Improving TCP in wireless networks with an adaptive machine-learnt classifier of packet loss causes

Ibtissam El Khayat, Pierre Geurts, and Guy Leduc

Department of Electrical Engineering and Computer Science
University of Liège
Institut Montefiore - B28 - Sart Tilman
Liège 4000 - Belgium

Abstract. TCP understands all packet losses as buffer overflows and reacts to such congestions by reducing its window and thus its rate. In hybrid wired/wireless networks where a non negligible number of packet losses are due to link errors, TCP is unable to sustain a reasonable rate. In this paper, we propose to extend TCP Newreno with a suitable packet loss classifier built by a supervised learning algorithm called 'decision tree boosting'. The learning set of the classifier is a data base of 25,000 packet loss events in a thousand of random topologies. Since a limited percentage of wrong classifications of congestions as link errors is allowed to preserve TCP friendliness, our protocol computes this constraint dynamically and tunes a parameter of the classifier accordingly to maximize the TCP rate. Our classifier outperforms the Veno and Westwood classifiers by achieving a higher rate in wireless networks while remaining TCP friendly.

1 Introduction

TCP has been deployed in the eighties. Its congestion control is based on the fact that packet losses are mainly due to buffer overflows and it works quite well in such situations. However, nowadays, many applications use TCP as their transport protocol and hence pass through wireless links, which become common in the Internet. Over these links, packet losses are not due anymore only to overflows but can also be caused by link errors. TCP, which has no mechanism to distinguish packet loss causes, 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.

To increase its throughput, TCP should avoid reacting to a packet loss due to a link error as it does when it faces a congestion. Two possibilities have been proposed in the literature. The first one is to hide link error losses from the sender (for example by splitting the TCP connection [1] or retransmitting in link layer). These kinds of solutions assume the support of the network. Splitting the

TCP connection has another important drawback which is the violation of the principle of end-to-end TCP: It allows to send an acknowledgment to the sender before the sink has received the packet. The second approach, which is the one adopted in this paper, is to endow one of the end systems with an algorithm that classifies the packet loss causes. Following this strategy, Veno [7] and Westwood [15] use some simple test to classify loss causes. Veno estimates the backlog and consider that the loss is due to a congestion if the backlog is higher than 3. Westwood classifies loss causes by a test equivalent to comparing the current RTT to 1.4.RTT$_{min}$ (where 1.4.RTT$_{min}$ is the smallest RTT estimated by TCP). In our opinion, one such simple test is not sufficient to make a good classification of loss causes in general and indeed, our simulations below will show that these two tests do not classify very well the loss causes in practice. Liu and al. [12] use hidden markov models based on RTT values to make the discrimination. It has been shown in [2] that there is no correlation between the round-trip-time and the loss cause. Indeed, a modification in the return path affects the round-trip-time without affecting the loss cause.

Therefore, we propose in this paper to infer a more complete model to classify loss causes that combines several indicators instead of one. Characterising analytically the network conditions leading to a certain type of packet loss is difficult because real networks are very complex systems but also because their behaviours depend on a large number of random external factors (user behaviours, current topologies,...) which are difficult to model analytically. On the other hand, it is quite easy to simulate the network behaviour (e.g. with a network simulator like `ns-2`) or to gather data from observation of the behaviour of a real network. This is the typical situation where automatic learning techniques are useful. These algorithms are general techniques to extract a model of a system only from data obtained either by direct observations or by simulation of this system. Of interest for our problem are supervised learning algorithms which focus on the approximation of an input/output relationship only from observations of examples of this relationship.

More specifically, in this paper, we propose to apply a particular learning algorithm called decision tree boosting to automatically design a model for discriminating the two possible packet loss causes and then use this model at best to improve the performance of TCP in wired/wireless networks. The paper is structured as follows. In Section 2, we give a short general introduction to supervised learning algorithms. The application of learning algorithms requires the generation of a database from which to infer a model. Section 3 describes how we generate this database and evaluates the performance of decision tree boosting applied to this problem. In Section 4, we explain the design of our improved TCP constructed upon Boosting. Finally, Section 5 evaluates our extension of TCP with several simulations.

