in a network.

DOI: 10.1002/nem

2.2.3. Application-type breakdown scheme

BLINC [9] is a connection-pattern-based classification method. It categorizes network traffic according to application type rather than a specific application name, such as Web, game, chat, P2P, DNS, FTP, streaming, mail and attack activities.This scheme resides between the former two schemes [1,16,29–36]. The application-type breakdown scheme helps network administrators or operators to understand dominant application types rather than specific application names or protocols.

2.2.4. Application breakdown scheme

The dominance of P2P networking in the Internet has had a huge influence on traffic classification research and has led to more sophisticated heuristics. In this context, many researchers have focused on identifying the exact application represented by the traffic [4–6,37–44]. Discovering byte signatures [5] has been a popular solution. Regardless of its proven accuracy, the signature-based solution possesses high processing overhead and privacy-breaching issues because it requires a packet header and payload analysis. Recently, machine-learning techniques which use statistical information of the transport layer [45] have been introduced to overcome privacy legislation related to packet payload inspection. They focus on the fact thatdifferent applications have different communication patterns (behaviors). Moreover, Szab *ó et al*. [46] introduced combinations of these existing methods in order to balance between the level of classification completeness and accuracy. All these efforts are presented to classify network traffic according to the name of application in use.

3. FINE-GRAINED TRAFFIC CLASSIFICATION

We propose a scheme, called fine-grained traffic classification, to provide more in-depth analysis. The key is to develop a method for categorizing single-application traffic into different traffic groups. In our previous work [2], we introduced the concept of the fine-grained traffic classification. This paper focuses more on a detailed method and highlights its effectiveness on detecting minor traffic compared to the conventional deep packet inspection (DPI)-based methods.

3.1. Significance of fine-grained traffic classification

The significance of fine-grained traffic classification derives from the traffic characteristics of Internet applications. Previous studies mainly grapple with detecting major applications such as P2P and streaming applications. Detecting these is heavily weighted toward classifying a few main functions (e.g. file transfer in P2P) that generate the largest volume. Neglect of minor traffic inevitably undermines the accuracy and completeness of classification.

We present a simple example of ignoring minor traffic: downloading a few music files using a P2P client. Assuming the total traffic volume generated by the client was approximately 45.9 Mbytes, the actual volume of downloads was only 24.1 Mbytes or less than 53% of the total generated traffic. This result implies that the ancillary (outside of actual downloading) functionalities of the P2P client generated a significant portion of the traffic.

Figure 1 compares the fine-grained traffic classification scheme with other traffic classification schemes in Section 2.2. It can classify various types of traffic caused by a single application. A single application typically has several functions, and each function will trigger traffic with unique characteristics. By identifying various types of functional traffic, undetected or unclassified traffic is less prevalent than the other schemes. While the others can do a simple top-*n* application analysis, the proposed scheme can provide a distinctive view on data analysis, such as measurement of pre-browsing time before download and raking of functions in use. It can serve as a new tool to analyze user behavior.

3.2. Method for fine-grained traffic classification

The key is to develop a method for categorizing single-application traffic into different traffic groups. This challenge differs from those facing traditional traffic classifications only in the degree of

DOI: 10.1002/nem

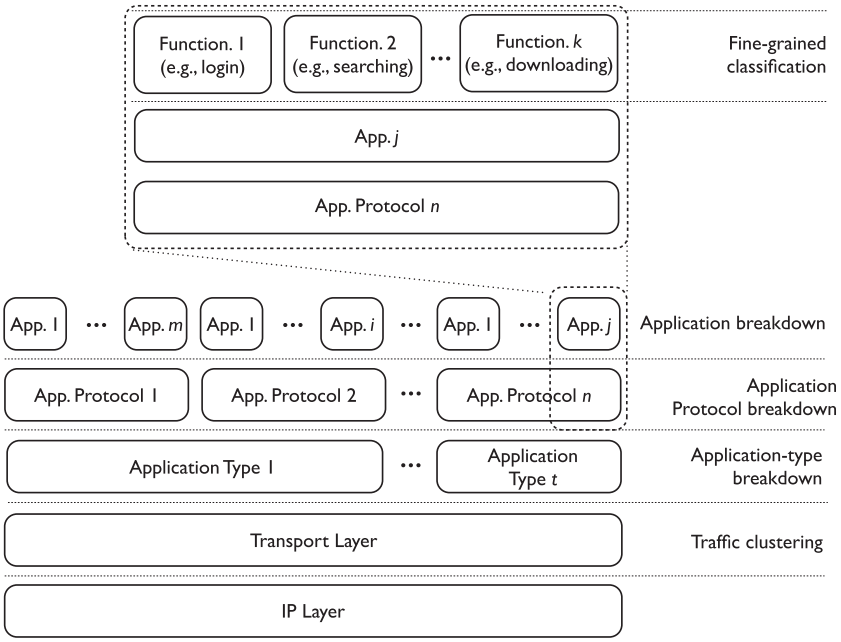

Figure 1. Relationship between fine-grained traffic classification and other traffic classification schemes

classification detail required. Accordingly, other existing methods, most of which use an application or protocol classifier, can be applied to simplify the fine-grained traffic classification problem. First, arbitrary classifiers (e.g. application signature, connection behavior model, statistical model) are built for each application, with each classifier corresponding to a distinct function within the application. A signature is selected as the classifier. As mentioned in Section 2, the signature-based (or payload) traffic classificationhas its limits in handling encrypted traffic, yet it is by far the most reliable in terms of accuracy. It is also convenient to apply new fine-grained signatures to existing traffic classification systems such as commercial traffic shapers and intrusion detection devices. The US government forecasts that the market share of DPI will increase continuously [47]. This implies that the signature-based approach is still reliable in practice despite its obvious drawbacks.

Our method for fine-grained traffic classification consists of four parts: (i) collection of input data; (ii) discrimination of the various functionalities within a single traffic of applications; (iii) classification filter extraction; and (iv) trafficclassification using fine-grained signatures.

Figure 2 illustrates the workflow of a fine-grained traffic classification process. It consists of offline and online components. The offline component builds a knowledge structure for the online traffic classification system, whereas the online component runs the actual traffic classification process. The offline process starts with input data collection. The functional separation of the offline process requires a collection of sanitized packets or packets belonging only to the target application as its input data. We have developed a packet dump agent to collect the packet trace for each process running in the operating system (OS). The collecting agent divides the packets by flow and stores them in separate packet dump files tagged with origin process names. It is important to keep datasets separated by flow and by process name in order to reduce unnecessary flow comparison overhead in the functional separation step. If a given dataset is a mixture of too many different applications, the discovery of common patterns within an application may be hindered. Use of this design feature is necessary in ensuring the efficiency and accuracy of functional separation, as it removes any uncertainty with respect to the traffic being fed to the functional separation algorithm.

We consider the example of a host running n different applications with each application executing m_i different functional modules (where i indicates the application index number),

DOI: 10.1002/nem

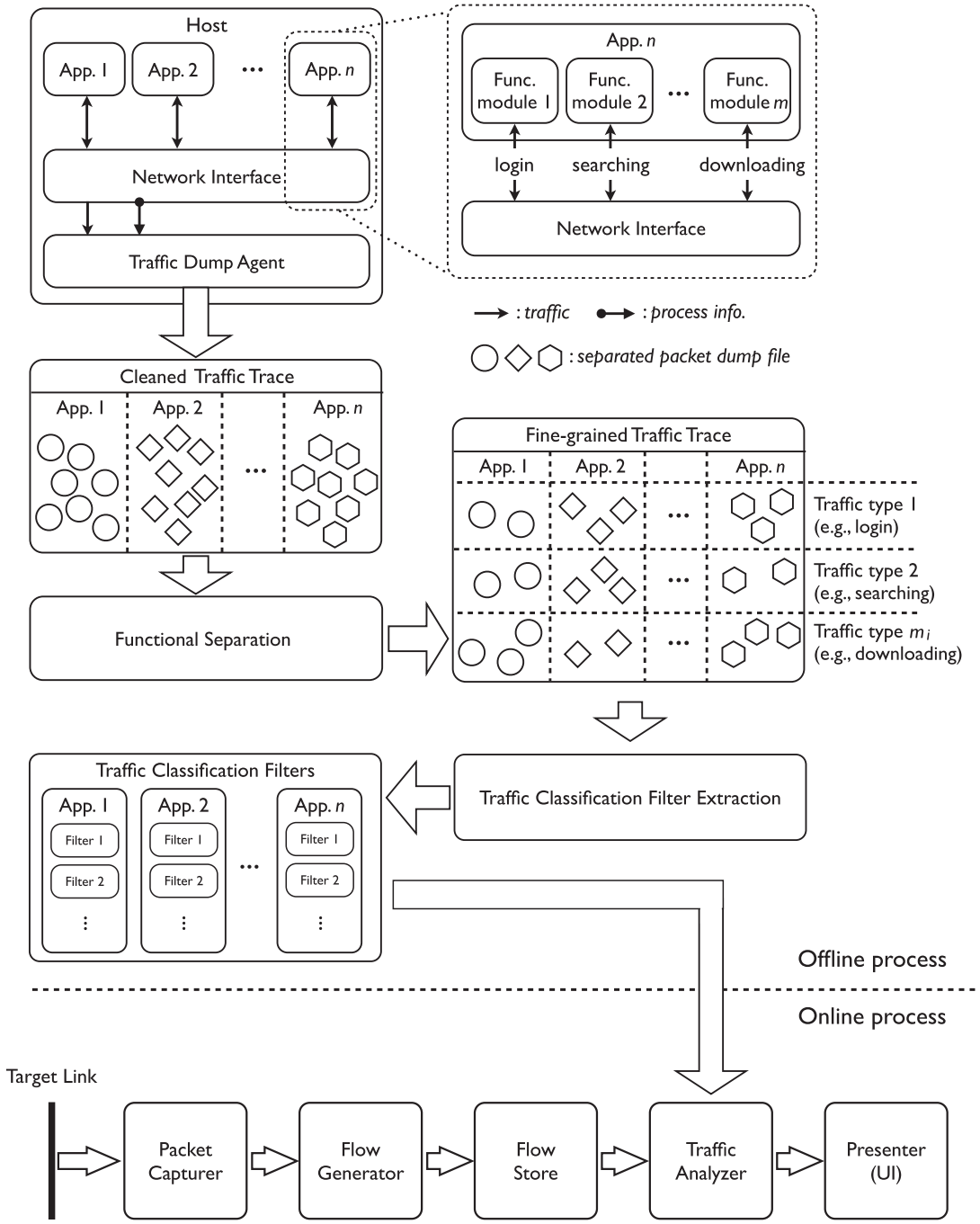

Figure 2. Abstraction of fine-grained traffic classification process

and each module generates differing types of traffic. The dump agent aggregates the resulting traffic data into flows and stores each flow in a separate file. Each flow is tagged with an application or process name acquired from the OS. The n stored groups of cleaned traffic, each labeled with an application name, are fed into the functional separation process, which classifies the cleaned traffic into m_i subcategories according to flow type. This functional separation produces $n \times m_i$ groups of flow data. Each group of flow is used as input for traffic classification filter extraction. In this case, the filter extraction's output will be $n \times m_i$ groups, where a single application can have at most m_i signatures.

DOI: 10.1002/nem

4. FUNCTIONAL SEPARATION

Functional separation (FS) is an attempt to classify individual flows according to their functionalities within an application. FS exploits the fact that equivalent functionalities share common characteristics. Here we adopt the TCP/UDP port number and payload contents as grouping criteria. Although assigned port numbers are not predictable, sessions sharing the same port number in a certain time slot can be placed in the same functional group. If a functional session allocates different port numbers, the actual packet contents are also examined in order to sort flows based on similarities of application protocol layers.

