ELSEVIER

Computer Networks

Scalable Classification of QoS for Real-Time Interactive Applications from IP Traffic Measurements

Stuart E. Middleton* and Stefano Modafferi

University of Southampton IT Innovation Centre, Southampton, SO16 7NS, UK

Abstract

Measurement of network Quality of Service (QoS) has attracted considerable research effort over the last two decades. The recent trend towards Internet Service Providers (ISP's) offering application-specific QoS is creating possibilities for more sophisticated QoS metrics to be offered by ISP's in service level agreements. This in turn could be used for the purposes of improved network optimization and billing according to application specific QoS guarantees. We report a scalable near real-time approach using passively logging IP traffic data for classification of application latency and packet loss across a range of real-time interactive applications. We run six experiments involving Minecraft, Quake 3 Urban Terror, VLC video streaming and the commercial Wirofon VOIP application. We use a mixture of laboratory and real-world deployments, with run times ranging from hours to days, and observe a combination of real and simulated ISP latency and packet loss events. Our binary classification (i.e. classes 'OK' or 'lag') 10-fold cross validation F1 scores are between 0.80 and 0.93 depending on application type. Our multi-class classification (i.e. classes representing discrete packet loss or latency ranges) 10-fold cross validation F1 scores for Minecraft are 0.89 for latency and 0.90 for packet loss. With new business models between ISP's and application developers being actively considered this work represents a significant contribution to the debate by providing scientific evidence relating to a novel approach to scalable QoS measurement

Keywords: Classification;IP traffic;QoS;VOIP;video streaming;gaming

* Corresponding author. Tel.: +44 23 8059 8866.
E-mail address: sem@it-innovation.soton.ac.uk.

1. Introduction

Measurement of network quality of service (QoS) and work towards a QoS-enabled internet has attracted considerable research effort over the last two decades. Achieving a consistent QoS is particularly important for real-time interactive applications such as gaming, on demand video streaming and voice over internet protocol (VOIP) applications.

Measuring client-server QoS across the internet is not a simple task. Most real-world application network traffic latency and packet loss originate from the complex packet routing ISP's have to manage. Typically internet network paths between an application's client and server include home gateways provided by internet service providers (ISP's), ISP network backbones, ISP partner network backbones and data centre gateways. Current standard practice for ISP's is to offer guaranteed QoS to customers based on bandwidth and availability (i.e. uptime). Other network QoS is offered simply as 'best efforts' with no QoS measurement or guarantee. Measurement of QoS is performed by the ISP and is not transparent to customers, with no access to the raw data. Cloud data centres offer more in the way of network QoS measurement and guarantees but only when related to the internal network resource at the data centre.

A trend is emerging, however, towards application providers and ISP's working together to provide premium QoS for paying customers. For example the Australian ISP iiNet [2] has offered a premium service to home users which provides additional bandwidth for Netflix's on-demand video streaming application. Under such business models ISP's offer increased QoS for all network traffic routed to application servers over and above their usual 'best efforts' network limits. There are currently parallel debates in major regions such as the US and EU over how this might conflict with net neutrality principles, with a recent US FCC [5] [8] ruling in favour of net neutrality being contested by telecommunications companies via the US court system. However the desire by ISP's to offer application-specific QoS is becoming clear and this is opening up the possibility for more sophisticated QoS metrics better suited to the demanding needs of applications.

A number of approaches have been proposed for directly measuring network-level QoS beyond simple bandwidth and availability, focussing primarily on the metrics throughput, latency, packet loss and jitter, which are important to a wide range of real-time interactive applications. Some methods [15] [32] use end to end measurement, exploiting active probing and/or application specific protocols to measure traffic between client and server. This is not practical when scaled across different application types as each application client and server needs network monitoring software added. Others approaches [4] [13] require access to the network switches/gateways along the entire network path between client and server, tracking individual packet round trip times and/or monitoring inter-packet arrival times. This is not scalable for realistic network infrastructures consisting of multiple ISPs since it requires ISPs to agree and deploy a common measurement technology.

In addition to direct measurement of QoS a number of approaches have been proposed to classify internet protocol (IP) traffic. Classification of IP traffic has importance for intelligent routing of packets to achieve improved QoS, and identification of rouge IP traffic from sources such as malware and denial of service attacks. A variety of machine learning techniques [1] [36] have been successfully used, with typical approaches working on network observations ranging from 10's of packets to 1000's of packets. This approach can scale since network packet logging can be setup by the application provider using off the shelf logging tools running on each server virtual machine instance in a data centre.

In real-time gaming applications latency and packet loss are key metrics, since delays in packets containing player instructions or game status updates can seriously affect the ability to play the game competitively and thus degrade the overall quality of experience (QoE). For video streaming and VOIP applications packet loss and jitter (i.e. variation of latency over time) are key to providing a stable stream of video or audio packets and allowing smaller buffer sizes for more interactive real-time playback. Each type of real-time interactive application also needs a minimum bandwidth allocation to be available to cope with peak-level application traffic throughput, but most applications try to minimize their network packet size requirement making QoS metrics for packet loss and latency much more important to the final application QoE than simple bandwidth.

Current IP traffic classification approaches focus only on classifying the type of application, not the QoS metrics themselves. We propose a novel indirect approach to the measurement of periods of high application latency and packet loss based on IP traffic classification. Because our classifier works using only passive server-side IP traffic log data, obtainable using standard tools such as tcpdump, we avoid the drawbacks associated with other more direct measurement techniques (i.e. requirements for access to network switches, end to end access, network congestion or use of application specific network protocols).

Our approach works in near real-time, requiring fully populated temporal segments before classification and thus lagging behind real-time by the temporal width of a segment (i.e. typically about 30 to 60 seconds behind real-time). We also show how multi-class classification could be used to create a type of 'virtual instrument', able to classify periods of according to discrete ranges of latency and packet loss. Our methods ability to classify QoS in near real-time is in fact a significant contribution to the state of the art over log-based analytical systems.

We report results from several experiments. We chose to test using real-world applications (i.e. Minecraft, Quake3 UT, VLC video streaming and Turk Telekom's Wirofon VOIP) after discussion with Europe's largest ISP Interroute and Turkey’s largest telecoms company Turk Telekom since these represent important classes of real-time interactive application that network providers want to support. Our experiments are a mixture of real-world server deployments, with real users from the general public, and controlled lab-based experiments. The experiment durations range from several hours to several days. Observed lag events in our experiments originate from either real ISP lag or simulated network lag introduced using NETEM on the application server. For ground truth we use a combination of application ping data (i.e. Minecraft heartbeat protocol and Quake3 UT ping) and timing records for the simulated NETEM lag events. We report results for binary lag classification based on 10-fold cross validation across 4 different application types and 5 different classifiers. We report results for discrete multi-class latency and packet loss classification based on 10-fold cross validation for 1 application type and 5 different classifiers. We find the results promising and well suited for use in near real-time QoS monitoring for the purposes of network optimization and billing (e.g. ISP service level agreement guarantees for QoS). The use of standard logging tools also means the raw data collection process should be transparent to any stakeholders involved (i.e. ISP's and application developers).

