PROVIDING QUALITY-OF-SERVICE SUPPORT TO LEGACY APPLICATIONS USING MACHINE LEARNING

Isara Anantavrasilp
Department of Computer Science, Technische Universität Dresden
Dresden, Germany
s9847947@inf.tu-dresden.de

Thorsten Schöler
Corporate Technology, Information and Communication, Siemens AG
Munich, Germany
thorsten.schoeler@siemens.com

ABSTRACT

Quality-of-Service (QoS) support is an essential feature in recent and upcoming networking standards. Applications running on these networks can specify appropriate service classes for their connection flows so that the flows are treated accordingly. However, current internet applications (legacy applications) cannot benefit from this facility as they are designed using the best-effort scheme. Effective QoS support to applications with unspecified service classes can be provided through our proposed intelligent QoS Manager.
This paper describes the important features that can be used to determine service classes and discusses how machine learning can be incorporated into a QoS Manager, enabling it to distinguish different types of application flows and assign them appropriate service classes. Experiments using 10-fold cross-validation (CV), 33% hold-out (HO) and the learner's specific features lead to the selection of PART (Frank & Witten, 1998) as the classifier of the framework. It achieves 91.55% and 93.29% prediction correctness in CV and HO experiments respectively.

KEYWORDS

Quality-of-Service, legacy applications, network, data mining, machine learning

1. INTRODUCTION

The differentiated services (DiffServ) QoS model (Blake et al., 1998) is supported by recent and upcoming network standards such as IPv6, UMTS, WiMAX or IEEE 802.11e. The application software in a network node can specify an appropriate service class to its network connection or flow. The packets within the flows are then marked with the "service class label" indicating the flow's type of service. As a consequence, the network will treat each packet differently according to its service class. This way, packets requiring, for instance, lower transfer delay or higher throughput will be served first, whereas packets that can sustain higher delay or less speed will be put on hold or even discarded.

Effective management of the QoS within the network requires the application software on the end-device to specify the required service quality level or "service class". We will call such applications "QoS-aware applications". In contrast, the commonly used applications at present, such as non-QoS-aware applications or legacy applications, are designed based on the best-effort scheme. They do not specify QoS requirement and hence cannot benefit from the QoS support from the network.

We thus propose an automatic QoS classification framework, which consists of an intelligent QoS Manager that can automatically assign proper service classes to flows from legacy applications, in order to provide support for legacy applications in a QoS-aware network. Since it is not pre-programmed, the assignment of service classes can be done regardless of the applications installed on the network device or the networks on which the device is operating. Moreover, the QoS Manager, located between the network layer and the data link layer, is able to learn the application's flow service class by itself. It observes the characteristics or "features" of the flows opened by QoS-aware applications and tries to associate them with the requested service class. The QoS Manager subsequently uses this knowledge to classify the flows from the legacy applications.

This paper describes the relevant features in the service class prediction. It also discusses how a machine learning technique can be incorporated into the end-device QoS Manager, so that the different types of connections can be classified and the appropriate service class can be assigned accordingly. Because of the self-learning ability, the proposed framework solution is not restricted to any QoS model and should work in any current or future networks that implement differentiated services or any service class-based QoS schemes.

The organization of the paper is as follows. Section 2 discusses current solutions to legacy flows. Section 3 gives the overview of the classification framework, followed by the introduction of service classes and their definitions in Section 4. In Section 5, the flow features and how they are extracted from the data flow are described. Section 6 explains machine learning methods and how they can be used in our domain. The evaluation of those algorithms is discussed in Section 7. Finally, Section 8 concludes the paper and presents some areas for future research.

2. RELATED WORKS

A proxy-based framework introduced in Tsetsekas et al. (2001) identifies the flows' QoS requirements at proxy servers according to their protocols. For a flow with an unknown protocol, a preliminary resource reservation is given, after which its resource usage will be periodically measured to provide a more accurate service quality for the flow. The knowledge of service quality requirements, however, is specific to that particular flow. In Miao et al. (2002), a QoS request can be set up for legacy applications without modification to the source code by employing the QoS Library Redirection. This library redirection is done by rewriting the communication library. When legacy applications open connections through the library, appropriate service qualities are assigned to the flow. Thus, QoS support can be added to the library instead of modifying the applications. Still, the QoS requirement for each application type must be specified beforehand.