Figure 3 illustrates the entire FS process, which consists of three different grouping/decomposition steps. The solid rectangles in the figure illustrate how the traffic data are organized after each grouping/decomposition step, and the dashed rectangles illustrate the information used in each step. The role of each grouping/decomposition step is described as follows:

1. *Port-relation grouping* (PRG) involves the assortment of flows based on their dependencies to assigned port numbers. In this step, flow information is generated from the traffic trace as 5-tuples, i.e. (*source IP address*, tdestination IP address, *source port*, *destination port*, *protocol*). During the PRG step, the port numbers are treated as indices without function-related information. For example, flows that use TCP port 21 are placed within the same PRG group; however, it is not assumed that the use of TCP port 21 signifies FTP control traffic.
2. *Contents-relation grouping* (CRG) measures the similarity between different PR groups. The CRG process is based on the payload content and communication patterns of each PR group. Depending on payload contents, a common byte pattern for each application protocol is determined. Communication patterns are also examined in terms of the number of source/destination ports used. By using the CRG process, unnecessary flow comparison overhead in content examination can be reduced.
3. *Contents-relation decomposition* (CRD) also involves dividing contents-relation groups based on content similarity. In the PRG step, flows are assorted according to port numbers instead of actual application protocol contents. By chance, it is possible that the original port-relation group might allocate the same port number to different functional flows.

Our proposed method for functional separation normally works on both the TCP and the UDP flows. For the functional separation step, we do not consider unintentional packet drops in manipulating flows. In the case of a general traffic monitoring and measurement system located in the middle of a specific network link, unintentional packet drops occur when the system is

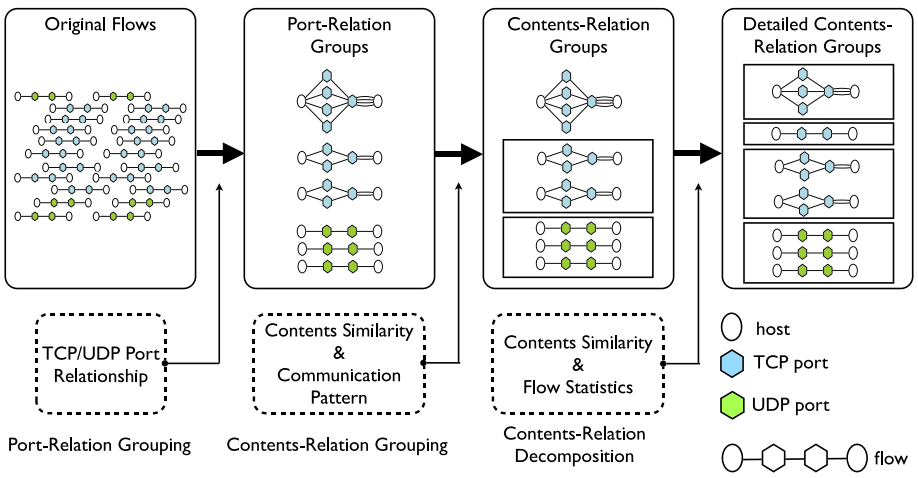

Figure 3. Overall process of functional separation

DOI: 10.1002/nem

exposed to an asymmetric routing environment or when the system has limited capacity for capturing packets. However, the offline processes (data collection, functional separation and classification filter extraction) of the fine-grained traffic classification function on an end host. The following subsections explain each functional separation step in detail.

4.1. Port-relation grouping

PRG classifies individual flows according to their dependencies on port numbers. PRG begins with the following assumptions:

1. Packets occurring close to each other in time and sharing the same 5-tuple (*source IP address*, *destination IP address*, *source port*, *destination port*, *protocol*) originated from the same functionality.
2. Reverse packets (in which the 5-tuple elements have been reversed, but the protocol remains the same) occurring within a small time interval (≤1 min) belong to the same functionality.

Algorithm 1. Pseudo algorithm for port-relation grouping

procedure FlowGrouping
input : Flows, $Flow = \{f_1, f_2, \cdots, f_n\}$
output: Port-relation groups, PR

```
1  begin
2      PR = {[ ], [ ], ··· , [ ]}                    // initialize port-relation group
3      F = {[ ], [ ], ··· , [ ]}                     // initialize bidirectional flow set
4      for 1 ≤ i ≤ n do
5          for i ≤ j ≤ n do
6              if f_i = reverse(f_j) then
7                  F ← f_i,f_j
8              end
9          end
10     end
11     for 1 ≤ i ≤ |F| do
12         if |PR| = 0 and i = 1 then
13             PR_0 ← F_i
14         end
15         else
16             for 1 ≤ j ≤ |PR| do
17                 if localPort(PR_j) = localPort(F_i) or
                   remotePort(PR_j) = remotePort(F_i) then
18                     PR_j ← F_i
19                 end
20             end
21         end
22     end
23 end
```


DOI: 10.1002/nem

Algorithm 1 describes the PRG process. A flow, $f_{(sip,dip,sport,dport,proto)}$, is a sequence of packets with the same 5-tuple header values. One TCP or UDP connection consists of two different flows. Therefore, flow f_a and its reverse flow f_b belong to the same bidirectional flow and the same PR group (lines 4–10). Because the input data are collected from a host, a bidirectional flow can be described as $F_{(localip,localport,remoteport,remoteip,proto)}$.

In a single execution of an application, different functionalities create multiple flows concurrently. Even though a port number that is to be allocated is determined at run time, flows that allocate the same port (either local or remote) simultaneously belong to the same PR group (lines 11–21). The best illustration of this lies in the behavior of a P2P client. A P2P client is required to allocate a port in order to upload data when the client joins a P2P network. Although the port number is not predictable, once the client allocates a port each flow established using the port will be designated for uploading data.

Figure 4 depicts an example of the application of PRG on BitTorrent traffic. The sample shows traffic captured when a BitTorrent client downloads a file, though only a part of the total traffic is shown. As detailed in the figure, a BitTorrent client uses both TCP and UDP flows in downloading

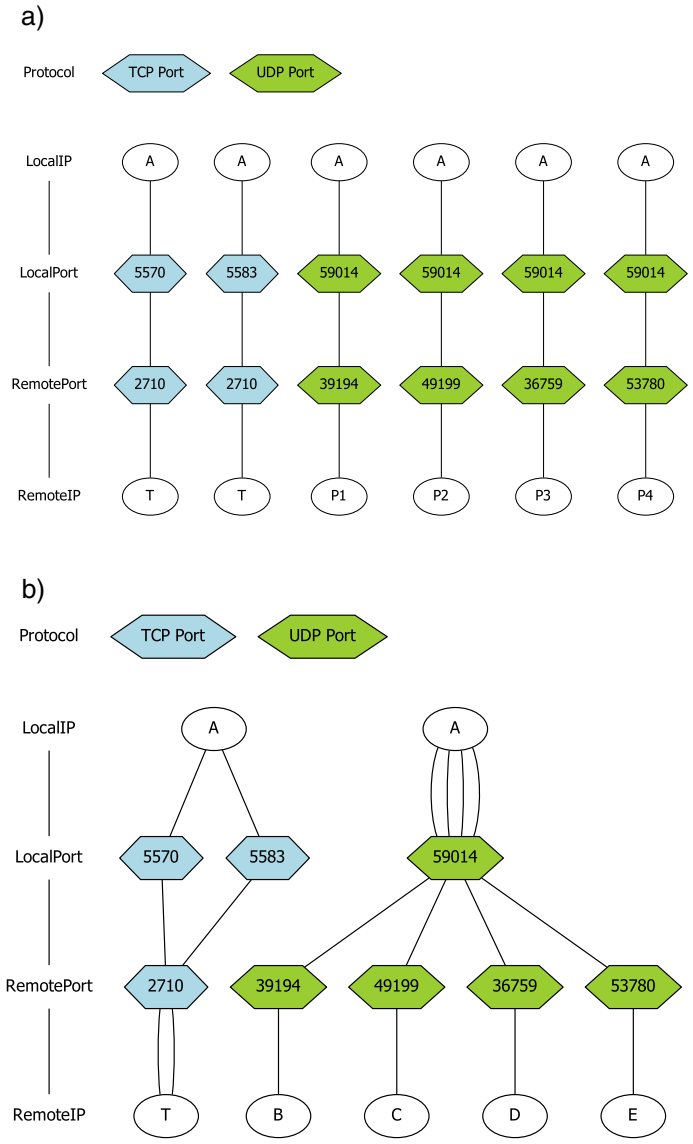

Figure 4. Example of port-relation grouping on BitTorrent traffic: (a) bidirectional flows; (b) port-relation groups

DOI: 10.1002/nem

a file. The TCP flows are used for obtaining the hash value of the file from trackers, whereas UDP flows are used for actual downloading. Figure 4(a) illustrates bidirectional flows (the aggregation of a flow and its reverse flow), and Figure 4(b) illustrates the PR groups created after the flows have been sorted based on either their local or remote port numbers. Connected graphs indicate PR groups. In this example, the output of PRG is two PR groups.

4.2. Contents-relation grouping

If different flows allocate different port numbers, PRG cannot classify them as belonging to the same functional group, even if they actually do. For example, since a P2P client connects to different peers at the same time, each flow might allocate to a random port. Thus it would be impossible to place all flows into one functional group using PRG, even if the flows are established for the same reason (e.g. for searching or downloading). To overcome this shortcoming of the PRG algorithm, we have developed a method called contents-relation grouping (CRG). The main objective of CRG is to connect preliminary PR groups according to content similarity interdependencies. To combine interrelated PR groups, we measure the degree of content similarity between the groups. If the similarity metric exceeds a certain threshold value, then the PR groups are merged into one CR group.

In order to compare content similarity, we have adopted a technique which is one of the main natural language-processing areas of research, i.e. document retrieval [48]. A key concept underlying document retrieval is that the degree of similarity of documents can be measured by the frequencies of keywords common to these documents. We define the following key terms for applying document similarity to traffic classification.

1. *Payload vector conversion*. To represent network traffic as text documents, vector space modeling (VSM) is used. VSM is an algebraic model that represents text documents as vectors. The objective of document retrieval is to find a subset of documents from a set of stored text documents D, which satisfies an information request or query Q. In a document space consisting of documents D_i, each can be identified by one or more index terms T_j that can be weighted according to importance [49].

A typical method for determining the significance of a term is to measure its occurrence. If t different index terms are present in document D_i, then D_i can be represented by a t-dimensional term-frequency vector $D_i = (d_{i1}, d_{i2}, \ldots, d_{ij})$, where d_{ij} represents the frequency of the jth term. Although text documents are composed of terms (words) that are the units of language that function as principal carriers of meaning, a packet does not have basic units with such definite meanings. To address this, we define the *term* of a payload as follows.

Definition 1: A term is n bytes of payload data within an i-byte sliding window, where the position of the sliding window can be $1, 2, \ldots, n - i + 1$. The size of the term set is $2^{8 \times i}$ and the length of a term is i.

If the length of a term is too short, that term cannot reflect the sequence of byte patterns in the payload. In this case, differences between byte pattern permutations, e.g. '0x01 0x02 0x03' and '0x03 0x01 0x02', cannot be recognized. On the otherhand, increasing the term length will cause the number of whole representative words to increase exponentially. A packet can be represented as a term-frequency vector called the *payload vector*.

Definition 2: If w_i is the ith occurrence of a term appearing repeatedly in a payload, the payload vector is defined as *payload vector* = $[w_1 w_2 \ldots w_n]^T$, where n is the size of the whole representative term set.

The sliding window size i is set to 2 because this is the simplest way to represent the order of content in payloads. If the term size is 2 bytes, the size of the term set will be 2^{16}. Therefore, the payload vector can be represented as a 2^{16}-dimensional term-frequency vector.

2. *Payload vector comparison*. Once packets are converted into vectors, the degree of similarity between packets can be calculated by measuring the distances between vectors. Here, we use Jaccard similarity as a distance metric. In our previous work [50], we compared three different similarity metrics: Jaccard similarity, cosine similarity and RBF, and Jaccard similarity showed

DOI: 10.1002/nem

the best performance. The Jaccard similarity $J(X,Y)$ draws word sets from comparison instances and uses them to evaluate similarity. $J(X,Y)$ is defined as the size of the intersection of the sample sets X and Y divided by the size of the union of the sets:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

One of the advantages of using Jaccard similarity instead of the Euclidean distance is that the similarity value can be normalized, enabling calculation using the dot (scalar) product, with the Jaccard similarity ranging from zero, for complete dissimilarity, to one, for complete vector similarity. As the payload vectors have very high dimensionality, they are statistically very sensitive. If two payload vectors are generated by different functionalities, then the contents of each payload will consist of distinct term (or binary) sequences and their vectors will also be very different. Because most signatures of application traffic represent a small portion of payload data, the other part of the payload's signature may be ignored as arbitrary binary data.

