# A Machine Learning Approach to End-to-End RTT Estimation and its Application to TCP

Bruno A. A. Nunes*[1], Kerry Veenstra*, William Ballenthin†, Stephanie Lukin‡ and Katia Obraczka*
*Department of Computer Engineering, Baskin School of Engineering, University of California, Santa Cruz, CA, USA
†Department of Computer Science, Columbia University, New York, NY, USA
‡Department of Computer Science, Loyola University Maryland, Baltimore, MD, USA
Email: {bnunes, veenstra, katia}@soe.ucsc.edu, wrb2102@columbia.edu and smlukin@loyola.edu

***Abstract*— In this paper, we explore a novel approach to end-to-end round-trip time (RTT) estimation using a machine-learning technique known as the *Experts Framework*. In our proposal, each of several "experts" guesses a fixed value. The weighted average of these guesses estimates the RTT, with the weights updated after every RTT measurement based on the difference between the estimated and actual RTT.**

**Through extensive simulations we show that the proposed machine-learning algorithm adapts very quickly to changes in the RTT. Our results show a considerable reduction in the number of retransmitted packets and a increase in goodput, in particular on more heavily congested scenarios. We corroborate our results through "live" experiments using an implementation of the proposed algorithm in the Linux kernel. These experiments confirm the higher accuracy of the machine learning approach with more than $40\%$ improvement, not only over the standard TCP, but also over the well known Eifel RTT estimator.**

## I. Introduction

Latency is an important parameter when designing, managing, and evaluating computer networks, their protocols, and applications. One metric that is commonly used to capture network latency is the end-to-end round-trip time (RTT) which measures the time between data transmission and the receipt of a positive acknowledgement.

Several network applications and protocols use the RTT to estimate network load or congestion, and therefore need to measure it frequently. The Transmission Control Protocol, TCP, is the best known example. TCP bases its error- and congestion-control functions on the estimated RTT instead of relying on feedback from the network. TCP's estimate of the RTT employs a widely used technique: the Exponential Weighed Moving Average (EWMA).

In this paper, we propose a novel RTT estimation technique that uses a machine-learning based approach called the Experts Framework [1]. As described in detail in Section III, the Experts Framework uses "on-line" learning, where the learning process happens in "trials". At every trial, a number of "experts" contribute to an overall prediction, which is compared to the actual value (e.g., obtained by measurement). The algorithm uses the prediction error to refine the weights of each expert's contribution to the prediction; the updated weights are used in the next iteration of the algorithm. We contend that by employing our prediction technique, network applications and protocols that make use of the RTT do not have to measure it as frequently and can rely on our predictions.

As an example application for the proposed RTT estimation approach, we use it to predict the RTT used by TCP's error and congestion control. Through extensive simulations and live experiments, we show that our machine-learning approach can adapt to changes in the RTT faster and thus predict its value more accurately than the current EWMA technique employed by most versions of TCP. As described in Section II, TCP uses the RTT estimate to compute its Retransmission Time-Out (RTO) timer [2], which is one of the main timers involved in TCP's error and congestion control. When the RTO expires the TCP sender considers the corresponding packet to be lost and therefore retransmits it. TCP's RTO relies on RTT predictions and measurements in order to set its value properly. RTT can be defined as the time that elapses from a packet leaving the sender until the reception at the sender of a positive acknowledgment for that packet. If the RTO is too long, it can lead to long idle waits before the sender reacts to the presumably lost packet. On the other hand, if the RTO is set to be too aggressive (too short), it might expire too often leading to unnecessary retransmissions. Needless to say that setting the RTO is critical for TCP's performance.

We can split this problem into two subproblems. First is how to predict the RTT of the next packet to be transmitted, and second, how the predicted RTT can be used to compute the RTO. In this paper, we focus on the first subproblem, i.e., the prediction of the RTT. We do so using a new approach based on machine learning. We then present results that show considerably more accurate RTT predictions when compared to TCP's original algorithm for RTT estimation, and we evaluate the impact of this increased accuracy on the network. Moreover, in order to evaluate the true impact of our machine learning solution on TCP, we also implemented a well cited work on the same field, the Eifel [3] retransmission timeout, on the Linux Kernel and compared the results of several file transfers using the machine learning solution, the standard TCP timer and the Eifel timer. The second subproblem, i.e., to set the RTO value in order to take advantage of the more accurate RTT predictions, is the focus of future work.