QoS support can also be realized by user assistance. In Dharmalingam & Collier (2002), a manager node is introduced into the network in order to interact and receive the QoS-support request for each application as well as appropriate "application type" directly from the user. The so-called Transparent QoS Mechanism is introduced by Shih et al. (2004). The mechanism, residing in the end-device, acts as a middleware that issues QoS request to the network on behalf of the application. The flows' QoS parameters are based on either user specifications or, for some support application types, statistical estimations.

While the aforementioned works are developed from different approaches, none can automatically classify the class of each flow without human assistance or predefined QoS assignment rules. Moreover, the statistical methods employed by the existing models do not provide any facility that allows the transfer of available resource usage knowledge to other unknown flows. The automated QoS classification framework we propose in this paper should therefore represent an innovation in the field, as it is the first framework that can accurately classify and assign correct service classes to the unseen and unknown non-QoS-aware flows through a machine learning method.

3. THE CLASSIFICATION FRAMEWORK

The goal of our classification framework is to enable a QoS management unit of a network device to assign a proper service class to any flow from any application by itself. The main challenges are that arbitrary, different applications are installed in different devices and that the devices can be equipped with more than one network interfaces or protocol stacks (which could be arbitrary). These make static rule-based classification impossible, requiring the framework to be able to work on arbitrary networks and with any kind of applications. Hence, our framework is designed to be independent from any device operating systems, applications, networks, QoS models or signaling protocols. Since the work we propose is focused only on flow classification, we assume that the underlying network is QoS-aware and that typical QoS management tasks, such as QoS negotiation, provision and policing, are carried out by the network protocol stack.

As shown in Figure 1, there are two sets of applications, namely the QoS-aware and legacy applications. When the QoS-aware application opens a flow, the flow classifier attaches a service class label to each packet of the flow according to the requested service level. Subsequently, the context manager observes the connection characteristics, extracts features, and stores these features along with the flow's service class into the feature database. The learner component then uses these data to find the relationship between flow features and flow service classes. Consequently, the flow classifier can use the knowledge (which, in our implementation, is encoded as a set of rules) to classify the legacy applications' flows. This classification can be done online in real-time.

It is possible to distinguish the legacy and QoS-aware flows from one another via dedicated flow-opening calls. A QoS-aware flow is opened with specified QoS requirements, whereas a legacy flow is opened using a non-QoS-aware API such as Winsock 1.0 without any specified QoS.

In our framework, the context manager employs WinPcap (Risso & Degioanni, 2001), which is Microsoft Windows port of UNIX's pcap library. WinPcap is written as a NDIS protocol stack and can transparently capture virtually every packet data coming in or out of the network interface. However, the captured data are packet-wise, meaning that WinPcap captures and forwards the packet data into the context manager packet by packet regardless of the flow to which it belongs. The context manager itself has to sort packets into designated flows, arranging them in relation to their remote and local addresses and ports.

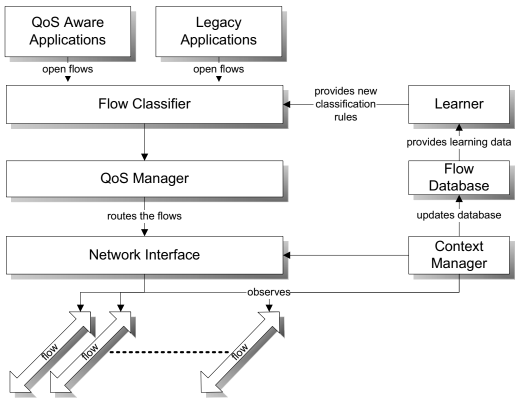

Figure 1. QoS manager architecture

4. SERVICE CLASSES

The definition of service classes follows the classes suggested in 3GPP (2003). However, the four classes proposed there are not enough to capture all the types of relevant connections. More specifically, within the conversational class, an audio/video conversation requires much higher data rate and is more sensitive to delay variation than data conversation (e.g. remote desktop or instant messaging). Therefore, splitting it into strict and relaxed conversational classes would yield better classification results. The background class is not considered because legacy applications are designed based on a best-effort scheme, whereby all services behave according to their true nature. Consequently, there is no true "background" service in such scheme and it would not make sense to observe background-service behaviors from legacy applications. The relevant service classes here are as follows:

