Online Available Bandwidth Estimation using Multiclass Supervised Learning Techniques

Sukhpreet Kaur Khangura*, Sami Akın

Department of Electrical Engineering and Computer Science

Leibniz University Hannover, Germany

Abstract

In order to answer how much bandwidth is available to an application from one end to another in a network, state-of-the-art estimation techniques, based on active probing, inject artificial traffic with a known structure into the network. At the receiving end, the available bandwidth is estimated by measuring the structural changes in the injected traffic, which are caused by the network path. However, bandwidth estimation becomes difficult when packet distributions are distorted by non-fluid bursty cross traffic and multiple links. This eventually leads to an estimation bias. One known approach to reduce the bias in bandwidth estimations is to probe a network with constant-rate packet trains and measure the average structural changes in them. However, one cannot increase the number of packet trains in a designated time period as much as needed because high probing intensity overloads the network and results in packet losses in probe and cross traffic, which distorts probe packet gaps and inflicts more bias. In this work, we propose a machine learning-based, particularly classification-based, method that provides reliable estimates utilizing fewer packet trains. Then, we implement supervised learning techniques. Furthermore, considering the correlated changes over time in traffic in a network, we apply filtering techniques on estimation results in order to track the changes in the available bandwidth. We set up an experimental testbed using the Emulab software and a dumbbell topology in order to create training and testing data for performance analysis. Our results reveal that our proposed method identifies the available bandwidth significantly well in single-link networks as well as networks with heavy cross traffic burstiness and multiple links. It is also able to estimate the available bandwidth in randomly generated networks where the network capacity and the cross traffic intensity vary substantially. We also compare our technique with the others that use direct probing and regression approaches, and show that ours has better performance in terms of standard deviation around the actual bandwidth values.

Keywords: Available Bandwidth Estimation, Machine Learning

1. Introduction

Due to intensive developments in network access technologies and applications in the Internet, performance monitoring through the estimation of end-to-end available bandwidth has been a significant research focus. There are many network applications that depend on quality available bandwidth estimates in order to improve their performance in, for instance, selecting the best route, detecting congestion, and balancing the traffic across a network to avoid stops, lags or buffering while streaming delay-sensitive contents. Here, the term *available bandwidth* refers to the residual capacity of a link that is left over after the existing (cross) traffic is served. Formally, given a link with capacity $C \in \mathbb{R}^+$ and cross traffic with long-term average rate λ, the available bandwidth in the link is given as $A = C - \lambda$ [1], where $A, \lambda \in [0, C]$. In general, the end-to-end available bandwidth in a network path consisting of $H \in \mathbb{Z}^+$ hops is defined by its tight link as $A = \min_{i=1..H}(A_i)$, where $A_i \in [0, C_i]$ is the available bandwidth in the i^{th} link and $C_i \in \mathbb{R}^+$ is the capacity of the same link. Specifically, the term *tight link* refers to the link that has the minimum available bandwidth in a network. Here, the tight link of a network differs from the bottleneck link of the same network such that the bottleneck link is the link with the minimum capacity. Hence, the capacity of a network is expressed as $C = \min_{i=1..H}(C_i)$.

1.1. Related Work

Several available bandwidth estimation techniques based on passive [2–4] and active [1, 3, 5–15] measurement tools were proposed in the literature. Unlike the former that performs non-intrusive traffic monitoring, the latter injects a *probe traffic* with packet size l and inter-packet gap g_{in} into a network, where g_{in} is also recognized as the input gap. Principally, given a pair of packets, g_{in} refers to the difference between the time instants when the packets enter the network. At the receiving end of the network, the time difference between the consecutive packets following the order at the input, known as the output gap, g_{out}, is measured, and is used along with l and g_{in} in order to estimate the available bandwidth. These techniques invoke the fluid-flow model, where cross traffic is assumed to be constant-rate and behaves like fluid, i.e., it is composed of infinitesimally small packets. Furthermore, they assume a network path with a

*Corresponding author

Email addresses: khangura.sukhpreet@ikt.uni-hannover.de (Sukhpreet Kaur Khangura), sami.akin@ikt.uni-hannover.de (Sami Akın)

lossless single link and the first-in, first-out (FIFO) multiplexing model.

Assuming that the cross traffic[1] in a network is constant-rate and has a fluid-like flow, the relation between the input and output gaps of a probe traffic, i.e., g_{in} and g_{out}, in a single link with FIFO multiplexing is expressed as $g_{\text{out}} = \max\left\{g_{\text{in}}, \frac{g_{\text{in}}\lambda + l}{C}\right\}$ [1]. Recall that l is the probe packet size. Here, $g_{\text{in}}\lambda$ represents the amount of cross traffic that enters between two consecutive packets in the probe traffic. Re-ordering the given relation, one can particularly obtain the characteristic *gap response curve*, which is defined as the ratio between the output and input gaps of the probe traffic, as follows:

$$\frac{g_{\text{out}}}{g_{\text{in}}} = \begin{cases} 1 & \text{if } \frac{l}{g_{\text{in}}} \leq C - \lambda, \\ \frac{l}{g_{\text{in}}C} + \frac{\lambda}{C} & \text{if } \frac{l}{g_{\text{in}}} > C - \lambda. \end{cases} \tag{1}$$

In a real setting, measured output gaps are highly noisy due to deviations from the fluid-flow model[2] and measurement inaccuracies, i.e., imprecise time-stamping at the receiving end. Therefore, one technique to mitigate the noise in g_{out} is to use packet trains rather than a packet pair [7, 16] or to send several packet pairs [17] as the probe traffic. With packet trains, we refer to sending more than two packets consecutively. For example, if we send $n > 2$ packets in series with fixed g_{in} between them, where $n \in \mathbb{Z}^+$, we will have $n - 1$ output gaps measured at the end of a network to use in averaging out the noise.

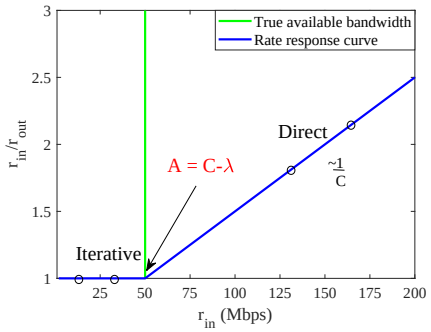
Now, let $g^{\text{j}}_{\text{out}} = t^{\text{j}+1}_{\text{out}} - t^{\text{j}}_{\text{out}}$ be the measured time difference between the j^{th} and $(j+1)^{\text{th}}$ transmitted packets for $j \in \{1, 2, \cdots, n-1\}$, where $t^{\text{j}}_{\text{out}}$ is the time[3] when the j^{th} packet arrives at the receiving end in the network. Then, one can calculate the output rate at the end of the network as $r_{\text{out}} = \frac{(n-1)l}{t^n_{\text{out}} - t^1_{\text{out}}} = \frac{l}{\frac{1}{n-1}\sum_{j=1}^{n-1} g^j_{\text{out}}}$. Moreover, because g_{in} is fixed at the input end, one can obtain the input rate as $r_{\text{in}} = \frac{l}{g_{\text{in}}}$. Given FIFO multiplexing, we have a rate-proportional capacity sharing between the cross and probe traffics; and therefore, the output rate of the probe traffic becomes $r_{\text{out}} = \frac{r_{\text{in}}}{r_{\text{in}}+\lambda}C$ if $r_{\text{in}} + \lambda > C$, and $r_{\text{out}} = r_{\text{in}}$ otherwise [5]. Now, we can express the ratio between r_{in} and r_{out}, which is also known as *rate response curve*, as

$$\frac{r_{\text{in}}}{r_{\text{out}}} = \begin{cases} 1, & \text{if } r_{\text{in}} \leq C - \lambda, \\ \frac{r_{\text{in}}+\lambda}{C}, & \text{otherwise.} \end{cases} \tag{2}$$

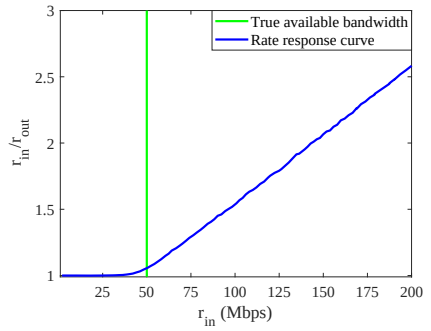
The advantage of (2) is the clear bend in the rate response curve at $r_{\text{in}} = C - \lambda$, which identifies the available bandwidth as shown in Fig. 1(a).

1.1.1. Model-based techniques

We can classify the active available bandwidth estimation methods, which are based on the constant-rate fluid-flow model,



(a) Fluid-flow model



(b) Exponential distributed packet inter-arrival times

Figure 1: Rate response curves in a single tight link with cross traffic that has (a) a constant-rate cross with infinitesimally small packets and (b) exponentially distributed packet inter-arrival times with discrete packets.

as *iterative* or *direct* probing techniques. The former search for the turning point in the rate response curve by sending repeated probes at increasing r_{in}, as long as $r_{\text{in}} = r_{\text{out}}$. When r_{in} reaches $C - \lambda$, the network saturates, and increasing the probe rate to a value beyond the available bandwidth results in self-induced congestion and causes $r_{\text{in}}/r_{\text{out}} > 1$. As a consequence, a queue builds up in the network buffers and one-way delays increase. These delays can be detected at the receiving end. This technique is implemented, for instance, in Pathload [3] and Pathchirp [9]. On the other hand, direct probing techniques estimate the upward segment of the rate response curve, i.e., the part where $r_{\text{in}} > C - \lambda$. This segment of the rate response curve is a function of C and λ. If we know C a priori, we can easily obtain the available bandwidth by setting $r_{\text{in}} = C$ and using (2) [10]. Particularly, we can express the available bandwidth as $A = C\left(2 - \frac{C}{r_{\text{out}}}\right)$. Another technique, assuming no knowledge about the capacity of a network in advance, is to send packet trains or pairs at least at two different input rates, where $r_{\text{in}} > C - \lambda$, to retrieve the two unknown network parameters, C and λ, as seen in TOPP [5], DietTOPP [6], and BART [12]. However, this requires strict choices of input rates such that r_{in} is always greater than the available bandwidth. This approach generally leads to congestion, and increasing traffic intensity by using packet trains results in performance degradation in other active traffics, e.g., delay and loss sensitive applications that carry live streaming video, voice over IP or multimedia teleconferencing [18].