3. *Flow similarity comparison*. The payload flow matrix (PFM) is defined as a matrix in which the ith row represents the payload vector of the ith packet in the flow. The PFM is a $k \times n$ matrix, where k is the number of packets and $n$is the dimension of the payload vectors.

Definition 3: PFM = $[p_1, p_2, \ldots, p_k]^T$, where p_i is the payload vector from Definition 2.

The similarity score between PFMs can be calculated by simply adding packet similarity values.

Definition 4: Similarity score = $\sum_{i=1}^{k} J(p_i, p_i')$, where p_i and p_i' are the ith packets of the first and second flows, respectively.

Algorithm 2. Pseudo algorithm for CRG using similarity

procedure Contents-relation grouping
input : PR groups, $PRG = \{PRG_1, PRG_2, \cdots, PRG_n\}$
output: CR groups, $CRG = \{CRG_1, CRG_2, \cdots, CRG_m\}$

```
1  begin
2  |  CRG = {[ ], [ ], ··· , [ ]}                    // initialize CR group
3  |  for 1 ≤ i ≤ n do
4  |  |  if i = 1 then
5  |  |  |  CRG[0] ← PRG_i
6  |  |  end
7  |  |  else
8  |  |  |  M = PFM(PRG_i)
9  |  |  |  for 1 ≤ j ≤ number of CR group do
10 |  |  |  |  Similarity{} ← SimilarityScore(CRG[j], M)
11 |  |  |  end
12 |  |  |  if Max(Similarity) ≥ threshold then
13 |  |  |  |  CRG[Max index] ← PR_i
14 |  |  |  end
15 |  |  |  else
16 |  |  |  |  CRG ← PR_i                    // create a new CR group
17 |  |  |  end
18 |  |  end
19 |  end
20 end
```


DOI: 10.1002/nem

Algorithm 2 describes the grouping process used by CRG. CRG reads bidirectional flows and groups them into CR groups based on similarity scores. If the CR group set is empty, the first flow F_1 creates a new flow group $FG[0]$ (lines 4–5). Otherwise, the input flow is compared with the existing CR groups and inserted into the CR group with the maximum flow similarity score (lines 11–12). When the maximum similarity score is less than a certain threshold value, a new CR group is created and the bidirectional flow F_i becomes a member of the new flow group (line 16).

Along with content similarity, two other factors are considered in the CRG process for efficiency. When two different PR groups are compared, the connection patterns of these groups are taken into consideration. We define the connection pattern of a PR group as {1:# of local ports:# of remote ports:# of destinations}. The first element will alwaysbe 1, as it indicates the number of source hosts, and functional separation data are collected from one selected host. Figure 5 shows an example of connection patterns of PR groups. PRG_1 and PRG_2 have a 1:3:1:1 pattern; PRG_3, PRG_4 and PRG_5 all have a 1:1:1:1 pattern.

If different PR groups are generated by the same functionality, they are likely to have common connection patterns. Thus the content similarity of PR groups that have the same connection patterns (i.e. $\{PRG_1,PRG_2\}$, $\{tPRG_3,PRG_4,PRG_5\}$) is measured.

In addition to connection patterns, we also consider the relationship between local and remote port numbers in bidirectional flows. In PRG, bidirectional flows are grouped according to the common local or remote port number.

In Figure 6(a) Host A operates as a server. Peers connect to Host A through A's listening port, a. The bidirectional flows F_{AB}, F_{AC}, and F_{AD} form a PR group, and their common local port number is a. If Host A operates as a client (Figure 6b), a bidirectional flow F_{AB} forms a PR group. If the protocol used in both cases is identical, then these PR groups should be merged. However, the PRG cannot group them because they have different port numbers. To solve this problem, CRG compares two different PR groups to see whether their local and remote port numbers are identical.

4.3. Contents-relation decomposition

In the CRG step, only PR groups with the same connection patterns or same representative ports are compared for possible merging into CR groups. Otherwise, PR groups will form independent CR groups of their own without content examination.

The PRG procedure classifies flows according to dependencies on the assigned port numbers. Such a grouping process works most effectively where a functionality is mapped to a single port number. However, the PRG cannot perceive differing functional traffic generated by a single port

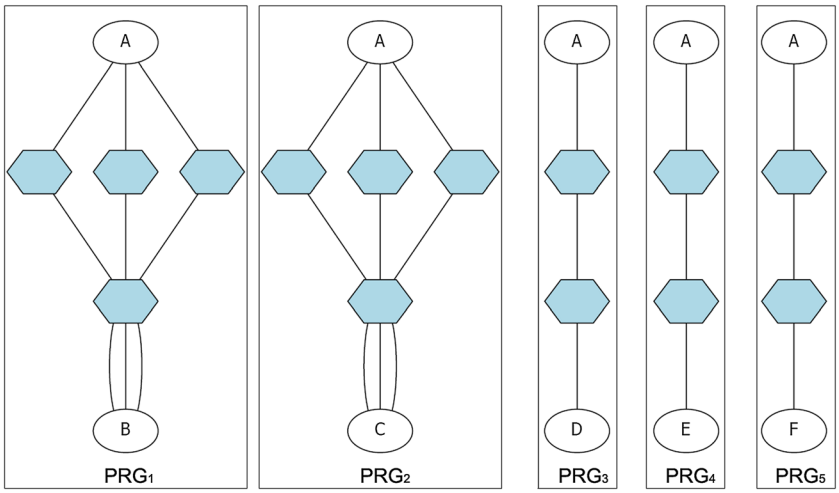

Figure 5. Example of connection patterns

DOI: 10.1002/nem

number. Thus the PR group representing TCP port 1863 may have different functionalities, such as retrieval of buddy lists or messaging. The contents-relation decomposition (CRD) process can compensate for this shortcoming of PRG processing.

Based on content similarity, CRD can differentiate functionalities within the CR group made up of PR groups. The method of calculation used to measure content similarity, shown in Section 4.2, can be used again, only in the opposite direction. In contrast to CRG, if the similarity metric between two flows in the CR group does not exceed a certain threshold value, then the CR group

Algorithm 3. Pseudo algorithm for CRD using similarity

procedure Contents-relation decomposition
input : CR groups, $CR = \{CR_1, CR_2, \cdots, CR_n\}$
output: CRD groups, $CRD = \{CRD_1, CRD_2, \cdots, CRD_m\}$

```
1  begin
2      FG = {[ ], [ ], ··· , [ ]}                          // initialize flow group
3      for 1 ≤ i ≤ n do
4          if isInPRGroup(CR_i) then
5              FG ← Decompose(CR_i)
6          end
7      end
8      n ← |FG|
9      CRD = {[ ], [ ], ··· , [ ]}                          // initialize CRD group
10     CRD[0] ← F_i                                         // create first CRD group
11     for 1 < i ≤ n do
12         M = PFM(F_i)
13         for 1 ≤ j ≤ |FG| do
14             Similarity{} ← SimilarityScore(CRD[j], M)
15         end
16         if Max(Similarity) ≥ threshold then
17             CRD[Max index] ← F_i
18         end
19         else
20             CRD ← F_i                                    // create a new CRD group
21         end
22     end
23 end
```

can be partitioned. Algorithm 3 describes the decomposition process of the CRD. First, the procedure examines each CR group to check whether or not a CR group is a single PR group. If so, it is disassembled into individual bidirectional flows (lines 4–6). If a CRD group set is empty, the first flow F_1 creates a new CRD group *CRD*[0] (lines 11–12). Otherwise, an input flow is compared with the existing CRD groups and inserted into the CRD group with the maximum flow similarity score (lines 18–19).

Figure 7 illustrates an example of functional separation processes, including RPG, CRG and CRD, operating on the MSN traffic. As mentioned above, MSN messenger uses port TCP 1863 to communicate with its server. Bidirectional flows F_1, F_2, F_3, and F_4 use TCP 1863 as their common

DOI: 10.1002/nem

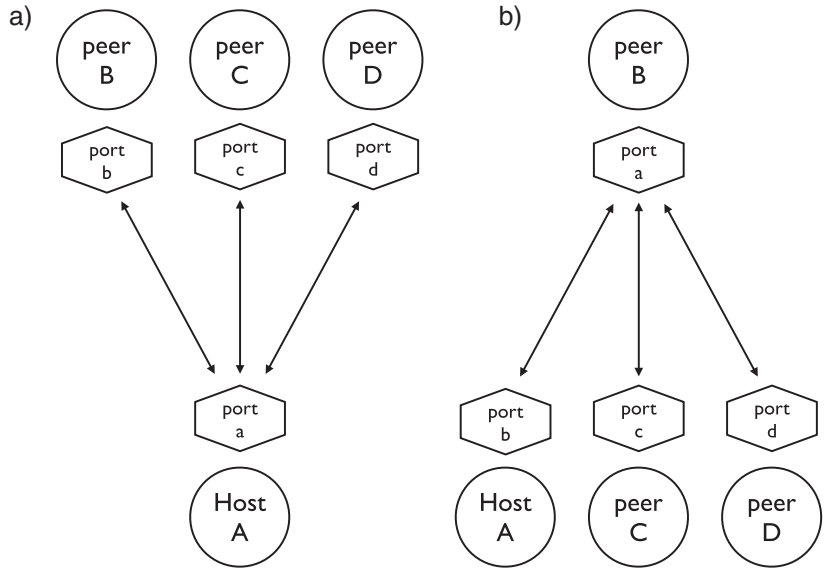

Figure 6. Connection behavior of P2P host A: (a) server mode; (b) client mode

remote port number. Based on PRG, these flows are all grouped into a PR group (PR_1), with a 1:4:1:1 connection pattern and representative port number TCP 1863. If an MSN server changes its port to 80 (we added this assumption to make it easier to understand the functional separation processes), bidirectional flows F_5, F_6, F_7 and F_8 form a different PR group (PR_2) likewise. If there is no other PR group with a 1:4:1:1 connection pattern or TCP 1863 as its representative port, PR_1 is not examined and simply forms a CR group without change. However, PR_2 has the same connection pattern. The similarity between these two PR groups is measured by CRG and a CR group (CR_1) is created. If F_1 and F_2 are generated by the same functionality and all other bidirectional flows are generated by another functionality, theresults of CRD are two different CRD groups, namely, CRD_1 $\{F_3, F_4, F_5, F_6, F_7, F_8\}$ and $CRD_2\{F_1, F_2\}$.

5. EVALUATION

In this section, we present our evaluation results and summarize a comparative analysis with other DPI-oriented methods.

5.1. Target applications

We selected 15 network applications based on their popularity as network and application types. The selected applications included both Internet and mobile applications. Application types included P2P (file sharing), messenger, Web storage (file hosting service), video/music streaming and online games.

Table 1 lists the selected applications and the amount of input traffic used by the functional separation process. The dataset was collected from the end host using the dump agent described in Section 3.2. Note that the functional separation process is a distinct process from the online traffic classification process in Figure 2. This process is for building a knowledge structure for the online traffic classification system. Therefore, the amount of the traffic data for the functional separation is not to be huge.

