A machine learning approach to improve congestion control over wireless computer networks

Conference Paper · December 2004

DOI: 10.1109/ICDM.2004.10063 · Source: IEEE Xplore

3 authors:

Pascal Geurts
Eindhoven University of Technology
74 PUBLICATIONS **812** CITATIONS
SEE PROFILE

Ibtissam El-Khayat
Ericsson
5 PUBLICATIONS **55** CITATIONS
SEE PROFILE

Guy Leduc
University of Liège
184 PUBLICATIONS **1,443** CITATIONS
SEE PROFILE

A Machine Learning Approach to Improve Congestion Control over Wireless Computer Networks

Pierre Geurts, Ibtissam El Khayat, Guy Leduc
Department of Electrical Engineering and Computer Science
University of Liège
Institut Montefiore - B28 - Sart Tilman
Liège 4000 - Belgium
{geurts, elkhayat, leduc}@montefiore.ulg.ac.be

Abstract

In this paper, we present the application of machine learning techniques to the improvement of the congestion control of TCP in wired/wireless networks. TCP is sub-optimal in hybrid wired/wireless networks because it reacts in the same way to losses due to congestion and losses due to link errors. We thus propose to use machine learning techniques to build automatically a loss classifier from a database obtained by simulations of random network topologies. For this classifier to be useful in this application, it should satisfy both a computational constraint and a time varying constraint on its misclassification rate on congestion losses. Several machine learning algorithms are compared with these two constraints in mind. The best method for this application appears to be decision tree boosting. It outperforms ad hoc classifiers proposed in the networking literature and its combination with TCP improves significantly the bandwidth usage over wireless networks and does not deteriorate the good behaviour of TCP over wired networks. This study thus shows the interest of the application of machine learning techniques for the design of protocol in computer networks.

1 Introduction

Computer networks are highly complex systems that have evolved greatly in the last decade. Nowadays networks combine a large diversity of different technologies (hybrid networks that combines wired, wireless, and satellite links) and must respond to a large variety of tasks with very different requirements in terms of quality of service (e.g. e-mail, videoconference, streaming, peer-to-peer). Furthermore, the behavior of these networks depend on a large number of external factors (user behaviours, load of the networks, link capacities...) that are difficult to appraise. All these factors make the analytical modeling of networks a very difficult (if not impossible) task. Yet, the good behaviour of computer networks requires the design of effective control strategies and the rapid adaptation of these strategies to new technologies. Hence, data mining and machine learning are certainly tools of choice to improve our understanding of networks and to help in the design of control solutions. Furthermore, as networks are computer based systems, it is quite easy to collect the data which are the basis of these methods, either from the observation of real networks or from the simulation of artificial ones (e.g. with a network simulator like `ns-2`[14]).

In this paper, we present an application of machine learning techniques to the improvement of congestion control protocol in hybrid wired/wireless computer networks. Congestion control in computer networks is necessary to share equitably the available bandwidth between several users and to avoid congestion collapse. Nowadays, most internet traffic is carried by TCP that is thus responsible for most congestion control. However, this protocol, which has been deployed in the eighties, has been designed for wired networks and hence its congestion control mechanism is not adapted to nowadays networks where wireless links are becoming more common. In this paper, we propose to apply machine learning techniques to improve the behaviour of TCP over hybrid wired/wireless networks. The main motivation of this paper is to illustrate the different problems we faced for the design of our solution from the machine learning point of view. The technical details about computer networks may be found in [7].

The paper is structured as follows. In the first section, we present the very basis of TCP and the drawback of its congestion control algorithm in the context of wireless networks. We next describe how we propose to solve this problem with machine learning techniques and what are the

main constraints on the machine learning solution. Section 3 describes the database that has been generated for this study. Then, Section 4 presents the different machine learning methods, essentially decision tree based methods, that have been compared for this problem. Experiments are described in Section 5. Finally, we end the paper with our conclusions and some future work directions.

2 TCP over wired/wireless networks

TCP, for "Transmission Control Protocol", is the most widely used protocol in the Internet. More than 90% of the traffic in nowadays networks is carried by TCP. Its success lies in its reliable file transfer and its capacity in avoiding congestion. In this section, we introduce the very basis of TCP congestion control and then we describe our proposed solution to improve its behaviour in the case of wireless networks. These descriptions are very much simplified. Further details about congestion control mechanisms in TCP may be found in [1].