The remainder of this paper is organized as follows. Section II presents related work, including a brief overview of TCP's original RTT estimation technique. In Section III, we describe our RTT prediction algorithm. Section IV and Section V present our evaluation methodology and results for simulation evaluation and real Linux implementations, respectively. Finally, Section VI concludes the paper and highlights directions for future work.

## II. Related Work

TCP uses a timeout/retransmission mechanism to recover from lost segments. While this timeout value needs to be greater than the current RTT to avoid unnecessary retrans-

[1]Financial support was granted by the CAPES Foundation Ministry of Education of Brazil, Caixa Postal 250, Brasilia - DF 70040-020 Brazil.

missions, it should not be so great that it results in network underutilization from protracted responses to losses and congestion. In Jacobson's well known work [2], two state variables EstimatedRTT and RTTVAR keep the estimate of the next RTT measurement SampleRTT and the RTT variation, respectively. RTTVAR is defined as an estimate of how much EstimatedRTT typically deviates from SampleRTT. EstimatedRTT is updated according to an exponential weighted moving average (EWMA) given by Equation 1, where $\alpha = \frac{1}{8}$. RTTVAR is calculated using Equation 2 which is also a EWMA, this time of the difference between SampleRTT and EstimatedRTT with gain $\beta$ typically set to $\frac{1}{4}$. Equation 3 sets the new value for the RTO as a function of EstimatedRTT and RTTVAR, where $K$ is usually 4.

$$\text{EstimatedRTT} = (1-\alpha)\cdot\text{EstimatedRTT} + \alpha\cdot\text{SampleRTT} \quad (1)$$

$$\text{RTTVAR} = (1-\beta)\cdot\text{RTTVAR} + \beta\cdot|\text{SampleRTT} - \text{EstimatedRTT}| \quad (2)$$

$$\text{RTO} = \max(\text{EstimatedRTT} + K\cdot\text{RTTVAR},\ 2\cdot\text{ticks}) \quad (3)$$

In prior work, a number of approaches have been proposed to estimate TCP's RTT. Allman and Paxson use trace-driven simulations to evaluate different RTT estimation algorithms, finding that the performance of the estimators is dominated by their minimum values and uninfluenced by the RTT sample rate [4]. This last conclusion was challenged by Ludwig and Sklower who propose a new algorithm called Eifel for estimating the RTT and setting the RTO [3]. Ludwig and Sklower identify several problems with TCP's original RTT estimation algorithm [2], including the observation that a sudden decrease in RTT causes RTTVAR and consequently RTO to increase unexpectedly. Our approach mitigates this behavior, since the predictions can follow quite closely any abrupt changes in the RTT. In Section IV we present comparative results that show that our proposed machine-learning approach not only outperforms TCP's original RTT estimator, but also the Eifel algorithm. Eifel was used as an alternative for comparison other than Jacobson's algorithm, since it is one of the most cite work on RTT estimation.

Lou and Huang propose an adaptive TCP control algorithm that adjusts the RTT estimation based on the ratio of previous and current bandwidth [5]. Ma et al. describe a TCP retransmission timeout algorithm that is based on recursive weighted median filtering [6]. Their simulation results show that, for Internet traffic with heavy tailed statistics, their method yields tighter RTT bounds than TCP's original RTT computation algorithm. Leung et al. present work that focuses on changing RTO computation and retransmission polices, rather than improving RTT predictions [7].

Machine learning has been used in a number of other applications. Helmbold et al. use an Experts Framework algorithm to predict hard-disk drive's idle time and decide when to attempt to save energy by spinning down the disk [8]. For this problem, the cost of making a bad decision (i.e., when spinning the disk down and back up costs more than simply leaving it on) is very well defined, since the decision of spinning down the disk does not affect the length of the next idle time.

Unlike the spin-down example, when predicting the RTT, every prediction causes the next RTT to be set to a different value, and that influences every event that happens thereafter. Thus, the problem of defining the cost of a bad RTT prediction is not as straightforward. Our solution to this problem is discussed in the Section III.

Moreover, in the spin-down cost problem, the traces used in the evaluation were "off-line" traces, i.e., traces captured from live runs and later on used only as input to the algorithms being evaluated. In the case of the spin-down problem it is OK to use off-line traces since the algorithm's estimations do not influence the outcome of the next measurement. In the case of TCP RTT estimation, however, as previously discussed, since the RTT estimations influence TCP timers and these timers affect the outcome of the next RTT measurement, off-line traces can be used to set and tune parameters of the algorithm, but they are not suitable for evaluating the performance of the system in the case of RTT predictions. The work presented in [9], [6], [10], [7] on RTT estimation compares their solution against TCP's original RTT estimation algorithm, but they base their evaluation on off-line traces.