DOI: 10.1002/nem

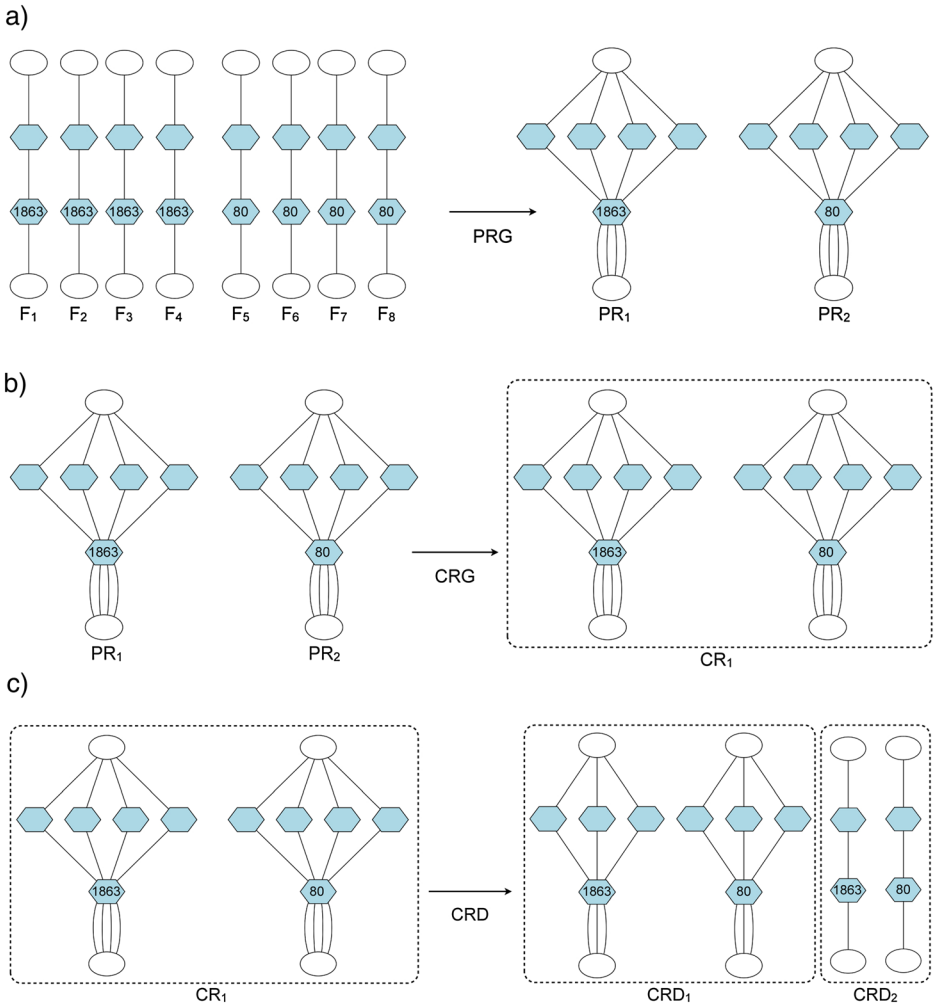

Figure 7. Example of functional separation on MSN traffic: (a) port-relation grouping; (b) contents-relation grouping; (c) contents-relation decomposition

Table 1. List of selected applications

			Operation		Dataset	
	Application type	Name	P2P	SC	Bytes	# of flow
Wired Internet applications	P2P (file sharing)	BitTorrent	◦		789 588 K	372
		LimeWire	◦		76 296 K	5 071
		Fileguri	◦	◦	78 768 K	11 044
	Messenger	MSN	◦	◦	81 420 K	224
		NateOn	◦	◦	306 388 K	140
	Web storage (file hosting service)	DropBox		◦	12 228 K	64
		uCloud		◦	95 176 K	32
	Video streaming	Gom	◦	◦	176 192 K	1 044
		PotPlayer	◦	◦	64 320 K	240
	Online game	Starcraft	◦	◦	4 118 K	584
		Starcraft 2		◦	48 815 K	98
Mobile applications	Web storage (file hosting service)	DropBox		◦	16 898 K	44
	Video streaming	TVPot		◦	82 502 K	422
	Music streaming	Bugs		◦	23 523 K	51
	Messenger	NateOn		◦	1 690 K	154

SC, server–client.

DOI: 10.1002/nem

5.2. Functional separation results

For each application, functional separation was carried out in order to generate arbitrary classifiers (e.g. application signature, connection behavior model, statistical model), with each classifier corresponding to a distinct function within the application.

We determined the threshold values for measuring the similarity experimentally. In our previous work [50], we examined three different similarity metrics—Jaccard similarity, cosine similarity and RBF—for the traffic classification per application (not per functionality). For each application, we tried many threshold values to find the one which maximized the accuracy against the ground truth. By trial and error, we could obtain threshold values for each application, but we used a fixed threshold value (0.7) in this paper for every application for the ease of experiment.

Table 2 shows the results of the functional separations. Since there was no 'ground truth' from the perspective of an application's functionality, we manually analyzed flows in each CRD group, labeling these by functionality. Further on, in the traffic classification step, this labeling process was used for analyzing the application usage pattern.

The '# of CRD group' field in the table counts only labeled groups; other groups are considered misclassified. We also considered CRD groups as misclassified if we could not label their exact functionalities. For example, we could not label passive FTPsessions without protocol semantics analysis. Thus, although the FTP sessions were grouped together, the passive FTP group was regarded as misclassified. Table 2's 'grouping ratios' entries indicate the proportion of labeled traffic to the entire traffictrace in terms of total bytes and number of flows, respectively.

Some misclassified flows did not carry application-level packet payloads aside from TCP/IP headers; additionally, some flows consisted of only one packet carrying the TCP SYN flag. As our CRG algorithm uses application-level packet payloads for grouping, the flow generated by applications exchanging only packets for TCP connection contained only TCP flags and thus could not be grouped with other PR groups owing to lack of payload data.

Classification accuracy can be measured in terms of either flow or byte accuracy. Flow accuracy denotes the proportion of correctly classified flow counts within the entire traffic dataset, whereas byte accuracy is a measure of the absolute number of bytes of traffic correctly classified by the classification algorithm. Erman *et al*. [51] argue that it is necessary to use byte accuracy to correctly evaluate the accuracy of traffic classification algorithms. This follows from the 'elephants andmice phenomenon' [28], in which the bulk of traffic byte data on the Internet is carried within a small number large flows, whereas the majority of flows transmit only a small percentage of total traffic in terms of bytes and packets. The authors analyzed the traffic dataset over a six-month-period; in terms of the number of bytes, the top 1% of flows accounted for over 73% of the traffic, with the top 5% accounting for 83% of traffic. When their classifier was applied, the traffic trace resulted in 99.9% flow accuracy, but with only 54% byte accuracy. The imbalanced accuracy results show the importance of using byte accuracy as a flow measure. Byte accuracy must also be used when evaluating the accuracy of traffic classification algorithms. With this in mind, we measured the grouping ratios in terms of both bytes and flow. These grouping ratios indicated how well functional separation distinguished each functionality. Even though the flow grouping ratios of LimeWire and uCloud were measured as 56.18% and 75%, respectively, the byte-grouping ratio for each of these applications was higher than 94%. In some applications, low flow grouping ratios were caused mainly by flows lacking payload content. However, these constitute a relatively small portion to the total byte flow and thus have a small effect on the byte-grouping ratio. Although the test cases were simple and limited, we could confirm that functional separation has a reasonable accuracy.

5.3. Traffic classification results

For the classification filter extraction, we used the LASER algorithm [6], which can generate an application signature automatically. This filter extraction (or signature generation) and traffic classification process, which uses extracted traffic classification filters, is based on DPI. We selected a DPI-based approach for a clear reason, as most previous studies demonstrate that, in terms of accuracy, the signature-based approach was the most reliable. Even some statistical (machine-learning) approaches

DOI: 10.1002/nem

Table 2. Functional separation result

Wired	Application	# of DCR groups	Grouping ratio		Details of DCR groups
			Byte	Flow	
Internet applications	BitTorrent	5	97.85%	99.98%	Tracker connection, Bulk transfer (UDP), Bulk transfer (TCP), Service discovery, DHT management
	LimeWire	5	99.51%	56.18%	Search, Bulk transfer, Peer info. exchange, uPnP discovery, DNS query (MDNS)
	Fileguri	7	100%	100%	Authentication, Advertisement, Peer info., Search, Bulk transfer, Connection management (UDP), Connection management (TCP)
	MSN	6	96.75%	99.99%	Advertisement, Authentication, Messaging, Remote assistant, Voice chat control, Voice chat
	NateOn	7	100%	100%	Advertisement, Authentication, Messaging, File transfer, Voice chat control, Voice chat, Remote assistant
	DropBox	4	100%	100%	Bulk transfer, Host discovery, Authentication, Net cache
	uCloud	3	99.89%	75%	Authentication, Bulk transfer, Version management
	Gom	4	99.99%	99.96%	Advertisement, Contents browsing, Streaming, Streaming management
	PotPlayer	6	99.90%	89.71%	Streaming (broadcast), Streaming (VOD), Service discovery, NTP, Contents browsing, Chat
	Stacraft	2	94.81%	96.92%	Battle net server connection (authentication + update + game list), Game data
	Stacraft 2	4	99.99%	94.12%	Game data, Battle net news, Battle net images, Authentication
Mobile applications	DropBox	3	100%	100%	Bulk transfer, Authentication, DNS query
	TVPot	4	100%	100%	Advertisement, Contents browsing, Streaming, DNS query
	Bugs	6	100%	100%	Top music chart, MV image, Album image, Music stream, Menu browsing, Authentication
	NateOn	3	100%	100%	Advertisement, Messaging, Text Chat

DOI: 10.1002/nem

used signature (or manual payload) inspection to generate the ground truths for validation [4,31,45,52,53].

For the dataset, we collected full packet traces from our campus backbone on two occasions: 3 h on 16 August 2007 (450 Gbytes) and 3 h on 7 October 2011 (37 Gbytes). For the latter dataset we capture the first 10 packets of each flow to save some storage. No port blocking or filtering policy was in effect at the time of measurement. The ground truth was verified by traffic measurement agent (TMA) [6]. This collects process and traffic information in allocation from the host OS directly; thus, the information is the closest possible ground truth available.

Classification filter extraction was also applied to some CRD groups that could not be labeled in the functional separation step (as shown under the 'Grouping ratio' heading in Table 2). Although concrete signatures could not be obtained for these CRDgroups, simple heuristics based on flow statistics and relationships, as well as on subnet information generated according to CRD group, were used as an alternative classifier for their function groups. A test of classification filter extraction was performed using BitTorrent, which has changed its protocol to allow downloading with UDP (for backward compatibility, BitTorrent still supports the TCP downloading protocol). BitTorrent sends only one UDP downloading request, and if a corresponding peer does not reply to this request the client connects again to the peer via TCP. However, the download request is not reissued. Based on this, we examined BitTorrent's download traffic in terms of the TCP connections established within the small time frame following the UDP download.

Table 3 shows the traffic identified by the proposed method. The classification accuracy in this table shows the ratio of correctly identified application traffic against the ground truth by TMA. It assesses how much of the traffic can be identified by multiple signatures of functional separation and the LASER algorithm. Owing to the lack of ground truth from the perspective of the application's functionality, the classification accuracy in functional level could not measured. Except for MSN (78.87%) and Starcraft (75.05%), the byte accuracy of all applications was higher than 91.39%, with the highest byte accuracy reaching 100%. As with the functional separation test, each application had lower flow accuracy thanthe byte accuracy. This can also be explained in terms of the 'elephants and mice phenomenon' described previously.

Figure 8 shows the cumulative probability distributions, in flow size (bytes), of three different applications—for BitTorrent and LimeWire, flows less than 1 kbyte in size constituted approximately 80% of the total number of flows. However, thecontribution of such flows to the total volume of traffic was very low, as shown in Figure 9. This is a good illustration of how a considerable portion of total traffic is generated by large flows, as more than 90% of the total traffic is caused by the largest 1% of flows. Such large flows usually correspond to file downloads. Therefore, even if our method cannot detect some 'mouse' flows, byte accuracy is rarely affected.

