Can Machine Learning Benefit Bandwidth Estimation at Ultra-high Speeds?[*]

Qianwen Yin Jasleen Kaur

University of North Carolina at Chapel Hill

Abstract. Tools for estimating end-to-end available bandwidth (AB) send out a train of packets and observe how inter-packet gaps change over a given network path. In ultra-high speed networks, the fine inter-packet gaps are fairly susceptible to noise introduced by transient queuing and bursty cross-traffic. Past work uses smoothing heuristics to alleviate the impact of noise, but at the cost of requiring large packet trains. In this paper, we consider a machine-learning approach for learning the AB from noisy inter-packet gaps. We conduct extensive experimental evaluations on a 10 Gbps testbed, and find that supervised learning can help realize ultra-high speed bandwidth estimation with more accuracy and smaller packet trains than the state of the art. Further, we find that when training is based on: (i) more bursty cross-traffic, (ii) extreme configurations of interrupt coalescence, a machine learning framework is fairly robust to the cross-traffic, NIC platform, and configuration of NIC parameters.

1 Introduction

End-to-end available bandwidth (AB) is important in many application domains including server selection[1], video-streaming[2], and congestion control[3]. Consequently, the last decade has witnessed a rapid growth in the design of AB estimation techniques[4–6]. Unfortunately, these techniques do not scale well to upcoming ultra-high speed networks [7][1]. This is because small inter-packet gaps are needed for probing higher bandwidth —such fine-scale gaps are fairly susceptible to being distorted by noise introduced by small-scale buffering.

Several approaches have been proposed to reduce the impact of noise [8–10], most of which apply smoothing techniques to "average-out" distortions. Due to the complex noise signatures that can occur at fine timescales, these techniques need to average out inter-packet gaps over a large number of probing packets—this impacts the overhead and timeliness of these techniques.

In this paper, we ask: can supervised machine learning be used to automatically learn suitable models for mapping noise-afflicted packet gaps to AB estimates? We design a learning framework in which the sender and receiver side inter-packet gaps are used as input features, and an AB estimate is the output. Extensive evaluations are conducted, and find that a machine learning framework can indeed be trained to provide robust bandwidth estimates, with much higher accuracy and using much smaller number of probing packets than the state of the art.

In the rest of this paper, we describe the challenges of AB estimation at ultra high-speed, and the state-of-art in Section 2. We introduce our machine learning framework in Section 3, and our data collection methodology in Section 4. In ~~Section 5, we~~ experimentally evaluate our approach, and conclude in Section 6.

[*] This material is based upon work supported by the National Science Foundation under Grant No. CNS-1526268, OCI-1127413.

[1] We focus on 10Gbps speed in this paper, and use jumbo frames of MTU=9000B.

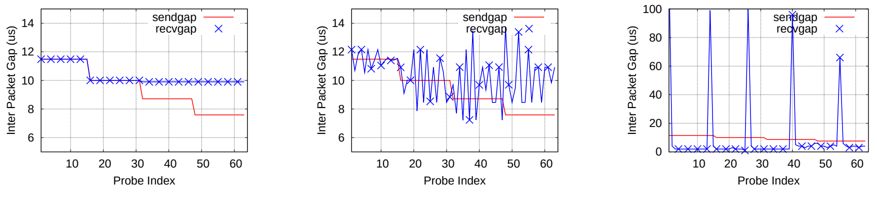
2 State of the Art

2.1 Background: Available Bandwidth Estimation

Main-stream bandwidth estimation tools adopt the *probing rate model*[11], which sends out streams of probe packets (referred to as **pstreams**) at a desired probing rate, by controlling the inter-packet send gaps as: $g_i^s = \frac{p_i}{r_i}$, where g_i^s is the send gap between the ith and *i-1*th packets, r_i is the intended probing rate, and p_i is the size of ith packet. The estimation logic is based on the principle of *self-induced congestion*— if $r_i > AB$, then $q_i > q_{i-1}$, where q_i is the queuing delay experienced by the ith packet at the bottleneck link, and AB is the bottleneck available bandwidth. Assuming fixed routes and constant processing delays, this translates to $g_i^r > g_i^s$, where g_i^r is the receive gap between the ith and *i-1*th packets. Most tools send out multiple packets (N_p) at each probing rate, and check whether or not the receive gaps are consistently higher than the send gaps. They try out several probing rates and search for the highest rate r_{max} that does *not* cause self-induced congestion. There are two dominant strategies for searching for r_{max}:

Feedback-based Single-rate Probing: The sender relies on iterative feedback-based binary search. The sender sends all packets within a pstream at the same probing rate, and waits for receiver feedback on whether the receive gaps increased or not. It then either halves or doubles the probing rate for the next stream accordingly. Pathload is the most prominent of such tools [4].