- *Strict conversational class* Real-time audio/video applications and symmetric applications that are sensitive to delay and delay variation, require relatively high data rate and are tolerable to transmission errors.
- *Relaxed conversational class* This class of applications is quite similar to the previous class. These applications – including Telnet, interactive games and instant messaging – are real-time and symmetric, and require low transmission delays. They, however, have much lower bandwidth requirements and are less sensitive to delay variation as well as to intolerable to transmission error.

- *Streaming class* As the name suggests, the streaming class includes services that serve streams of data such as online audio/video streaming and data transfers (FTP or file download). Both media streaming and data transfers are categorized in 3GPP (2003) as one group. However, it is important to recognize that they are fundamentally different. While both of them are not sensitive to delay, streaming media might require specific minimum service quality levels (such as data rate and jitter) to properly operate whereas data transfers might not. Nonetheless, most streaming applications do behave like file transfer and often use TCP as transport protocol or even encapsulate packets in HTTP similar to file transfer protocols. We thus categorize both media streaming and file transfer into the same class, although it might lead to overestimating the service quality requirements of file transfer connections. Feasibility and advantages of separating these two types of connections will be investigated in our future works.
- *Interactive class* All server access applications fall into this class. The main characteristic of these applications is the request-and-response manner – that is, the client application requests a service from a server and then the server responds. Some examples of this class of applications are web browsers, email clients, and the like. This kind of service is asymmetric, non-delay sensitive, and independent of high bandwidth. The main requirement, however, is error intolerance.

It is clear at this point that the service classes are distinguished by four main QoS parameters – namely, delay sensitivity, delay variation tolerability, data rate and error tolerance (i.e. bit-error rate). However, most of these parameters are impossible to measure directly. For instance, the required delay or delay variation of a flow cannot be determined unless they are specified. The classification of the flows hence requires other indirect features in order to distinguish their different types. These features are discussed in the next section.

5. OBSERVABLE FEATURES

Since it is impossible to distinguish connection types by observing the specified service quality directly, one alternative is to identify other features, which are not the specified service quality parameters but can be used to discriminate the service classes. Such features are called the observable features.

5.1 Separating Service Classes

In order to find the observable features that can separate the connection types from each other, it is necessary to first understand the nature of the flows. The strict and relaxed conversational classes share some characteristics such as symmetry (both sides of connection communicate to each other equally) and connection time (e.g. a conversation would take longer than loading a web page) but they are different in error tolerance and data volume. Strict conversational flow consumes much more bandwidth and is tolerable to errors, while relaxed conversational class behaves diametrically. Therefore, we believe that the transport protocol and data volume could be the key differences between them.

Streaming is typically carried out in a request-and-response manner. Once the connection is established (either an online video broadcast or a simple file download), the client does not send anything back to the server apart from TCP packet acknowledgements. This is also true for the interactive class. We can thus use this asymmetry feature to separate streaming and interactive classes from the conversational classes. The streaming and interactive classes can in turn be distinguished from each other by data volume and burstiness. A streaming flow is usually sent in a non-bursty manner and its data volume is much higher than that of an interactive one (e.g. online video versus web browsing). In an interactive connection flow, on the other hand, the response from the server is small and short and the request-and-response sequence occurs more often, making the connection appear bursty in contrast to the steady data rate of the streaming class.