By manually analyzing traffic not identified by our method, we were able to make further conclusions. As part of their protocol operations, some applications use well-known protocols. BitTorrent clients, for

Table 3. Classification results: accuracy of the proposed method

		Classification accuracy	
Type	Name	Byte	Flow
Wired Internet application	BitTorrent	91.39%	65.78%
	LimeWire	99.78%	89.15%
	Fileguri	99.84%	99.49%
	MSN	78.87%	87.50%
	NateOn	100%	100%
	DropBox	100%	100%
	uCloud	100%	100%
	Gom	99.79%	97.31%
	PotPlayer	99.90%	81.36%
	Starcraft	75.05%	91.84%
	Starcraft 2	99.98%	88.23%
Mobile application	DropBox	99.97%	54.55%
	TVPot	100%	100%
	Bugs	100%	100%
	NateOn	99.90%	60.87%

DOI: 10.1002/nem

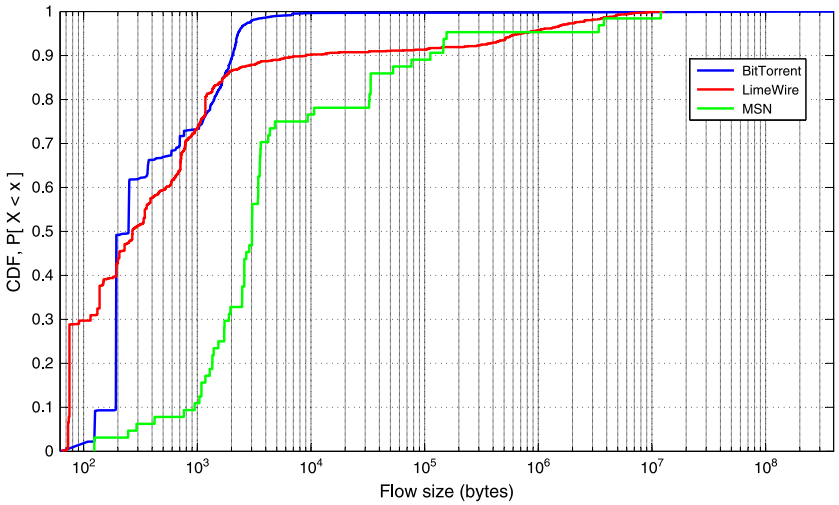

Figure 8. CDFs of flows generated by three different applications

example, use the SSDP protocol for service discovery, even though SSDP traffic payloads lack BitTorrent-specific features.

Another cause of non-identification was the fact that flows not having payload data were not included in the '# of CRD groups' tally in Table 2. Although this lack of traffic data degraded the flow identification ratio, the byte identification ratio was barely affected owing to the small byte size of the unrecorded flow. The main cause of the low byte accuracy of MSN was that the session initiation protocol (SIP)-based flow used in voice chat carries no MSN-specific contents as payload. Although MSN creates only one SIP session for voice chat, the size of this flow is relatively large. Therefore, the misclassified large flows degraded byte accuracy but not flow accuracy.

5.4. Comparison with conventional DPI solutions

We compared our method with two different conventional DPI-based traffic classification methods: L7-filter and OpenDPI:

- *L7-filter* [54] uses regular expression to match application signatures with packet payload application layer data. The current version supports 113 application protocols.
- *OpenDPI* [55] is a software library designed to classify Internet traffic. It is derived from the commercial PACE product [56]. In addition to signature matching, OpenDPI incorporates connection behavior and statistical analysis. The current version supports 101 protocols.

Table 4 shows the traffic identified by our method, L7-filter, and OpenDPI. Fine-grained traffic classification produced the highest identification ratio among the three approaches. L7-filter and OpenDPI showed comparatively poorer results in terms of both byte accuracy and flow accuracy. Even though the difference between L7-filter and OpenDPI in terms of accuracy was negligible, of the two, OpenDPI performed better.

To determine performance variance in the application of the fine-grained traffic classification method, we analyzed the classification results of applying OpenDPI to BitTorrent, LimeWire and MSN, as shown in Table 5. For each application, traffic was classified into different application protocol groups. Except in the case of unknown traffic, these classifications were correctly carried out. For example, our scheme confirmed that BitTorrent actually uses the HTTP protocol for discovering content, whereas LimeWire uses the MDNS protocol for discovering peers. However, our method would provide no means for matching these protocols to their originating applications.

Newer generations of applications tend to adopt multiple application protocols and support various functions (Figure 10). OpenDPI does not classify application traffic into layers higher than the application protocol, resulting in a low classificationratio for application layers. Based on functional separation, fine-grained traffic classification can detect application traffic regardless of its protocol.

DOI: 10.1002/nem

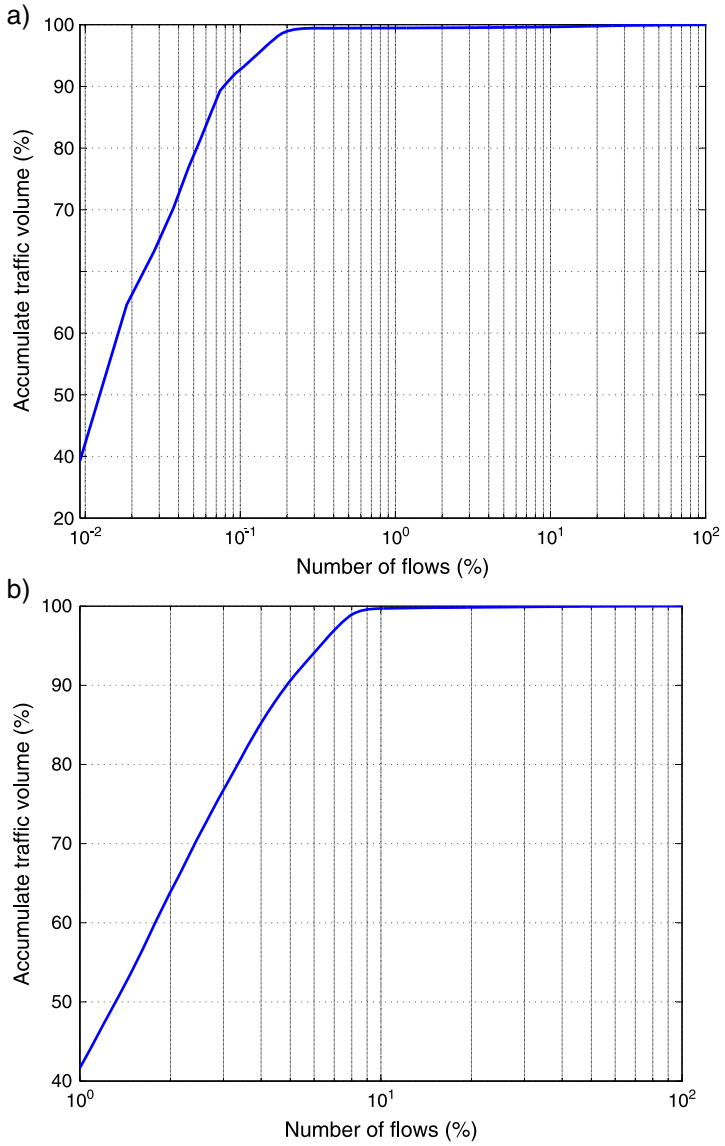

Figure 9. Contribution of the top *n*% of flows in traffic volume: (a) BitTorrent; (b) LimeWire

Table 4. Accuracy of fine-grained traffic classification vs. L7-filter and OpenDPI

	Identification ratio (accuracy)					
	Fine-grained traffic classification		L7-filter		OpenDPI	
Application	Byte	Flow	Byte	Flow	Byte	Flow
BitTorrent	91.39%	65.78%	16.57%	19.02%	16.44%	9.01%
LimeWire	99.78%	89.15%	97.85%	9.98%	99.73%	46.39%
MSN	78.87%	87.50%	17.64%	4.69%	17.64%	7.81%

Additionally, our proposed method classifies application traffic into functional layers, enriching the quality of information obtained from traffic classification results in the process.

Figure 11 illustrates a functional decomposition of traffic. Each color indicates a function within a single application. As demonstrated previously, fine-grained traffic classification can classify different traffic types within a single application according to their functionalities. A network

DOI: 10.1002/nem

Table 5. Classification results of OpenDPI

Application	Classified protocol	Proportion (%)	
		Byte	Flow
BitTorrent	Unknown	83.54	90.50
	HTTP	0.02	0.60
	BitTorrent	16.44	8.89
LimeWire	Unknown	1.38	52.95
	HTTP	0.01	0.51
	MDNS	0.00	0.07
	SSDP	0.00	0.07
	Gnutella	98.60	46.39
MSN	Unknown	0.63	6.76
	HTTP	0.45	45.95
	Flash	2.29	9.46
	MSN	17.64	6.76
	STUN	0.02	5.41
	RTP	18.77	1.35
	RDP	59.41	1.35
	SSL	0.78	22.97

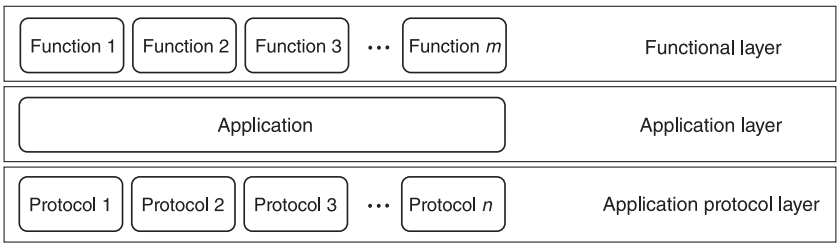

Figure 10. Application broken into layers

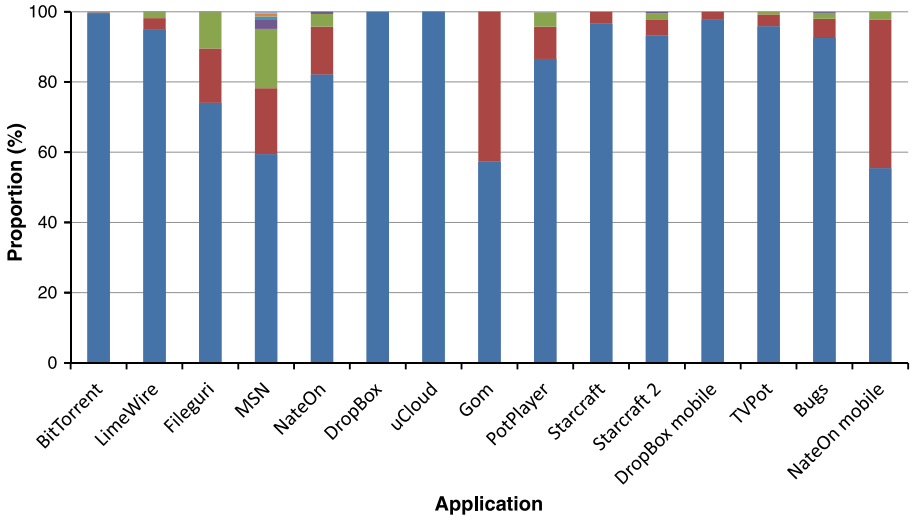

Figure 11. Traffic composition of each functionality

operator could use this information to perform quality-of-service (QoS) control for different traffic classes within an application.

The comparisons described here show that fine-grained traffic classification generally outperforms conventional signature-based classification methods. As discussed in Section 3.1, many traffic classification

DOI: 10.1002/nem

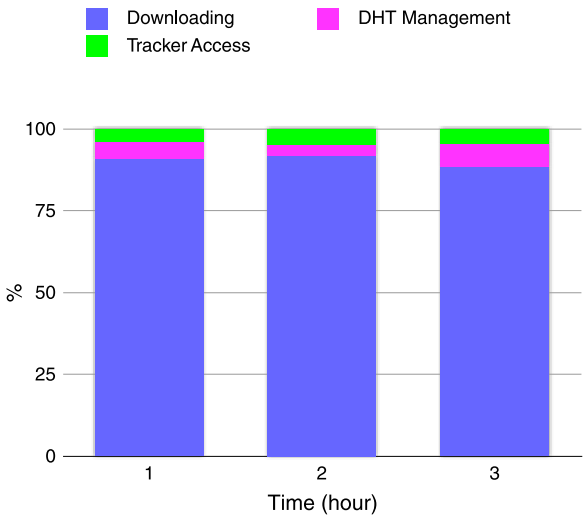

Figure 12. BitTorrent's workloads according to functionality

studies have focused on classifying major functions (such as file transfer in P2P) that generate high traffic volume. Thus signature-based classification misses a large portion of the application traffic data generated by other functions. Based on our manual analysis of classification results, both L7-filter and OpenDPI can only detect MSN file transfers and chat traffic, along with a portion of BitTorrent signaling traffic. Comparison of these results with the results of fine-grained traffic classification confirms that the proposed method can increase accuracy and completeness.