Multi-rate Probing: The sender uses **multi-rate** probing without relying on receiver feedback—each pstream includes $N = N_r \times N_p$ packets, where N_r is the number of probing rates tried out. The sender then looks for the highest probing rate that did not result in self-congestion. Fig 1(a) illustrates a multi-rate pstream with $N_r = 4, N_p = 16$. The receive gaps are consistently larger than the send gaps since the third probing rate, so the second probing rate (r_{max}) is taken as an estimate of the AB. Multi-rate probing facilitates the design of light-weight and quick tools [7]. Pathchirp is the most prominent of such tools [5].



(a) Noise-Free (b) After Bottleneck (c) At Receiver

Fig. 1: Inter-Packet Gaps $N_r = 4, N_p = 16$

2.2 Challenge: Noise in Ultra High Speed Networks

End-to-end bandwidth estimation tools face three major challenges at ultra high-speed: accurately creating fine-scale inter-packet gaps at the sender, dealing with the presence of noise along the path, and precisely timestamping packet arrival at the receiver. [2] To address the first challenge, we use the framework described in [10], in which approporiate-sized IEEE 802.3x PAUSE frames — "dummy" frames that get dropped by the first switch on the path, are inserted for creating fine-scale inter-packet gaps. We focus on the remaining two allenges in this paper.

[2] The first and thrid can be well addressed with specialized NICs [12], or with recent advances in fast packet I/O frameworks such as netmap [13]. In this study, however, we focus on end systems with standard OSes and commodify network hardwares.

Any resource that is shared can be temporarily unavailable, even if it is not a bottleneck resource over larger timescales—a packet may have to wait in a transient queue at such a resource. In ultra-high speed networks, the magnitude of distortions created by queuing-induced noise are comparable to (or even larger than) the changes in inter-packet gaps that need to be detected for bandwidth estimation. [10] identifies two main noise sources:

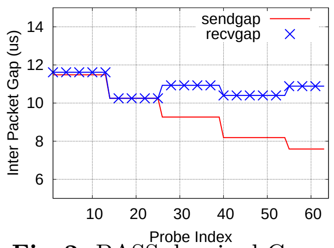


Fig. 2: BASS-denoised Gaps

Bursty cross-traffic at bottleneck resources. If the cross-traffic that shares a bottleneck queue varies significantly at short timescales, then all packets sent at a given probing rate may not consistently show an increase in receive gaps. For instance, Fig 1(b) plots the inter-packet gaps observed right after the bottleneck queue, for the same pstream as in Fig 1(a). Due to the bursty cross-traffic, the receive gaps are consistently larger than the send gaps only for the 4th probing rate (resulting in an over-estimation of AB).

Transient queuing at non-bottleneck resources. Even though a resource may not be a network bottleneck, it can certainly induce short-scale transient queues when it is temporarily unavailable while serving competing processes or traffic. *Interrupt Coalesence* is a notable source of such noise [14, 8]. It is turned on by default at receivers, forcing packets to wait at the NIC before being handed to OS for timestamping, even if the CPU is available—the waiting time (a.k.a interrupt delay) can be significant compared to the fine-scale gaps needed in ultra high-speed networks. Fig 1(c) plots the inter-packet gaps observed at the receiver (g_i^r) for the pstream in Fig 1(a). We find that these gaps are dominated by a **"spike-dips"** pattern—each spike corresponds to the first packet that arrives after an interrupt and is queued up till the next interrupt (thus experiencing the longest queuing delay). The dips correspond to the following packets buffered in the same batch. With the "spike-dips" pattern, an consistently increasing trend of queuing delays will not be observed in any pstream, leading to persistent over-estimation of AB.

2.3 State of the Art: Smoothing Out Noise

Several approaches have been proposed to deal with the impact of noise on bandwidth estimation [4, 8–10]. In general, all of these approaches employ *denoising techniques* for smoothing out inter-packet receive gaps, before feeding them to the bandwidth estimation logic. The recently-proposed Buffering-aware Spike Smoothing (BASS) [10] has been shown to outperform the others on 10 Gbps networks with shorter streams, and is summarized below.

BASS works by detecting boundaries of "buffering events" in recvgaps— each "spike" and the following dips correspond to packets within the *same* buffering event. Based on the observation that the average receiving rate within a buffering event is the same as that observed before the buffering was encountered, BASS recovers this quantity by carefully identifying buffering events and smoothing out both sendgaps and recvgaps within each. The smoothed gaps are then fed into an AB estimation logic. Fig 1(c) plots the BASS-smoothed gaps for the pstream in Fig 2. In [10], BASS was used within both single-rate and multi-rate probing frameworks. For single-rate probing, BASS helped achieve bandwidth estimation accuracy within 10%, by using pstreams with at least 64 packets. For multi-rate probing, BASS-smoothed gaps were fed to a variant of the Pathchirp bandwidth estimation logic, and estimation accuracy of mostly within +/-10% was achieved using multi-rate pstreams with N=96 packets and 50% probing range[3].