2 Supervised learning

Supervised learning is the part of the field of machine learning which focuses on modeling input/output relationships. More precisely, the goal of supervised learning is to identify a mapping from some input variables to some output variable on the sole basis of a sample of observations of these variables. Formally, the sample of observations is called the learning sample LS and is a set of input/output pairs, $LS = \{< x_1, y_1 >, < x_2, y_2 >, ..., < x_N, y_N >\}$, where x_i is the vector of values of the input variables (also called the attributes) corresponding to the i^{th} observation (also called an object) and y_i is its output value. Attribute values may be discrete or continuous. The goal of supervised learning can be formulated as follows: From a learning sample LS, find a function $f(x)$ of the input attributes that predicts at best the outcome of the output attribute y for any *new unseen* value of x. When the output takes its values in a discrete set $\{C_1, C_2, ..., C_m\}$, we talk about a classification problem and when it is continuous, we talk about a regression problem.

This problem is solved by a (supervised) learning algorithm. Loosely speaking, a learning algorithm receives a learning sample and returns a function f (an hypothesis or a model) which is chosen in a set of candidate functions (the hypothesis space). There exist many learning algorithms, which differ mainly in the hypothesis space but also in the optimization algorithm that searches this space for a good model. Among the most popular supervised learning algorithms, there are decision trees and neural networks. In this paper, we will use an algorithm called decision tree boosting, which is a powerful extension of decision tress.

For a complete reference on supervised learning, see for example [9].

3 Loss classification by supervised learning

In this section, we focus on the problem of the derivation and the evaluation of a model for predicting loss causes by using supervised learning techniques. The question of the application of this model to improve TCP will be addressed in the next sections.

3.1 The Database

To solve our problem of losses classification, each observation $< x_i, y_i >$ of the learning sample will be an input/output pair where the inputs x_i are some variables that describe the state of the system 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. We describe below the way we generated our observations and we discuss our choice of input variables.

Database generation The database was generated by simulations with the network simulator `ns-2`. 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 that correspond to more than one thousand different random topologies.

To generate a random topology[1], we first select a random number of nodes (between 10 and 600) and then choose randomly the connections between these nodes. The links are chosen simplex to avoid symmetrical topologies. The bandwidth, the propagation delay and the buffer size of the links were chosen randomly. The bandwidth is chosen between 56Kb/s and 100Mb/s while the propagation delay varies between 0.1ms and 500ms. As Droptail is the most widely deployed policy [4], our simulations all use this latter policy.

The number of wireless links, their place in the topology, the error model and the loss rate were also drawn at random. The error models are either the simple uniform error model, to mimic random losses, or the two-state Gilbert-Elliott model, to mimic bursty losses. These two models are often used to simulate wireless losses (eg. [15], [8]).

Concerning the traffic, 60% of the flows at least were TCP Newreno flows and the others were chosen randomly among TCP and other types of traffic proposed by `ns-2` and based on UDP. The senders, the receivers, the packets size and the duration of each traffic were set randomly. Thus, the database contains losses belonging to short and long TCP sessions. This random choice of traffic length allows us to avoid making any assumption about the network load which is randomized in the database.

The choice of the inputs At the end system, the information we can measure to predict a congestion is the inter-packet times and the one-way delay. Subsequently we will also use the term queuing delay to denote the one way delay. These measures can be obtained at both sides. The one-way delay, computed by one of the two entities, is the difference between the timestamp of the acknowledgment and the timestamp of the TCP packet, and is actually the real one-way delay minus the difference between the clocks of the sender and the receiver. This latter difference is not important in our study since we will see below that our inputs are based only on relative variation of the one-way delay.

To compute our inputs, we use the values of the inter-packet times and the one-way delay for the three packets following the loss[2] and the packet that precedes it. To make the model independent of the absolute values of these measures, we normalized these values in different ways. To this end, we relate them

[1] The TCL code used to generate our topologies can be found at [11]

[2] We consider only losses detected by triple duplicates.

using various functions to 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 the last two round-trip-times before the loss. In total, this normalization results in about 40 inputs.

3.2 Decision tree boosting

There exist many different learning algorithms that could be used for our problem and in fact, we have carried out experiments with several of them. However, for the sake of brevity, we concentrate our discussion in this paper on the method that gives the best results for our problem, which turns out to be decision tree boosting.