2.1 TCP congestion control

A congestion arises in the network when routers are not able to process the data that arrive to them and this causes buffer overflows at these routers that result in the loss of some packets of data. The only solution to relax a congestion is to reduce the load of the network. TCP congestion protocol is based on the definition of a congestion window (denoted cwnd) that controls the rate at which packets are sent by a sender to a receiver and a mechanism to adapt the value of this window size according to the state of the network. The idea is to increase the rate steadily (i.e. additively) when everything works fine and to reduce it more abruptly (i.e. multiplicatively) as soon as a loss occurs.

In TCP, every packet sent from a sender to a receiver gives rise to an acknowledgment packet sent in the opposite way by the receiver. The congestion window determines the maximum number of packets that can wait for an acknowledgment at the sender side at each time. A normal scenario of communication between a sender and a receiver is represented in the left part of Figure 1, starting with a congestion window of size 1. The sender starts by sending one packet. Then, each time it gets an acknowledgment, it sends a new packet. When it gets the acknowledgment of the last packet of the previous group of cwnd packets, it increments cwnd by 1 and then sent two packets in burst instead of one[1]. The round-trip-time (RTT) is defined as the time between the sending of a packet and the reception of its acknowledgment by the sender. Hence, as seen in Figure 1, the number

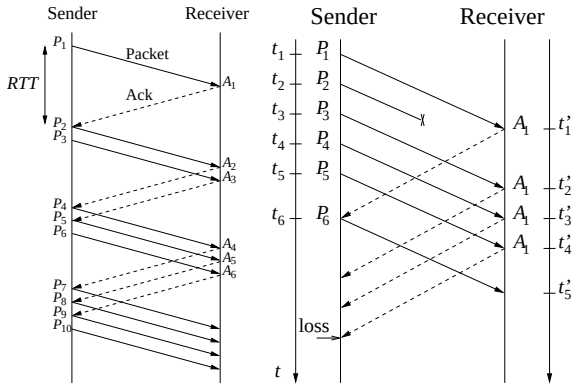

Figure 1. Congestion control without loss (left) and with loss (right)

of packets sent during each RTT is exactly equal to the window size and hence the instantaneous throughput of TCP is equal to $(cwnd.\text{size\_of\_packets}/RTT)$. Hence, the additive increase of the window size leads to a linear growth of the bandwidth occupancy and allows the sender to slowly probe the available bandwidth.

The right part of Figure 1 shows a scenario where cwnd=5 and the second packet is lost. The receiver knows that a packet has been lost when it receives a packet before its predecessors in the sequence order. In this case, it sends to the sender the acknowledgment of the last packet it has received in the sequence. As packets may not be received in order for other reasons than packet loss (e.g. two packets may follow different routes), a packet loss is detected only after three duplicate acknowledgments [2]. At the detection of at least three duplicates, the sender divides its congestion window by a factor 2, starts retransmission of the lost packet, and enters in a so-called fast recovery phase. The description of this latter phase is not useful for the comprehension of this paper. Further details may be found in [1].

This congestion control mechanism thus makes the hypothesis that packet losses are due to buffer overflows and it works fairly well in the case of wired networks where this is indeed the case. However, in wireless links, losses caused by error in the link (for example by signal fading) are not so unlikely. The TCP protocol as described above has no mechanism to distinguish loss caused by a link error from losses caused by a congestion and it reduces systematically its rate whenever it faces a packet loss. This reduction is not justified when there is no congestion and the consequence is that the throughput of TCP over wireless link is lower than what it could be. Several studies have highlighted the bad behaviour of TCP over wireless link (e.g. [17]).

[1] Actually, cwnd is initialized by a "slow-start" phase that accelerates the growth of the congestion window to a certain level

[2] Packet losses are also detected if after some fixed timeout the sender has not received any acknowledgment from the receiver. This type of packet loss will not be considered in this paper.