For many applications of bandwidth estimation, that need to probe for bandwidth regularly and frequently, large probe streams pose a significant issue in terms of timeliness, overhead and responsiveness— both the duration for which each pstream overloads the network, and the total time needed to collect AB estimates, increase linearly with N (when N_r is fixed). Even a 96-packet pstream can last several milliseconds in a gigabit network—such a duration is too long in the context of ultra-high speed congestion control [3].

3 A Learning Framework for Bandwidth Estimation

It is important to note that noise can distort gaps within a pstream with several different signatures, each with its own magnitude of gap-distortion, and each with its own timescale and frequency at which it manifests itself (as exemplified in Figs 1(b) and 1(c)). When simple smoothing heuristics are used by the state of the art for dealing with such diversity in noise, they result in an *underfit* model—expectedly, these techniques need to smooth over a *large* number of probe packets in order to be robust. The main hypothesis of this work is that machine learning (ML) can improve our understanding of the noise signature in gaps, with even shorter probe streams than the state of the art.

In this paper, we propose to use supervised learning to automatically derive an algorithm that estimates AB from the inter-packet send and receive gaps of each pstream. Such an algorithm is referred to as a learned "model". We envision that the model is learned *offline*, and then can be incorporated in other AB estimation processes. Below, we briefly summarize the key components of this framework.

Input Feature Vector The input feature vector for a pstream is constructed from the set of send gaps and receive gaps, $\{g_i^s\}$ and $\{g_i^r\}$. Fourier transforms are commonly used in ML applications, when the input may contain information at multiple frequencies [15, 16]—as discussed before, this certainly holds for the different sources of noise on a network path. Hence, we use as a feature vector, the fourier-transformed sequence of send and receive gaps for a pstream of length N: $x = FFT(g_1^s, ..., g_N^s, g_1^r, ..., g_N^r)$.

Output The output, y, of the ML framework is the AB evaluation. For *single-rate* pstreams, the AB estimation can be formulated as a classification problem: $y = 1$ if the probing rate exceeds AB, otherwise $y = 0$. For *multi-rate* pstreams, it can be formulated as a regression problem, in which $y = AB$.

Learning Techniques We consider the following ML algorithms—ElasticNet [17], which assumes a polynomial relationship between x and y; RandomForest [18], AdaBoost [19] and GradientBoost [20], which ensemble multiple weak models into a single stronger one; Support Vector Machine(SVM) [21], which maps x into a high dimensional feature space and constructs hyperplanes separating y values in the training set.[4]

[3] Probing range is given by: $\frac{r_N}{r_1} - 1$.

[4] Our evaluations revealed that models trained with ElasticNet and SVM result in considerable inaccuracy. For brevity, we don't present their results.

Training-and-testing The success of any ML framework depends heavily on good data collection—data that is accurate as well as representative. Section 4 describes our methodology for generating hundreds of thousands of pstreams under a diverse set of conditions—it also describes how we collect the ground-truth of AB, AB_{gt}, for each pstream. The knowledge of AB_{gt} allows us to compute an expected value, y_{exp}, of the output of the ML framework—both for single-rate as well as multi-rate pstreams.

We use data from the above pstreams to "train" each of the learning techniques, and then "test" them on pstreams not included in the training set. In each experiment in Section 5, we generate more than 20000 pstreams, of which 10000 are used for training and the remaining for testing.[5]

Metrics Each "test" that is run on a pstream, yields an estimate of the output, y. For single-rate pstream, the accuracy of the model is quantified by the *decision error rate*, which is the percentage of pstreams, for which: $y \neq y_{exp}$. For multi-rate pstream, we quantify *relative estimation error* as: $e = \frac{y - AB_{gt}}{AB_{gt}}$.

4 Data Collection

The success of a ML framework depends on its ability to work with a diverse and representative set on input data. We use a carefully-designed experimental methodology for obtaining such data. A salient feature of our methodology is that all evaluations are performed on a 10 Gbps testbed.

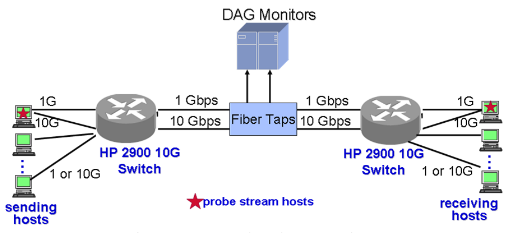


Fig. 3: Testbed Topology

Testbed We use the dedicated network illustrated in Fig 3 in this study. The switch-to-switch path is a 10Gbps fiber path. The two end hosts involved in bandwidth estimation are connected to either side of the switches using 10Gbps Ethernet. The testbed includes an additional 10 pairs of hosts, each equipped with a 1Gbps NIC, that are used to generate cross traffic sharing the switch-to-switch link. For each experiment, we collect packet traces on the switch-to-switch link using fiber splitters attached to an Endace DAG monitoring NIC which provides timestamps at 10 nanosecond accuracy.