We outline in section 2 relevant existing work in the context of our proposed approach. The approach itself is detailed in section 3 and experimental results reported in section 4. We end with a discussion of these results in section 5 and both conclusions and future work in section 6.

2. Related Work

A lot of research has been conducted into various network level QoS measurements on infrastructure ranging from ISP backbones to data centre networks and wireless mobile networks. Network QoS metrics for real-time interactive applications [29] include throughput, latency, jitter and packet loss. Typically these applications work over the TCP [6] and UDP [7] network protocols. Application QoS metrics include game frame rate, speech quality and video buffering time. We limit our review of related work to network QoS metrics. Broadly speaking measurement techniques can be classified as active (i.e. generating traffic) or passive (i.e. listening only) and can be either application independent or utilize application specific protocols (e.g. Minecraft's TCP packet protocol supporting heartbeats).

Active probing [15] [21] [32] can be used to measure both latency, jitter and packet loss, and there are a number of standard network tools such as ping, traceroute and iperf available on many operating systems.

Packets are sent from client to server and the one-way, round trip and/or hop transfer times calculated. Active probing usually needs a well-known port to be opened on the server machine. One disadvantage of active approaches is that they can congest the network they are trying to measure, especially where there are lots of clients accessing a single gateway. Another problem is that active pings will typically not trigger ISP traffic shaping effects [27] in the way more realistic application traffic might (e.g. large data downloads). Lastly the requirement for clients to run active probe software is an unrealistic one for home users wanting to run a real-time interactive application.

Passive logging approaches can be used to record network IP traffic data at different network switches and/or computers. Logging IP traffic data from network switches has become increasingly easy with the development of software-defined networking (SDN) tools such as OpenFlow. This has given rise to a number of passive end to end measurement approaches [4] [13] [35] [25] [26] where switch counters are polled to measure directional packet flows between switches. Switches are typically configured to filter and/or sample packets prior to logging to avoid excessive computation, thus measurement accuracy is related to how fine-grained the flow sampling and per-flow state tracking is. Typical results [16] [22] report sampled measurement precision ranging from 0.80 to 0.95. The advantage of using passive switch counters is that there is no network congestion. The disadvantage is that access to the network switches is required at each measurement point.

Logging of IP traffic directly from client and/or server network cards can be achieved with standard tools such as tcpdump. For each log the mean inter-packet arrival times can be calculated over a given sample period as well as incoming and/or outgoing mean packet sizes and mean packet counts. IP traffic logs can be used [32] to look at end to end latency, across the whole ISP wide-area network, 'last-mile' latency, to the first hop in the ISP network, and 'latency under load' measured during network saturation conditions. The measurement of 'latency under load' is particularly useful when trying to identify 'bufferbloat' in video streaming applications, where the bandwidth limits is throttled for upload and download.

Although end to end measurement avoids traffic congestion it can be hard to implement since access to both the client and server machines is often not realistic. If only server-side IP traffic logs are available then end to end QoS can be accurately modelled for non-interactive applications, such as FTP file transfers, where metrics such as server-side inter-packet arrival times can infer latency. However for interactive applications where network load can change erratically based on user behaviour so some sort of application-specific approach is usually required.

Another use of passive IP traffic logs is for IP traffic type classification [1] [36] which is most closely related to our approach. Classifying network traffic in terms of application type can guide network routing decisions and can be used to filter or block packets from malware or denial of service attacks. Approaches use machine learning techniques including kNN, Linear Discriminant Analysis (LDA) [23], Bayesian classifiers [33], Expectation Maximization (EM) clustering [3] and decision trees [11] [28] [37] to classify a variety of application traffic types such as FTP, HTTP, Skype, BitTorrent, games, video streaming and music downloads. Results for application classification using supervised learning [37] report F1 scores up to 0.98. There has also been work [21] on QoS profiling using linear models for the purposes of predicting bulk data transfer latency between data centres via an ISP backbone, with error rates up to 3%. Our work goes beyond the state of the art on IP traffic classification of application type and looks at the more difficult task of classifying periods of application packet loss and latency, ultimately providing an indirect way to quantify the severity of these periods.

For video streaming and VOIP applications passive QoS measurement techniques are mostly used with latency, packet loss and jitter QoS metrics. Representative examples include end to end measurement of Real-time Transport Protocol (RTP) traffic using traceroute [18], video packet flow statistics extracted from tcpdump data [38] and application QoS metrics for video buffering time [30] derived from TCP traffic data.

For VOIP applications Skype and RTP-based application examples include packet loss and jitter modelling [9] [10] using network packet logging tools like Wireshark. There have also been large scale studies looking at MPEG-2 video streaming [24] and Skype [31] [19] traffic. These studies examine the link between network QoS, specifically latency, jitter and packet loss, and QoE as measured by mean opinion scores from user surveys.

For gaming applications latency and packet loss are very important, with many games (e.g. Minecraft and Quake3 UT) providing in-built application-specific active end to end protocols for measuring it. Typically 'acceptable' [17] game latencies range from 100ms for first person shooter (FPS) and racing games, 500ms for sports and role playing games (RPG), and 1000ms for real-time strategy games (RTS). Active QoS measurement approaches include active ping latency measurement for cloud-hosted FPS and RTS [14] games, Passive QoS measurement approaches use sampling latency and packet loss from tcpdump traffic data [39] for Massively Multiplayer Online Role-Playing Games (MMORPG's) like World of Warcraft and, inter-packet arrival times [12] for FPS's like Quake3 and Halo2.

3. Materials and Method

3.1. Network Traffic Monitoring

Our approach is based on the classification of passively logged IP network traffic. We instrument each application server (e.g. Minecraft server in a data centre) by running tcpdump locally and logging the PCAP network packet data running on the application server's port. These log files are parsed by our 'network packet parser' tool which continually extracts any Ethernet, IP, TCP and UDP packet header information, looking for metadata on timestamp, source IP address, destination IP address and the packet body size. Incoming packet header information is sampled at a 1 second sample period and source IP address information removed for privacy protection. For each packet type (i.e. TCP or UDP) a row is inserted into a MySQL table containing columns for sample index, timestamp, count of the number of packets in a sample period and the aggregated packet size from these packets.