[1] When the cross traffic in a network is constant-rate, λ refers to its rate. Otherwise, λ denotes the average cross traffic rate.

[2] Transmitted packets are not infinitesimally divisible. Moreover, there are packet losses in both the probe and cross traffic.

[3] We refer to the time when the first bit of a packet arrives at the receiving end of the network.

The aforementioned techniques usually assume a fluid-flow structure for cross traffic and that its rate stays constant over a long period of time. In reality, the certain minimum packet size and the stochastic nature of packet arrivals in cross traffic process cause deviations from the fluid-flow model. In addition, the estimation quality declines with noisy measurements and multiple links in real-time networks even if longer packet trains are employed. Particularly, the output rate of a probe traffic indeed becomes $r_{\text{out}} = \frac{r_{\text{in}}}{r_{\text{in}}+\lambda}C + \zeta$ if $r_{\text{in}} + \lambda > C$, and $r_{\text{out}} = r_{\text{in}} + \zeta$ otherwise, where ζ is the noise term. Therefore, it is of importance to consider the randomness in the cross traffic of a network. In order to improve available bandwidth estimation and reduce the impacts of randomness, statistical post-processing techniques such as Kalman filter [12, 19], majority rule [3], averaging repeated measurements [9, 10], and linear regression [6] are used. Albeit longer packet trains and statistical post-processing techniques reduce the bias in available bandwidth estimates, they are not able to tackle the deviations from the deterministic fluid-flow model. We can see the impacts of these deviations in real-time measurements in Fig. 1(b). Unlike the deterministic fluid-flow model in Fig. 1(a), the sharp bend around $r_{\text{in}} = C - \lambda$ that marks the available bandwidth is not clearly apparent in Fig. 1(b). This elastic deviation from the fluid-flow model leads to biased estimates. Furthermore, it is difficult to tailor methods to specific hardware implementations that influence the measurement accuracy. Therefore, the fundamental limitations on the model-based state-of-the-art bandwidth estimation methods make researchers explore the use of machine learning tools in bandwidth estimation.

1.1.2. Machine Learning Techniques

Machine learning techniques take attention in [20–26]. The authors in [20] have available bandwidth estimates from the network traffic data collected by passive measurements, while the authors in [21–23] use active probes to estimate the available bandwidth. The latter obtain bandwidth estimates by using Network Simulator (NS)-2 simulations in [21], ultra-high-speed 10 Gbps networks in [22], and the operational Long Term Evolution networks in [23]. What is common in these active probing methods is the use of packet chirps [9] that are probes composed of several packets sent at increasing rates. The rate increase is achieved by either a geometric reduction of input gaps [21, 27], or concatenating several packet trains with increasing rates to a multi-rate probe [22], or a linear increase in packet size [23]. Packet chirps let the detection of the turning point in (1), i.e., the available bandwidth, with the use of a single probe. They are, however, susceptible to random fluctuations [14]. Furthermore, the authors in [21] study the packet bursts, i.e., back-to-back probes, and conclude that the packet bursts are not adequate to estimate the available bandwidth. The authors in [22] consider constant-rate packet trains in an iterative manner to attain the available bandwidth. Here, machine learning solves a classification problem to estimate whether the rate of a packet train exceeds the available bandwidth. Depending on the result, the rate of the next packet train is decreased or increased in a binary search until the probe approaches the available bandwidth [3]. The authors in [22] give, however, preference to packet chirps.

Recently, in [24, 25], we implement a regression-based supervised learning technique for bandwidth estimation. Because this is a scale-invariant approach, we are able to apply the method to networks with arbitrarily changing capacity. Unlike the binary search performed in [22], our method consecutively chooses the probe rate that is expected to improve the bandwidth estimate the most, and hence, can effectively control the estimation procedure in an iterative method. In addition to the regression-based estimator, we also implement a binary classifier in [25]. Unlike the classifier in [22] where the classifier takes only one probe rate as input, our classifier uses a feature vector of probe rates to classify the probe rates into two groups, i.e., the rates above and below the available bandwidth, which leads to more accurate estimates. More recently, we employ a reinforcement learning-based multiclass estimation algorithm in [26]. This approach does not require a training phase unlike supervised learning-based techniques and declares the available bandwidth as the input rate that maximizes a reward metric, which is indirectly a function of g_{in}, g_{out} and l, after an exploration-exploitation mechanism is employed. For more details on reinforcement learning, we refer interested readers to [28]. However, because there is no training phase and the system learns a network in an online manner, the convergence time may be longer in certain network scenarios, e.g., when the available bandwidth changes very frequently.

1.2. Contributions

In this paper, we propose a machine learning-based method that provides available bandwidth estimates accurately and fast using fewer probe packets. Particularly, we formulate available bandwidth as a multiclass classification problem. We divide a range of available bandwidth, e.g., from zero bits/second to the capacity of a network, into subranges and assign a class to each subrange. Specifically, we design a classifier that takes as input a feature vector, composed of r_{in}, r_{out}, and results in the class that the available bandwidth belongs to. We evaluate our technique in controlled experimental setups in a network testbed employing supervised learning techniques such as support vector machines (SVMs), k-nearest neighbor (k-NN), bootstrap aggregation (Bagging), adaptive boosting (AdaBoost) and neural networks (NNs). We show that our method performs better than the fluid-flow model-based direct probing technique that employs Kalman filter, which has been used in the literature [12], in network scenarios where the deterministic fluid-flow model assumptions fail due to the bursty cross traffic nature and multiple links. We show that our method converges faster and its estimates have less variations around the available bandwidth. We further compare our classification-based method with the regression-based counterpart. Ours outperforms its counterpart in general. In addition, we consider network scenarios where the tight link capacity and the cross traffic intensity vary over time randomly. We make the aforementioned comparisons and show that our method generates better results. Finally, assuming that the available bandwidth changes over time in a correlated way, we apply filtering techniques to bandwidth estimates and decrease the estimation error.

2. Multiclass classification-based Method

In this section, we present our supervised learning-based approach, where we address the available bandwidth estimation as a multiclass classification problem. We initially discuss how available bandwidth classes are formed and define a feature vector for the classifier. Then, we present the implemented supervised learning techniques.

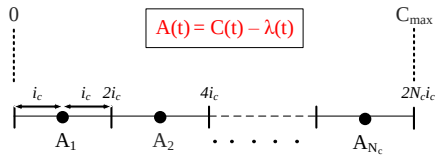
2.1. Available Bandwidth Classes

Let us consider a network path with capacity $C(t)$ and cross traffic rate $\lambda(t)$ that vary over time as shown in Fig. 2(a), where $C(t) \in [C_{\min}, C_{\max}]$, $\lambda(t) \in [0, C(t)]$, and $C_{\min}, C_{\max} \in \mathbb{R}^+$. One can think of a path where the channel capacity varies over time due to environmental conditions, for instance, in wireless networks. Therefore, the available bandwidth is not constant, i.e., $A(t) = C(t) - \lambda(t)$. Here, we divide the range[4] $[0, C_{\max}]$ into N_c subranges, where each subrange is denoted by one class, A_c, for $c \in \{1, 2, \cdots, N_c\}$, and each subrange covers a distance of $2i_c$ units. For instance, the centers of classes A_1 and A_2 are $2i_c$ units apart from each other as seen in Fig. 2(a). Basically, given a feature vector, our estimator returns the class that the actual available bandwidth belongs to and sets the estimate to the class center. For instance, if the estimator returns A_c as the estimated class, then the available bandwidth estimate is $(2c-1)i_c$ units. Here, one can observe that the smaller the subrange a class refers to, the more classes we need to assign to cover the entire range from 0 to $C_{\max}$. Particularly, there is a trade-off between the number of classes and the subrange a class encompasses; the training process takes longer with more classes and we need to obtain more experimental data for training, but the deviation around the actual available bandwidth decreases. In order to understand the aforementioned technique, let us consider the experimentally obtained rate response curve shown in Fig. 2(b) as an example, where $C_{\max} = 200$ Mbps and $i_c = 10$ Mbps. The entire range is divided into 10 regions.

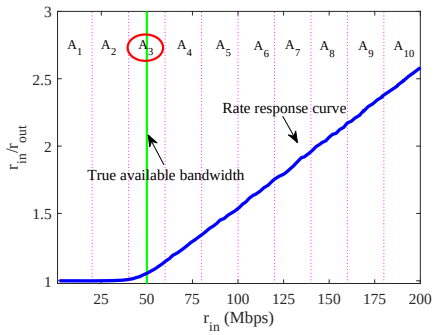
We define the input to the classifier as a p-dimensional feature vector, where each element is the ratio between the input and corresponding output rates, i.e., $\frac{r_{\text{in}}}{r_{\text{out}}}$. Here, r_{in} values are spaced equidistantly with increment δ_r, and hence we have a vector of input rates $\mathbf{r}_{\text{in}} = [r^1_{\text{in}} \ \cdots \ r^p_{\text{in}}]$, where $r^i_{\text{in}} = i\delta_r$ for $i \in \{1, \cdots, p\}$. Considering the case $C_{\max} = 200$ Mbps as an example, one can set $\delta_r = 25$ Mbps and $p = 8$[5]. Subsequently, a probe stream comprising p probe trains is sent to the network with defined p input rates, and the corresponding output rates are obtained. Hence, a feature vector, $f = \left[\frac{r^1_{in}}{r^1_{out}}, ..., \frac{r^p_{in}}{r^p_{out}}\right]$, is provided as an input to the classifier. If the classifier returns A_3 as the estimated class, then the available bandwidth estimate $(2c-1)i_c$ is 50 Mbps.