Pstream Generation We use the Linux kernel modules implemented in [10] for sending and receiving pstreams. An *iperf* client is first used to generate data segments with an MTU size of 9000 bytes. A sender-side Linux Qdisc scheduler then turns the stream of these data segments into pstreams of a specified size and average probing rate. Inter-packet sendgaps are enforced by inserting appropriately-sized Ethernet PAUSE frames sent at link speed. [10] shows that these modules ensure gap accuracy within 1μs, even when probing at 10Gbps. At the receiver, packet arrival timestamps are recorded in an ingress Qdisc with microsecond precision. In each experiment summarized in Section 5, more than 20000 pstreams are generated, with their average probing rate ranging from 5 Gbps to 10 Gbps.

[5] In our Python implementation with scikit-learn [22] library, we use its automatic parameter *tuning* feature for all ML methods, and use 5-fold cross-validation to validate our results.

Calculating AB_{gt} The first and last packet from every pstream are located in the packet trace, the bytes of cross traffic between those two packets are counted and then cross traffic throughput is computed. AB_{gt}, the groundtruth of AB for that pstream is calculated by subtracting cross traffic throughput from the bottleneck capacity.

Cross Traffic Generation: Incorporating Diversity in Burstiness One major source of noise considered in this paper is fine-timescale burstiness in cross-traffic encountered at the bottleneck. In order to incorporate *diversity* in such burstiness in our data set, we generate serveral cross-traffic models.

BCT: We first ran a modified version of SURGE [23] program to produce bursty and synthetic web traffic between each pair of cross-traffic generators. An important consideration is that to study the impact of other factors, cross traffic should be consistently repeated across experiments. Thus, we record packet traces from each of the SURGE senders, and then *replay* these in all experiments on the same host using tcpreplay [24]. We denote the aggregate traffic of the replayed traces as "BCT". The average load of BCT is 2.4 Gbps.

SCT: We then generate a smoother version of BCT by running a token bucket Qdisc on each sending host. The resultant aggregate is referred to as **"SCT"**.

CBR: To obtain the least bursty cross-traffic (constant bit-rate, CBR) on the switch-to-switch link, we use iperf to create UDP flows between host pairs. We experiment with CBR traffic generated at 50 different rates, ranging from 1 Gbps to 5 Gbps.

UNC1-3: We also use three 5-minute traces collected at different times on a 1 Gbps egress link of the UNC campus network. For each trace, we run a corresponding experiment in our testbed, in which the trace is replayed concurrently by 10 cross-traffic senders (with random jitter in their start times). We label the resultant aggregate traffic aggregates as UNC1, UNC2, and UNC3, respectively. The average load of UNC1 is 3.10Gbps, UNC2 is 2.75Gbps, and UNC3 is 3.28Gbps.

Table 1: Cross Traffic Burstiness

label	Burstiness 5-95% Gbps
BCT	1.15 – 3.94
SCT	1.78 – 3.31
UDP	range $\sim$ 0.51
UNC1	2.23 – 4.05
UNC2	1.84 – 3.77
UNC3	2.31 – 4.29

Table 1 quantifies the burstiness of each of the above traffic aggregates, by listing the 5th and 95th percentile load offered by each on the bottleneck link. In most experiments reported in Section 5, we use BCT as the cross-traffic—Section 5.2 considers the others too.[6]

Incorporating Diversity in Interrupt Coalescence Section 5 describes how we also experiment with diversity in the other major source of noise—receiver-side interrupt coalescence. We rely on two different NIC platforms in this evaluation: *NIC1*, a PCI Express x8 Myricom 10 Gbps copper NIC with the myri10ge driver, and *NIC2*, an Intel 82599ES 10Gbps fiber NIC.

5 Evaluation

The two major sources of noise considered in this study are cross-traffic burstiness and receiver-side interrupt coalescence. In this section, we first present experiments conducted under conditions (BCT cross-traffic, and default configuration of interrupt coalescence on NIC1) similar to those used to evaluate BASS.

[6] Note that replayed traffic retains the burstiness of original traffic aggregate, but does not retain responsiveness of individual TCP flows. However, the focus of this paper is to evaluate denoising techniques for accurate AB estimation —this metric is not impacted by the responsiveness of cross traffic, but only by its burstiness.

Later, we explicitly control for, and consider the impact of cross-traffic burstiness and interrupt coalescence. '

5.1 Performance with BCT, and default interrupt coalescence

BASS has been shown to yield good bandwidth estimates on 10 Gbps networks, when used with single-rate pstreams of length $N = 64$, and multi-rate pstreams with $N = 96, N_r = 4$ [10]. In this section, we first evaluate our ML model under similar conditions, and then consider even *shorter* pstreams.