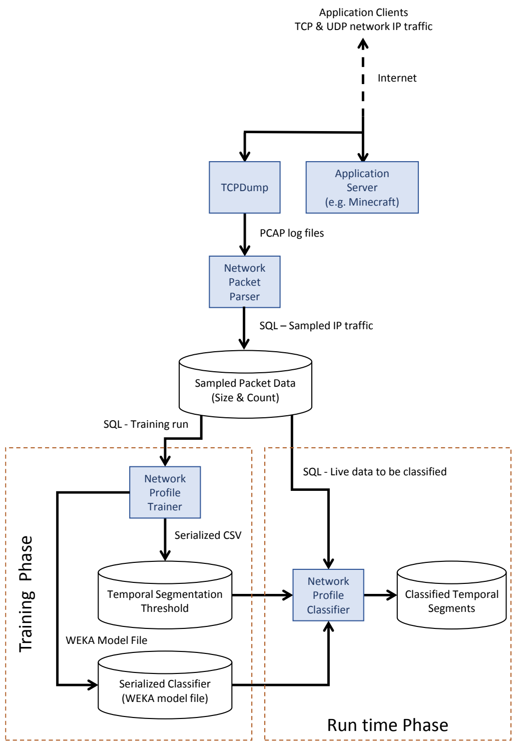
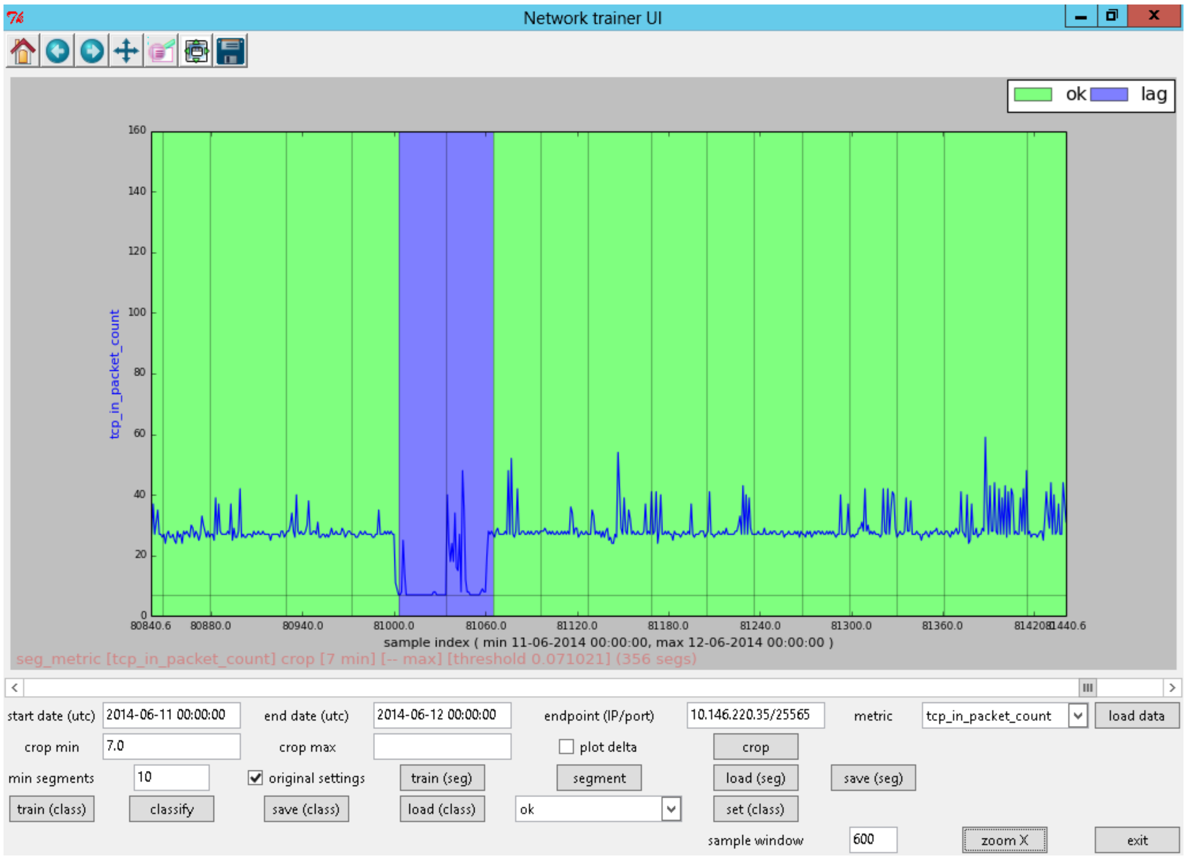

Application Clients
TCP & UDP network IP traffic
Internet
TCPDump
Application Server (e.g. Minecraft)
PCAP log files
Network Packet Parser
SQL – Sampled IP traffic
Sampled Packet Data (Size & Count)
SQL - Training run
SQL - Live data to be classified
Network Profile Trainer
Serialized CSV
Training Phase
Temporal Segmentation Threshold
WEKA Model File
Serialized Classifier (WEKA model file)
Network Profile Classifier
Classified Temporal Segments
Run time Phase

Fig. 1. Information dataflow used for all our experiments.

The rest of the processing dataflow can be seen in Fig. 1. We first run the application server for a pre-defined training period (e.g. 1 day) and upload all sampled network traffic data to the database. Our 'network profile trainer' tool displays the sampled training data, temporally segments it and facilitates manual labelling of lag periods. Once training data has been labelled a WEKA supervised learning classifier [20] is created and serialized. The 'network profile classifier' is then run using this serialized classifier and all future real-time application network traffic classified using it. Classified temporal segments are continually stored in a MySQL database audit table shortly after they are classified. A simple SQL query can report the number of application lag periods and/or total duration of these lag periods for purposes such as billing.



seg_metric [tcp_in_packet_count] crop [7 min] [-- max] [threshold 0.071021] (356 segs)

start date (utc) 2014-06-11 00:00:00
end date (utc) 2014-06-12 00:00:00
endpoint (IP/port) 10.146.220.35/25565
metric tcp_in_packet_count
crop min 7.0
crop max
☐ plot delta
min segments 10
☑ original settings
ok
sample window 600

Fig. 2. Screenshot from the network profile training tool loaded with sampled Minecraft network packet data. The x-axis is sample index number, with a 1 second period, and y-axis is TCP packet IN count (i.e. number of packets per second arriving at server from clients). The vertical grey lines indicate temporal segments. The green colour indicates segments manually labelled as 'OK', and purple colour indicates segments labelled as 'lag'.

3.2. Temporal Segmentation of Network Traffic

Temporal segmentation is the process of dividing up a time series into a series of segments, each with their own start and end time, according to a segmentation algorithm. Our 'network profile trainer' tool provides a way to visualize network traffic datasets, temporally segment them and allow manual classification of each of the temporal segments. A screenshot from the 'network profile trainer' tool is shown in Fig. 2.

First sampled network packet logs are loaded from a MySQL table for visualization and human inspection by a human analyst using our tool. Metrics are available for each network protocol (i.e. TCP and/or UDP) with sampled data for packet count & packet size in both the incoming & outgoing directions. A human analyst manually inspects the visualized data and decides upon the chosen metric to use for temporal segmentation (e.g. TCP_packet_count_IN).