2.2 Proposed machine learning solution

A straightforward solution to increase the throughput of TCP over wireless links is to prevent it from reducing its rate when it faces a loss due to a link error as it does when it faces a congestion. Several approaches have been proposed in the networking literature (e.g. [2, 15]) that assumes the support of the network. The approach adopted in this paper is to endow one of the end systems (i.e the sender or the receiver) with an algorithm able to determine the cause of a packet loss only from information available at this end system. With such a classifier, the modified protocol will proceed as follows: each time a loss is detected by a triple duplicate, its cause is determined by the classification model. If the loss is classified as due to a congestion, the sender proceeds as usual (i.e. it divides its congestion window by two). Otherwise, it maintains its congestion window constant. The advantage of such end-to-end methods is that they do not require the support of the networks and hence they can be more easily deployed in practice.

Several rough classification rules for losses have been proposed in the literature based on heuristics or analytical derivations (e.g. in Veno [11] or Westwood [16]) but our experiment in Section 5.1 will show that these rules do not classify very well the loss causes in practice. In this paper, we propose to use supervised machine learning techniques to automatically derive a classification model for loss causes. This model will use as inputs statistics computed on packets received by the sender or by the receiver and it will be induced from a database obtained by simulations of random network topologies.

2.3 Constraints on the classifier

For this application to be practically feasible and useful, there are several specific constraints that should be taken into account when choosing a particular classification model. They are discussed below.

2.3.1 Computational constraint

In this application, the computing times to make a prediction and the computer resource needed to store the model are not to be neglected. What we can afford according to these criteria will depend on which side we decide to put the classification algorithm. If the sender is in charge of the classification, using a relatively complex classification algorithm could be a problem if the sender has a great number of clients (which is usually the case). So, it could be desirable to trade some accuracy for more efficiency. Of course, if the sender has a benefit in providing the highest rate to the receivers, or if it has a few number of clients (e.g. proxy), it may still prefer to use more accurate classifier at the expense of computational costs.

On the other hand, if the receiver is in charge of the classification, the complexity of the classifier will not be an issue anymore (providing it remains reasonable) and there is no reason to avoid using the more accurate models. However, in this case, once a loss is classified, the receiver will need to send the classification result to the sender or at least a message where it asks the sender not to decrease its rate. The problem is that by allowing this kind of exchange, we will open greatly the door to distrusted receivers. Hence, we cannot decide straightforwardly to implement the classifier at the receiver side.

This discussion shows that in this application the computational constraint should be taken into account when selecting a model and thus there is an interest even in less accurate models for computational efficiency reasons.

2.3.2 Adaptive misclassification costs

Since TCP is so popular in nowadays networks, one important constraint that any new protocol should satisfy before being practically used is that it should be TCP-Friendly ([9], [13]). A TCP-Friendly protocol is a protocol that, when sharing the network with TCP, allows this latter to have a throughput similar to the one it would get if it were in competition with another TCP in similar conditions. This constraint ensures that the new protocol would not be unfair towards the most widely used protocol on the Internet.

If our classifier misclassifies quite often congestion losses, it will be unfair towards TCP since it will force its competitor to decrease its rate to avoid the congestion and at the same time it will maintain its congestion window unchanged. Even if it only competes with the same protocol, a global misclassification of congestion losses will lead the network to a blackout as nobody will reduce its congestion window. So, for their own benefit and the benefit of the community, one should not use a classifier that makes many errors on the detection of losses due to congestion. On the other hand, a classifier that perfectly detects congestion losses and misclassifies many link error losses will be no better than TCP and hence is not worth consideration.

The best strategy is thus to determine the maximum admissible error on the classification of congestion losses that ensures TCP-friendliness and choose a classifier that satisfies this constraint and minimizes errors on the link error losses. Actually, we show in [7] that it is possible to derive analytically an upper bound on the misclassification error that ensures TCP-friendliness. This bound is of the form:

$$Err_C \leq f(RTT, p), \tag{1}$$

where Err_C is the probability of misclassifying a congestion loss as a link error loss, RTT is the round-trip-time, and p is the actual loss rate, i.e. the percentage of lost packets whatever the cause. However, RTT and p, and conse-

quently the bound on Err_C, depend on the particular network topology and furthermore they may change with time. We thus have two solutions to select a classifier. First, we can determine a conservative value of Err_C such that (1) is satisfied for all reasonable values of RTT and p and then choose a classifier that satisfies this bound on Err_C. However, as usually allowing greater error on congestion losses will allow to reduce error on link error losses, a better solution is to dynamically adapt the classifier such that it always satisfies the constraint for the current (estimated) value of RTT and p. This is the strategy adopted in this paper.