Single-Rate Probing: We first train models of different ML algorithms with $N = 64$, and test them on pstreams probing at 9 discrete rates, ranging from 5-9 Gbps (with BCT, the average AB is around 7.6 Gbps).

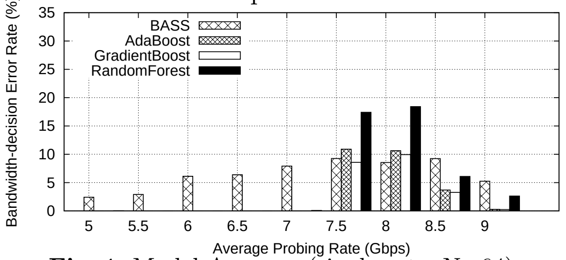


Fig. 4: Model Accuracy(single-rate, N=64)

The bandwidth-decision errors observed at each rate are plotted in Fig 4. We find that (unlike BASS) each of the three ensemble methods leads to negligible error when probing rate is far below or above avail-bw. When probing rates are close to the AB , both BASS and the ML models encounter more ambiguity. AdaBoost and GradientBoost perform comparable to BASS. RandomForest performs worse than the two boosting methods, which agrees with the findings in [25].[7] In the rest of the paper we focus our discussion on *GradientBoost*.

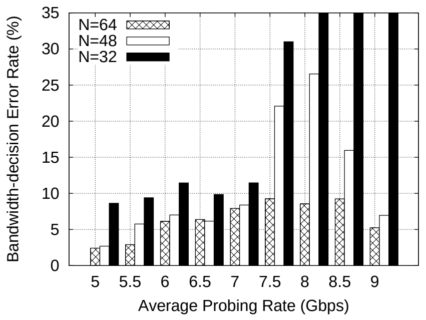


Fig. 5: BASS (single-rate)

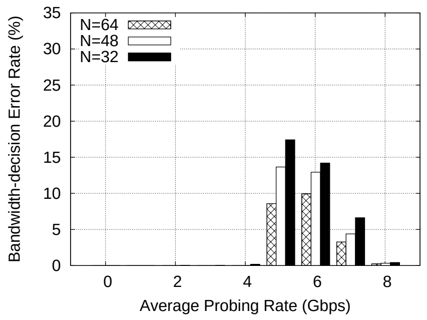


Fig. 6: GradientBoost (single-rate)

We then consider shorter pstreams by reducing N to 48 and 32, respectively, and compare the accuracy in Figs 5 and 6. We find that the performance of BASS degrades drastically with reduced N: for $N = 32$ error rate can exceed 50% when the probing rate is higher than 8Gbps! Although GradientBoost also yields more errors with smaller N, the error rate is limited to within 20% even with $N = 32$.

Multi-Rate Bandwidth Estimation We next train models with multi-rate pstreams of $N = 96, N_r = 4$ and probing range 50%. Fig 7 plots the distributions

[7] Each weak model in RandomForest is learned on a different subset of training data. The final prediction is the average result of all models. AdaBoost and GradientBoost follow a boosting approach, where each model is built to emphasize the training instances that previous models do not handle well. The boosting methods are known to be more robust than RandomForest [25], when the data has few outliers.

of relative estimation error using BASS and the learned GradientBoost model—ML significantly outperforms BASS by limiting error within 10% for over 95% pstreams! We further reduce N to 48 and 32, and find that $N = 48$ maintains similar accuracy as $N = 96$, while $N = 32$ leads to some over-estimation of bandwidth.

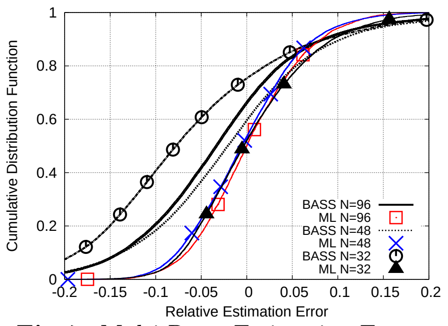


Fig. 7: Multi-Rate: Estimation Error

Based on our experiments so far, we conclude that *our ML framework is capable to estimating bandiwidth with higher accuracy and small pstreams than the state of the art, both with single-rate as well as multi-rate probing techniques*. In what follows, we focus on multi-rate probing with $N = 48$ and $N_r = 4$.

We next consider the impact of prominent sources of noise, namely, cross-traffic burstiness, and receiver-side interrupt coalescence. It is worth noting that the literature is lacking in controlling for and studying the following factors, each of which is a significant one for ultra-high-speed bandwidth estimation—this is a novelty of our evaluation approach.

5.2 Impact of Cross-traffic Burstiness

We repeat the experiments from Section 5.1, with BCT replaced by each of the other five models of cross-traffic. Fig 8 plots the results—the boxes plot the 10-90% range of the relative estimation error, and the extended bars plot the 5-95% ranges. The left two bars for each cross-traffic type compare the performance of BASS and our ML model. We find that the performance of both BASS and our ML model is relatively insensitive to the level of burstiness in cross-traffic. However, in each case, ML consistently outperforms BASS.