Our tool then calculates a set of statistical measures (i.e. mean, max, min and standard deviation) over sample windows before and after each sample point in the time series for the chosen metric. We found a sample window of 30 samples worked well. We use the difference between the previous sample window and the next sample window for the basis of deciding when to segment the time series. From analysis of many hours of IP traffic logs we found that some interactive events (e.g. players leaving, videos ending) caused drops in IP traffic that looked very similar to drops in network quality. We found that the most reliable indication for a change in the network quality was a change in the standard deviation of the packet count and/or packet size metrics. We think the reason for this is related to the different strategies applications employ when trying to limit the effects of poor network QoS, such as resending packets and appending lost status information to future packets. A temporal segment is triggered whenever the difference in standard deviation between previous and next sample windows jumps above a threshold value.

```
Algorithm: get_temporal_segmentation_threshold( $N_{target\_segments}$ )

$N_{sample\_window}$ = 30
Threshold = 0.1
MinSampleWidth = 30

# load metric data (e.g. sampled TCP packet count IN)
Load sampled dataset D

# calculate change in stdev from previous window to next window
for SampleIndex in 0 ... $N_{samples}$ do
   PrevD = D[ SampleIndex - $N_{sample\_window}$ : SampleIndex - 1 ]
   NextD = D[ SampleIndex + 1 : SampleIndex + $N_{sample\_window}$ ]
   Norm = max( mean( PrevD ), mean( NextD ) )
   $\Delta_{SampleIndex}$ = abs( stdev( NextD ) - stdev( PrevD ) ) / Norm
end

# increase threshold until we get the target number of segments
do
  $N_{segments}$ = 0
  Threshold = Threshold * 1.1
  for SampleIndex in 0 ... $N_{samples}$ do
    if $\Delta_{SampleIndex}$ >= Threshold
    then
      $N_{segments}$ = $N_{segments}$ + 1
      SampleIndex = SampleIndex + MinSampleWidth
  end
while $N_{segments}$ > $N_{target\_segments}$

return Threshold
```

Fig. 3. Algorithm for getting the temporal segmentation threshold. Algorithm uses a Python-like syntax where subsets are represented using a formulation SET [start_index:end_index].

```
Algorithm: apply_temporal_segmentation ( SegmentationThreshold )

$N_{sample\_window}$ = 30
MinSampleWidth = 30

# load metric data (e.g. sampled TCP packet count IN)
Load sampled dataset D
$N_{samples}$ = size(D)

# calculate change in stdev from previous window to next window
for SampleIndex in 0 ... $N_{samples}$ do
    PrevD = D[ SampleIndex - $N_{sample\_window}$ : SampleIndex - 1 ]
    NextD = D[ SampleIndex + 1 : SampleIndex + $N_{sample\_window}$ ]
    Norm = max( mean( PrevD ), mean( NextD ) )
    $\Delta_{SampleIndex}$ = abs( stdev( NextD ) - stdev( PrevD ) ) / Norm
end

# segment dataset using the segmentation threshold
$N_{segments}$ = 0
for SampleIndex in 0 ... $N_{samples}$ do
    if $\Delta_{SampleIndex}$ >= SegmentationThreshold
    then
        add segment to D
        $N_{segments}$ = $N_{segments}$ + 1
end

# calculate statistical features for each new segment
for Segment in 0 ... $N_{segments}$ do
    calc feature mean, max, min, stdev, duration for $D_{segment}$
    calc feature Δmean, Δmax, Δmin, Δstdev for $D_{segment}$ - $D_{segment-1}$
    calc feature Δmean, Δmax, Δmin, Δstdev for $D_{segment+1}$ - $D_{segment}$
    add features to $D_{segment}$
end

return D
```

Fig. 4. Algorithm for temporal segmentation. Algorithm uses a Python-like syntax where subsets are represented using a formulation SET [start_index:end_index].

The algorithm to determine the segmentation threshold is shown in Fig. 3. A target value for the minimum number of temporal segments is first chosen after manual inspection of the data using our training tool. We use ground truth data (i.e. known lag events) to decide how many lag periods we think are present in each training dataset, and therefore how many temporal segments we think is sensible to have as a target value.

Once a segmentation threshold is decided the training dataset is segmented using the algorithm shown in Fig. 4. A set of statistical features are calculated for each temporal segment to be used later by the classifier. Features for training include segment duration and all combinations of mean/stdev/min/max/sum/delta values for TCP/UDP packet count IN/OUT and TCP/UDP packet count IN/OUT. These features represent all the data available from the logged and sampled packet headers, and we let the classifier choose which features provide the best discriminating power for classification. Typical temporal segment periods range between 30s and 60s, with longer segments only occurring during periods of long term inactivity (e.g. no players on game server). Our tool's visual interface is used to manually label each segment according to the chosen class set (e.g. binary labels {ok,lag}). Class labels can be seen as the coloured segments in Fig. 2.

Both the class label and statistical features for each temporal segment are used to train a WEKA classifier. We can configure our tool to use any WEKA supervised learning classifier allowing easy comparison of classifiers. The trained classifier is serialized ready for use with the 'network profile classifier' tool.

3.3. Classification of Temporally Segmented Network Traffic

The 'network profile classifier' tool starts by loading a WEKA classifier model created during the training phase. The tool then continually polls (e.g. every 60s) for new network traffic samples appearing in the MySQL database. When new sample data arrives it is appended to an in-memory unsegmented sample buffer and temporally segmented. Any new segments are classified and the result written to a MySQL table for audit purposes. We only accept classification results whose probability, as reported by the chosen WEKA classifier, is above a configurable quality threshold. Classifications below the quality threshold are rejected and the temporal segment listed as unlabelled. The per-classifier type quality threshold was chosen empirically after manually comparing false positive rates verses recall for different settings during the training phase. Each classifier algorithm has its own characteristics and quality thresholds across classifiers used very different values. We found quality thresholds for a single classifier type across applications did not vary as much, thus a single quality threshold can be applied to each classifier irrelevant of the application it is classifying.

The final database of classified temporal segments is an audit record with server endpoint (i.e. IP address and port), start time, end time and classification label. Stakeholders (i.e. ISP's and application developers) can easily download this audit data to see how many lag periods were detected and what the total duration of them was in seconds. This information could be used to compare against any QoS guarantees made by an ISP in their SLA with the application developer.

It should be noted that the classification process is always one temporal segment behind real-time to ensure that the last segment has been fully populated with samples. As such our approach is only able to run in near real-time, something which is acceptable for QoS billing but would not be suitable for use in any real-time control loops (e.g. real-time QoS driven load balancing of resources).

We explored events where players left, and there was no activity, and cases where the lag was so bad players dropped their connection in some detail since these are particularly difficult hard to differentiate. This differentiation does in fact provide the strongest reason for using a machine learning approach. Patterns are application specific but often looking at the standard deviation and/or duration of the preceding and following temporal segments surrounding a drop event revealed useful pattern. This type of contextual pre-requisite information is easily processed by a machine learning algorithm but is much more difficult to factor into conventional statistical models based only on metrics such as a moving average.

4. Results

4.1. Classification of Time Periods with High Latency and/or Packet Loss for Real-time Interactive Applications

In order to empirically evaluate how accurate our approach is when classifying application lag periods we conducted 4 experiments profiling different real-time interactive applications under both real-world and laboratory conditions. The focus of these experiments was to measure the classification performance of different classifiers and determine which performed best and how performance varied across different application types.

For the first experiment we worked with Interroute to setup a laboratory experiment using a local-area network, a VLC video streaming server and 3 VLC clients. All 3 clients had a 1 Gbit network link but 2 clients were routed via a slower 100 Mbps firewall to simulate a variety of connection options. The streamed video (H.264) was a standard video Interroute used for video testing. The experiment scenario ran for 1.5 hours with clients requesting on-demand streaming at different times throughout. We ran tcpdump on the VLC streaming server and the PCAP traffic log data was parsed and profiled. We scheduled a CRON job running NETEM to alternate simulated 1 minute periods of network latency (100ms +/-10ms distribution normal) and packet loss (2%) every 10 minutes.