Mirza et al. propose a throughput-estimation tool based on Support Vector Regression modeling [11]. It predicts throughput based on multiple real-value input features. To the best of our knowledge, to date no attempt to use *on-line learning algorithms* to predict network conditions has been reported.

## III. Proposed Approach

In this section we present the *Fixed-Share Experts* algorithm as a generic solution for on-line prediction. Later we describe its application to the problem of predicting the round-trip time of a TCP connection.

### A. The Fixed-Share Experts Algorithm

Our RTT prediction algorithm is based on the *Fixed-Share Experts Algorithm* [1] which uses "on-line learning" combining the predictions of a set of fixed *experts* denoted by $\{x_1, ..., x_N\}$.

In on-line learning, the learning process happens in trials. During every trial $t$ the algorithm receives the predictions from $N$ experts and uses them to output a master prediction $\hat{y}_t$. After trial $t$ is completed, the true outcome $y_t$ is known and the experts incur a loss. The loss, computed at trial $t$ for every expert $i$, is a function $L_{i,t}(x_i, y_t)$ used to update a set of *weights* to be used to compute the next prediction for trial $t+1$. This set of weights is denoted by $W_{t+1,i} = \{w_{t+1,1}, ..., w_{t+1,i}, ..., w_{t+1,N}\}$. The weight $w_{t,i}$ should be interpreted as a measurement of the confidence in the quality of the $i$-th expert's prediction at the start of trial $t$. In the initialization of the algorithm we make $w_{1,i} = \frac{1}{N}, \forall i \in \{1, ..., N\}$. The algorithm updates the expert's weight every trial after computation of the loss at trial $t$ by multiplying the weight of the $i$-th expert by $e^{-\eta L_{i,t}(x_i, y_t)}$. The *learning rate* $\eta$ is used to determine how fast the updates will take effect, dictating how rapidly the weights of misleading experts will be reduced. Table I summarizes the fixed-share experts algorithm.

After updating the weights, the algorithm also "shares" some of the weight of each expert among other experts. Thus, an expert who is performing poorly and had its weight severely compromised can quickly regain influence in the master prediction once it starts predicting well again. The amount of sharing can be changed by using the $\alpha$ parameter, called the *sharing rate*. This allows the algorithm to adapt to

burst behavior. Indeed, in Section IV, we use, among others, burst traffic scenarios to evaluate the performance of our algorithm.

In [1] the basic version of the Experts Framework is presented, proving bounds for different loss functions. The algorithm is also analyzed for different prediction functions, including the weighted averaging we use. The implementation described in this paper, with the intermediate pool variable, costs $O(1)$ time per expert per trial.

**Parameters:** $\eta > 0$ and $0 \leq \alpha \leq 1$
**Initialization:** $w_{1,1} = ... = w_{1,N} = \frac{1}{N}$
**1) Prediction:**
$\hat{y}_t = \frac{\sum_{i=1}^{N} w_{t,i} \cdot x_i}{\sum_{i=1}^{N} w_{t,i}}$
**2) Computing the Loss:** $\forall i : 1, ..., N$
$L_{i,t}(x_i, y_t) = \begin{cases} (x_i - y_t)^2 & , x_i \geq y_t \\ 2 \cdot y_t & , x_i < y_t \end{cases}$
**3) Exponential updates:** $\forall i : 1, ..., N$
$w'_{t,i} = w_{t,i} \cdot e^{-\eta L_{i,t}(y_t, x_i)}$
**4) Sharing weights:** $\forall i : 1, ..., N$
$\mathsf{pool} = \sum_{i=1}^{N} \alpha \cdot w'_{t,i}$
$w_{t+1,i} = (1 - \alpha) \cdot w'_{t,i} + \frac{1}{N} \cdot \mathsf{pool}$

TABLE I
THE FIXED-SHARE EXPERTS ALGORITHM.

### *B. Applying Experts to TCP's RTT Prediction*

To apply the presented algorithm to TCP's RTT-prediction problem, the experts predictions $x_i$ shown in Table I serve as predictions for the next RTT measured in ticks. $y_t$ is the RTT value at the present trial (equivalent to the SampleRTT), and $\hat{y}_t$ is the algorithm's prediction (equivalent to the RTT prediction or the EstimatedRTT mentioned in Section II).