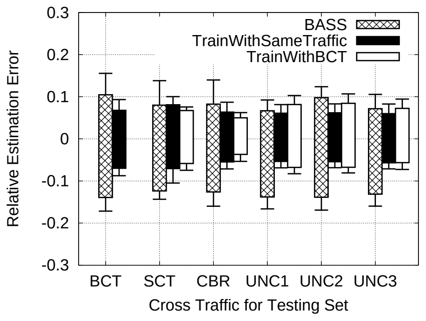


Fig. 8: Test with Same/Smoother Traffic

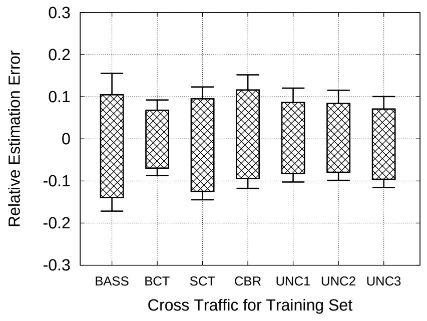


Fig. 9: Train with Smoother Traffic

In the above experiments, the ML model was trained and tested using pstreams that encounter the *same* type of cross-traffic model. In practice, it is not possible to always predict the cross-traffic burstiness on a given network path. We next ask the question: how does our ML framework perform when burstiness encountered in the training vs testing phases are different? Intuitively, a model learned from bursty cross-traffic is more likely to handle real-world cases where traffic is bursty; however, it is more subjective to overfitting — the model may try to "memorize" the noisy training data, leading to poor performance for conditions with smoother traffic.

Training With Smoother Traffic We next employ the models trained with each cross-traffic type to test pstreams that encounter the more bursty BCT in Fig 9. We find that, ML outperforms BASS in all cases; but models learned with smoother traffic lead to higher errors than the one learned with BCT. This is to be expected—bursty traffic introduces a higher degree of noise. We conclude that it is *preferable to train an ML model with highly bursty cross-traffic, to prepare it for traffic occuring in the wild.*

Testing With Smoother Traffic We use the model trained with BCT, to predict AB for pstreams that encounter other types of cross-traffic. In Fig 8, we find that the BCT-derived model gives comparable accuracy as the one trained with the same cross-traffic type as the testing set. Thus, *a model learned from more bursty cross traffic is robust* to testing cases where cross traffic is less bursty.

5.3 Impact of Interrupt Coalescence Parameter

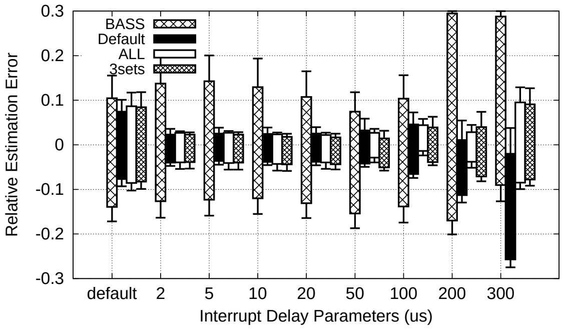


Fig. 10: Impact of ICparams in Training Set

Interrupt coalescence by a NIC platform is typically configured using two types of parameters (*ICparam*): "rx-usecs", the minimum time delay between iterrupts, and/or "rx-frames", the number of packets coalesced before NIC generates an interrupt. By default, NICs are configured to use some combination of both of these parameters—our experiments presented so far use the default configuration on NIC1, which roughly boils down to a typical interrupt delay of about 120μs.

Different *ICparam* lead to different "spike-dips" patterns in the receive gaps, in terms of the heights of the spikes, as well as the distances between neighboring spikes. We next study the impact of having different *ICparam* in the training vs testing data sets—a model learned with one parameter may fail on pstreams that encounter another. We first apply the previously learned ML model (with *ICparam=default*) to testing scenarios when rx-usecs is set to a specific value—ranging from 2μs to 300μs. Fig 10 compares the estimation accuracy of BASS and the ML model (left two bars in each group). The box plots the 10%-90% relative error, and the extended bar plots the 5%-95% error. We find that BASS severely over-estimates AB when interrupt delay is significant (rx-usecs $\geq$ 200μs), while the ML model yields better accuracy. This highlights the model of carefully studying the impact of *ICparam*—this factor was not considered in the BASS evaluations in [10]. We also find that the ML model consistently under-estimates AB when rx-usecs=300μs.