This implies two constraints on the classifier. First, it should be possible to tune the classifier in order to get different tradeoffs between the two kinds of error. In preference, this tuning should not require to rerun the learning algorithm or to store several models (corresponding to different error costs) because of the computational constraint. Second, the model should be as good as possible independently of the error costs. This will imply to evaluate our classifiers using ROC curves (see Section 4.2).

3 Database generation

To solve our problem of losses classification, each observation $< x_i, y_i >$ of our learning sample will be an input/output pair where the inputs x_i are some variables that describe the state of the network at the occurrence of a loss and the (discrete) output y_i is either C to denote a loss due to a congestion or LE to denote a loss due to a link error.

To make the model generally applicable, the observations in the database must be as much as possible representative of the conditions under which we will apply the classification model. So, the database generation should take into account all the uncertainties we have a priori about the topology of the networks, the user behaviours, and the protocols. The way we generated our observations is described in Section 3.1. Another important question is the choice of the input variables which is discussed in Section 3.2.[3]

3.1 Simulations

The database was generated by simulations with a network simulator called `ns-2`[14]. To generate our observations of losses, we have used the following procedure: a network topology is generated randomly and then the network is simulated during a fixed amount of time, again by generating the traffic randomly. At the end of the simulation, all losses that have occurred within this time interval are collected in the database. This procedure is repeated until we have a sufficient number of observations in the database. In practice, the larger the learning sample, the better it is for supervised learning algorithms. In our study, we have collected 35,441 losses (among which 22,426 are due to congestion) that correspond to more than one thousand different random network topologies.

To generate a random topology, we first select a random number of nodes and then choose randomly the presence or the absence of a connection between two nodes and the bandwidths of these connections. Concerning the traffic, 60% of the flows at least were TCP flows[4] and the others were chosen randomly among TCP and other types of traffic proposed by the network simulator `ns-2`. The senders, the receivers, and the duration of each traffic were set randomly. The number of wireless links, their loss rate, and their place in the network were also randomized. Further details about our simulations may be found in [7].

3.2 Input variables selection

The choice of the inputs variables is mainly directed by two constraints. First, of course, it should be possible to predict a congestion from the observation of these variables. Second, for our classification models to be practically useful, these variables should also be measurable, either at the receiver or at the sender side.

At the end system (sender or receiver), the only information we can measure to predict a congestion is some statistics on the packet departure and arrival times. The one-way delay of a packet is the delay between the instant the packet was sent and the instant it is received (e.g. the one-way delay of the first packet in right part of Figure 1 is $t'_1 - t_1$). The inter-packet time corresponding to a packet is the difference between the time it is sent and the time the previous packet was sent (e.g. the inter-packet time of the second packet in Figure 1 is $t_2 - t_1$). These two indicators are certainly influenced by the state of congestion of the network and thus we believe that it should be possible to predict the occurrence of a congestion by monitoring these indicators for a number of packets.

To define our inputs, we thus compute these two values for the three packets following the loss and the packet that precedes it. In Figure 1, they are respectively packets P_3,P_4,P_5, and P_1. Since the absolute value of these indicators are highly dependent on the particular network topology, it is not a good idea to introduce them as it in the inputs. Instead, we propose to use as inputs only relative values of these measures normalized in different ways.

To this end, we further compute the average, the standard deviation, the minimum, and the maximum of the one-way delay and inter-packet time for the packets that are sent during each round-trip-time and we maintain these values at

[3] The database is available electronically at http://www.montefiore.ulg.ac.be/~geurts/BD-Fifo.dat.gz.

[4] We have chosen a particular variant of TCP called Newreno but any other version of TCP could be used.

each time for the last two round-trip-times before the current time. Our final inputs are then various simple functions (for example, the ratio) relating these (4×2) values at the occurrence of a loss to the inter-packet times and the one-way delays of the three plus one packets surrounding the loss. All in all, this results in a set of 40 numerical input variables.