We want the loss function $L_{i,t}(x_i, y_t)$ to reflect the real cost of making wrong predictions. In our implementation (see Table I), the loss function has different penalties for overshooting and undershooting, as they have different impacts on the system's behavior and performance. An underestimate of the RTT could contribute to an RTO computation that is less than the next measured RTT, causing unnecessary timeouts and retransmissions. We thus employ the following policies. If the measured RTT $y_t$ is higher than the expert's prediction $x_i$, but still close, then it means that this expert is contributing for a spurious timeout and should be penalized more than other experts that overshoot by a little. Big over-shooters are also severely penalized. The non-trivial issue here is identifying the appropriate cost for miss-predicting the RTT. In this case, the cost could be simply the difference between prediction and measurement, or maybe a factor thereof. It is still possible to improve the loss function by improving the definition of cost in our particular RTT prediction problem. Further investigating appropriate loss functions is the subject of future work.

Setting the value $x_i$ of the experts is referred to as setting the *experts spacing*. To space the experts is to determine the experts' values and their distribution within the prediction domain. When predicting RTTs, the experts should be spaced between $\mathsf{RTT_{min}}$ and $\mathsf{RTT_{max}}$, defined in some implementations (and on the network simulator used in this work) to be 1 and 128 ticks, respectively. Based on observations of several RTT datasets, we concluded that the the majority of the RTT measurements are concentrated in the lower part this interval. For that reason we found that exponentially spacing the experts, instead of uniformly (or linearly) spacing them, leads to better predictions. The exponential function used in our experiments is $x_i = \mathsf{RTT_{min}} + \mathsf{RTT_{max}} \cdot 2^{\frac{(i-N)}{4}}$. The $\frac{1}{4}$ multiplicative factor in the exponent of the spacing function was experimentally chosen to smooth out its growth. This increases the difference between the experts and generates diversity among them, which is good for the sake of increasing predictions granularity and accuracy.

The algorithm, as stated in Table I, will continually reduce the experts' weights towards zero. Thus, in order to avoid underflow issues in our implementation, we periodically rescale the weights.

## IV. SIMULATION RESULTS

In this section we discuss simulation results obtained from applying the experts framework and compare the results with Jacobson's performance. We evaluate the RTT prediction accuracy, as well as goodput, number of retransmitted packets and size of congestion window.

We evaluated the proposed RTT estimation approach using a variety of scenarios; in this section, we present results of three of these scenarios. These scenarios were chosen so that we could study the impact of mobility and network density in the RTT fluctuations and not only fluctuations due to traffic changes and congestion. In Scenario I we study a mobile ad-hoc network composed of 20 nodes. Scenario II is a static wireless network also composed of 20 nodes uniformly distributed over the simulated area. Like in Scenario I, Scenario III is also a mobile network composed of 20 nodes but with a traffic pattern different from Scenarios I, as described in the following sections. In this scenario we also vary mobility by changing the average speed of the mobile nodes.

Each point in every plot presented in this section is the mean value of the given metric for 24 simulations runs with a confidence level of 90%. For the simulations, the QUALNET [12] simulator was used. The simulation area in all experiments is $1500m$ X $1000m$, and the simulated time is 25 minutes. The routing protocol used was AODV [13], 802.11 [14] as the medium access protocol, and Random Way Point (RWP) mobility model was used for all mobile scenarios, with zero seconds of pause and velocity between [1,50] m/s. All nodes run File Transmission Protocol (FTP) applications to generate the TCP flows, and packet size is fixed to 512 bytes. The parameters are common for every simulation scenario unless stated otherwise.

*Impact of Parameters ($N, \eta, \alpha$)*

We experiment with several combinations of the Fixed-Share algorithm parameters. The number of experts $N$ affects the granularity over the range of values the RTT can assume. In our experiments, $N > 100$ had no major impact on the prediction accuracy. The learning rate $\eta$ is responsible for how fast the experts are penalized for a given loss. We want to avoid too low a value of $\eta$ since it increases the algorithm's convergence time; conversely, if $\eta$ is too large, it forces the expert's weights toward zero too quickly. If this second case occurs, then as weight rescaling kicks in, the algorithm assigns similar weights to all experts, making the algorithm's master prediction fluctuate undesirably around the mean value of the

experts' guesses. We chose a learning rate in the interval $1.7 < \eta < 2.5$ as it provides good prediction results for all the scenarios tested.