[4] One can define a range with borders from a non-zero value to another value that is greater than $C_{\max}$.

[5] Notice that here the maximum input rate $p\delta_r$ is not limited by the maximum capacity, $C_{\max}$. However, in order to prevent any congestion, it is better to limit the maximum input rate to a value smaller than $C_{\max}$.



(a) Available bandwidth classes



(b) Rate response curve

Figure 2: (a) Available bandwidth classification between zero Mbps to $C_{\max}$ and (b) its projection onto the rate response curve of a single tight link of capacity $C = 100$ Mbps in the presence of exponential cross traffic with average rate $\lambda = 50$.

2.2. Machine Learning Techniques

To evaluate our machine learning-based approach for bandwidth classification, we consider the following supervised machine learning techniques. However, one can easily adapt the proposed scheme to other supervised learning techniques.

2.2.1. SVMs

SVMs are supervised learning algorithms, which can be used for classification and regression problems. An SVM-based classifier can project a feature vector to a higher dimensional space and find a hyper-plane that separates two given classes. In this paper, we use the Gaussian radial basis function. However, one can use polynomial kernel functions as well. As SVMs are originally designed for binary classification [29], error-correcting output codes are used to reduce a multiclass problem to a set of multiple binary classification problems [30]. As described in [30], the output coding for multiclass problems is composed of two stages. In the training stage, multiple independent binary classifiers are constructed, where each classifier is trained to distinguish between two disjoint subsets of class labels. In the second stage, i.e., the classification part, the estimations of trained binary classifiers are combined to test instances, and a voting scheme is used to decide the class.

2.2.2. k-NN

A k-NN algorithm is a non-parametric lazy supervised machine learning technique. The principle behind k-NN is the majority voting rule applied over k nearest neighbors, where the test data is assigned to the class that is chosen by the majority vote of its k nearest neighbors. k-NN mainly involves two hyper-parameters, i.e., the number of neighbors involved in the decision, k, and the distance function denoting how far

neighbors are from each other. Small k provides a flexible fit, which has low bias but high variance. On the other hand, large k averages more voters in each estimation and hence is more resilient to outliers, which means lower variance but increased bias. Regarding a bias-variance trade-off, we use weighted k-NN, where we set the nearest neighbor $k = 10$. Further, each nearest neighbor is given a weight of $1/d^2$, where we set d as the Euclidean distance metric. We choose k-NN for the bandwidth classification problem because it is non-parametric; and therefore, we can avoid mismodeling the underlying distribution of the data. In real bandwidth estimation problems, which deviate from the fluid-flow model assumptions, it is difficult to find the underlying data distribution that matches the theoretical distributions. For example, choosing a learning model that assumes a Gaussian distribution for non-Gaussian distributed data will cause the algorithm to make poor estimations. In contrast to SVM, k-NN is instance-based, and hence, does not explicitly learn a model. Instead, it chooses to memorize the training instances, which are subsequently used as *knowledge* in the estimation phase. The minimal training phase of k-NN results in memory cost as we have to store possibly a huge data set. Moreover, it causes computational cost in the testing phase as well because classifying a given feature vector requires a run-down of the whole data set.

2.2.3. Bagging

The main causes of error in learning models are noise, bias, and variance. However, for applications, an accurate and reliable available bandwidth estimation with less bias and variance is of utmost importance, e.g., to minimize the adverse effects of quality variance and video freezing on viewers in rate-adaptive applications, such as Dynamic Adaptive Streaming over the Hypertext Transfer Protocol. In order to obtain a better estimation performance, we consider ensemble learning methods for classification such as bagging and boosting to decrease the variance and bias in the model, respectively. Bagging uses a bootstrapping technique to randomly sample the input data with a replacement to create multiple data subsets. Then, a set of models is trained on each of these subsets, and their estimations are aggregated to make the final combined estimation by averaging. The averaging of multiple decision trees reduces the variance and avoids over-fitting. The performance of the ensemble technique depends on the setting of both the ensemble and that of the weak learners. A large number of trees increase the performance and make the estimations more stable, but having many trees slows down the computation, which is not desired for real-time bandwidth estimations. We choose the hyper-parameters to have a trade-off between the estimation power and the computational speed. Therefore, we set the number of ensemble learning cycles to be 30.

2.2.4. AdaBoost

In order to reduce the bias in bandwidth estimation, we consider AdaBoost, which is another ensemble-based algorithm. Instead of training parallel models as in Bagging, AdaBoost is an iterative technique because it uses multiple iterations to generate a single composite strong learner. AdaBoost fits a classifier to the original dataset. If an observation is misclassified, it tries to increase the weight of this observation. The subsequent classifier acknowledges the updated weights and attempts to reduce the classification error of its predecessor. The procedure is repeated until the required accuracy is achieved. To achieve the balance between the speed and the accuracy, we set the learning rate to 0.1 and the number of ensemble members to 30.

2.2.5. NNs

NNs are complex models composed of simple elements, called neurons, which try to mimic the way human brains develop classification rules. We use a feed-forward multi-layer perceptron consisting of three different layers of neurons: an input layer, a hidden layer, and an output layer, with each layer receiving inputs from previous layers, and passing outputs to further layers. The neural net iterates for a predetermined number of iterations, called epochs. After each epoch, the weights between the neurons are updated considering the cost function at the end of the network. We have a shallow NN consisting of one hidden layer with 40 neurons. For training of the NN, we first implement an auto-encoder for each layer separately and then fine-tune the network using a scaled conjugate gradient. We optimize the cross-entropy error function requiring approximately 1000 epochs until the convergence is achieved.

3. Evaluation Metrics

In order to gain a better assessment of the effectiveness of the aforementioned machine learning models in the testing phase, we use the K-fold cross-validation technique. In the K-fold cross-validation technique, the entire data set is split into K subsets, where each subset is used for testing and the remaining $K - 1$ subsets are used for training. Because more partitions lead to a smaller bias but a higher variance, we set $K = 5$ to have a balanced bias-variance trade-off. We employ the performance measures described in [31], i.e., *accuracy*, *precision*, and *recall*, to experimentally substantiate our classification-based available bandwidth estimation technique. The accuracy of estimates provides the overall effectiveness of the method and calculates the ratio of the number of predicted available bandwidth estimates that are classified correctly to the total number of predicted available bandwidth estimates. We calculate the average accuracy $\mathrm{Acc}_{\mathrm{avg}}$ over N_{c} available bandwidth classes as follows:

$$\mathrm{Acc}_{\mathrm{avg}} = \frac{\sum_{c=1}^{N_{\mathrm{c}}} \frac{\mathrm{TP}_c+\mathrm{TN}_c}{\mathrm{TP}_c+\mathrm{FN}_c+\mathrm{FP}_c+\mathrm{TN}_c}}{N_{\mathrm{c}}}.$$

Recalling from Sec. 2.1 that A_c is the class representing c^{th} subrange of the entire possible range of available bandwidth, i.e., $[0, C_{max}]$, we have true positives (TP_c) as the number of correct bandwidth estimates classified to A_c, false positives (FP_c) as the number of incorrect estimates assigned to A_c, true negatives (TN_c) as the number of correct estimates not assigned to A_c, and false negatives (FN_c) as the number of incorrect estimates not assigned to A_c. However, the accuracy parameter is not always the best measure to determine the effectiveness of a classifier in multi-class classifications due to the class imbalance problem.

The randomly changing capacity and cross traffic of a network path may cause the number of instances of one class to exceed than the others. Here, each instance refers to the ratio of input and output rates, i.e., $r_{\text{in}}/r_{\text{out}}$. With imbalanced data, the test instances belonging to a small class are misclassified more often as compared to the prevalent classes with more instances. Therefore, we consider the precision and recall parameters as well.

The precision parameter indicates how precise the estimates are, i.e., the bandwidth estimates that are correctly classified to the available bandwidth class, A_c, out of all the bandwidth estimates, including the incorrect ones that are classified to A_c. The recall parameter shows the sensitivity of the classifier, i.e., how many of all the actual bandwidth estimates that belong to A_c are detected correctly. High scores for both the precision and recall parameters show that the classifier returns accurate results as well as a majority of all the positive results. Herein, since the precision and recall parameters are inversely related to each other, we compute F-measure by taking their harmonic mean as follows:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{3}$$

The F-measure score takes a value between zero and one. The higher the F- measure score of a classification technique is, the better its performance is. Particularly, when the F-measure score of a classifier is one, the classifier separates the classes without error. The overall F-measure score of the entire classification problem can be computed by two different averaging techniques, i.e., macro-averaging (M) and micro-averaging (μ). In micro-averaging, the F-measure score is computed globally over all the categorical decisions, i.e., Precision_μ and Recall_μ are obtained by summing over all the individual decisions:

$$\text{Precision}_\mu = \frac{\sum_{c=1}^{N_c} \text{TP}_c}{\sum_{c=1}^{N_c} (\text{TP}_c + \text{FP}_c)} \quad \text{and} \quad \text{Recall}_\mu = \frac{\sum_{c=1}^{N_c} \text{TP}_c}{\sum_{c=1}^{N_c} (\text{TP}_c + \text{FN}_c)}. \tag{4}$$

In macro-averaging, Precision_M and Recall_M are obtained for each class A_c, and then average is taken:

$$\text{Precision}_M = \frac{\sum_{c=1}^{N_c} \frac{\text{TP}_c}{\text{TP}_c+\text{FP}_c}}{N_c} \quad \text{and} \quad \text{Recall}_M = \frac{\sum_{c=1}^{N_c} \frac{\text{TP}_c}{\text{TP}_c+\text{FN}_c}}{N_c}. \tag{5}$$

Now, we can plug (4) and (5) into (3) in order to calculate the F-measure score.