4 Machine learning techniques

In this study, we propose to use mainly decision tree based algorithms. Indeed, they provide usually very good results for classification problems. Furthermore, they are among the fastest methods when it comes to make a prediction. Nevertheless, we also compare these methods to neural networks and the k-nearest neighbors. Section 4.1 briefly describes these algorithms and Section 4.2 explains ROC curves that are used to evaluate them.

4.1 Algorithms

Decision trees are among the most popular supervised learning algorithms. A decision tree partitions the input space into hyper-rectangular regions where the classification is constant. The partition is determined recursively by successive binary axis-parallel cuts. The main advantage of this method for our application is that pruned trees are usually quite small and the computation of a prediction that consists in a sequence of comparisons is very fast. To build a decision tree, we have adopted the standard CART algorithm described in [6] (with cost-complexity pruning by ten-fold cross-validation).

Decision tree ensembles. One drawback of standard regression trees is that they suffer from a high variance which sometimes prevents them from being competitive in terms of accuracy with other learning algorithms. Ensemble methods are generic techniques that improves a learning algorithm by learning several models (from the same learning sample) and then by aggregating their predictions. In our experiment, we have used four ensemble methods that have been proposed for decision trees:

Bagging [4]. In bagging, each tree of the ensemble is built by the CART algorithm (without pruning) but from a bootstrap sample drawn from the original learning sample. The predictions of these trees are aggregated by a simple majority vote.

Random forests[5]. This method is a slight modification of bagging, which improves often both accuracy and computational efficiency. In this method, when splitting a node, k variables are selected at random among all input variables, an optimal split threshold is determined for each one of these and the best split is selected among these latter. In our experiments, the value of k has been fixed to its default value which is equal to the square root of the number of attributes.

Extra-trees[12]. Unlike bagging and random forests, this method generates each tree from the whole learning sample. During tree growing, the best split is selected among k totally random splits, obtained by choosing k inputs and split thresholds at random. The main advantage of this algorithm is its computational efficiency since there is no threshold optimization stage. In the experiments, the value of k for this method has also been fixed to the square root of the number of attributes.

Boosting. While in the three preceeding methods the different trees are built independently of each other, boosted trees are built in a sequential way. Each tree of the sequence is grown with the classical induction algorithm (with pruning) but by increasing the weights of the learning sample cases that are misclassified by the previous trees of the sequence. Furthermore, unlike in previous methods, when making a prediction, the votes of the different trees are weighted according to their accuracy on the learning sample. In our experiments, we use the original Adaboost algorithm proposed in [10].

Usually all these methods improve very much the accuracy with respect to a single tree. However, since they require to grow T decision trees, the computing times for testing and the memory required to store a model is T times higher than for a classical decision tree.

Multilayer perceptrons [3] Multilayer perceptrons are a particular family of artificial neural networks. Neural networks represent a model as the interconnection of several small units called perceptrons that compute a weighted average of their inputs and send this average through a non linear functions (usually a hyperbolic tangent). This method usually gives more accurate models than decision trees but it is also much more demanding in terms of computing times and computer resources. In our experiments, we have used a Levenberg-Marquard optimization algorithm to learn neural networks.

k-Nearest Neighbors. To make a prediction at some point of the input space, the k-NN algorithm simply computes the k nearest neighbors of this point in the learning sample (according to the euclidian distance) and then outputs the majority class among the k neighbors. An important drawback of this method for this application is that the computation of a prediction is quite demanding and furthermore it requires to store the entire learning sample.

Of course, there exist many other learning algorithms that could have been used for our application but we believe that these methods already provide interesting results.

4.2 ROC curves

One very desirable characteric of the chosen classifier discussed in Section 2.3.2 is that it should be possible to adjust its error on congestion losses dynamically. Actually, all classification methods presented in the previous section, not only provide a class prediction for each value of the inputs x but also provide an estimate of the conditional probability of each class, C or LE, given the inputs x. In the two class case, the default use of the model is to classify a loss as a congestion loss if the probability estimate $\hat{P}(C|x)$ is greater than 0.5. However, by using a user defined threshold P_{th} different from 0.5, we can change the misclassification probability of each class and hence adjust our classifier so that it satisfies (1).