When sharing is not enabled ($\alpha = 0$) the outcome of the algorithm is given only by decreasing exponential updates, making it hard for the algorithm to follow abrupt changes in the RTT measurements: experts that experience a prolonged poor performance lack influence because their weights have become too depreciated. In this case, it would take more trials so that these experts start gaining greater importance in the master prediction. Enabling full sharing ($\alpha = 1$), similar weights are assigned to every expert, and the master prediction fluctuates close to a mean value among the experts guesses. Thus, all reported results use $N = 100$, $\eta = 2$ and $\alpha = 0.08$.

### A. Scenario I - MANET

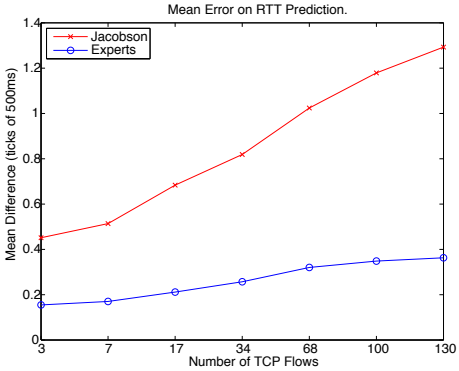


Fig. 1. Mean absolute difference between the predictions and the measured RTT for both algorithms on a MANET 20 with nodes.

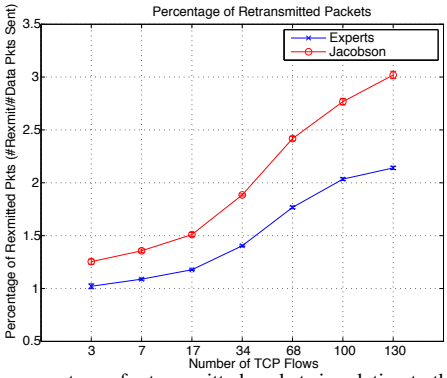


Fig. 2. The percentage of retransmitted packets in relation to the total number of packets transmitted on a MANET 20 with nodes.

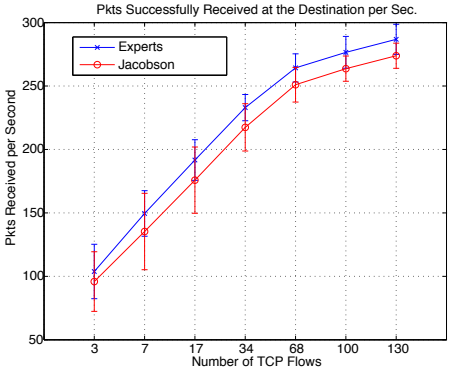


Fig. 3. Total number of packets delivered over the duration of the simulation for a MANET 20 with nodes.

First we considered a Mobile Ad Hoc Network (MANET) composed of 20 nodes. This scenario was built to let us evaluate the performance of the RTT prediction algorithms when the paths in the network and the RTT vary considerably. Although our goal was to show the algorithm's response to RTT fluctuations, we also varied the number of TCP flows during the simulations to change the load and congestion level in the network. We evaluated scenarios with concurrent flow counts of 3, 7, 17, 34, 68, 100 and 130. Although the flows were evenly distributed among nodes, they started in random times across the duration of the simulation and presented random sizes that varied from 1000 to 100000 packets.

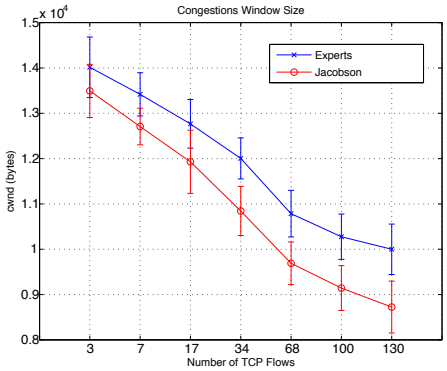


Fig. 4. cwnd over different congestion scenarios on a MANET 20 with nodes.

On Figure 1 we can compare the accuracies of the Experts Framework and Jacobson's algorithms. Note that the more accurate RTT prediction of the machine-learning approach leads to a more efficient setting of the RTO timer, which reduces considerably the relative number of packets retransmitted, especially under heavy traffic conditions. As shown in Figure 2, retransmissions are reduced substantially (by as much as over 30%) without decreasing the number of packets sent. Note also the improvement in the total number of packets received (Figure 3).