For the second experiment we worked with Turk Telekom to setup an experiment using their commercial Wirofon VOIP service. Three clients were setup to initiate calls and 3 clients to receive calls, each communicating via the internet to reach the central Wirofon server. Clients connected and disconnected over a period of 1.5 hours, playing sections from a standard Turk Telekom VOIP test call featuring both audio (G.711Mu) and video (H.264) communication. We ran tcpdump on the Wirofon server and the PCAP traffic log data was parsed and profiled. We used the same simulated network latency and packet loss as the first experiment.

For the third and fourth experiments we setup an Amazon EC2 cloud instance and installed both Minecraft and Quake 3 Urban Terror (UT) servers. These were made accessible to the general public to play on and advertised via the gaming community server listings. We used a custom Minecraft CraftBukkit server build which logged Minecraft heartbeat statistics for players sampled over 60s periods to give us a ground truth metric for in-game lag. We used the Quake3 network protocol to request player ping statistics, sampled over 60s periods, to again log a ground truth metric for in-game lag. We scheduled a CRON job running NETEM to alternate simulated 1 minute periods of network latency (1000ms +/-100ms distribution normal) and packet loss (5%) every 30 minutes. The experiment ran for 2 weeks with players logging on and off as they wished. For analysis purposes we looked at 3 days of recorded data where there were 48 player connections (14 Quake3 UT, 34 Minecraft) to our server.

In all of the experiments we provisioned enough server-side CPU, RAM and network bandwidth to ensure the applications performed without resource limits. As such any observed network lag came either from the client's ISP network links or the simulated NETEM lag events.

For each logged dataset we used ⅔ of the data as training data (i.e. 60 mins for experiments 1 & 2 and 2 days for experiments 3 & 4) and ⅓ of the data for testing (i.e. 30 mins for experiments 1 & 2 and 1 day for experiments 3 & 4). We used our network profiling tool to temporally segment and manually label both the training and test data using binary classes {'ok', 'lag'}. Labels were decided based on the ground truth (i.e. application specific ping & heartbeat data and time periods during which NETEM was active). The dataset breakdown of ground truth labels can be seen in Table 1. The number of ground truth ISP lag events was dependent on who was playing at the time and the quality of their ISP connection to the server, but for test datasets was always more frequent than the NETEM events.

Table 1. Ground Truth Breakdown for Temporally Segmented Datasets Exp1.

Dataset	Duration	Total segs	ISP lag segs	Dataset
Minecraft (train)	2 days	1876	102 (5%)	107 (5%)
Minecraft (test)	1 day	356	69 (19%)	15 (4%)
Quake3 UT (train)	2 days	165	13 (12%)	29 (17%)
Quake3 UT (test)	1 day	144	21 (14%)	6 (4%)
VLC (train)	60 mins	81	n/a	10 (12%)
VLC (test)	30 mins	52	n/a	8 (15%)
VOIP (train)	60 mins	39	n/a	8 (20%)
VOIP (test)	30 mins	31	n/a	8 (25%)

We trained 5 different WEKA classifiers using only the temporally segmented training data. The classifiers used were a J48 decision tree with no pruning, a NaiveBayes classifier with a kernel density estimator, an IBk

K-nearest neighbour classifier with a K value of 20, RandomForest bagging classifier with a 500 feature limit and a Support Vector Machine (SVN) using Sequential Minimal Optimization (SMO). The classifier configurations were chosen after an optimization process involving inspection of each classifiers results and some trial and error.

Metrics

Precision (P) = tp/(tp + fp) tp = true positive, fp = false positive
Recall (R) = tp/(tp + fn) tn = true negative, fn = false negative
F1 measure = 2*PR/(P+R)

Mean Precision = $\sum P(\text{fold}_N) / N_{\text{folds}}$
Mean Recall = $\sum R(\text{fold}_N) / N_{\text{folds}}$
Mean F1 = $\sum F1(\text{fold}_N) / N_{\text{folds}}$
$N_{\text{folds}} = 10$

95% Confidence Interval = Mean ± 1.96 x STDev / $\sqrt{N_{\text{folds}}}$

Fig. 5. Metric definitions used in all experiments.

The test data was temporally segmented and then divided into 10 sequential folds, each containing 10% of the total number of temporal segments. Each classifier classified each fold without any knowledge of the ground truth labels we had for the segments. A true positive (TP) occurred when the classifier was correct, a false positive (FP) when it was incorrect and a false negative (FN) when the classifier confidence was below the quality threshold and no classification result was generated. There are never any true negatives (TN) since our binary classes {'ok','lag'} cover all possible states. This allowed us to perform a standard 10-fold cross validation analysis calculating the mean and standard deviation for each classifier's precision, recall and F1 score (see Fig. 5. for metrics).

We ran the 10-fold cross validation analysis twice, once using a low classifier quality threshold and once using a high classifier quality threshold. The low quality threshold (i.e. 0.0) accepted all classification results and hence provided a 100% recall. The high quality threshold (i.e. 0.7 to 0.99 depending on classifier type) was chosen to give the best precision whilst still having a usable recall score.

Results using the low quality threshold can be seen in Fig. 6, which shows the per-classifier per-application mean precision/recall/F1 scores across all 10 folds along with error bars showing the 95% confidence intervals. The SVM + SMO classifier was the most robust classifier choice across all applications, giving F1 scores ranging from 0.78 to 0.89. Results using the high quality threshold can be seen in Fig. 7. The SVM + SMO classifier was again the most robust classifier choice across all applications, giving precision scores ranging from 0.73 to 0.86.

We also analysed the total time, in seconds, for correctly classified temporal segments (i.e. TP's) and total time for incorrectly classified temporal segments (i.e. FP's). The fraction of time spent between TP segments and FP segments was very similar to the previous precision results, and there was no classifier bias to longer or shorter segments. This is an important finding if our approach was used for QoS billing, since financial penalties could be applied to service providers going above an allowed count of classified lag periods or going above an overall time allowance for lag periods.