Furthermore, we calculate the empirical standard deviation (SD) as a metric to measure the precision of the bandwidth estimates:

$$SD = \sqrt{\frac{\sum_{t=1}^{K_r} (\hat{A}_c(t) - A(t))^2}{K_r - 1}}, \tag{6}$$

where t is the discrete time index and K_r is the number of measurements. Here, we assume that a measurement takes place in a certain time period, and during one period the available bandwidth stays constant. Therefore, one can consider t as the time frame index. In (6), $A(t)$ is the actual available bandwidth value and $\hat{A}_c(t)$ is the bandwidth estimate after the classification is performed. Technically, $\hat{A}_c(t)$ is the center of the class to which the bandwidth is assigned.

Notice also that even if the classifier makes a correct classification, there will still be an error because each class defines a range of real numbers and the maximum error will occur when the available bandwidth is at the edge of the class. For instance, let A_1 be the class the available bandwidth belongs to and assume that the classifier estimates its class correctly. The difference between the available bandwidth and the estimate, which is the center of the subrange, will be maximum when the available bandwidth is either zero or $2i_c$ as can be seen in Fig. 3(a). Moreover, in the case of a wrong estimate of a class, the maximum possible error will be equal to the difference between the estimated class center and the far edge of the actual class. For example, let A_1 be the actual class of the available bandwidth and A_2 be the estimated class. As can be seen in Fig. 3(b), the maximum error occurs when the available bandwidth is at the far edge of A_1 to A_2, which is the edge 0. Therefore, in addition to the SD metric provided in (6), we calculate an upper bound on the SD metric as follows:

$$SD_u = \sqrt{\frac{\sum_{t=1}^{K_r} (i_c + |\hat{A}_c(t) - A_c(t)|)^2}{K_r - 1}}, \tag{7}$$

where $A_c(t)$ is the class of the available bandwidth. (7) gives us the maximum deviation that we may observe in our measurements at any time.

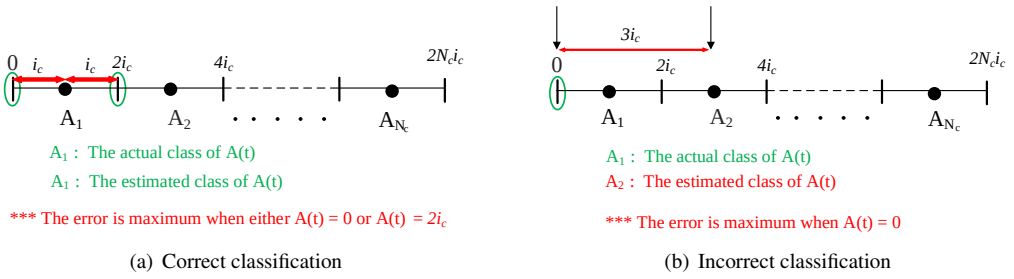
4. Performance Analysis

In this section, we initially describe our testbed, and then, discuss the model-based direct probing technique that employs a Kalman filter [12]. Finally, we present our experimental results.

4.1. Experimental Setup

We generate different data sets for training and evaluation using a controlled network testbed, which is located at Leibniz Universität Hannover, and comprises 80 servers. We connect each server deploying minimum four network switches with 1 Gbps and 10 Gbps link capacities. We use the Emulab software [32] to manage the testbed, where we configure the servers as hosts and routers, and connect them using Virtual Local Area Networks to implement the desired topology. We set up a dumbbell topology with multiple links, as shown in Fig. 4. We use the MoonGen software [33] for the emulation of the link capacities that differ from the native physical Ethernet capacity. To achieve an accurate spacing of packets that matches the emulated capacity, we fill the gaps between packets with dummy frames by using MoonGen, which are later discarded at the output of the link. We use the "forward rate Lua script" for the MoonGen to achieve the desired forwarding rate at the transmission and reception ports.

We create cross traffic models that have different distributions by employing distributed internet traffic generator [34]. Each cross traffic is *single-hop-persistent*, i.e., at each link, another



(a) Correct classification (b) Incorrect classification

Figure 3: The maximum error in the case of (a) correct and (b) incorrect bandwidth classifications.

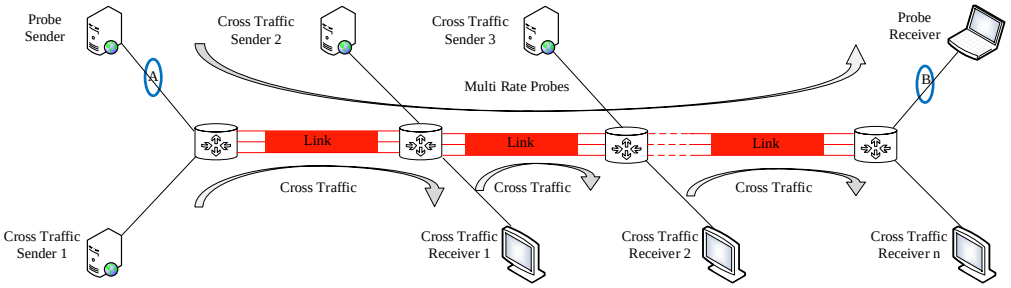


Figure 4: Dumbbell topology set up using the Emulab and MoonGen software.

cross traffic is multiplexed. The probe traffic that we deploy to estimate the end-to-end available bandwidth is *path-persistent*. Particularly, it travels the entire network path from the probe sender to the probe receiver. We use real-time User Datagram Protocol (UDP) data emitter and collector, known as RUDE and CRUDE [35], respectively, to generate the UDP probe streams. A probe stream consists of a series of p packet trains, each having n packets. These p different packet trains, respectively, correspond to p different probe rates that successively increase with increment rate δ_r. In order to reduce the amount of probe traffic injected into network path, we set $p = 6$, and to reduce the impact of noise on the output gaps, we compute r_{out} over $n = 100$ packets. The packet trains are sent at six different rates with the rate increment, i.e., $p = 6$ and $\delta_r = 25$ Mbps unless otherwise stated. We set the packet length to $l = 1514$ bytes, including the Ethernet header both in the probe and cross traffic. We use "libpcap" to capture the packets at the probe sender and receiver, and the packet timestamps are generated at points A and B, respectively, as shown in Fig. 4. We also use a specific *Endace* Data Acquisition and Generation measurement card to obtain the accurate reference timestamps. We use the timestamps to compute r_{in} and r_{out} for each packet train.

In order to investigate the sensitivity of our method with respect to different network parameters, we set up a number of topologies, where the link capacities and the cross traffic intensities are chosen randomly. The capacity in a link, $C_i(t)$, is chosen as a random number in the interval [10, 200] Mbps, and the cross traffic intensity in the same link is chosen relative to the link capacity as $\lambda_i(t) = U[0,1]C_i(t)$ Mbps, where U denotes uniform distribution. For each random network configuration, 100 repeated experiments are performed. Particularly, having more than 100 different network configurations, we run more than 10000 experiments to generate different data sets for training and testing. Recall that the network capacity is $C(t) = \min(C_i(t))$ and the available bandwidth is $\min(C_i(t) - \lambda_i(t))$.

4.2. Model-based Reference Implementation

In order to understand how much performance gain our estimation technique realizes, we compare its performance with that of the direct probing technique that employs Kalman filter [12]. Although available bandwidth estimation tools significantly differ in the selection and amount of probe traffics in experimental testbeds, we tailor our technique and the direct probing technique to work on the same database in order to provide a fair ground for comparison. Hence, in the following, we describe the direct probing technique.

Direct Probing: In order to implement Kalman filter in the direct probing technique, we invoke the inter-packet strain as $\xi(t) = \frac{r_{\text{in}}}{r_{\text{out}}} - 1$ for $r_{\text{in}} > C(t) - \lambda(t)$, and $\xi = 0$ otherwise [12]. Now, inserting $\xi(t)$ into (2), we obtain $\xi(t) = r_{\text{in}}\frac{1}{C(t)} + \frac{\lambda - C(t)}{C(t)}$ when $r_{\text{in}} > C(t) - \lambda(t)$. Subsequently, we define $\alpha(t) = \frac{1}{C(t)}$ and $\beta(t) = \frac{\lambda(t) - C(t)}{C(t)}$ and achieve the strain parameter as

$$\xi(t) = \begin{cases} 0, & \text{if } r_{in} \leq C(t) - \lambda(t), \\ \alpha(t) r_{in} + \beta(t), & \text{otherwise.} \end{cases} \tag{8}$$