Our use of the Experts Framework not only improved the RTT predictions, thereby avoiding unnecessary retransmissions, but also avoided unnecessary triggering of congestion-control mechanisms. Consequently, TCP's congestion window (cwnd) is higher on average, as shown in Figure 4, which shows the average cwnd over the duration of the whole simulation for different congestion scenarios. We can see in this figure how the gap between algorithms increases with increasing congestion conditions. That means that the consequences for increasing traffic, i.e. greater RTT fluctuations, is much less severe when using the experts approach.

We ran similar experiments using a MANET composed of just 10 nodes (less dense network). However, the difference between the algorithms' accuracy for the 10-node experiments did not change much, where the mean error when using the standard TCP timer was always more then double the mean error for the machine-learning approach. We omitted the figures here due to space restrictions and given the similar nature of the results using 10 and 20 nodes.

### B. Scenario II - Static Network

In this experiment, our goal was to isolate the effect of congestion on the performance of the proposed RTT estimator. Therefore, we did not take mobility into account, by simulating

a static wireless ad hoc network. Here we varied traffic the same way we did for Scenario I.

Figure 5 shows the accuracy of the prediction algorithms. Again, like in the previous scenario, the increasing traffic degrades the performance of the algorithms although, this degradation is much larger for Jacobson's RTT predictor. Moreover, when comparing the accuracy results for the static scenario with the mobile scenario with 20 nodes (Figure 1) one can notice that the error is greater for the static scenario for both algorithms. This can be explained by the fact that in the static scenario, since routes between source and destination are less volatile, some nodes start acting as "bottlenecks", once they can be part of the route path for many flows. For these nodes, queue lengths are greater and that generates greater fluctuations in the RTT as the traffic changes along the simulation. These larger fluctuations are responsible for the accuracy degradation. Figure 6 shows the average queue sizes in bytes for both mobile and static scenarios, for different number of flows. As expected, the average queue length for the mobile scenario is much lower.

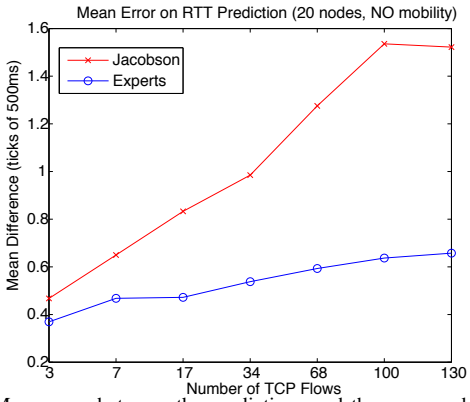


Fig. 5. Mean error between the predictions and the measured RTT for the static network

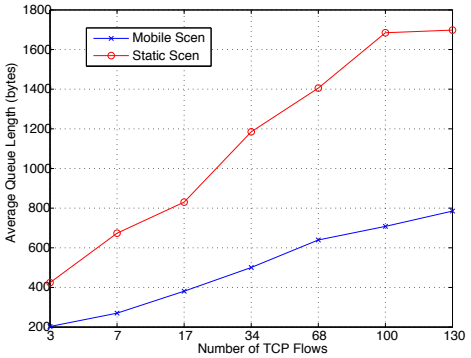


Fig. 6. Average queue length in bytes for both mobile and static scenarios.

In the static scenario the results for the number of retransmitted packets were also lower when using the machine learning approach, as we can see in Figure 7. In this case, the number of retransmissions were smaller than in the mobile scenario, for both algorithms, given that in the static scenario there were no losses due to route failure.

### C. Scenario III - Bursty Traffic

In this scenario we give traffic more of a bursty characteristic. The objective of implementing this scenario is to evaluate the behavior of the predictor and the network performance,

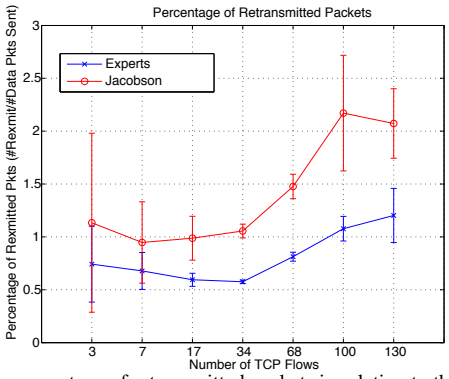


Fig. 7. The percentage of retransmitted packets in relation to the total number of packets transmitted for both algorithms for the static scenario.

under different traffic fluctuations. Moreover, the expected behavior in this case would be a larger improvement in the network metrics when using the machine learning approach, given that it can take advantage of burst characteristics due to sharing. Here, we simulate a network with 20 mobile nodes in which every node starts a TCP flow of 1000 packets every 200 seconds during 90 minutes of simulation. Thus, nodes would transmit for a while and then remain silent until the next cycle of 200 seconds. We also vary the velocity of the nodes between [1, 10]m/s, [20, 30]m/s, [40, 50]m/s.