5.2 Capturing the Flow Characteristics

Table 1. Observable features

Feature Description	Possible Value	Data Type
Transport Protocol	TCP / UDP	Nominal
Remote Port	TCP/UDP Port	Integer
Connection Time	Millisecond	Integer
Traffic (Data Volume, Number of Packets) (Total, Sent, Received,	Bytes /	Integer / Real
Sent:Received Ratio)	No. of Packets	Integer / Real
Throughput (Data Volume, Number of Packets) (Total, Sent,	Bytes /	Integer / Real
Received) (Peak / Average / Energy / Area Above Mean)	No. of Packets	Integer / Real

The process of capturing flow characteristics prompts us to explore various features ranging from those that are directly observable from the packet headers, such as IP addresses and ports, to the ones that have to be further processed such as connection throughput or traffic volume of a particular flow. The features of the former type that can be observed directly are called "raw features" and the features of the latter type that require preprocessing are called "extracted features". In other words, the raw features are processed to obtain the extracted features. We summarize a total of 35 features in Table 1. These features are predetermined to reflect the following characteristics: error tolerance, data volume (or traffic), connection time, symmetry and burstiness. Error tolerance can be detected by the flow's transport protocol. Connection time and data traffic are directly measurable, while connection symmetry can be determined by comparing the amount of incoming and outgoing traffic. Burstiness is, however, more complicated. The QoS manager has to measure the data rate in real-time in order to get the sense of burstiness, since traditional network monitoring tools such as records of average and peak speed are not sufficient to determine burstiness. This requires an identifier for "bursts", for which we employ the mathematical concepts of *energy* and *area above mean*. The energy of a flow is first calculated by squaring up the recorded throughput so that the peaks become more noticeable. The sum of squared peaks is then used as one of the features. The area above mean is calculated by summing up the areas above the average throughput. The more bursty flows will have more area above the average throughput than non-bursty flows. Nevertheless, the values of both energy and area above mean depend not only on the traffic characteristics, but also on the overall data volume and connection time. Short and bursty traffic, for example, will have significantly lower energy or area above mean than long and constant one. Therefore, both values are to be normalized first by dividing them with the average throughput. Furthermore, throughput and traffic are measured in both incoming and outgoing directions to get an extra indication of the flow's symmetry.

6. MACHINE LEARNING ALGORITHMS

Machine learning is a set of methods that are trained to describe structural pattern in a set of data, where each instance in the data set is a list of features extracted from each collected example. To be precise, let X be a set of data instances, and $f(x)$ be a function that maps the instance to a target value of set C (that is, $f : X \rightarrow C$). The goal of machine learning is to come up with a function $g: X \rightarrow C$ where $g(x) = f(x)$ for all $x \in X$. In our domain, the data set X is the set of collected flow behaviors and C the set of service classes. The algorithm can be trained by presenting a set of *training examples*; each is a pair of an instance and the target value: $(x, f(x))$ where $x \in X$. Learning with a pair of an example and a target value is called "supervised learning", which suits our task perfectly. Here the target values are service classes (in contrast to "unsupervised learning" where $f(x)$ is not given in the example).

The collected flow characteristics in our domain are the training examples, modeled as a vector of features, while the flows' service classes are modeled as the target value. Following is a simplified version of a training example:

```
(<Protocol=TCP, ConnTime=2326, RmtPort=80, DataVolumeIn=50382, DataVolumeOut=9540>,
                              ServiceClass=interactive)
```

Because the learner's task is essentially to learn the function that maps the flow characteristics to a service class, we only focus on supervised learning algorithms in the present paper. Moreover, since the learner must be integrated in the QoS manager, we are looking for algorithms whose learned classification models are easy to interpret and process. Algorithms that can be transformed into IF-THEN rules are the most preferable since such rules can be easily interpreted and implemented into the flow classifier. Therefore, we do not seek to evaluate all available supervised learners and focus instead only on the ones that meet the aforementioned criterion.