A natural way to evaluate our models independently of the value of P_{th} is to use receiver operating characterisic (ROC) curves (see [8] for an introduction). A ROC curve plots for every possible value of the threshold P_{th} the true positive rate versus the false positive rate of a given class (among two classes). In our case, the true positive rate is taken as $1 - Err_C$ and the false positive rate is the probability of misclassifying a link error loss, that will be denoted Err_{LE}. A classifier is thus strictly better for our application than other classifiers if for every possible value of Err_C the value of Err_{LE} is lower than with the other classifiers or, in other words, if its ROC curves lies to the left of the ROC curves of other classifiers. While ROC curves are two-dimensional, a common one-dimensional summary of a ROC curve to compare classifiers is the area under the ROC curve (AUC) which we will also use to rank our packet loss classifiers. ROC curves and AUC are computed according to the algorithms presented in [8].

5 Experiments

First, we evaluate classifiers and then we evaluate the improvement obtained by the enhanced version of TCP with the best classifier only.

5.1 Comparison of loss classifiers

To make a reliable estimate of the error of each model, we have randomly divided the whole database into two parts: a learning set of 25,000 cases and a validation set containing the remaining 10,441 cases. A classification model is built with every method on the learning sample and its ROC curve, AUC, and error rate are evaluated on the validation sample. The methods are compared from these criteria in Table 1 and in Figure 2. Since computing times for testing are important for our application, we also give in Table 1 the time needed to classify the validation set with each method. For ensemble methods, we build $T = 25$ trees. As the ROC curves of all ensemble methods are very close, we only give the results obtained by boosting in Figure 2. The value of k for the k nearest neighbors was determined by leave-one-out cross-validation (the optimum is $k = 7$). For the MLP, we tried several structures of one and two layers with a number of neurons going from 10 to 50 neurons in each layer. Table 1 only shows the best result that was obtained with two layers of 30 neurons.

Table 1. Comparison of different ML methods

Method	AUC	Error (%)	Time (msec)
DT	0.9424	8.92	150
Bagging	0.9796	6.65	650
Random forests	0.9823	6.48	600
Extra-trees	0.9813	6.91	940
Boosting	0.9840	6.34	570
MLP	0.9761	7.67	1680
k-NN	0.9541	10.16	316,870
Veno	0.7260	34.52	-
Westwood	0.6627	41.54	-

The results are quite good, especially considering the diversity of the network topologies represented in the database. The decision tree has the lowest AUC but it is by far the fastest method to make a prediction. The k-NN has a better AUC than decision tree but its ROC curve is worst for small values of Err_C, which is the region of interest of our application. It is also the slowest method in terms of computing times. MLP improves upon both methods in terms of accuracy but it remains below ensemble methods in terms of accuracy and computing times. All ensemble methods are very close but boosting is superior along the two criteria. As it combines pruned trees, it also builds the less complex models.

For comparison, we put also in Table 1 the result obtained on the validation set by two ad-hoc classification rules that have been proposed in the networking literature in the context of two extensions of TCP for wireless networks called Veno [11] and Westwood [16]. The results obtained by these rules are far below the results obtained by machine learning algorithms. This clearly shows the interest of a machine learning approach in the context of this application, both to improve existing classifiers but also to assess their validity.

Another reason to prefer one method over the others that does not appear in Figure 2 is the discretization of the ROC curves, i.e. the number of sensible values of P_{th}. Indeed, in our application, it is highly desirable to be able to finely control the value of Err_C by means of P_{th}. From this point of view also, all methods are not equal. Figure 3 illustrates these differences by drawing Err_c versus P_{th} for some methods. Decision tree and k-NN provide only a few sensible values of P_{th} (about 80 for the pruned tree