Decision tree induction [3] is one of the most popular learning algorithm. A decision tree represents a classification model by a tree where each interior node is labeled with a test on one input attribute and each terminal node is labeled with a value of the output (here C or LE). To classify an observation with such a tree, we simply propagate it from the top node to a terminal node according to the test issues and the prediction for this observation is the class associated with the terminal node.

In supervised learning, ensemble methods are generic techniques that improve a learning algorithm by learning several models (from the same learning sample) and then aggregating their predictions. Boosting [6] is a particular ensemble method where the models are built in sequence. Each model is built by increasing the weights of the learning sample cases that are misclassified by the previous models in the sequence. Then, the prediction of the resulting ensemble of models is the majority class among the classes given by all models. When applied on the top of decision trees, this method often improves very importantly the accuracy with respect to a single tree. Actually, this combination of boosting and decision tree is often considered as one of the best supervised learning algorithm for classification problems[9].

In our experiments, we have used our own implementation of the algorithm for decision tree induction proposed in [3] and the number of trees constructed by boosting was fixed to 25.

3.3 Model evaluation

Usually, the error rate of the model at classifying loss causes in the learning sample is very small since the learning algorithm explicitely tries to minimize this error. Thus, this error is not a good indication of the ability of the model at classifying losses in new unseen topologies. To get a more reliable estimate of the error of the classifier, we have thus randomly divided the database into two parts: a learning sample that is used to learn the model and a test sample on which the resulting classifier is tested. Since the losses in both samples are obtained from different topologies, the error rate of the model at classifying losses in the test sample gives a good idea of the probability of misclassification of our model in a new situation.

When evaluating the model on the test sample, there are two errors of interest: the probability that the model misclassifies a congestion as a link error and the probability that it misclassifies a link error as (a loss due to) a congestion. We will denote these errors respectively Err_C and Err_{LE}. Of course, it is important to minimize these two types of errors but we will show later that if one decreases the other increases. So, independently of the application of the model, it is very desirable to be able to favour the accuracy of the prediction of one type of loss over the other.

Actually, Boosting does not only provide a class prediction for each value of the inputs x but also provides an estimate of the probability of each class, C or LE, given x, i.e. two numbers $\hat{P}(C|x)$ and $\hat{P}(LE|x)$ such that $\hat{P}(C|x) + \hat{P}(LE|x) = 1$. The class given by the model is then LE if $\hat{P}(LE|x)$ is greater than a threshold P_{th} and C otherwise. By default, the value of P_{th} is fixed to 0.5 so as to treat each class fairly. However, by changing the threshold, we can easily favour the accuracy of the prediction of one class over the other. By taking P_{th} lower than 0.5, more losses will be predicted as due to link error and hence, we will decrease Err_{LE} and increase Err_C. On the opposite, by taking P_{th} greater than 0.5, we will decrease Err_C and increase Err_{LE}. So, this parameter allows us to obtain different classification models with different tradeoffs between the two types of error. It is also important to note that the user can choose the tradeoff that fits his application without re-running the learning algorithm. All he has to do is to change the value of P_{th} when making a prediction with the model.

3.4 Results

The database of 30,441 losses has been randomly divided into a learning sample of 25,000 cases and a test sample with the remaining 10,441 cases. The decision tree boosting algorithm has been run on the learning sample and tested on the test sample. For a value of $P_{th} = 0.5$, the boosting model misclassifies only 6.34% of the losses in the test sample. This result is very good considering the large diversity of the topologies in the test sample and the fact that these topologies were not seen by the learning algorithm. For comparison, Veno and Westwood misclassify respectively 34.5% and 41.5% of the losses in the test sample. So, although these two simple models can still be good at predicting loss causes in some topologies, the boosting classifier is much better in average.

We also evaluated the two types of errors Err_C and Err_{LE} on the test sample for different values of P_{th} going from 0.02 to 0.98 by step of 0.02. The left part of Figure 1 plots Err_C in function of Err_{LE} when varying P_{th}. The closest the curve to the origin the best the model. We can see clearly that the curve is not far from the origin. There is clearly a tradeoff between the two types of error. It is possible to reduce the error of one class to zero but this is always at the expense of the other. One important thing to note is that we can decrease greatly the error on the classification of LE without increasing too much the error on the detection of congestions.