We have examined a wide range of learning algorithms with different biases, including decision tree learners, rules generators and Bayesian classifier. Decision tree algorithms work from top-down, seeking the best attribute to separate the classes at each node in the tree. This means that the shortest tree is assumed to be the best and always preferred. For decision trees, we have evaluated two algorithms: J4.8, an implementation variant of the classical C4.5 (Quinlan, 1993), and Random Forest (Breiman, 2001). Rule generators, in contrast to the decision tree learners, concentrate on one class at a time. In general, they try to discover the rules that cover as many instances in a class as possible, while excluding as many instances of the other classes as possible. Two rule generator algorithms, RIPPER (Cohen, 1995) and PART (Frank & Witten, 1998) are considered here. Lastly, the Naïve Bayes classifier (John & Langley, 1995) also features in our experiments by virtue of its simplicity and speed. It employs Bayes theorem and "naïvely" assumes that all attributes are not independent from one another.

7. MACHINE LEARNING ALGORITHMS EVALUATION

In this section, the evaluation of various machine learning algorithms is described. We will begin with a discussion of the data set used in the experiments, followed by the experiment strategy and results..

7.1 The data Set

The data set contains 438 flow samples, which are equally distributed into different classes and contain features of a flow as shown in Table 1. The samples are collected from various types of real-world connections from a variety of networks including Ethernet, 802.11b and 802.11g Wireless LAN. All the data are from FlowStat, an advanced network monitoring tool based on WinPcap, which is developed exclusively for the task and can capture the packets of each flow directly at the driver level. Moreover, we include various types of connections from a large variety of applications to ensure that the classifier learn from the characteristics of a service class that are really common to that particular class. For the strict conversational class, the samples are collected from the flows belonging to, for instance, video conference applications (MSN and Yahoo messenger) and interactive online games (Half-Life 2, Gunbound). For the relaxed conversational class, flow data come from remote desktop applications (VNC, Telnet), instant messaging applications (MSN, Yahoo Messenger) and interactive GIS software (Google Earth). The flow samples of the streaming class come from various streaming and file transfer applications. The media streaming data are collected from both media player software, such as Winamp, and web browsers. FTP software (NetTransport), web browsers and software updates (Half-Life 2, Gunbound) provide the sources for file transfers data. Finally, for the interactive class, we gather flows opened by web browsers such as Firefox, Internet Explorer and Opera, as well as Email clients such as Thunderbird. The diversity of software in our experiments ensures that the data set is not biased and that the characteristics are learned from the *type* of applications, not *individual* applications.

One could argue that the size and diversity of the data may not be essential characteristics in an actual implementation, especially when the features are discriminative. Nevertheless, they can prove to be useful in the evaluation of the learning algorithms, as the larger and more diverse data set would improve the accuracy of the classifier by exposing it to more information and variety.

7.2 Experiment Strategy

Evaluations of each method are conducted in three phrases: 10-fold cross validation (Kohavi, 1995; Stone, 1997), 33% hold-out and test against the training set itself. Generally, machine learning algorithms use one set of data, called the “training set”, to train the algorithms and then compare the results to another unseen set of data, the “test set”, to evaluate the prediction correctness of the algorithms. In our experiments, we split our data such that two-thirds of the samples are used as the training set and the rest as the test set. This is called a 33% hold-out test. However, a hold-out test is expensive as one-third of data cannot be used to train the algorithms and the sample used for training might not be representative or relevant. (One can also increase the size of the training set, but the test-set could become too small and lead to inaccurate results.) The cross-validation technique is therefore employed to reduce the bias caused by hidden data. In cross-validation, one selects a number of folds or partitions of the data, denoted by *k*. The data are subsequently divided into *k* equal partitions. Next, each data partition is held out to be used as the test set while the rest are used as the training set. This process is repeated until every partition is used as test set (i.e. *k* iterations). As a result, each learning algorithm is evaluated *k* times over *k* different sets. The correctness from all iterations is then averaged to find the overall accuracy. In our evaluation, *k* is set to 10 as it tends to produce the best accuracy estimation (Kohavi, 1995).

The learning algorithms are normally evaluated only by hold-out or cross-validation procedures. However, in the real implementation, the selected technique will be trained by the whole data set to maximize the classification ability, and hence it does make sense to evaluate the learned model using the training set itself to see the maximum classification ability. Additionally, since some of the features are numeric, feeding is preceded by discretization for algorithms that cannot directly handle numerical values.