5.5. *Use cases of fine-grained traffic classification*

5.5.1. *User behavior analysis*

As a test, we analyzed average search counts used in initializing downloading P2P applications. The resulting transaction ratio of searches to downloads was 56 392:1. Empirically, we confirmed that the Fileguri client generated approximately 6000 TCP transactions with a single keyword search. This led us to conclude that a Fileguri user performs about 9.398 searches on an average before downloading from a P2P network.

5.5.2. *Workload analysis*

Most wired Internet access plans involve flat-rate billing. However, Internet access for smartphones often involves usage-based billing. As of March 2011, there are more than 200 000 Android and approximately 300 000 iPhone applications available, with these numbers continuously increasing [57]. From a billing perspective, the amount of traffic generated by such applications is crucial. As a test, we analyzed BitTorrent workloads owing to file downloading. Figure 12 illustrates the results, broken down by functionalities. From 10% to 15% of the traffic was generated by functionalities other than downloading. Such traffic, though essential to downloading, is undesirable from a user's standpoint. Analyzing and comparing workload structures of different applications could thus prove useful to users, by enabling them to obtain a breakdown by application of the relative proportion of unwanted traffic; this, in turn, could provide insight into optimizing application use based on data plans or billing schemes.

6. CONCLUSION

Many variations of traffic classification have been proposed to obtain better classification accuracy and traffic composition information. To overcome known difficulties, we proposed the fine-grained traffic classification scheme which allowedseparation of traffic with respect to application functionalities. As a part of this, we developed the function separation method and compared its performance with the existing

DOI: 10.1002/nem

DPI-oriented frameworks. Overall, our scheme showed an increase in accuracy of 1–75% and identified the traffic portions of newly discovered functionalities which proved for completeness (Figure 11).

For future work, we consider enhancing the functional separation method's labeling process. As the immediate identification and classification of all network flow still remain impossible, there is still much room for improvement. We also plan to analyzeuser behavior, an area which has received under considerable attention recently.

APPENDIX
VALIDITY OF ML ALGORITHMS ON FINE-GRAINED TRAFFIC CLASSIFICATION

The signature-based approach has several drawbacks on handling encrypted traffic. We compare the functional separation and ML approaches to show any superiority of the functional separation. We applied a clustering algorithm to Fileguri and NateOn, which have the most complex functionalities among the selected applications.

As briefly mentioned in Section 2, ML approaches can be categorized into supervised learning (or classification) and unsupervised learning (or clustering). Supervised learning requires a training phase to grasp the interrelationship between various features and classes. Training acquires a pre-labeled dataset. For example, flows should be labeled with their original application for traffic classification. Conversely, unsupervised learning automatically discovers the nature of different classes (clusters) in the dataset without any prior guidance.

Considering the functional separation problem, classifying different traffic generated by different functionalities within an single application, functional separation is close to unsupervised learning because the prior knowledge of functionalities is notavailable. The number of functionalities is not predefined and it is impossible to characterize different functional traffic groups.

Feature selection

Candidate features

For feature selection, we relied on previous traffic classification research using unsupervised learning. Table A.1 describes a selection of research and their feature set. The feature set includes flow statistics such as packet length, packetinter-arrival time, flow duration, etc. Bernaille *et al*. [58] used packet length and showed that more than 80% of total flows were correctly identified and the highest accuracy reached 99%. Table A.2 illustrates the candidate features.

Feature selection algorithm

Feature selection is a crucial process for building robust learning models [59,60]. There has been much attention on finding an effective feature selection for traffic classification [60,61]. We measured the impact of each candidate feature by applying the Relief algorithm and selected the final features for functional separation.

Relief algorithm

Relief is an instance-based feature-ranking algorithm introduced by Kira *et al*. [62] and improved by Kononenko [63]. The Relief algorithm has been known as the most successful feature selection method for classification [64]. The Relief family of algorithms identifies the importance of features based on the distance of nearest hits and nearest misses. Nearest hits denote data points within the same class and nearest misses refer to data points from different classes. It is suitable for high-dimensional data. It also works when there is no linear relationship among features.

The Relief weights can be calculated as follows:

$$w_i = w_i + dist\left(x^{(i)}, \mathrm{NM}^{(i)}(\rightarrow x)\right) - dist\left(x^{(i)}, \mathrm{NH}^{(i)}(\rightarrow x)\right) \quad (1)$$

where $x^{(i)}$ denotes the ith feature of a data point $\rightarrow x$. $\mathrm{NM}^{(i)}(\rightarrow x)$ and $\mathrm{NH}^{(i)}(\rightarrow x)$ indicate the ith feature

DOI: 10.1002/nem

Algorithm A.1. Relief algorithm for identifying discriminating features

procedure Relief

input : m training vectors of n attributes and the class label for each vector

output: the weigh vector $\vec{w}$ $(= < w_1 \cdots w_n >)$

```
begin
    for 1 ≤ i ≤ n do
        w_i ← 0
    end
    foreach data point x⃗ do
        NH ← nearest hit
        NM ← nearest miss
        for 1 ≤ j ≤ n do
            w_j = w_j + dist(x^(j), NM^(j)(x⃗)) - dist(x^(j), NH^(j)(x⃗))
        end
    end
end
```

Table A.1. Traffic classification research using clustering and their feature set

Title	ML algorithms	Features	Traffic composition
Flow clustering using machine learning techniques [15]	Expectation Maximization	Packet length statistics Inter-arrival statistics Byte counts Connection duration Idle time	HTTP, SMTP, FTP, NTP, IMAP, DNS, etc.
Automated traffic classification and application identification using machine learning [65]	AutoClass (Bayesian clustering)	Packet length statistics Inter-arrival statistics Flow size (bytes) Flow duration	Half-Life, Napster, AOL, HTTP, DNS, SMTP, Telnet, FTP
Traffic classification on the fly [58]	SimpleKMeans	Packet length of the first few packets	eDonkey, FTP, HTTP, KaZaA, NTP, SMTP, SSH, HTTPS, POP3
Identifying and discriminating between web and peer-to-peer traffic in the network core [32]	K-Means	Total number of packets Mean packet length Mean payload length Flow duration Flow size (bytes) Mean inter-arrival time	Web, P2P, FTP, others
Traffic classification using clustering algorithms [18]	K-Means	Total number of packets	HTTP, P2P, SMTP, IMAP, POP3, MSSQL, others
	DBSCAN AutoClass	Mean packet length Mean payload length Flow size (bytes) Mean inter-arrival time	

Statistics include mean/min./max./SD.

DOI: 10.1002/nem

Table A.2. List of candidate features

Index	Feature
1	Protocol
2	Number of packets
3	Flow size in bytes
4	Flow duration
5	Average packet size
6	Minimum packet size
7	Maximum packet size
8	Average packet inter-arrival time
9	Minimum packet inter-arrival time
10	Maximum packet inter-arrival time
11	Average payload size
12	Minimum payload size
13	Maximum payload size
14	Size of 1st packet in the flow
15	Size of 2nd packet in the flow
16	Size of 3rd packet in the flow
17	Size of 4th packet in the flow
18	Size of 5th packet in the flow

of nearest hit and nearest miss respectively. Algorithm A.1 shows the Relief algorithm. The function *dist* computes the difference between the values of feature for two instances. For discrete attributes, the difference is either 1 (the values are different) or 0 (the values are the same), while for continuous attributes the difference is the actual difference normalized to the interval [0,1].

Selected feature set

Table A.3 shows the final feature set. The weight of each feature fluctuates significantly from application to application. For example, protocol has the lowest weight (0) when identifying functionalities of Fileguri. It has the highest weight (0.351) when identifying functionalities of NateOn. Figure A.1 shows the feature weights of two different applications. Thus it is difficult to select features which are commonly efficient for all application traffic. We selected a different

Table A.3. Weights of features by the Relief algorithm

	Fileguri	NateOn
Protocol	0.000 ± 0.000	0.351 ± 0.004
Number of packets	0.085 ± 0.007	0.015 ± 0.003
Flow size in bytes	0.050 ± 0.009	0.003 ± 0.003
Flow duration	0.242 ± 0.031	0.115 ± 0.013
Average packet size	0.152 ± 0.011	0.018 ± 0.006
Minimum packet size	0.193 ± 0.010	0.253 ± 0.007
Maximum packet size	0.305 ± 0.008	0.030 ± 0.010
Average packet inter-arrival time	0.129 ± 0.019	0.027 ± 0.015
Minimum packet inter-arrival time	0.085 ± 0.027	0.042 ± 0.009
Maximum packet inter-arrival time	0.180 ± 0.023	0.038 ± 0.015
Average payload size	0.152 ± 0.010	0.018 ± 0.006
Minimum payload size	0.000 ± 0.000	0.351 ± 0.004
Maximum payload size	0.304 ± 0.008	0.029 ± 0.010
Size of 1st packet in the flow	0.128 ± 0.028	0.248 ± 0.011
Size of 2nd packet in the flow	0.086 ± 0.011	0.003 ± 0.001
Size of 3rd packet in the flow	0.332 ± 0.016	0.108 ± 0.012
Size of 4th packet in the flow	0.301 ± 0.018	0.242 ± 0.018
Size of 5th packet in the flow	0.317 ± 0.019	0.248 ± 0.017

DOI: 10.1002/nem

Table A.4. Final feature set of each application

	Fileguri	NateOn
Protocol		◦
Number of packets		
Flow size in bytes		
Flow duration	◦	◦
Average packet size	◦	◦
Minimum packet size	◦	◦
Maximum packet size	◦	
Average packet inter-arrival time	◦	
Minimum packet inter-arrival time	◦	
Maximum packet inter-arrival time	◦	
Average payload size	◦	
Minimum payload size		◦
Maximum payload size	◦	
Size of 1st packet in the flow	◦	◦
Size of 2nd packet in the flow		
Size of 3rd packet in the flow	◦	◦
Size of 4th packet in the flow	◦	◦
Size of 5th packet in the flow	◦	◦