Following the fluid-flow model assumption, we can see that $\xi(t)$ is zero in the absence of congestion, and it grows proportionally to the probe traffic rate, r_{in}, when the probing rate exceeds the available bandwidth. As can be seen in (8), the model is piece-wise linear and has a sharp turn at $r_{\text{in}} = C(t) - \lambda(t)$, which inhibits the direct application of Kalman filter. In order to overcome the problem, we feed only those measurements

that satisfy $r_{\text{in}} > \hat{A}(t-1)$ to the filter, where $\hat{A}(t-1)$ is the available bandwidth estimate by the direct probing technique in the $(t-1)^{\text{th}}$ time frame. Since, the direct probing technique seeks to estimate the upward line segment of the rate response curve, which is determined by $C(t)$ and $\lambda(t)$, we can express the state vector of Kalman filter as $\mathbf{x}(t) = \left[\alpha(t)\,\beta(t)\right]^{\dagger}$, where $\dagger$ is the transpose operator. Assuming that the network statistics remain constant over a long period of time compared to the time period, t, we set the state transition matrix of Kalman filter to the identity matrix, and hence, we define the state transition process as $\mathbf{x}(t) = \mathbf{x}(t-1) + \mathbf{w}(t)$, where $\mathbf{w}(t)$ is the process noise vector, which is zero-mean and Gaussian-distributed with 2×2 covariance matrix Q [12]. The authors in [12] define Q as a parameter that indicates the deviation of the system from the fluid-flow model, and use it as a tuning parameter to increase the estimation quality. Subsequently, we define the measurement model as $\mathbf{z}(t) = \mathbf{H}(t)\mathbf{x}(t) + \mathbf{v}(t)$, where $\mathbf{z}(t)$ and $\mathbf{v}(t)$ are the $p \times 1$-dimensional observation and noise vectors. Recall that in each measurement, we send packets at p different input rates, r_{in}, and hence, we have the $p \times 1$-dimensional observation vector. Herein, $\mathbf{v}(t)$ is zero-mean and Gaussian distributed with $p \times p$ covariance matrix R. Moreover, the observation matrix, $\mathbf{H}(t)$ is a $p \times 2$ matrix and is given as

$$\mathbf{H}(t) = \begin{bmatrix} r_{\text{in}}^{1}(t) & & 1 \\ & \vdots & \\ r_{\text{in}}^{p}(t) & & 1 \end{bmatrix},$$

where $r_{\text{in}}^{i}(t)$ for $i \in \{1, \cdots, p\}$ is a function of time because we choose the input rates at t such that the input rates are greater than the available bandwidth estimate at $t-1$, i.e., $r_{\text{in}}^{i}(t) > \hat{A}(t-1)$. We finally note that we define $Q = \eta\mathbf{I}$ and set $\eta = 10^{-2}$ in order to guarantee a faster convergence to the actual available bandwidth with less variations in the estimates.

4.3. Experimental Results

We evaluate the performance of the proposed available bandwidth estimation technique in testbed scenarios that capture the nature of practical settings very closely. When creating the training and test data, we do not assume an equally likely distribution of instances to investigate the impact of *class imbalance problem* on the performance of our classification-based estimator. Furthermore, we employ cross traffic models with the exponential and Pareto distributions to reflect a moderate and heavy burstiness nature of a real network, respectively. We further consider multiple links with discrete and random traffic patterns. We generate a training set in the presence of exponentially distributed cross traffic as discussed in Sec. 4.1.

By choosing the capacity and cross traffic intensity values randomly, we generate available bandwidth values between 10 Mbps to 145 Mbps. We define the available bandwidth class centers from 1 to 200 Mbps, where each bandwidth covers a subrange of 1 Mpbs. Particularly, the set of available bandwidth classes is $\{1, 2, \cdots, 200\}$. For each available bandwidth, we repeat the experiment 100 times.

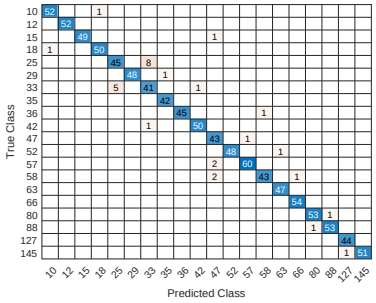


Figure 5: The confusion matrix obtained after using the SVM-based estimator with different network capacities and cross traffic intensities during the testing phase.

4.3.1. Single link

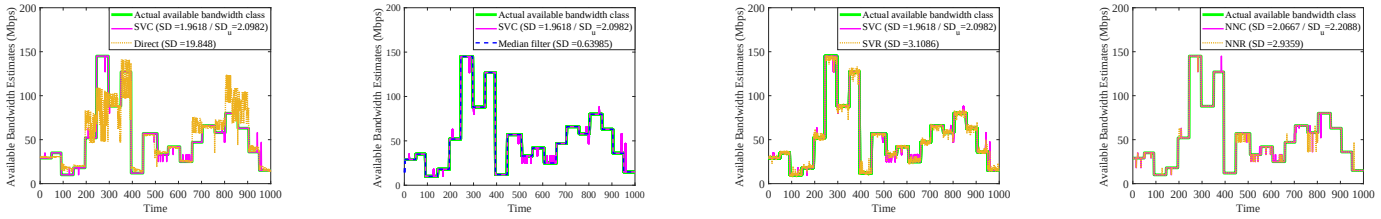
In this section, we show the testing results of an estimator in single link networks, which is trained in a network with cross traffic having exponentially distributed packet interarrival times, in the same network. Then, we test the same estimator in networks with no and heavy burstiness. Furthermore, we decrease the number of packet trains from six to five and four, i.e., we set $p = 5$ and $p = 4$, and test the estimator.

Exponentially Distributed Cross Traffic and Mutually Exclusive Classes. In Table 1, we show the averaged accuracy, micro-averaging and macro-averaging results after employing the SVM, k-NN, Bagging, AdaBoost and NN techniques. Notice that precision, recall and F-measure score are equal when calculating the micro-averaging results. This is owing to that a false positive with respect to one class will be a false negative with respect to another class in the case of misclassification. For instance, this can be also seen in the confusion matrix of the SVM-based estimator in Fig. 5, where one can show $\sum_{c=1}^{N_c} \text{FP}_c = \sum_{c=1}^{N_c} \text{FN}_c = 30$ by counting all the false instances. Moreover, as can be seen in Table 1, while all the classifiers achieve comparable accuracy values, the SVM-based estimator outperforms the rest with 99.7%. It exceeds the others in other metrics as well. Its better performance is due to the reason that the SVM-based estimator projects the data to a higher dimensional feature space in order to separate the classes efficiently. Moreover, since we test all the techniques in the network we train them, looking at the results, we can say that our estimator neither over-fits nor under-fits; particularly, we are able to generalize the network model very well.

We compare the performance of the SVM-based classifier (SVC) with that of the direct probing technique described in Sec. 4.2 under the same conditions. We use the same testing data set when running both the techniques. As can be seen in Fig. 6(a), the SVM-based estimator achieves more accurate estimates considering the SD and corresponding upper bound results (SD_u). Since the bandwidth values stay constant for a certain period, the direct probing technique can converge the actual bandwidth. However, it is not able to track the sudden changes in the bandwidth. Another reason for falling behind the SVM-based estimator is that six measurements are not enough to estimate the upward line segment of the rate response curve

Table 1: Testing performance in a single link with the presence of cross traffic that has exponentially distributed packet inter-arrival times.

Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9970	0.9964	0.9941	0.9922	0.9969
Precision_{μ}	0.9700	0.9640	0.9410	0.9220	0.9690
Recall_{μ}	0.9700	0.9640	0.9410	0.9220	0.9690
F1_{μ}	0.9700	0.9640	0.9410	0.9220	0.9690
Precision_{M}	0.9709	0.9648	0.9433	0.9237	0.9699
Recall_{M}	0.9711	0.9645	0.9426	0.9240	0.9690
F1_{M}	0.9710	0.9646	0.9429	0.9238	0.9694



(a) SVM and direct probing comparison

(b) Bandwidth estimate after median filter

(c) SVM-based classification and regression comparison

(d) NN-based classification and regression comparison

Figure 6: Available bandwidth estimates and their SD in the presence of cross traffic that has exponentially distributed packet inter-arrival times using $p = 6$ packet trains in single tight link network.

when the available bandwidth is close to the network capacity.

Since the bandwidth values stay constant over a certain time period, we consider the correlation among the bandwidth values over time and implement a one-dimensional median filter to smooth the estimates. Herein, one can observe that the classification errors look like the salt-and-pepper noise. Therefore, the median filter works very well in smoothing the classification errors. We set the filter order to five because increasing the order more leads to increased delays. Fig. 6(b) compares the results of estimated classes with filtered classes. The SD has been significantly reduced after applying a median filter to the estimated available bandwidth classes.

We further compare the SVM-based (SVC) and NN-based (NNC) classifiers results, which show the highest two performances in Table 1, with the results of the corresponding regression-based techniques that are employed in [24, 25]. In the SVM-based regression (SVR) estimator, we set the Gaussian radial basis function as the kernel, and in the NN-based regression (NNR) estimator, we use a shallow NN consisting of one hidden layer with 40 neurons. As can be seen in Fig. 6(c) and Fig. 6(d), although the results are comparable for both the classification and regression-based estimators, the SD values are better when implementing the classification based estimators.

Bursty Cross Traffic. In order to evaluate our method in the presence of cross traffic with an unknown burstiness, we test the aforementioned estimators, which are trained in a network with cross traffic having exponentially distributed packet inter-arrival times, in the following cross traffic models:

1. No burstiness with constant packet inter-arrivals.
2. Heavy burstiness with Pareto-distributed inter-arrival times with infinite variance and shape parameter $\alpha_s = 1.5$.

Recall that the burstiness of cross traffic can cause queueing at a link even if the probe rate is below the available bandwidth, i.e., if $r_{\text{in}} < C(t) - \lambda(t)$, which leads to deviations from the ideal rate response curve. We show these deviations in Fig. 1(b) and we observe the maximum deviation when $r_{\text{in}} = C(t) - \lambda(t)$. Moreover, the strong deviation blurs the bend that marks the available bandwidth and causes an estimation bias. Therefore, we test our bandwidth estimators in networks having different cross traffic types.