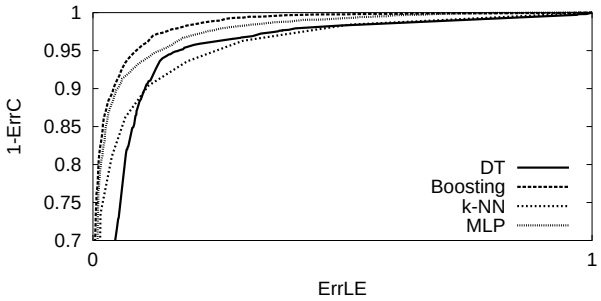

Figure 2. Comparison of the ROC curves of different ML methods

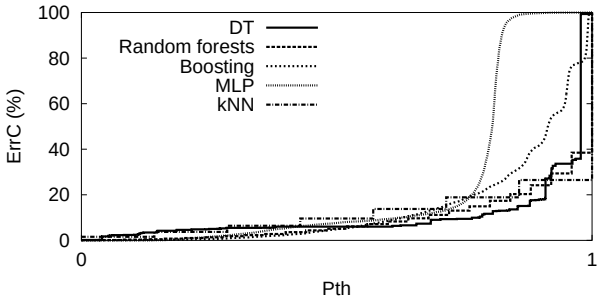

Figure 3. Err_C versus P_{th} for different methods

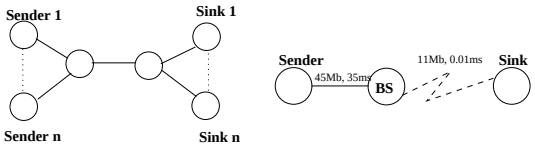

Figure 4. Two topologies used in our simulations

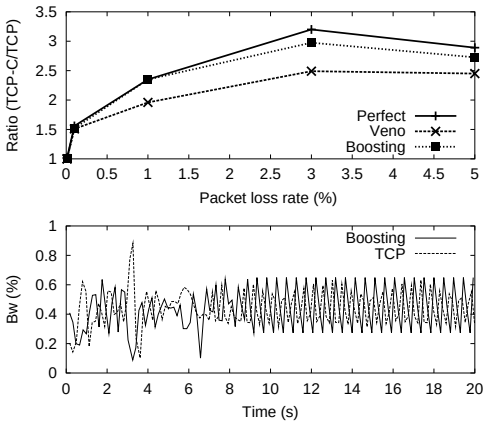

Figure 5. Top, the gain in wireless link, bottom, TCP-Friendliness

on Figure 3 and 7 for k-NN). MLP on the other hand allows to control very finely the value of Err_C. From this point of view, boosting is also superior to other ensemble methods (represented by Random forests in Figure 3). Indeed, other ensemble methods build unpruned trees that provide only 0-1 conditional probability estimates. Hence, if T trees are built, there are only T different possible values of P_{th}. Boosting, on the other hand, builds pruned trees that provide finer probability estimates and thus its Err_C curve is much smoother.

This analysis tells us that boosting is the best candidate for our application. Our experiments with the simulated protocol in the next section will thus focus on this method.

5.2 Simulations of the modified protocol

We propose to use the boosting classifier in the following way to improve TCP. Each time a loss occurs, the minimal value of P_{th} such that (1) is satisfied is determined from the current (estimated) values of RTT and p. Then, the ensemble of trees together with this value of P_{th} are used to classify the loss. If the loss is predicted as caused by a congestion, TCP works as usual. Otherwise, it maintains its congestion window constant. The correspondence between Err_C and P_{th} is obtained on the validation set by inverting the curve in Figure 3.

The two main criteria to evaluate this protocol are bandwidth usage in the case of wireless links and TCP-friendliness in the case of wired networks. In this paper, we only describe two simple experiments showing the interest of the machine learning solution along these criteria. A more detailed validation of the protocol may be found in [7].

Improvement on wireless links. To illustrate the gain of our protocol over wireless link, we simulated the simple hybrid network illustrated in the left part of Figure 4 with the network simulator `ns-2`. The first link (Sender to BS) is wired, and the second part, which connects the base station to the sink, is wireless. The bottleneck is the wireless link, which has a bandwidth equal to 11Mb/s. We compare the ratio between the throughput obtained by the classifier and the one obtained by a standard TCP when we vary the packet loss rate from 0 to 5% over the wireless link. Each simulation is run 50 times. To have some points of comparison, we run also simulations with two other classification rules: the classification rule proposed in Veno and, as our system is simulated, a hypothetic perfect classification rule. The graph at the top of Figure 5 illustrates the ratio obtained by TCP enhanced with the three classifiers. A standard TCP would have a ratio of 1 whatever the loss rate over the wireless link. The superiority of boosting over Veno and standard TCP is clear. It is also very close to the perfect model. Its gain with respect to TCP reaches about 300% when the loss rate is equal to 3%. In other words, the file tranfer with

our protocol is three times faster than with standard TCP.