Machine learning performs best when the training set is representative of conditions encountered during testing. To achieve this, we create a training set that for each *ICparam*, include 5000 pstream samples that encounter it—we denote this as "ALL-set". As shown in Fig 10, the model learned from "ALL-set" reduces error to within 10% for most pstreams that encounter extreme rx-usecs values. In practice, however, *all* possible configurations of *ICparam* at a receiver NIC may not be known. We next ask: does there exist a model, which is trained with only a limited set of $ICparams$, but which manages to apply to all configurations? To study this, we minimize the training set to only include two extreme values (2us and 300us), in addition to the default setting. We refer to this set as "3sets". Fig 10 shows that "3sets" is sufficient to train an accurate ML model, which provides similar accuracy as "ALL-set".

5.4 Robustness and Portability across NICs

Different NIC platforms may interpret and implement interrupt coalescence differently. For instance, NIC-2 relies an adaptive interrupt behavior, even though it allows us to specify "rx-usecs" and "rx-frames". Fig 11 illustrates that on this NIC by plotting the distribution of number of frames coalesced per interrupt—we find that "rx-frames" takes no effect when rx-usecs $\leq 12\mu s$. But "rx-usecs" is not respected once it exceeds 12μs; the distribution mainly depends on rx-frames. This unpredictability is quite different from what we observed on NIC1—we next study if our ML framework will work on such a NIC as well.

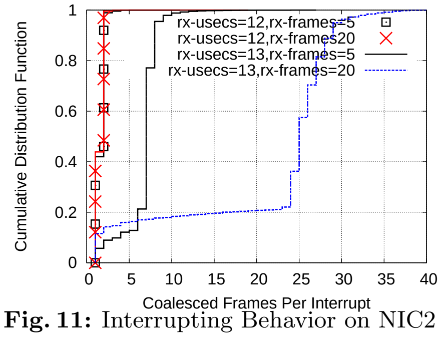


Fig. 11: Interrupting Behavior on NIC2

We repeat the experiments of Section 5.3, but use NIC2 instead of NIC1 for collecting both the training and testing data. We consider the following $ICparams$ for NIC-2: rx-usecs from 2 to 10μs, and rx-frames from 2 to 20 (rx-usecs=100). Models are learned from training sets consisting of different combinations of $ICparams$ in training scenarios, namely, the "Default", "All-set", and the "3sets"(default,rx-frames=2 and rx-frames=20). Fig 12 plots the estimation errors for these three environments. We find that compared with Fig 10, the estimation error is generally higher on NIC-2 than NIC-1, presumably due to greater unpredictability in its interrupting behavior. As before, the "3sets" on NIC-2 outperforms BASS significantly, and gives comparable accuracy as "All-set"—which agrees with our observation with NIC-1.

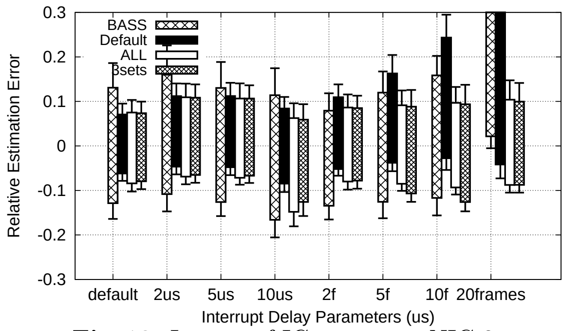
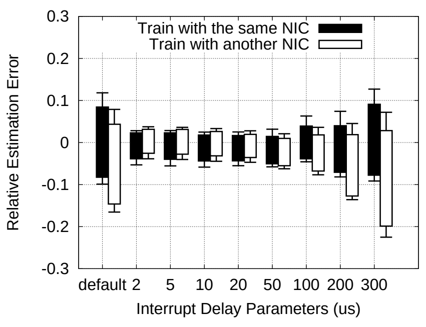


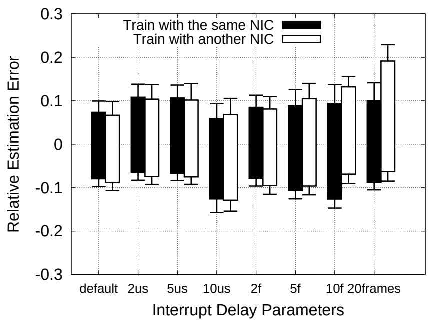
Fig. 12: Impact of ICparams on NIC-2

Cross-NIC Validation To investigate the portability of a learned model across NICs, we next perform a cross-NIC validation: the model trained with data collected using one NIC is tested on data collected on a different NIC. We use only *ICparam=3sets*, and plot the results in Figs 13(a) and 13(b). In general, we find that the cross-NIC models generally give comparable accuracy as models trained on the NIC itself. The notable exceptions occur for extreme values of $ICparam$— *rx-usecs=300μs* on NIC1 and *rx-frames=20* in NIC2.