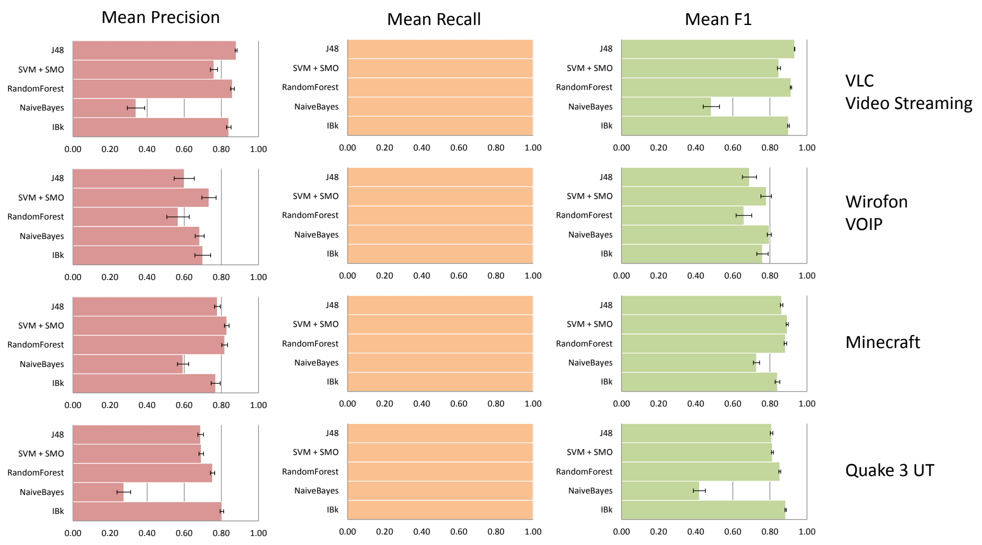


Fig. 6. 10-fold cross validation classification performance for periods of lag across 5 supervised classifier types and 4 different interactive application types, using a low classifier confidence threshold to give the best F1 scores. VOIP had the lowest overall F1 score (0.80) with VLC the best (0.93). SVM + SMO classifier was the most robust choice across application types. 95% confidence intervals are shown as error bars.

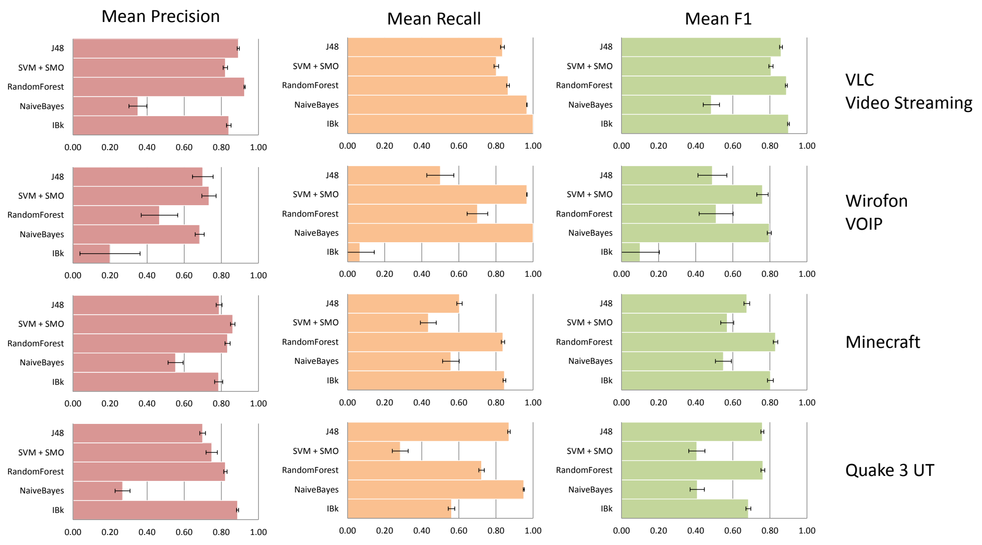


Fig. 7. 10-fold cross validation classification performance for periods of lag across 5 supervised classifier types and 4 different interactive application types, using a high classifier confidence threshold to give the best precision scores. VOIP had the lowest overall precision score (0.73) with VLC the best (0.93). SVM + SMO classifier was the most robust choice across application types. 95% confidence intervals are shown as error bars.

4.2. Classification of Time Periods with Discretized Ranges for Latency and Packet Loss

We also looked at how our IP traffic classification approach could be used as a kind of 'virtual instrument' to measure discrete ranges of application latency and packet loss. The focus of this experiment was to classify IP traffic in terms of discrete ranges for latency and packet loss, and examine the multi-class classification accuracy when doing this.

We setup a laboratory experiment using a pre-commercial stage federated testbed made up of FIWARE Lab instances created under the XIFI project [34]. We provided a virtual machine for a custom CraftBukkit Minecraft server and 2 virtual machines for clients running 5 Minecraft bots each (i.e. a maximum of 10 bot players connected at any one time). The Minecraft bots were creating using the open source Mineflayer Javascript API and acted like simple real-world players issuing commands to the Minecraft server as they walked across the Minecraft world, wandering and mining randomly in equal measure. Clients and servers were remotely distributed (i.e. clients in France, server in Italy) via a L3VPN 50 Mbps network link, and we provisioned enough server-side CPU, RAM and network bandwidth to ensure the Minecraft server ran without resource limits. The experiment scenario ran for 6 hours with the 10 Minecraft bot clients connecting, playing the game and disconnecting at different times throughout the test period. We ran tcpdump on the Minecraft server and the PCAP traffic log data was parsed and profiled. We scheduled a CRON job running NETEM to alternate simulated 1 minute periods of discrete network latency {250ms, 500ms, 750ms, 1000ms +/- 10% distribution normal} every 5 minutes for a total of 3 hours, then scheduled packet loss {2%, 4%, 6%, 8%, 10%} in the same way for a total of 3 hours.

Table 2. Ground Truth Breakdown for Temporally Segmented Datasets Exp2.

Dataset	Duration	Total segs	NETEM lag segs
Minecraft banded latency (train)	120 mins	153	26 (17%)
Minecraft banded latency (test)	60 mins	101	18 (18%)
Minecraft banded packet loss (train)	120 mins	154	32 (20%)
Minecraft banded packet loss (test)	60 mins	94	18 (19%)

We divided our 6 hour dataset into two, with the first 3 hours focussed on discrete latency classification and the last 3 hours on discrete packet loss classification. We used ⅔ of each of the two datasets as training data (i.e. 120 mins) and ⅓ of the data for testing (i.e. 60 mins). We used our network profiling tool to temporally segment and manually label both the training and test data using discrete multi-class labels {'0ms', '250ms', '500ms', '750ms', '1000ms'} and {'0%', '2%', '4%', '6%', '8%', '10%'} respectively. Labels were decided based on the ground truth metrics (i.e. time periods during which NETEM was active). The dataset breakdown of ground truth labels can be seen in Table 2.

The same experimental procedure was applied as for the experiments in the previous section. We trained 5 different WEKA classifiers using only the temporally segmented training data. The test data was temporally segmented and then divided into 10 sequential folds, each containing 10% of the total number of temporal segments. We ran the 10-fold cross validation analysis using the low classifier quality threshold since we wanted to primarily look at F1 scores.

Results for the discrete latency classes can be seen in Fig. 8, and results for discrete packet loss classes in Fig. 9. The IBk classifier was the most robust classifier, giving the best F1 score of 0.89 for latency and 0.90 for packet loss, although RandomForest results were very similar.