TCP-friendliness. To test the behaviour of our protocol when it competes with another TCP, we use the topology illustrated in the right part of Figure 4 with $n = 2$ which is often used to test the fairness of a new protocol. The experiment consists in running a standard TCP in competition with our TCP plus the boosting classifier. The bottom graph of Figure 5 illustrates the evolution of the throughput obtained by each traffic. This figure shows that the link is fairly shared between the two protocols. The same experiment was run with Veno's classification rules and in this case, the bandwidth used by Veno was five times higher than that of TCP. This unfairness is not surprising as Veno's classification rule misclassifies many congestions and, hence, it reduces its bandwidth less often than a standard TCP.

6 Conclusions

In this paper, we have presented the application of machine learning techniques to the improvement of congestion control protocol in wireless networks. The idea is to enhance TCP with a loss classifier and to adapt its behaviour according to the predictions of this classifier. A particularity of this application from the machine learning point of view is the need to adapt dynamically the misclassification costs of the classifier that requires to evaluate machine learning algorithms with ROC curves. A database has been generated by simulating a thousand random network topologies and several machine learning algorithms have been compared for this task. Decision tree boosting turns out to be the best method in terms of accuracy. Its application in the context of TCP shows a significant improvement of bandwidth usage in wireless networks and no deterioration in traditional wired networks. Furthermore, the resulting classifier also compares very favorably to existing ad-hoc classification rules that have been proposed in the networking literature. This study thus shows the interest of the application of machine learning techniques to help designing new protocol in computer networks.

There nevertheless remain several possible improvements. From the point of view of computing times, the model induced by boosting may still be too complex for some practical applications. Several techniques could be used to reduce its computational requirement. For exemple, we could try to reduce the number of trees in the ensemble or build smaller trees. As the size of decision trees is proportional to the size of the learning sample, it is also possible to use subsampling techniques to reduce the size of our models. Another drawback of our approach is that we have to store and update continuously about 40 variables based on different statistics measured on the packets. It would be desirable to reduce this set of variables as much as possible for computational efficiency reason. In all cases, the selection of the best tradeoff between accuracy and computational efficiency will require to consider the real impact of a reduction of accuracy on the behaviour of the protocol.

References

[1] M. Allman, V. Paxson, and W. Stevens. RFC 2581: TCP congestion control, 1999.

[2] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. *15th International Conference on Distributed Computing Systems*, 1995.

[3] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.

[4] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[5] L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[6] L. Breiman, J. Friedman, R. Olsen, and C. Stone. *Classification and Regression Trees*. Wadsworth International (California), 1984.

[7] I. El Khayat, P. Geurts, and G. Leduc. Classification of packet loss causes in wired/wireless networks by decision tree boosting. Submitted, 2004.

[8] T. Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report, HP Labs, 2004.

[9] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*, pages 43–56, Stockholm, Sweden, August 2000.

[10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the second European Conference on Computational Learning Theory*, pages 23–27, 1995.

[11] C. P. Fu and S. C. Liew. TCP veno: TCP enhancement for transmission over wireless access networks. *IEEE (JSAC) Journal of Selected Areas in Communications*, Feb. 2003.

[12] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. Submitted, 2004.

[13] M. Mathis, J. Semke, Mahdavi, and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *ACM Computer Communication Review*, 3:67–82, July 1997.

[14] S. McCanne and S. Floyd. *The LBNL Network Simulator.* Lawrence Berkeley Laboratory, 1997.

[15] C. Parsa and J. J. Garcia-Luna-Aceves. Improving TCP performance over wireless networks at the link layer. *Mob. Netw. Appl.*, 5(1):57–71, 2000.

[16] R. Wang, M. Valla, M. Sanadidi, B. Ng, and M. Gerla. Efficiency/friendliness tradeoffs in TCP westwood. In *In Proceedings of the Seventh IEEE Symposium on Computers and Communications, Taormina, July 2002*.

[17] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen. TCP performance issues over wireless links. *Communications Magazine, IEEE*, 39(4):52–58, 2001.