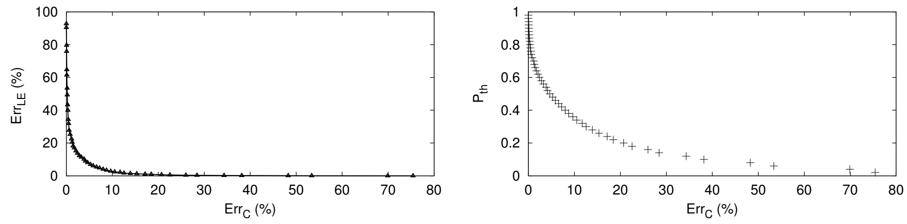

Fig. 1. At the left: The classification error obtained by Boosting. At the right: Err_C in function of P_{th}

On Figure 1, the point corresponding to TCP, which has no mechanism to distinguish loss causes, is $(Err_C, Err_{LE}) = (0, 100\%)$. The operating points corresponding to the classification rules used by Veno [7] and by Westwood [15] are respectively $(Err_C, Err_{LE}) = (54.29\%, 0.50\%)$ and $(Err_C, Err_{LE}) = (63.20\%, 4.25\%)$. These results are again much worse than what we obtain with our approach. For example, for the same value of Err_{LE} as Veno and Westwood, boosting gives respectively an Err_C of about 22% (for $P_{th} = 0.18$) and 8% (for $P_{th} = 0.4$).

In terms of computing times to make a classification, it is clear that the boosting model is more expensive than Veno's or Westwood's rule. However, computing times remain very reasonable. Assuming the inputs have been computed, one classification with boosting requires about 250 simple comparisons. To give a rough idea, in our implementation[3], the classification of the 10,441 losses in the test sample requires about 570 msec, i.e. about 55μsec per classification. So, the computational cost of the classification should not be an issue in most cases.

4 Enhancement of TCP with the loss classifier

Given a classification model for loss causes, we propose to modify TCP (Newreno) in the following way: When a loss is detected by triple duplicates, the result of its classification by the model is checked. If the cause is classified as congestion, the sender acts normally (i.e. it divides its congestion window by two), otherwise it maintains its congestion window constant.

However, we have at our disposal, not only one, but several classification models corresponding to different tradeoffs between the two types of error (by changing P_{th}). So, the question now is which value of P_{th} should be chosen with the double goal of improving TCP in wireless case and maintaining TCP-Friendliness. To ensure TCP-Friendliness, it is sufficient to maintain Err_C very close to zero. But, when Err_C is very low, Figure 1 shows that the corresponding Err_{LE} is too high to allow our model to really improve TCP in wireless case. So, the solution is to determine the lowest Err_{LE} that still preserves TCP-Friendliness. Since the two errors evolve in opposite direction to a change in

[3] The classifier is implemented in C and is run on a pentium 4 2.0 GHz

P_{th}, this is equivalent to determining the highest Err_C that preserves TCP-Friendliness. By definition, a protocol is considered TCP-Friendly if the ratio between its rate and that of a competing TCP belongs to $[1/K, K]$ with $K \leq 1.78$ [5]. So, the value of Err_C (and hence of P_{th}) should be chosen as the largest value that still fulfills this condition.

To determine the target value of Err_C, let us suppose that we run one TCP NewReno, referred to simply as TCP in the sequel, and one TCP NewReno equipped with our Boosting classifier on the same network path[4] and that both flows lose a proportion p of their packets. According to Padhye et al. [14], the throughput of the TCP flow is equal to:

$$B_{tcp} = \frac{1}{RTT\sqrt{\frac{2p}{3}} + T_0 min(1, 3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

with the assumption that there is no delayed acknowledgment ($b = 1$).[5]

The TCP NewReno equipped with our Boosting classifier is a normal TCP except that it reacts only to a proportion $p(1 - Err_C)$ of packet losses instead of p. Its throughput over the network path is then equal to:

$$B_C = \frac{1}{RTT\sqrt{\frac{2pY}{3}} + T_0 min(1, 3\sqrt{\frac{3pY}{8}} pY(1+32p^2Y^2)},$$

where $Y = 1 - Err_C$. So, using these equations, it is possible to compute, for a given RTT and loss rate p, the largest value of Err_C such that $\frac{B_C}{B_{tcp}} < K$. However, since RTT and p are changing with time, instead of choosing a fixed value of P_{th}, we propose to dynamically adapt the value of P_{th} to the current values of RTT and p.