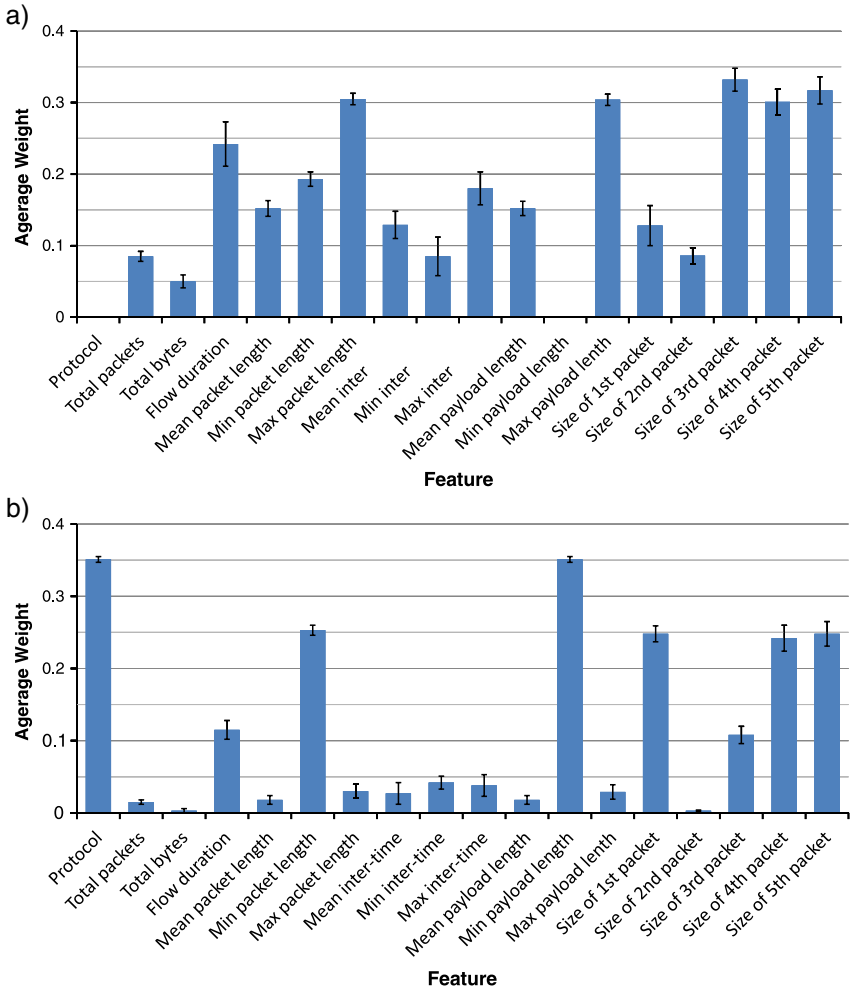

Figure A.1. Average weight of each feature: (a) Fileguri; (b) NateOn

DOI: 10.1002/nem

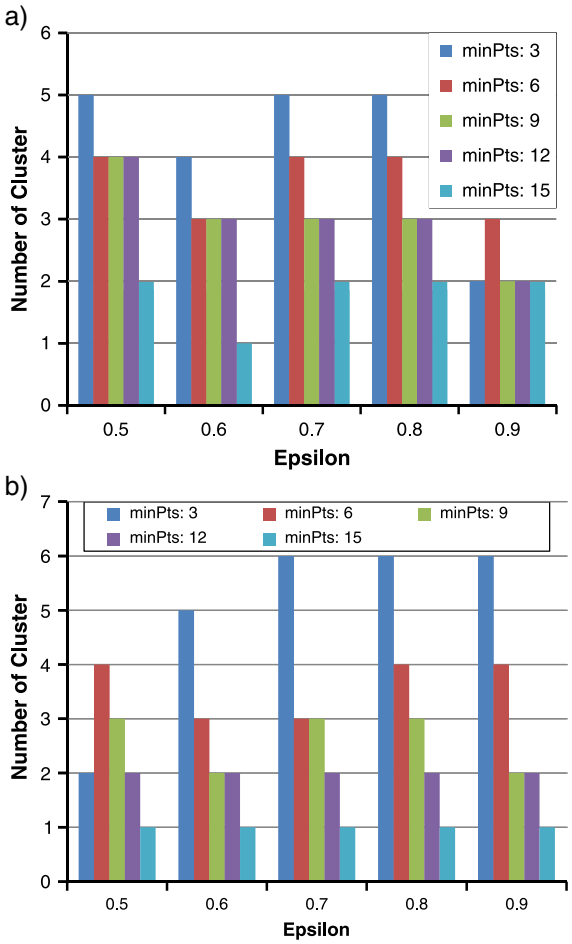

Figure A.2. Number of clusters using DBSCAN: (a) Fileguri; (b) NateOn

feature set for each application to resolve this problem. We removed the features with a weight value of less than 0.1 (Table A.4).

Clustering algorithms

Previous traffic classification studies using clustering utilized the following algorithms: Expectation Maximization (EM), AutoClass, SimpleKMeans, K-Means and DBSCAN (Table A.1). DBSCAN was used in clustering for functional separation amongthese algorithms owing to the characteristics of the clustering algorithms and the functional separation problem. The other algorithms require the number of clusters as an input parameter. However, the functional separation starts without any prior information on the number of clusters (functionalities in application). Therefore, we compared our method to DBSCAN.

DBSCAN algorithm

DBSCAN [66] is a density-based clustering algorithm. It finds a number of clusters starting from the estimated density distribution of corresponding nodes. An object p is directly density-reachable from an object q if both objects are located within a given distance ò. If p is surrounded by sufficient number of points, objects which are closer than ò in terms of distance, p and those objects are considered as a cluster.

DOI: 10.1002/nem

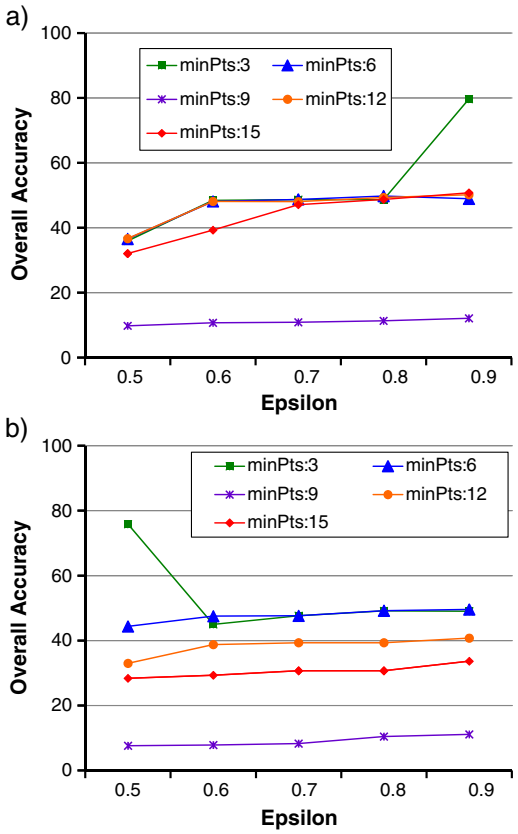

Figure A.3. Clustering accuracy of clustered data instances: (a) Fileguri; (b) NateOn

Clustering results

DBSCAN has two input parameters: ò and *minPts*. Each parameter denotes the radius of the ϵ distance and minimum number of data objects required in an ϵ distance range, respectively.

We first examined the number of clusters before analyzing traditional accuracy metrics such as *precision* and *recall*. Figure A.2 shows the results of DBSCAN in the number of output clusters. The numbers of labeled classes in the input data are seven for both applications. Because multiple clusters can be mapped to a functional group, the number of clusters may be more than that of the original classes, which is still acceptable considering the characteristics of the functional separation problem. DBSCAN determines the number of clusters automatically; however, it does not work well on discriminating different types of traffic within a single application. The number of output clusters is less than the original number of classes in every execution with different parameters. Such observation implies that the clustering algorithms cannot detect a clear boundary between different traffic characteristics using statistical features.

In the case of NateOn, we can get close to the original number of classes by adjusting the parameters. However, the overall accuracy is not acceptable.

Figure A.3 shows the overall accuracy of clustering. The unclustered data instances is considered as mis-clustered. The highest accuracy is about 80% in both applications. The cause of such low accuracy is the unclustered data instances described in Figure A.4. DBSCAN can label noise data. This is one of the strengths of DBSCAN when applied to general clustering problems. However, noise data instances can be considered as mis-clustered in functional separation. In the worst case, the unclustered ratios reach 61.22% and 67.21% for each. This implies that the accuracy of clustering results is less than 40%.

DOI: 10.1002/nem

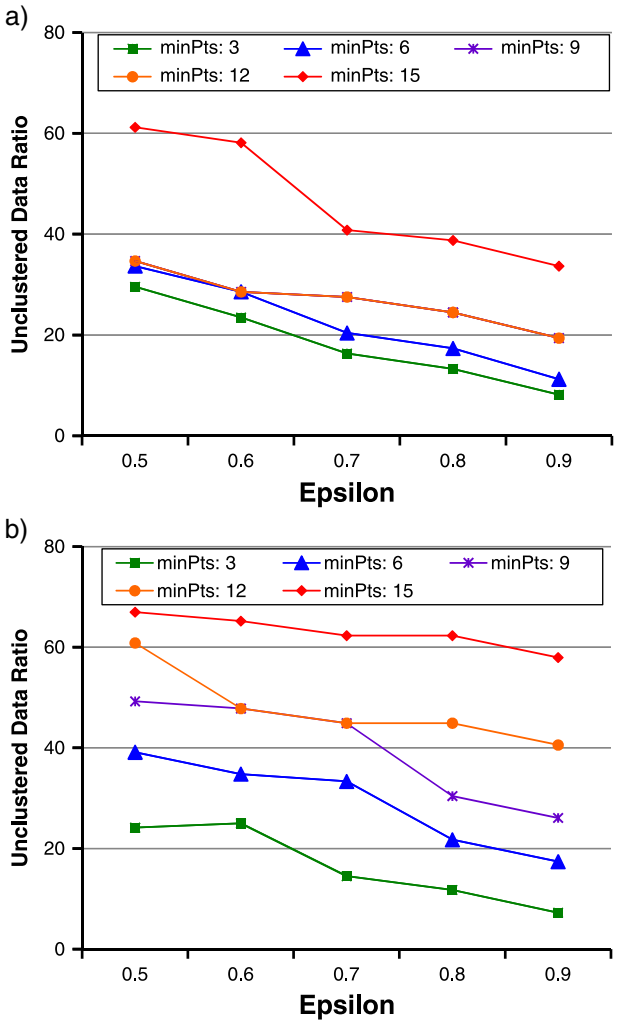

Figure A.4. Unclustered data ratios: (a) Fileguri; (b) NateOn

We need to maximize the number of clusters close to the real number of classes and minimize the mis-clustered ratio to apply clustering algorithms to the functional separation problem. However, we could not get any proper result. Our experiment here showed that the traditional clustering algorithms are not suitable for identifying different traffic characteristics of a single application.

ACKNOWLEDGEMENTS

This research was partially supported by the MSIP, Korea, under the ITRC support program supervised by the NIPA (NIPA-2013-H0301-13-3002), World Class University program funded by the MEST through the NRF of Korea (R31-10100), and the research fund of Hanyang University (HY-2012).

REFERENCES

1. Moore AW, Papagiannaki K. Toward the accurate identification of network applications. In *Proceedings of Passive and Active Network Measurement*, Boston, MA, March 2005; 41–54.