We conduct the experiments on WEKA, an open source data mining software (Witten & Frank, 2005), as the evaluation processes are automated and the integration of WEKA and learned classification models into other projects is easy.

7.3 Experiment Results

Table 2 presents the experiment results, showing the comparison of the classification correctness of each method. The first column of the table indicates the method. The second, third and last columns indicate the cross-validation, hold-out and training set evaluation results, respectively.

Table 2. Experiment results

Method	CV	HO	TS
J4.8	92.92%	92.62%	97.03%
Random Forest	93.15%	88.59%	99.77%
PART	91.55%	93.29%	98.86%
RIPPER	91.32%	88.59%	97.03%
Naïve Bayes	83.79%	85.23%	87.67%

Every learning method produces excellent results, especially Random Forest, which achieves the highest correctness percentage in cross-validation test. All learners obtain rather low correctness in the hold-out tests but still remain on an acceptable level. In the training set test, Random Forest again yields a distinctive result by predicting almost all examples correctly. However, this could be the effect of overfitting, especially upon comparison of the results of the hold-out tests. At any rate, the exceptional performances from all learners show that the exploited features have discrimination power. In contrast, J4.8 and PART perform consistently well in all tests and produce a rule set that can be used directly in the flow classifier. PART is more preferable, however, as the rule set generated by PART is more compact than the tree generated by J4.8. The reason for this is that PART is explicitly designed to produce a compact rule set (Frank & Witten, 1998). Figures 2 and 3, respectively, illustrate the results of classification models of J4.8 and of PART over the 33% hold-out test set. Note that the rule set generated by PART is a “decision list”, which has to be executed in order such that the rules order cannot be shuffled.

The decision tree in Figure 2 and decision list in Figure 3 show that each type of flows behaves as expected and the predetermined features have the ability to express their characteristics. Consider, for instance, the root node of the tree in Figure 2. It indicates that all connections of strict conversation class ultilize UDP protocol, which is supported by the fact that they are error tolerant and require high data rate. This is also confirmed by rule 4 of the decision list in Figure 3. Following the right branch of the root, the ratio of outgoing and incoming data volumes is then tested. The lower ratio (i.e. lower outgoing data, comparing to incoming ones) leads to streaming and interative classes. Rule 1 of the decision list also agrees with this.

In our experiments, the misclassified instances come mostly from the streaming flows with fluctuated traffic, leading to their misclassification into the interactive class. These are usually file transfer flows. Conversational flows that are somehow fluctuated, albeit rare, could be misclassified as interactive as well.

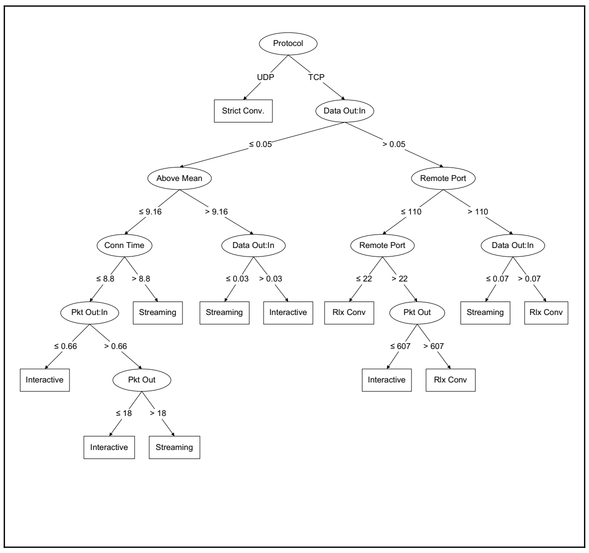

Figure 2. J4.8 decision tree

1 Protocol = TCP ∧ DataOut:In ≤ 0.046768 ∧ AboveMean ≤ 9.157046 ∧ ConnTime > 8.8→ **Streaming**

2 Protocol = TCP ∧ RemotePort > 110 ∧ DataOut:In > 0.070932→ **Relaxed Conv**

3 Protocol = TCP ∧ PktOut ≤ 187 ∧ TPUTDataInPeak ≤ 28128→ **Interactive**

4 Protocol = UDP→ **Strict Conv**

5 RemotePort > 22 ∧ DataVolTotal ≤ 1552134 ∧ Remote Port ≤ 554 ∧ TPUTPktInAvg ≤ 3.41 ∧ TPUTDataTotalPeak ≤ 77538→ **Interactive**