To this end, after each loss, we compute the loss rate p obtained over the whole session. Then, from this estimation and the RTT, we compute the highest value of Err_C such that $B_C/B_{tcp} < K$. Once the bound on Err_C is found, we retrieve the value of P_{th} that provides an error on congestion lower than this bound. The correspondence between P_{th} and Err_C is obtained on our test set and is illustrated in the right part of Figure 1.

Since P_{th} is adapted dynamically after each loss, from now on, we refer to TCP NewReno equipped by this classifier as *"Boosting-adapt"*.

In all our experiments below, we have used a value of K equal to 1.15 instead of the standard value of 1.78. There are two reasons for that. The first one is that we consider that 1.78 is too high; a smaller value will provide better friendliness. The second reason is that we prefer a more conservative choice of K in our case. Indeed, the value of Err_C for a given P_{th} in Figure 1 is only an estimate from the test sample of the true value. Hence, for a given situation, the choosen value of P_{th} can, in practice, lead to a higher Err_C than expected. By adopting a lower K, we minimize the impact of such situation.

[4] We focus here only on TCP-Friendliness in the wired case.

[5] Using $b = 1$ in the formula was recommended in RFC-3448.

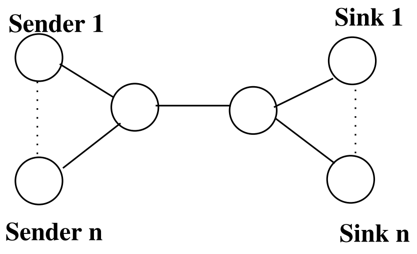

Fig. 2. Topology 1

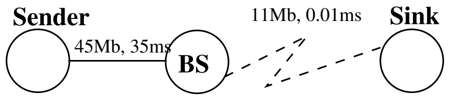

Fig. 3. Topology 2

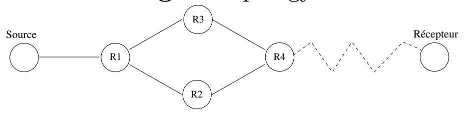

Fig. 4. Topology 3

5 Simulations with the learned models

In this section, we evaluate Boosting-adapt in wireless and wired networks. The experiments in the case of wireless networks show the gain we can obtain in this kind of networks where most of the losses are due to link errors. The experiments related to wired networks concern the TCP-Friendliness and the bandwidth usage.

For comparison, we test also the classification rule used by Veno. We have chosen Veno because both its errors are lower, than the ones of Westwood. All the experiments have been done with `ns-2`.

5.1 The improvement in lossy links

We test the topology used in [15] and illustrated in Figure 3. It represents a hybrid network. The first part, Sender-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 Boosting-adapt and the one obtained by TCP when we vary the packet loss rate from 0 to 5% over the wireless link. Each simulation is run 50 times. To have a good point of comparison, we run also simulations with an artificial TCP that classifies perfectly the cause of loss detected by triple duplicates. The graph at Figure 5 illustrates the ratio obtained by TCP with the three classifiers, the perfect one, Veno, and Boosting-adapt. Boosting-adapt is much better than Veno and also very close to the perfect model. Its gain with respect to TCP is not far from 300% when the loss rate is equal to 3%.

5.2 TCP-Friendliness and link capacity usage

Wired network In the previous section we have tested Boosting-adapt over a wireless link, and we have compared the gain it obtained with those of TCP-Perfect and Veno.

In this section we compare its fairness towards TCP in the wired case, which is an important criterion that should be fulfilled by the classification model. To

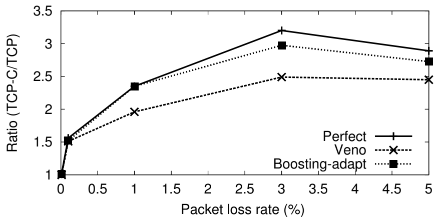

Fig. 5. The gain in lossy links

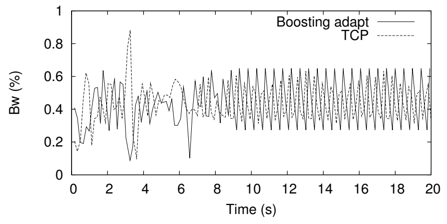

Fig. 6. TCP-Friendliness in wired case

Table 1. The bandwidth usage

	Newreno	Veno	Boosting-adapt
Throughput(%)	95.06	93.90	97.82
Goodput (%)	92.50	86.96	93.18

test TCP-Friendliness, we use the topology illustrated in Figure 2 with $n = 2$, often used to test the fairness. The experiment consists of running a Newreno in competition with Boosting-adapt. Figure 6 illustrates the throughput obtained by each flow and shows that the share is fair.