Since the measured RTT fluctuation for this scenario is much greater, the mean prediction error in this case is larger than in the previous scenarios (figure omitted due to space concerns). We also report for this scenario a much lower number of retransmitted packets, which reflected on the cwnd, allowing a better goodput when using our machine learning approach, as seen in Figure 8.

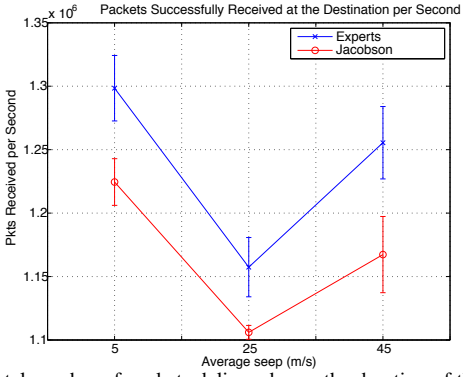


Fig. 8. Total number of packets delivered over the duration of the simulation for different node mean velocities and traffic in burst.

## V. Linux Implementation and Experiments

We implemented the machine-learning algorithm and the Eifel algorithm in the kernel of Linux version 2.6.28.3. We modified the function `tcp_rtt_estimator()` to return the output of the evaluated RTT prediction algorithm. Our implementation of the Eifel algorithm, to the best of our knowledge, is faithful to the algorithm described in [3] for predicting the RTT and setting the RTO.

We acquired data from live runs of file transfers using our modified TCP Kernel modules that implement the Eifel and the proposed prediction algorithm. Data collection happened over 30 file transfers of a 16 MB file. To help filter out the effects of gradual network changes, we interleaved the transfers that

| | Scenario1 | | | Scenario2 | | | Scenario3 | | |
|---|---|---|---|---|---|---|---|---|---|
| **Metric** | Eifel | Jacobson | **Experts** | Eifel | Jacobson | **Experts** | Eifel | Jacobson | **Experts** |
| **Error** | 11.21 | 8.19 | **5.10** | 114.52 | 74.11 | **67.64** | 298.24 | 199.23 | **131.65** |
| **cwnd** | 61.55 | 69.82 | **74.87** | 55.38 | 66.71 | **74.89** | 18.91 | 31.08 | **38.11** |
| **rexmits** | 26.40 | 31.12 | **13.02** | 314.20 | 367.70 | **250.80** | 204.62 | 363.21 | **159.61** |

TABLE II

PREDICTION ERROR (IN TICKS), CWND (IN PACKETS) AND NUMBER OF RETRANSMISSIONS AVERAGED OVER 10 RUNS OF THE SAME EXPERIMENT.

were controlled by our machine-learning approach, the Eifel retransmission timer, and Jacobson's algorithm. In total there were 10 runs of each algorithm for each of 3 scenarios.

The live experiments used a different set of scenarios than the simulations. In *Scenario 1* the source of the file transfer was a Linux machine containing the modified modules for the Experts and Eifel algorithms, and the original Kernel code and TCP timer. This machine was connected to the wired campus network at the University of California, Santa Cruz. The destination was another Linux machine connected to the Internet, physically located in the state of Utah in the USA. *Scenario 2* was similar to Scenario 1, except that the source was now connected wirelessly to a 802.11 access point, which was connected to the Internet through the UCSC campus network. *Scenario 3* was a full wireless scenario, where both source and destination were connected to the same 802.11 access point. All the measurements were collected at the source of the file transfer.

Figure 9 shows around 200 trials of one of the file transfers. It is possible to notice how much faster the machine learning algorithm can respond to sudden changes in the RTT value and how much closer it can follow the real measurements.