6 Port > 22 ∧ DataVolTotal ≤ 1552134 ∧ Port ≤ 554 ∧ TPUTDataOutAvg ≤ 2286.05 ∧ EnergyPktOut ≤ 600 ∧ EnergyDataIn > 1080000→ **Streaming**

7 Conn Time > 70.16 ∧ Port ≤ 554→ **Relaxed Conv**

8 Port ≤ 554 ∧ TPUTDataOutPeak > 1836 ∧ TPUTDataOutAvg > 308.96→ **Interactive**

9 EnergyDataIn ≤ 1020000→ **Streaming**

10 → Interactive

Figure 3. PART extracted decision list

8. CONCLUSION AND FUTURE WORKS

In this paper, we propose an automated QoS classification framework that employs machine learning to classify the service classes of legacy flows. We study the relevant features, which can be exploited in flow classification, as well as how to extract them. The performances of different classifiers are also compared. The high level of accuracy achieved by all learners indicates that the predetermined features can be used to distinguish the service classes from each other.

The learning results, while encouraging, still have room for improvement. The learning framework operates only on a single device and the data can only be collected from a single user. This begs the question whether learning from and classifying data from multiple users is possible. Because of their dissimilarities, the feasibility and possibility of separating streaming and data transfer connections also need to be investigated. Furthermore, QoS policing can also be enhanced using the framework, as it can not only classify non-QoS-aware flows but also detect QoS-aware flows that behave differently from the requested service class.

ACKNOWLEDGEMENT

Isara Anantavrasilp would like to thank Surapa Thiemjarus, Arsalan Minhas and Tansir Ahmed for their valuable discussions and suggestions. This research is sponsored by Ernst von Siemens scholarship granted by Siemens AG.

REFERENCES

3GPP, 2003. TS 22.105: Services and service capabilities (Release 6). *Third Generation Partnership Project Technical Specification*. Third Generation Partnership Project.

Blake, S. et al., 1998. *RFC 2475: An architecture for differentiated services*. IETF.

Breiman, L., 2001. Random Forest, *In Machine Learning*, Vol. 45, pp. 5–32.

Cohen, W. W., 1995. Fast effective rule induction. *In Proceedings of the Twelfth International Conference on Machine Learning*.

Dharmalingam, K. and Collier, M., 2002. Transparent QoS support of network applications using Netlets. *In Lecture Notes in Computer Science*, Vol. 2521, pp. 206–215.

Frank, E. and Witten, I.H., 1998. Generating accurate rule sets without global optimization. *Proceedings of the Fifteenth International Conference on Machine Learning*.

John, G. H. and Langley, P., 1995. Estimating continuous distributions in Bayesian classifiers. *In Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345.

Kohavi, R., 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection*. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1137–1143

Miao, Y. B., et al., 2002. A transparent deployment method of RSVP-aware applications on Unix. *In Computer Network*, Vol. 40, pp. 45–56.

Quinlan, R., 1993. *C4.5: Programs for machine learning*, Morgan Kaufmann.

Risso, F. and Degioanni, L., 2001. An architecture for high performance network analysis. *Proceedings of Sixth IEEE Symposium on Computers and Communications*, pp. 686–693.

Shih, C. et al., 2004. A transparent QoS mechanism to support IntServ/DiffServ networks. *Proceedings of First IEEE Consumer Communications and Networking Conference*.

Stone, M., 1977. Asymptotics for and against cross-validation. *In Biometrika,* Vol. 64, pp. 29–35.

Tsetsekas, C. et al., 2001. Supporting QoS for legacy applications. *In Proceedings of the First International Conference on Networking - Part 2*.

Witten, I.H. and Frank, E., 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.