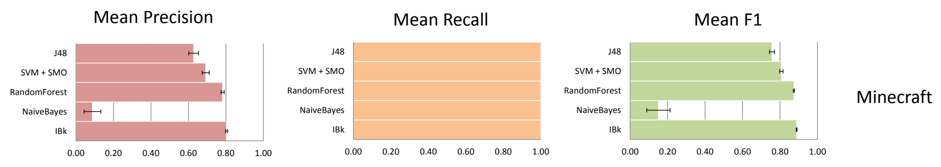


Fig. 8. 10-fold cross validation classification performance for periods of latency banded into 5 classes (0ms to 1000ms) across 5 supervised classifier types for the Minecraft interactive application, using a low classifier confidence threshold. Minecraft bots were used instead of players for this experiment. IBk classifier gave the best results with a precision of 0.80 and F1 of 0.89. Standard confidence intervals are shown as error bars.

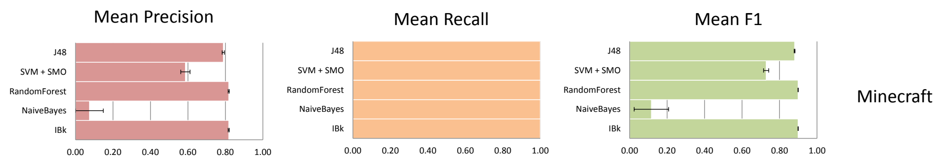


Fig. 9. 10-fold cross validation classification performance for periods of packet loss banded into 6 classes (0% to 10%) across 5 supervised classifier types for the Minecraft interactive application, using a low classifier confidence threshold. Minecraft bots were used instead of players for this experiment. RandomForest and IBk classifiers gave the best results with a precision of 0.82 and F1 of 0.90. Standard confidence intervals are shown as error bars.

5. Discussion

Overall binary classification performance was good across the different application types, with the best F1 scores ranging from 0.80 to 0.93. We found that the VOIP application IP traffic was the hardest to classify, as shown by both the higher than average confidence intervals associated with VOIP results and lower F1 scores. We think this is due to a combination of factors, including the complex interaction between video and audio IP traffic that VOIP applications generate and the fact that the experiment training dataset size was relatively small (i.e. 60 mins) compared to the larger (i.e. 2 days) datasets for the gaming applications. Classification results would improve with access to more training data.

We observed most of the classification errors seen were associated with lag events followed by drop outs as people gave up on the application and disconnected, sometimes reconnecting shortly afterwards. This type of error might be reduced with more training data, or considering additional temporal segments either side of the target segment. We had no reported testbed downtime during our trials but as we did not have access to the detailed day to day maintenance logs for our testbeds (i.e. amazon EC2, FIWARE Lab instances) we could not verify this independently.

We consider these F1 scores viable for use within a commercial QoS billing environment alongside traditional guarantees of bandwidth and availability. For example an ISP could offer an application-specific SLA with a guarantee to deliver no more than a certain count/duration of QoS lag periods in a specific billing period (e.g. guaranteed maximum of 1 hour total time in a 'lag' state for a 1 month service provisioning contract). Classification error could be factored into guarantees, revising down measurement values according to a 95% confidence interval margin of 'reasonable' error. For example a mean precision score of 0.8 means on average there are 25 false positives for every 100 correctly classified lag events; this error rate could be explicitly discounted in the SLA to ensure fair billing.

The results obtained when using our IP traffic classification approach as a multi-class 'virtual instrument' are also promising, with F1 scores for some classifier types almost as good as the binary classification results. The choice of classifier is important however, with instance-based (i.e. IBk) and bagging classifiers (i.e. RandomForest) performing well whilst other classifier types were not able to handle the multi-class scenario as effectively. In situations where direct measurement of latency and packet loss is impossible (e.g. access to both client and server is unavailable) or undesirable (e.g. active measurement techniques would degrade network performance) our discrete multi-class measurement approach can offer the most value.

Our approach is able scale to any real-time interactive application since we do not use application-specific protocols. New application types simply require training data to be acquired using a tool such as tcpdump, and lag periods labelled using our profiler tool. After this the chosen classifier can be run on live application server sampled IP traffic logs, classifying temporal segments for the new application type. The use of a sampling strategy also allows this approach to scale as the number of servers are increased. Messages containing sampled log data, sent to the central classifier via a high performance RabbitMQ message bus, are orders of magnitude smaller that the raw network packet data logged. For example a message containing 60 samples, each of 1 second duration, is about 4kBytes in size, whereas packet data logged for tcpdump for 60 seconds of traffic would be around 15Mbyte in size for the Minecraft application.

6. Conclusions And Future Work

More sophisticated and scalable ways to measure QoS are needed as the trend develops towards application providers and ISP's working together to provide premium QoS for paying customers.

This paper presents two sets of empirical evaluations of a novel approach to QoS classification of network IP traffic. The first evaluations quantify binary classification performance for application lag periods across 4 type of interactive real-time applications. Our experimental studies report results ranging from laboratory-based tests, running over a few hours, to real-world deployments running over several days. The second evaluations quantify multi-class classification performance using discrete ranges of latency and packet loss. In both sets of evaluations we compared 5 different supervised learning classifiers and reported classification precision/recall/F1 scores with associated confidence intervals to demonstrate their statistical significance.

We have shown it is possible to classify application packet loss and latency and have quantified the error associated with such classifications. We also showed how it is possible to indirectly measure application packet loss and latency. Our classification approach to QoS measurement uses passive IP traffic logging, avoiding the need for end-to-end access and access to any intermediate network links.

The use of classifiers trained for each application type means we are able to differentiate multiple types of application lag. For example we could label player lag during a fight sequence differently than player lag during a map upload at the start of a game. This means we can add new application level granularity to QoS measurements, something that could represent real added value over existing application neutral measurement approaches to QoS.

The ability to measure application latency and packet loss from standard passively logged IP network traffic has a number of advantages. First the measurements are passive so do not effect network performance in any way. Second the measurements are transparent, using industry standard technology such as tcpdump so all stakeholders (i.e. ISP's and application developers) can trust them. This could allow progress beyond the current marketplace norm where ISP traffic measurements are largely kept secret from application developers. Lastly the approach is scalable across a range of interactive applications, allowing ISP's to provide a common trusted approach to a wide range of application developers without the need for any detailed application network protocol know-how.

Of course whilst a QoS classification technique to measure application latency and packet loss might be technically viable, work at a business/political level would be required before it could be adopted in the current ISP marketplace. However, with new business models between ISP's and application developers being actively considered in the marketplace today this work represents a significant contribution to the debate by providing scientific evidence relating to new QoS measurement approaches.

Acknowledgements

The work presented in this paper is part of the research in the EU FP7 OFERTIE project (contract number 318665) and EU FP7 XIFI project (contract number 604590). We would like to acknowledge the help provided by Antonios Kollaras (Interroute), Burak Görkemli (Argela Turk Telekom), Jack Edge & Martin Hall-May (IT Innovation Centre) when setting up and running testbeds for our network experiments.