In Table 2 and Table 3, we show the averaged accuracy, and the micro-averaged and macro-averaged F-measure results when there is cross traffic with constant and Pareto-distributed packet inter-arrival times, respectively. The trained classifier performs well when the burstiness in the cross traffic decreases as can be seen in Table 2. However, the misclassification increases significantly when the cross traffic becomes bursty as can be seen in Table 3. Notice that the F-measure score decreases. We can explain this with the class overlap problem caused by the noise introduced by the increased burstiness. The noise introduced by the increased burstiness of cross traffic causes instances from nearby classes to reside in overlapping regions in the feature space, which makes it difficult for the classifier to assign instances to correct classes, hence the misclassification error rate increases. Therefore, one solution to this problem is to train the estimator in the presence of cross traffic having a heavy bursty nature.

In Fig. 7(a), we compare the results obtained by implementing the direct probing technique and the NN-based estimator.

Recall that the NN-based estimator shows the highest accuracy in Table 2. The NN-based estimator, which is trained in a network with cross traffic having exponentially distributed packet inter-arrival times, performs almost with no misclassification in the case of no burstiness. In Fig. 7(b), we see that the SD decreases after applying a median filter to the estimates obtained by the NN-based estimator. In Fig. 7(c) and Fig. 7(d), we show the results for both the classification and regression-based methods using NN and SVM, respectively. While all of them perform well and provide comparable results, the NN-based regression estimator has the minimum SD.

Furthermore, in Fig. 8(a), we compare the results obtained from the direct probing technique with the SVM-based estimator, which shows the highest accuracy in Table 3 in the case of cross traffic with Pareto-distributed packet inter-arrival times. Again, the SVM-based estimator, which is trained in a network with cross traffic having exponentially distributed packet inter-arrival times, performs better than the direct probing technique. In Fig. 8(b), we show that the estimates are improved after applying the median filter to the estimates of the SVM-based estimator. In Fig. 8(c) and Fig. 8(d), we compare the classification and regression-based methods again using SVM and NN, respectively. In all techniques, the bias in the estimates increases with the increasing burstiness while the classification methods have less SD. Here, we can easily conclude that the NN-based regression estimator is better than the other when there is no burstiness in cross traffic, while utilizing SVM and classification improves the estimation quality.

Different Number of Packet Trains. In general, the excessive amount of probe traffic is undesired in networks. In the previous sections, we use a probe traffic with $p = 6$ packet trains to estimate the bandwidth thereby reducing the amount of artificial traffic injected to the network. Now, we investigate how much we can decrease the probe traffic without a significant impact on the performance of the classification-based method.

As we can interpret from the rate response curve in Fig. 1(a), the probe rates that are greater than or equal to the available bandwidth, i.e., $r_{\text{in}} \geq C(t) - \lambda(t)$, provide the relevant information to estimate the available bandwidth, but meanwhile, cause congestion in the network. Therefore, in order to investigate the effect of the number of packet trains on the performance and harness the congestion risk, we exclude the highest probing rates while setting the number of packet trains to $p = 5$ and $p = 4$, respectively. Considering again a single link with cross traffic having exponentially distributed packet inter-arrival times, we show the results in Table 4 and Table 5, when the number of packet trains is five and four, respectively. We can see that the performance decreases with the decreasing number of packet trains. However, the decrease in accuracy is negligible, and the misclassification performance is affected slightly. Noting the trade-off between the amount of injected traffic and the performance loss, we can still use four packet trains if we need to avoid network congestion.

In Fig. 9(a) and Fig. 10(a), we compare the results obtained by the direct probing technique with the ones by the SVM-based estimator when we have $p = 5$ and $p = 4$, respectively. Despite the noise increase in the estimates with the decrease in the probe traffic, the results obtained by the SVM-based estimator are still more accurate than the ones by the direct probing technique. Moreover, the direct probing technique has more SD. This is again because when the available bandwidth approaches the capacity, there is no inter-packet strain measurement left on the upward segment of the rate response curve due to omitting the higher probing rates. Furthermore, in Fig. 9(b) and Fig. 10(b), we show that after applying the median filter to the estimates of the SVM-based estimator, the estimates become less noisy again. Finally, we compare the SVM-based classification and regression estimators in Fig. 9(c) and Fig. 10(c). The results are close to each other. Finally, we note the noise in the estimates increases with the less packet trains.

4.3.2. Multiple Links

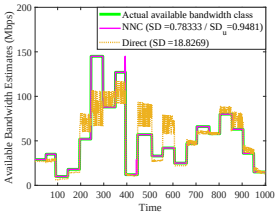
Networks with multiple links pose a well-known challenge in available bandwidth estimation due to the fact that the linear model in Fig. 1(a) does not reflect the nature of multiple hop networks thoroughly. Hence, we extend our testbed from a single link network to a four-link (four-hop) network. In order to generate the training and test data, we send a probe traffic consisting of six packet trains through the network with one link initially, and then, two, three and four hops, respectively, where each link is exposed to cross traffic with exponentially distributed packet inter-arrival times with average rate $\lambda = 50$ Mbps. In all the network settings, we set the probe stream as path-persistent and the cross traffic as single hop-persistent. The maximum capacity of each link is 100 Mbps. Following the acquisition of the data, we train the estimator.

In Table 6, 7 and 8, we show the averaged accuracy, and the micro-averaged and macro-averaged F-measure results for the 2-hop, 3-hop and 4-hop networks, respectively. Compared to the results of a single tight link in Table 1, we see that the misclassification errors increase with the increase in the number of multiple links. We can explain this with the fact that the probe traffic has a constant-rate, i.e., the input rate is r_{in}, only in the first link, and the packet inter-arrival times of the probe traffic in the following links become randomized. This increased randomness in the probe traffic in the second link and the others along with the random cross traffic causes more noise in the output rate measurements. Hence, the classification becomes more difficult for the estimator. Particularly, we need more data for the training of the estimator in order to capture the randomness of the network with multiple links.

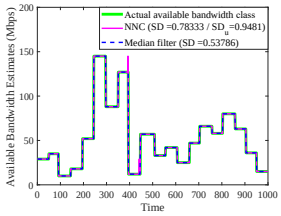
In Fig. 11(a), Fig. 12(a) and Fig. 13(a), we compare the results obtained by the direct probing technique with the ones by the SVM-based estimator in the 2-hop, 3-hop, and 4-hop networks, respectively. Although the noise in the estimates increases with the increase in the number of links, the results obtained by the classification-based methods are more accurate than that of the direct probing technique. Furthermore, after applying the median filter to the of the estimates of the SVM-based estimator, we filter out the noise in the estimates as can be seen in Fig. 11(b), Fig. 12(b) and Fig. 13(b). Finally, we compare the SVM-based classification and regression estimators. Noting that the results are comparably closer to each other,

Table 2: Testing performance in a single link with the presence of cross traffic that has a constant bit rate.

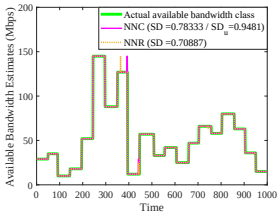
Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9996	0.9996	0.9991	0.9990	0.9998
Precision_{μ}	0.9960	0.9960	0.9910	0.9900	0.9980
Recall_{μ}	0.9960	0.9960	0.9910	0.9900	0.9980
F1_{μ}	0.9960	0.9960	0.9910	0.9900	0.9980
Precision_{M}	0.9964	0.9961	0.9926	0.9916	0.9981
Recall_{M}	0.9962	0.9956	0.9898	0.9888	0.9979
F1_{M}	0.9963	0.9958	0.9912	0.9902	0.9980



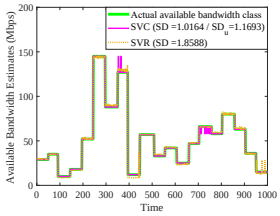
(a) NN and direct probing comparison



(b) Bandwidth estimate after median filter



(c) NN-based classification and regression comparison

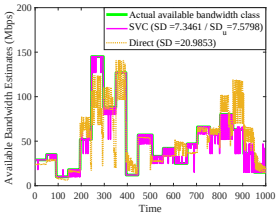


(d) SVM-based classification and regression comparison

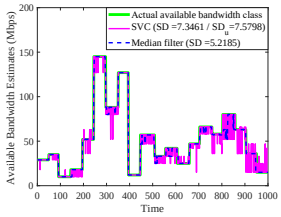
Figure 7: Available bandwidth estimates and their SDs in the presence of cross traffic that has a constant bit rate using $p=6$ packet trains in single tight link network.

Table 3: Testing performance in a single link with the presence of cross traffic that has Pareto-distributed packet inter-arrival times.

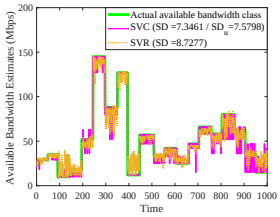
Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9746	0.9724	0.9734	0.9684	0.9735
Precision_{μ}	0.7460	0.7240	0.7340	0.6840	0.7350
Recall_{μ}	0.7460	0.7240	0.7340	0.6840	0.7350
F1_{μ}	0.7460	0.7240	0.7340	0.6840	0.7350
Precision_{M}	0.8018	0.7394	0.7528	0.7051	0.7576
Recall_{M}	0.7518	0.7238	0.7347	0.6856	0.7323
F1_{M}	0.7760	0.7315	0.7436	0.6952	0.7447



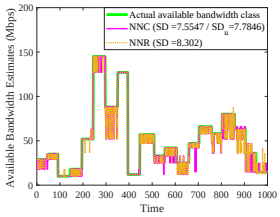
(a) SVM and direct probing comparison



(b) Bandwidth estimate after median filter



(c) SVM-based classification and regression comparison

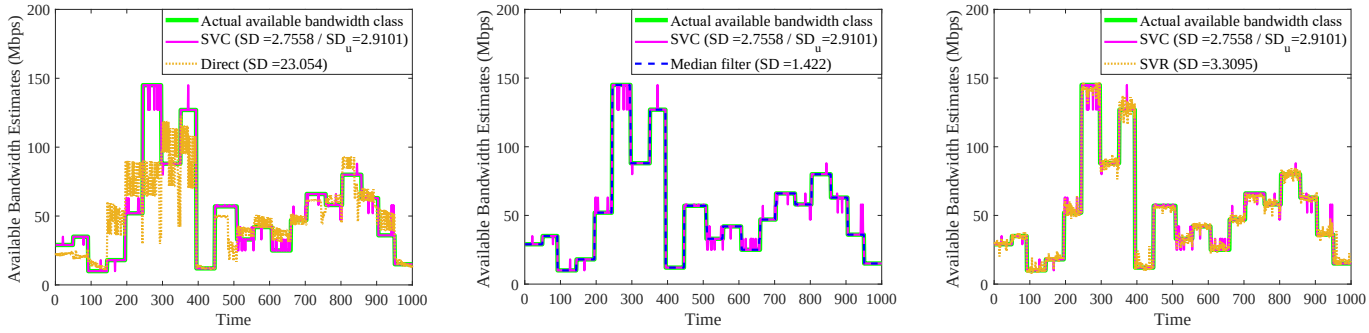


(d) NN-based classification and regression comparison

Figure 8: Available bandwidth estimates and their SDs in the presence of cross traffic that has Pareto-distributed packet inter-arrival times using $p=6$ packet trains in single tight link network.