Table II summarizes results from live experiments for the three scenarios studied. This table shows an improvement in the RTT prediction from 40% in Scenario 1 up to 51% in Scenario 3 when comparing the accuracy between the experts algorithm and the standard TCP predictor. This difference is even higher when comparing to Eifel. The other performance metrics—average number of retransmissions and cwnd—also improved considerably when applying our machine-learning approach. On the other hand, Eifel has the advantage of not requiring any parameters to be set since gains are computed "on-the-fly". In the case of Jacobson's algorithm, even though a couple of parameters have to be set in advance, it is a much simpler and easier to implement algorithm. However, trading-off complexity to achieve significantly higher performance is consistent with the steady increase of processing and storage capabilities available in computing and communication devices. Thus, given the superior performance illustrated by our results, we can conclude that our approach gives a good trade-off between higher complexity and performance improvement.

## VI. CONCLUSIONS

In the present work, we proposed a novel approach to end-to-end RTT estimation using a machine learning technique known as the Experts Framework. We showcase our approach as an alternative to TCP's RTT estimator and show that it yields higher accuracy in predicting the RTT than the standard algorithm used in most TCP implementations. The proposed machine learning algorithm is able to adapt very quickly to changes in the RTT. Our simulation results show a considerable reduction in the number of retransmitted packets, while increasing goodput, particularly in more heavily congested scenarios. We corroborate our results by running "live" experiments on a Linux implementation. These experiments confirm the higher accuracy of the machine learning approach with more than 40% improvement, not only over the standard TCP predictor, but also when comparing to another well know solution, the Eifel retransmission timer.

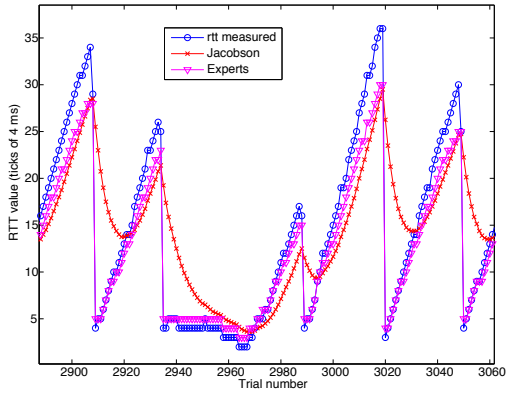


Fig. 9. RTT values measured on the Linux kernel, and predictions made by Jacobson's algorithm and the proposed predictor using the Experts framework.

## REFERENCES


[1] M. Herbster and M. K. Warmuth, "Tracking the best expert," *Mach. Learn.*, vol. 32, no. 2, pp. 151–178, 1998.

[2] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 157–187, 1995.

[3] R. Ludwig and K. Sklower, "The eifel retransmission timer," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 3, pp. 17–27, 2000.

[4] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, (New York, NY, USA), pp. 263–274, ACM, 1999.

[5] W. Lou and C. Huang, "Adaptive timer-based tcp control algorithm for wireless system," in *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, vol. 2, pp. 935 – 939 vol.2, june 2005.

[6] L. Ma, G. Arce, and K. Barner, "Tcp retransmission timeout algorithm using weighted medians," *Signal Processing Letters, IEEE*, vol. 11, pp. 569 – 572, june 2004.

[7] K. Leung, T. Klein, C. Mooney, and M. Haner, "Methods to improve tcp throughput in wireless networks with high delay variability [3g network example]," in *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, vol. 4, pp. 3015 – 3019 Vol. 4, sept. 2004.

[8] D. P. Helmbold, D. D. E. Long, and B. Sherrod, "A dynamic disk spin-down technique for mobile computing," in *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, (New York, NY, USA), pp. 130–142, ACM, 1996.

[9] M. Haeri and A. Rad, "Tcp retransmission timer adjustment mechanism using model-based rtt predictor," in *Control Conference, 2004. 5th Asian*, vol. 1, pp. 686 – 693 Vol.1, july 2004.

[10] D. Ngwenya and G. Hancke, "Estimation of srtt using techniques from the practice of spc and change detection algorithms," in *AFRICON, 2004. 7th AFRICON Conference in Africa*, vol. 1, pp. 397 –402 Vol.1, sept. 2004.

[11] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to tcp throughput prediction," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, no. 1, pp. 97–108, 2007.

[12] "QualNet." `http://www.scalable-networks.com`.

[13] C. Perkins and E. Royer, "Ad-hoc on-demand distance vector routing," in *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pp. 90–100, 1997.

[14] I. . W. Group, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*. IEEE Std. 802.11, 345 E. 47th St, New York, NY 10017, USA: IEEE Computer Society, 1997.