5.5 Implementation Overhead



(a) NIC-1



(b) NIC-2

Fig. 13: Cross-NIC Evaluation

Table 2: Per-pstream Evaluation Overhead

single-rate N=48	BASS	Random Forest	AdaBoost	Gradient Boost
CPU Time(s)	1.7e-6	2.1e-6	1.0e-4	7.7e-6
Memory	-	3.8MB	3.3MB	248KB
multirate N=48	BASS	Random Forest	AdaBoost	Gradient Boost
CPU Time(s)	8.1e-6	2.5e-5	6.3e-5	7.2e-6
Memory	-	237MB	2.5MB	260KB

The benefits of our ML framework are achieved at the cost of system overhead in the testing phase[8]—the whole model has to be loaded into memory, resulting in more memory usage; also, the estimation time in testing phase increases with model complexity. Table 2 lists the memory and CPU cost incurred by different models trained with *ICparam=3sets*, for generating a single estimate. The memory usuage shown is the relative increment compared with BASS. We find that GradientBoost reports similar costs for both single-rate and multi-rate probing frameworks. For multi-rate probing, it takes comparable CPU usuage as BASS, and only 260KB more memory, which is negligible for modern end hosts with gigabits of RAM.

Although the above numbers are implementation-specific, it is important to understand the implementation complexity. In our evaluations, the offline-learned GradientBoost model consists of 100 base estimators, each with a decision tree with height less than 3— the memory cost of maintaining 100 small trees, as well as the time complexity in tree-search (upper-bounded by 300 comparisons), are both affordable in modern end-systems, in both user and kernel space. In practice, network operators can program the training process with any preferred ML library and store the learned model. The stored model contains parameters that fully define the model structure —thus, it can be easily ported to other development platforms. Even a Linux kernel module, such as the ones used in bandwidth-estimation based congestion-control [3], can load the model during module initialization, and can faithfully reconstruct the entire model in order to estimate AB.

6 Conclusion

In this paper we apply ML techniques to estimate bandwidth in ultra-high speed networks, and evaluate our approach in a 10Gbps testbed. We find that supervised learning helps to improve estimation accuracy for both single-rate and multi-rate probing frameworks, and enable shorter pstreams than the state of the art. Further experiments show that: (i) a model trained with more bursty cross traffic is robust to traffic burstiness; (ii) the ML approach is robust to

[8] Since models are trained off-line, the training overhead is not of concern.

interrupt coalescence parameters, if default and extreme configurations are included in training; and (iii) the ML framework is portable across different NIC platforms. In further work, we intend to conduct evaluations with more NICs from different vendors, and investigate the practical issues of generating training traffic in different networks.

References

1. Dykes et al. An empirical evaluation of client-side server selection algorithms. In *INFOCOM 2000*, 2000.
2. Aboobaker et al. Streaming media congestion control using bandwidth estimation. In *Management of Multimedia on the Internet*. 2002.
3. Konda and Kaur. Rapid: Shrinking the congestion-control timescale. In *INFOCOM 2009*. IEEE.
4. Jain and Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *PAM*, 2002.
5. V. Ribeiro et al. pathchirp: Efficient available bandwidth estimation for network paths. In *PAM*, volume 4, 2003.
6. A. Cabellos-Aparicio et al. A novel available bandwidth estimation and tracking algorithm. In *NOMS*. IEEE, 2008.
7. A. Shriram and J. Kaur. Empirical evaluation of techniques for measuring available bandwidth. In *INFOCOM*. IEEE, 2007.
8. S.R. Kang and D. Loguinov. Imr-pathload: Robust available bandwidth estimation under end-host interrupt delay. In *PAM*. 2008.
9. S.R. Kang and D. Loguinov. Characterizing tight-link bandwidth of multi-hop paths using probing response curves. In *IWQoS*. IEEE, 2010.
10. Q. Yin et al. Can bandwidth estimation tackle noise at ultra-high speeds? In *ICNP*. IEEE, 2014.
11. J. Strauss et al. A measurement study of available bandwidth estimation tools. In *the 3rd ACM SIGCOMM conference on Internet measurement*.
12. Ki-Suh K. Lee. Sonic: Precise realtime software access and control of wired networks. In *NSDI*, 2013.
13. Luigi Rizzo. netmap: A novel framework for fast packet i/o. In *USENIX Annual Technical Conference*, pages 101–112, 2012.
14. R. Prasad et al. Effects of interrupt coalescence on network measurements. *PAM*, 2004.
15. T.G. Dietterich. Machine-learning research. 1997.
16. T.T. Nguyen el al. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials*, 2008.
17. H. Zou et al. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2005.
18. A. Liaw et al. Classification and regression by randomforest. *R news*, 2002.
19. Y. Freund et al. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 1997.
20. J.H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 2001.
21. C. Cortes et al. Support-vector networks. *Machine learning*, 1995.
22. F. Pedrogosa et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 2011.
23. P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 1998.
24. Aaron A. Turner and M. Bing. Tcpreplay, 2005.
25. T. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 2000.