2. Park B, Won YJ, Hong JW-K. Towards fine-grained traffic classification. *Communications Magazine, IEEE* 2011; **49**: 104–111.
3. IANA. Iana port numbers list. Available: http://www.iana.org/ assignments/port-numbers [8 August 2012].
4. Karagiannis T, Broido A, Faloutsos M, Claffy K. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, Taormina, Sicily, October 2004; 121–134.
5. Sen S, Spatscheck O, Wang D. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th International Conference on World Wide Web*, New York, May 2004; 512–521.
6. Park BC, Won Y, Kim MS, Hong J. Towards automated application signature generation for traffic identification. In *Network Operations and Management Symposium*. IEEE, April 2008; 160–167.
7. Haffner P, Sen S, Spatscheck O, Wang D. Acas: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data, MineNet â€™05*. ACM: New York, 2005; 197–202.
8. Kim H-A, Karp B. Autograph: toward automated, distributed worm signature detection. In *Proceedings of the 13th Usenix Security Symposium*, UASA, San Diego, CA, August 2004; 271–286.
9. Karagiannis T, Papagiannaki K, Faloutsos M. Blinc: multilevel traffic classification in the dark. In *Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Philadelphia, PA, August 2005; 229–240.
10. Iliofotou M, Pappu P, Faloutsos M, Mitzenmacher M, Singh S, Varghese G. Network monitoring using traffic dispersion graphs. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, San Diego, CA, October 2007; 315–320.
11. Kim SS, Reddy A. Image-based anomaly detection technique: algorithm, implementation and effectiveness. *IEEE Journal on Selected Areas in Communications* 2006; **24**: 1942–1954.
12. Iliofotou M. Exploring graph-based network traffic monitoring. In *Proceedings of the 28th IEEE International Conference on Computer Communications Workshops, INFOCOMâ€™09*. IEEE Press: Piscataway, NJ, 2009; 353–354.
13. Frank J. Machine learning and intrusion detection: current and future directions. In *National 17th Computer Security Conference*, Washington, DC, October 1994.
14. Nguyen T, Armitage G. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys Tutorials, IEEE* 2008; **10**(4): 56–76.
15. Mcgregor A, Hall M, Lorier P, Brunskill J. Flow clustering using machine learning techniques. In *Proceedings of the 5th International Workshop on Passive and Active Network Measurement*, 2004; 205–214.
16. Moore AW, Zuev D. Internet traffic classification using Bayesian analysis techniques. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '05*. ACM: New York, 2005; 50–60.
17. Roughan M, Sen S, Spatscheck O, Duffield N. Class-of-service mapping for qos: a statistical signature-based approach to IP traffic classification. In *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement, IMC '04*. ACM: New York, 2004; 135–148.
18. Erman J, Arlitt M, Mahanti A. Traffic classification using clustering algorithms. In *Proceedings of the ACM SIGCOMM Workshop on Mining Network Data*, Barcelona, August 2006; 281–286.
19. Auld T, Moore AW, Gull SF. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks* 2007; **18**(1): 223–239.
20. Crotti M, Dusi M, Gringoli F, Salgarelli L. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review* 2007; **37**(1): 5–16.
21. Williams N, Zander S, Armitage G. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM Computer Communication Review* 2006; **30**: 7–15.
22. Thompson K, Miller G, Wilder R. Wide-area internet traffic patterns and characteristics. *Network, IEEE* 1997; **11**: 10–23.
23. Moore D, Keys K, Koga R, Lagache E, Claffy KC. The coralreef software suite as a tool for system and network administrators. In *Proceedings of the 15th USENIX conference on System administration*, San Diego, CA, December 2001; 133–144.
24. Claffy K. Internet traffic characterization. PhD thesis, University of California, San Diego, CA, 1994.
25. Ma J, Levchenko K, Kreibich C, Savage S, Voelker GM. Unexpected means of protocol inference. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, IMC â€™06*, New York, 2006; 313–326.
26. Cho K, Fukuda K, Esaki H, Kato A. Observing slow crustal movement in residential user traffic. In *Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT â€™08*. ACM: New York, 2008; 12:1–12:12.
27. Soule A, Salamatia K, Taft N, Emilion R, Papagiannaki K. Flow classification by histograms: or how to go on safari in the internet. *SIGMETRICS Performance Evaluation Review* 2004; **32**: 49–60.
28. Lan K-C, Heidemann J. A measurement study of correlations of internet flow characteristics. *Computer Networks* 2006; **50**: 46–62.
29. Iliofotou M, Pappu P, Faloutsos M, Mitzenmacher M, Varghese G, Kim H. Graption: automated detection of p2p applications using traffic dispersion graphs. Technical report UCR-CS-2008-06080, University of California, Riverside, CA, June 2008.
30. Zuev D, Moore AW. Traffic classification using a statistical approach. *Passive and Active Network Measurement* 2005; **3431**(3): 321–324.
31. Erman J, Mahanti A, Arlitt M, Cohen I, Williamson C. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation* 2007; **64**: 1194–1213.

DOI: 10.1002/nem

32. Erman J, Mahanti A, Arlitt M, Williamson C. Identifying and discriminating between web and peer-to- peer traffic in the network core. In *Proceedings of the 16th International Conference on World Wide Web, WWW â€™07*. ACM: New York, 2007; 883–892.
33. Iliofotou M, Pappu P, Faloutsos M, Mitzenmacher M, Singh S, Varghese G. Network traffic analysis using traffic dispersion graphs (tdgs): techniques and hardware implementation. Technical report UCR-CS-2007-05001, University of California, Riverside, CA, 2007.
34. John W, Tafvelin S. Heuristics to classify internet backbone traffic based on connection patterns. In *International Conference on Information Networking*, January 2008; 1–5.
35. John W, Tafvelin S, Olovsson T. Trends and differences in connection-behavior within classes of internet backbone traffic. In *Proceedings of the 9th International Conference on Passive and Active Network Measurement, PAM '08*. Springer: Berlin, 2008; 192–201.
36. Pietrzyk M, Costeux J-L, Urvoy-Keller G, En- Najjary T. Challenging statistical classification for operational usage: the ADSL case. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference, IMC â€™09*. ACM: New York, 2009; 122–135.
37. Karagiannis T, Broido A, Brownlee N, Claffy K, Faloutsos M. Is p2p dying or just hiding? [p2p traffic measurement]. *In Proceedings of IEEE Global Telecommunications Conference (GLOBECOM)*, Vol. 3, Dallas, TX, 2004; 1532–1538.
38. Bartlett G, Heidemann J, Papadopoulos C. Inherent behaviors for on-line detection of peer-to-peer file sharing. In *IEEE Global Internet Symposium*, May 2007; 55–60.
39. Bonfiglio D, Mellia M, Meo M, Rossi D, Tofanelli P. Revealing Skype traffic: when randomness plays with you. *SIGCOMM Computer Communication Review* 2007; **37**: 37–48.
40. Constantinou F, Mavrommatis P. Identifying known and unknown peer-to-peer traffic. In *Fifth IEEE International Symposium on Network Computing and Applications*, July 2006; 93–102.
41. Plissonneau L, Costeux J-L, Brown P. Analysis of peer-to-peer traffic on ADSL. In *Passive and Active Measurement Workshop*, Boston, MA, March 2005; 69–82.
42. Bartlett G, Heidemann J, Papadopoulos C, Pepin J. Estimating P2P traffic volume at USC. Technical report ISI-TR-645, University of Southern California, 2007.
43. Duffield N, Lund C. Predicting resource usage and estimation accuracy in an IP flow measurement collection infrastructure. In *Proceedings of the 3rd ACM SIGCOMM Conference on Internet Measurement, IMC '03*. ACM: New York, 2003; 179–191.
44. Suh K, Figueiredo DR, Kurose J, Towsley D. Characterizing and detecting SKYPE-relayed traffic. In *Proceedings of the 25th IEEE International Conference on Computer Communications, INFOCOM '06*, Barcelona, April 2006; 1–12.
45. Kim H, Claffy K, Fomenkov M, Barman D, Faloutsos M, Lee K. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT '08*, Madrid, December 2008; 1–12.
46. Szabo G, Szabo I, Orincsay D. Accurate traffic classification. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM '07*, June 2007; 1–8.
47. Deep packet inspection (DPI): U.S. government market forecast 2013–2018. Available: http://www.marketresearchmedia.com/ [20 August 2012].
48. Salton G, Buckley C. Term weighting approaches in automatic text retrieval. Technical report, Cornell University, Ithaca, NY, 1987.
49. Salton G, Wong A, Yang C-S. A vector space model for automatic indexing. *Communications of the ACM* 1975; **18**(11): 613–620.
50. Chung JY, Park B, Won YJ, Strassner J, ki Hong JW. An effective similarity metric for application traffic classification. In *Proceedings of the 12th IEEE/IFIP Network Operations and Management Symposium, NOMS 2010*, Osaka, May 2010; 286–292.
51. Erman J, Mahanti A, Arlitt M. Byte me: a case for byte accuracy in traffic classification. In *Proceedings of the 3rd Annual ACM Workshop on Mining Network Data, MineNet '07*. ACM: New York, 2007; 35–38.
52. Bernaille L, Teixeira R, Salamatian K. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference, CoNEXT '06*. ACM: New York, 2006; 6:1–6:12.
53. Lim Y-S, Kim H-C, Jeong J, Kim C-K, Kwon TT, Choi Y. Internet traffic classification demystified: on the sources of the discriminative power. In *Proceedings of the 6th International Conference, Co-NEXT '10*. ACM: New York, 2010; 9:1–9:12.
54. L7-filter. Available: http://l7-filter.clearfoundation.com/ [8 August 2012].
55. Opendpi. Available: http://www.opendpi.org/ [8 November 2011].
56. Ipoque. Available: http://www.ipoque.com/ [8 November 2011].
57. DISTIMO. The battle for the most content and the emerging tablet market. http://www.distimo.com [8 August 2012].
58. Bernaille L, Teixeira R, Akodkenou I, Soule A, Salamatian K. Traffic classification on the fly. *SIGCOMM Computer Communication Review* 2006; **36**: 23–26.
59. Dash M, Liu H. Feature selection for classification. *Intelligent Data Analysis* 1997; **1**: 131–156.
60. Guyon I, Elisseeff A. An introduction to variable and feature selection. *Journal of Machine Learning Research* 2003; **3**: 1157–1182.

DOI: 10.1002/nem

61. Hall M, Holmes G. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions on Knowledge and Data Engineering* 2003; **15**: 1437–1447.
62. Kira K, Rendell LA. A practical approach to feature selection. In *Proceedings of the Ninth International Workshop on Machine Learning, ML '92*. Morgan Kaufmann: San Francisco, CA, 1992; 249–256.
63. Kononenko I. Estimating attributes: analysis and extensions of relief. In *Proceedings of the European Conference on Machine Learning on Machine Learning*, Secaucus, NJ. Springer: New York, 1994; 171–182.
64. Robnik-Šikonja M, Kononenko I. Theoretical and empirical analysis of relieff and rrelieff. *Machine Learning* 2003; **53**: 23–69.
65. Sander J, Ester M, Kriegel H-P, Xu X. Density-based clustering in spatial databases: the algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery* 1998; **2**: 169–194.
66. Zander S, Nguyen T, Armitage G. Automated traffic classification and application identification using machine learning. In *Proceedings of the The IEEE Conference on Local Computer Networks 30th Anniversary, LNC 2005*. Sydney, November 2005; 250–257.

AUTHORS' BIOGRAPHIES

Byungchul Park is a postdoctoral researcher at INRIA Sophia Antipolis-Méditerranée, France. Prior to INRIA, he was a postdoctoral researcher at Division of IT Convergence Engineering, POSTECH, Korea. He received his B.S. (2006) and Ph.D. (2013) in Computer Science and Engineering from POSTECH, Korea. His main research interests are network measurement and traffic analysis, content-centric networking.

Youngjoon Won is an assistant profesor at Hanyang University, Seoul, Korea. Prior to Hanyang University, he was a researcher at IIJ Research, Japan and a postdoctoral researcher at INRIA, France. He received his B. Math (2003) from the University of Waterloo, Canada, and M.S. (2006) and Ph.D. (2010) from POSTECH.

Jae Yoon Chung is a Ph. D. candidate in the Department of Computer Science and Engineering at POSTECH. He also received his B.S. in the Department of Computer Science and Engineering at POSTECH. His research interests include network traffic measurement and classification.

Myung-Sup Kim received his B.S., M.S., and Ph.D. degree in Computer Science and Engineering from POSTECH, Korea, in 1998, 2000, and 2004, respectively. From September 2004 to August 2006, he was a postdoctoral fellow in the Department of Electrical and Computer Engineering, University of Toronto, Canada. He joined Korea University, Korea, in September 2006, where he is currently working as an associate professor in the Department of Computer and Information Science. His research interests include Internet traffic monitoring and analysis, service and network management, and Internet security.

James Won-Ki Hong is currently the Chief Technology Officer and Senior Executive Vice President for KT (Korea Telecom) since March 2012, where he is responsible for leading the R&D effort of KT and its subsidiary companies. He is Chairman of National Intelligence Communication Enterprise Association, and Chairman of ICT Standardization Committee in Korea. His interests include network innovation, such as software-defined networking (SDN), cloud computing, mobile services, IPTV, ICT convergence technologies (e.g., Smart Home, Smart Energy, Healthcare), big data analytics and intelligence. Before taking the role of CTO at KT, he had worked at POSTECH for 17 years as a professor including Head of Dept. of Computer Science and Engineering, Dean of Graduate School for Information Technology, Director of POSTECH Information Research Labs (PIRL) and Head of the Division of IT Convergence Engineering. He was also co-founder and CTO of Netstech, a Palo Alto, USA-based startup developing network integrated ultra-dense, blade servers from 2000 to 2002. Over the past 20 years, James has been an active volunteer in various committees in IEEE, ComSoc, and KICS. He has served as Steering Committee Chair of IEEE NOMS, IM and APNOMS, as well as Chair of CNOM and KNOM. He has also been serving as EiC of Wiley's International Journal of Network Management (IJNM) and ComSoc Technology News (CTN) as well as an editorial member of the IEEE TNSM, JNSM and JCN. James received his HBSc and MSc degrees in Computer Science from the University of Western Ontario, Canada in 1983 and 1985, respectively, and the Ph.D degree in Computer Science from the University of Waterloo, Canada in 1991.

DOI: 10.1002/nem