References

[1] A. Callado, C. Kamienski, S. Fernandes, D. Sadok, G. Szabo, B.P. Ger, A survey on internet traffic identification and classification, IEEE Communications Surveys and Tutorials (2009), vol. 11, no. 3, pp. 37–52.
[2] A. Hern, Netflix in row over net neutrality support, theGuardian online newspaper. [Online]. Available: http://www.theguardian.com/technology/2015/mar/05/netflix-row-net-neutrality-support-australia
[3] A. McGregor, M. Hall, P. Lorier, J. Brunskill, Flow clustering using machine learning techniques, In Proceedings of the PAM'04 (2004), pp. 205–214.
[4] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, H.V. Madhyastha, FlowSense: Monitoring Network Utilization with Zero Measurement Cost, In Passive and Active Measurement (2013), Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp 31-41.
[5] D. Reisinger, Net neutrality rules get published -- let the lawsuits begin, CNET blog post. [Online]. Available: http://www.cnet.com/uk/news/fccs-net-neutrality-rules-hit-federal-register-lawsuit-underway/
[6] DARPA, Transmission control protocol, RFC 793 (1981).
[7] DARPA, User datagram protocol, RFC 768 (1980).
[8] Federal Communications Commission, Protecting and Promoting the Open Internet, Final Rule 80 FR 19737 (2015), FCC.
[9] G. Exarchakos, L. Druda, V. Menkovski, A. Liotta, Network analysis on Skype end-to-end video quality, International Journal of Pervasive Computing and Communications (2015), Vol. 11, Iss 1 pp. 17 - 42.
[10] H. Toral-Cruz, A.K. Pathan, J.C.R. Pacheco, Accurate modeling of VoIP traffic QoS parameters in current and future networks with multifractal and Markov models, Mathematical and Computer Modelling (2013), Vol 57, pp 2832–2845.
[11] J. But, P. Branch, T. Le, Rapid identification of BitTorrent traffic, In Proceedings of the 35th IEEE LCN'10 (2010), pp. 536–543.
[12] J. Saldana, J. Fernández-Navajas, J. Ruiz-Mas, J.I. Aznar, E. Viruete, L. Casadesus, First Person Shooters: Can a Smarter Network Save Bandwidth without Annoying the Players?, IEEE Communications Magazine (2011).
[13] J. Suh, T. Kwon, C. Dixon, W. Felter, J. Carter, OpenSample: A low-latency, sampling-based measurement platform for SDN, IBM Research Report (2014).
[14] K. Chen, Y. Chang, P. Tseng, C. Huang, C. Lei, Measuring the Latency of Cloud Gaming Systems, In Proceedings of the 19th ACM International Conference on Multimedia (2011).
[15] K.P. Gummadi, S. Saroiu, S.D. Gribble, King: Estimating Latency between Arbitrary Internet End Hosts, In the Proceedings of SIGCOMM IMW 2002 (2002), Marseille, France.
[16] L. Jose, M. Yu, J. Rexford, Online measurement of large traffic aggregates on commodity switches, In Proceedings of the USENIX HotICE Workshop (2011).
[17] M. Claypool, K. Claypool, Latency and player actions in online games, Communications of the ACM (CACM) (2006), Vol 49, Issue 11, pp 40-45.
[18] M. Ellis, C. Perkins, D.P. Pezaros. End-to-end and network-internal measurements of real-time traffic to residential users, In Proceedings of the second annual ACM conference on Multimedia systems (MMSys '11) (2011).
[19] M. Fiedler, T. Hossfeld, P. Tran-Gia, A Generic Quantitative Relationship between Quality of Experience and Quality of Service, IEEE Network (2010).
[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA Data Mining Software: An Update, SIGKDD Explorations (2009), Volume 11, Issue 1.
[21] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, J. Shi, Use of Network Latency Profiling and Redundancy for Cloud Server Selection, In Proceedings of the IEEE 7th International Conference on Cloud Computing (CLOUD) (2014), pp.826-832.

[22] M. Malboubi, L. Wang, C.N. Chuah, P. Sharma, Intelligent SDN based traffic (de)aggregation and measurement paradigm (iSTAMP), In Proceedings of the IEEE INFOCOM'14 (2014).
[23] M. Roughan, S. Sen, O. Spatscheck, N. Duffield, Class-of-service mapping for QoS: A statistical signature-based approach to IP traffic classification, In Proceddings of the 4th ACM SIGCOMM IMC'04 (2004), pp. 135–148.
[24] M. Venkataraman, M. Chatterjee, Quantifying Video-QoE Degradations of Internet Links, IEEE/ACM Transactions On Networking (2012), Vol. 20, No. 2.
[25] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with OpenSketch, In Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI'13) (2013), vol. 13, pp. 29–42.
[26] N.L.M. van Adrichem, C. Doerr, F.A. Kuipers, OpenNetMon: network monitoring in openflow software defined networks, In Proceedings of the IEEE Network Operations and Management Symposium (NOMS) (2014).
[27] P. Kanuparthy, C. Dovrolis, End-to-end Detection of ISP Traffic Shaping using Active and Passive Methods, Technical Report, Georgia Tech (2011).
[28] P.A. Branch, A. Heyde, and G.J. Armitage, Rapid identification of Skype traffic flows, In Proceedings of the 18th NOSSDAV'09 (2009), pp. 91–96.
[29] R. Beuran, M. Ivanovici, B. Dobinson, Network quality of service measurement system for application requirements evaluation, In Proceedings of IEEE SPECTS (2003), pp. 380–387.
[30] R.K.P. Mok, E.W.W. Chan, R.C.K. Chang, Measuring the Quality of Experience of HTTP Video Streaming, In Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM) (2011), Dublin, pp 485 - 492.
[31] S. Chen, C. Chu, S. Yeh, H. Chu, P. Huang, Modeling the QoE of Rate Changes in Skype/SILK VoIP Calls, IEEE/ACM Transactions On Networking (2014), Vol. 22, No. 6.
[32] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, A. Pescapè, Measuring Home Broadband Performance, Communications of the ACM (CACM) (2012), vol. 55, no. 11.
[33] S. Zander, T. Nguyen, G. Armitage, Automated traffic classification and application identification using machine learning, In Proceedings of the IEEE 30th LCN'05 (2005), pp. 250–257.
[34] S.E. Middleton, S. Modafferi, Experiences Monitoring and Managing QoS using SDN on Testbeds Supporting Different Innovation Stages, Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft) (2015), London, UK.
[35] S.R. Chowdhury, M.F. Bari, R. Ahmed, R. Boutaba, Payless: a low cost network monitoring framework for software defined networks, In Proceedings of the 14th IEEE/IFIP Network Operations and Management Symposium (NOMS’14) (2014).
[36] T. Nguyen, G. Armitage, A survey of techniques for internet traffic classification using machine learning, IEEE Communications Surveys and Tutorials (2008), vol. 10, no. 4, pp. 56–76.
[37] T.T.T. Nguyen, G. Armitage, P. Branch, S. Zander, Timely and Continuous Machine-Learning-Based Classification for Interactive IP Traffic, IEEE/ACM Transactions On Networking (2012), Vol. 20, No. 6.
[38] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, H. Yan, Prometheus: toward quality-of-experience estimation for mobile apps from passive network measurements, In Proceedings of the 15th Workshop on Mobile Computing Systems and Applications (HotMobile '14) (2014), New York, NY, USA
[39] X. Wang, T. Kwon, Y. Choi, M. Chen, Y. Zhang, Characterizing the Gaming Traffic of World of Warcraft: From Game Scenarios to Network Access Technologies, IEEE Network (2012).