Table 4: Testing performance in a single link with the presence of cross traffic that has exponentially distributed packet inter-arrival times when the number of packet trains is 5.

Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9935	0.9929	0.9934	0.9908	0.9935
Precision_{μ}	0.9350	0.9290	0.9340	0.9080	0.9350
Recall_{μ}	0.9350	0.9290	0.9340	0.9080	0.9350
F1_{μ}	0.9350	0.9290	0.9340	0.9080	0.9350
Precision_{M}	0.9354	0.9309	0.9372	0.9107	0.9364
Recall_{M}	0.9354	0.9290	0.9349	0.9077	0.9364
F1_{M}	0.9354	0.9299	0.9360	0.9092	0.9364

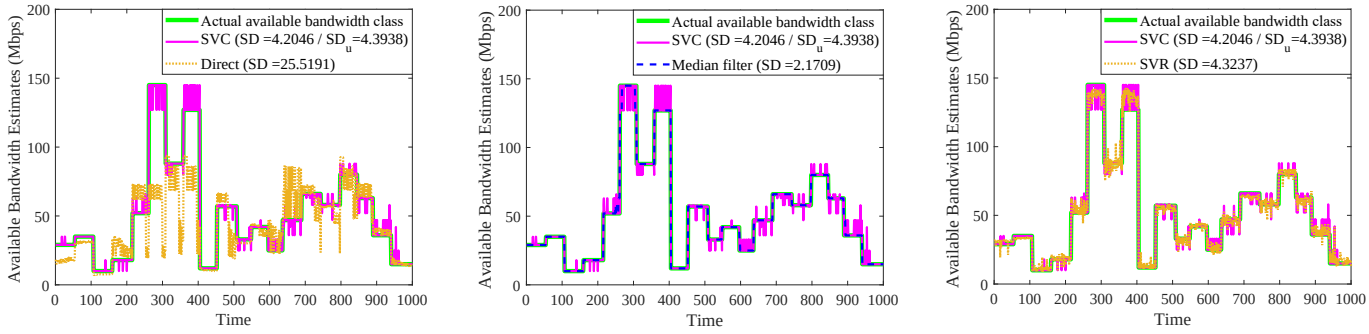


(a) SVM and direct probing comparison (b) Bandwidth estimate after median filter (c) SVC and SVR comparison

Figure 9: Available bandwidth estimates and their SDs in the presence of cross traffic that has exponentially distributed packet inter-arrival times using $p=5$ packet trains in a single tight link network.

Table 5: Testing performance in a single link with the presence of cross traffic that has exponentially distributed packet inter-arrival times when the number of packet trains is 4.

Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9863	0.9853	0.9857	0.9809	0.9853
Precision_{μ}	0.8630	0.8530	0.8570	0.8090	0.8530
Recall_{μ}	0.8630	0.8530	0.8570	0.8090	0.8530
F1_{μ}	0.8630	0.8530	0.8570	0.8090	0.8530
Precision_{M}	0.8611	0.8521	0.8581	0.8153	0.8604
Recall_{M}	0.8591	0.8498	0.8571	0.8118	0.8521
F1_{M}	0.8601	0.8509	0.8576	0.8135	0.8562

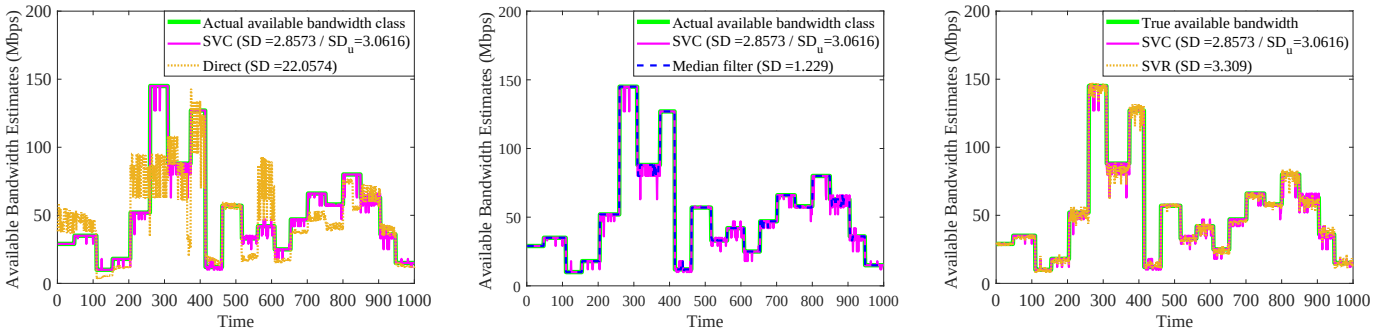


(a) SVM and direct probing comparison (b) Bandwidth estimate after median filter (c) SVC and SVR comparison

Figure 10: Available bandwidth estimates and their SDs in the presence of cross traffic that has exponentially distributed packet inter-arrival times using $p=4$ packet trains in a single tight link network.

Table 6: Testing performance in a 2-hop network with the presence of cross-traffic that has exponentially distributed packet inter-arrival times.

Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9814	0.9774	0.9811	0.9790	0.9813
Precision_{μ}	0.8140	0.7740	0.8110	0.7900	0.8130
Recall_{μ}	0.8140	0.7740	0.8110	0.7900	0.8130
F1_{μ}	0.8140	0.7740	0.8110	0.7900	0.8130
Precision_{M}	0.8265	0.8019	0.8256	0.8000	0.8356
Recall_{M}	0.8155	0.7802	0.8179	0.7895	0.8225
F1_{M}	0.8210	0.7909	0.8217	0.7947	0.8290

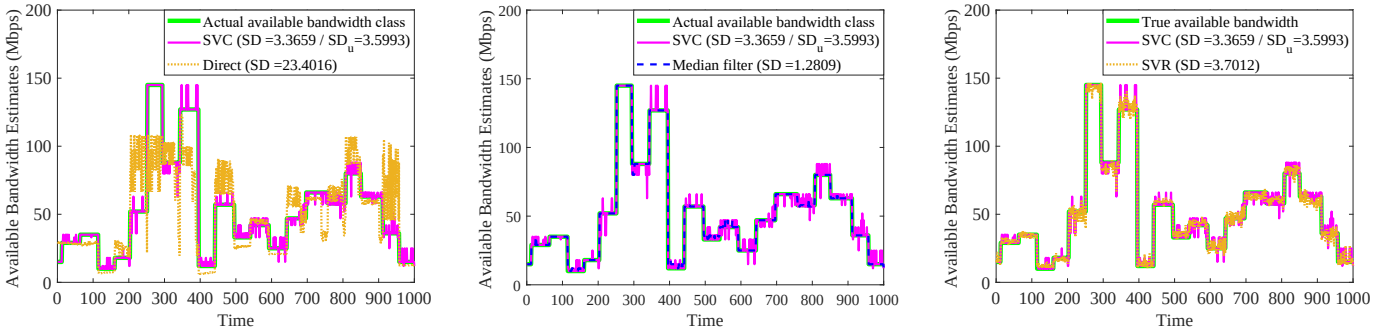


(a) SVM and direct probing comparison (b) Bandwidth estimate after median filter (c) SVC and SVR comparison

Figure 11: Available bandwidth estimates and their SDs in the presence of cross traffic that has exponentially distributed packet inter-arrival times using $p = 6$ packet trains in a 2-hop network.

Table 7: Testing performance in a 3-hop network with the presence of cross-traffic that has exponentially distributed packet inter-arrival times.

Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9749	0.9705	0.9727	0.9652	0.9727
Precision_{μ}	0.7490	0.7050	0.7270	0.6520	0.7270
Recall_{μ}	0.7490	0.7050	0.7270	0.6520	0.7270
F1_{μ}	0.7490	0.7050	0.7270	0.6520	0.7270
Precision_{M}	0.7571	0.7117	0.7401	0.6648	0.7339
Recall_{M}	0.7470	0.7040	0.7264	0.6598	0.7248
F1_{M}	0.7520	0.7078	0.7332	0.6623	0.7293

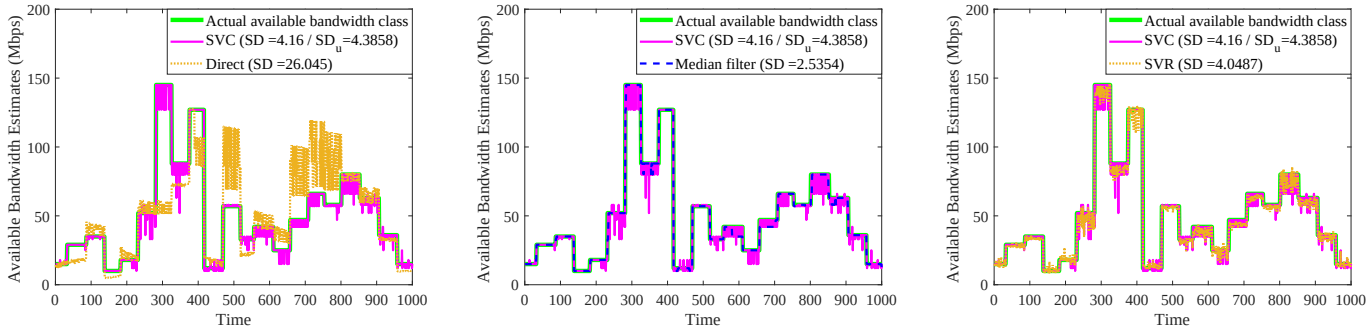


(a) SVM and direct probing comparison (b) Bandwidth estimate after median filter (c) SVC and SVR comparison

Figure 12: Available bandwidth estimates and their SDs in the presence of cross traffic that has exponentially distributed packet inter-arrival times using $p = 6$ packet trains in a 3-hop network.

Table 8: Testing performance in a 4-hop network with the presence of cross-traffic that has exponentially distributed packet inter-arrival times.

Parameters	Algorithms				
	SVM	k-NN	Bagging	AdaBoost	NN
Acc_{avg}	0.9735	0.9700	0.9701	0.9613	0.9704
Precision_{μ}	0.7350	0.7000	0.7010	0.6130	0.7040
Recall_{μ}	0.7350	0.7000	0.7010	0.6130	0.7040
F1_{μ}	0.7350	0.7000	0.7010	0.6130	0.7040
Precision_{M}	0.7639	0.7323	0.7371	0.6457	0.7346
Recall_{M}	0.7368	0.7001	0.7112	0.6085	0.7086
F1_{M}	0.7501	0.7158	0.7239	0.6265	0.7214



(a) SVM and direct probing comparison (b) Bandwidth estimate after median filter (c) SVC and SVR comparison

Figure 13: Available bandwidth estimates and their SDs in the presence of cross traffic that has exponentially distributed packet inter-arrival times using $p = 6$ packet trains in a 4-hop network.

the classification has less SD, and the noise in the estimates increases with the increasing number of links.

5. Conclusion

We have investigated the use of machine learning tools in available bandwidth estimation after formulating it as a multiclass classification problem. Defining a $p-$dimensional feature vector, instances of which are functions of the input and output rates of a defined probe traffic, we have employed five different machine learning techniques, namely, SVM, k-NN, Bagging, AdaBoost and NN. We have conducted a comprehensive experimental study in a controlled network testbed and induced network models ranging from deterministic scenarios with constant-rate cross traffic to stochastic ones with bursty nature. We have shown that using machine learning techniques, we can improve available bandwidth estimates significantly with six packet trains and reduce the overhead in the network caused by active probing. This holds true in networks with randomly changing capacity and cross traffic intensity. We have further investigated the impact of reducing the probe traffic on the estimation performance. We have found out that although there is a trade-off between the amount of injected probe traffic and the error rate of an estimator, we are able to obtain significantly good results even by using four packet trains. Furthermore, by applying a median filter to estimation results, we have been able to track the changes in the available bandwidth. The classification-based estimation outperforms the direct probing technique that employs a Kalman filter. Finally, we note that the classification-based technique performs very close to its regression-based counterparts, even results in better SD values around the available bandwidth.

References

[1] X. Liu, K. Ravindran, D. Loguinov, A queueing-theoretic foundation of available bandwidth estimation: single-hop analysis, IEEE/ACM Transactions on Networking 15 (4) (2007) 918–931.

[2] M. Zangrilli, B. B. Lowekamp, Applying principles of active available bandwidth algorithms to passive tcp traces, in: International Workshop on Passive and Active Network Measurement, Springer, 2005, pp. 333–336.

[3] M. Jain, C. Dovrolis, Pathload: A measurement tool for end-to-end available bandwidth, in: Passive and Active Measurement Workshop, 2002.

[4] C. L. T. Man, G. Hasegawa, M. Murata, A merged inline measurement method for capacity and available bandwidth, in: International Workshop on Passive and Active Network Measurement, Springer, 2005, pp. 341–344.

[5] B. Melander, M. Bjorkman, P. Gunningberg, A new end-to-end probing and analysis method for estimating bandwidth bottlenecks, in: IEEE Globecom, 2000, pp. 415–420.

[6] A. Johnsson, B. Melander, M. Björkman, Diettopp: A first implementation and evaluation of a simplified bandwidth measurement method, in: Swedish National Computer Networking Workshop, 2004.

[7] C. Dovrolis, P. Ramanathan, D. Moore, What do packet dispersion techniques measure?, in: IEEE INFOCOM, 2001, pp. 905–914.

[8] M. Jain, C. Dovrolis, End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput, IEEE/ACM Transactions on Networking 11 (4) (2003) 537–549.

[9] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, L. Cottrell, pathChirp: Efficient available bandwidth estimation for network paths, in: Passive and Active Measurement Workshop, 2003.

[10] J. Strauss, D. Katabi, F. Kaashoek, A measurement study of available bandwidth estimation tools, in: ACM Internet Measurement Conference, 2003, pp. 39–44.
[11] N. Hu, P. Steenkiste, Evaluation and characterization of available bandwidth probing techniques, IEEE Journal on Selected Areas in Communications 21 (6) (2003) 879–894.
[12] S. Ekelin, M. Nilsson, E. Hartikainen, A. Johnsson, J.-E. Mangs, B. Melander, M. Bjorkman, Real-time measurement of end-to-end available bandwidth using Kalman filtering, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), 2006, pp. 73–84.
[13] X. Liu, K. Ravindran, D. Loguinov, A stochastic foundation of available bandwidth estimation: Multi-hop analysis, IEEE/ACM Transaction on Networking 16 (1) (2008) 130–143.
[14] J. Liebeherr, M. Fidler, S. Valaee, A system theoretic approach to bandwidth estimation, IEEE/ACM Transactions on Networking 18 (4) (2010) 1040–1053.
[15] R. Lübben, M. Fidler, J. Liebeherr, Stochastic bandwidth estimation in networks with random service, IEEE/ACM Transactions on Networking 22 (2) (2014) 484–497.
[16] V. Paxson, End-to-end internet packet dynamics, IEEE/ACM Transactions on Networking 7 (3) (1999) 277–292.
[17] S. Keshav, A control-theoretic approach to flow control, in: Proc. ACM SIGCOMM, 1991, pp. 3–15.
[18] A. Shriram, J. Kaur, Empirical evaluation of techniques for measuring available bandwidth, in: IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications, IEEE, 2007, pp. 2162–2170.
[19] Z. Bozakov, M. Bredel, Online estimation of available bandwidth and fair share using Kalman filtering, in: IFIP Networking, 2009.
[20] A. Eswaradass, X.-H. Sun, M. Wu, A neural network based predictive mechanism for available bandwidth, in: Parallel and Distributed Processing Symposium, 2005.
[21] L.-J. Chen, A machine learning-based approach for estimating available bandwidth, in: TENCON, 2007, pp. 1–4.
[22] Q. Yin, J. Kaur, Can machine learning benefit bandwidth estimation at ultra-high speeds?, in: Passive and Active Measurement Conference, 2016, pp. 397–411.
[23] N. Sato, T. Oshiba, K. Nogami, A. Sawabe, K. Satoda, Experimental comparison of machine learning-based available bandwidth estimation methods over operational LTE networks, in: IEEE Symposium on Computers and Communications (ISCC), 2017, pp. 339–346.
[24] S. K. Khangura, M. Fidler, B. Rosenhahn, Neural networks for measurement-based bandwidth estimation, IFIP Networking Conference.
[25] S. K. Khangura, M. Fidler, B. Rosenhahn, Machine learning for measurement-based bandwidth estimation, Computer Communications 144 (2019) 18 – 30.
[26] S. K. Khangura, S. Akın, Measurement-based online available bandwidth estimation employing reinforcement learning, ITC 31- Networked Systems and Services.
[27] P. Hága, S. Laki, F. Tóth, I. Csabai, J. Stéger, G. Vattay, Neural network based available bandwidth estimation in the ETOMIC infrastructure., in: TRIDENTCOM, 2007, pp. 1–10.
[28] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, MIT press, 2018.
[29] C. Cortes, V. Vapnik, Support-vector networks, Machine learning 20 (3) (1995) 273–297.
[30] T. G. Dietterich, G. Bakiri, Solving multiclass learning problems via error-correcting output codes, Journal of artificial intelligence research 2 (1994) 263–286.
[31] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Information processing & management 45 (4) (2009) 427–437.
[32] D. S. Anderson, M. Hibler, L. Stoller, T. Stack, J. Lepreau, Automatic online validation of network configuration in the emulab network testbed, in: IEEE International Conference on Autonomic Computing, 2006, pp. 134–142.
[33] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, G. Carle, MoonGen: A Scriptable High-Speed Packet Generator, in: ACM Internet Measurement Conference, 2015.
[34] S. Avallone, S. Guadagno, D. Emma, A. Pescape, G. Ventre, D-ITG distributed internet traffic generator, in: Quantitative Evaluation of Systems, 2004, pp. 316–317.
[35] J. Laine, S. Saaristo, R. Prior, Real-time udp data emitter (rude) and collector for rude (crude) (2000).
URL `https://sourceforge.net/projects/rude/`