For comparison, we have used Veno in the same scenario and the share ratio was slightly higher than five. This is not surprising since Veno is very bad at detecting congestion loss (its Err_C is high) and hence, it reduces its bandwidth less often than TCP.

Link capacity usage Too much misclassifications of congestion losses can also lead to the underuse of the link when several TCPs equipped with a loss classifier compete over the same bottleneck. Indeed, if the modified TCPs do not react to packet losses due to a real congestion, then the congestion will actually worsen. Thus, instead of reacting to triple duplicates as TCP would, the modified TCPs would wait for the timeout expiration and thus get in average less throughput than a normal TCP. To show that Boosting-adapt does not suffer from link underuse, we have used an aggregate of 4 similar flows, competing over one link (topology of Figure 2 with $n = 4$) and have computed their throughput and goodput. For comparison, we have also run an aggregation of TCP equipped by Veno in the same situation. The results are given in Table 1. We can see that the throughput and the goodput of Boosting-adapt exceed those of Newreno while those of Veno are much lower.

Route change and network reordering Route changes and failures have not been taken into account in the generation of the database, but we think that they will not affect the robustness of our approach. Indeed, our model has a memory of 2 RTTs (for the computation of the inputs). Thus, in the worst case, the classifier will misclassify all wireless losses happening during the two RTTs following the route change (just like a standard TCP). After this transition,

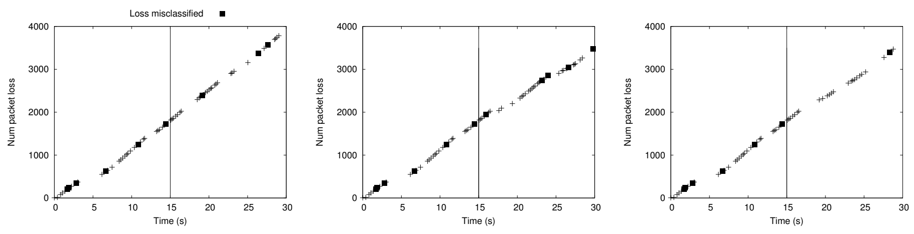

Fig. 7. Losses classification. From left to right, top to bottom: shorter, equal, and longer paths.

the path will become "stable" again and the classifier will retrieve its ability of discriminating loss causes. For the same reason, network reordering was not taken into account during the generation of the database. In addition, reordering seems to be rare in actual networks [10].

To confirm our statements, we use the topology of Figure 4 where Boosting-adapt is used between the sender and the sink and where the link R2-R4 breaks down after 15 seconds. We study the cases where the new path is longer, shorter and of the same length as the old one.

The graphs in Figure 7 show the packets lost in the three cases with the misclassified losses represented by a square. In the three cases, the discrimination quality is not deteriorated after the route change.

In the case of a shorter new path, we have observed a case of reordering. Some packets arrive to the sink before their predecessors and they lead to triple duplicates at the sender side. The sender uses the classifier and concludes that the "loss" is not due to a congestion and then does not decrease its rate.

6 Conclusion

In this paper, we have applied a supervised learning algorithm, called decision tree boosting, to automatically infer a loss cause classifier from a database of losses observed in a large number of random topologies. The resulting classifier has shown very good accuracy at classifying losses in random topologies that were not seen by the learning algorithm. Then, we have proposed to use this loss classifier to improve the performance of TCP in wired/wireless networks. To this end, we have shown how to adapt dynamically the classifier to ensure TCP friendliness. The new protocol, called Boosting-adapt, has shown a very good behaviour in wireless networks. It offered a high gain in throughput over wireless links and, at the same time, it preserved TCP-Friendliness in all topologies it has been tested over.

We see two potential limitations to our approach. First, we have not taken into account losses detected by timeout expiration, which were thus all considered as signs of congestion. However, even without classifying such losses, the throughput gains observed in our simulations are already excellent and we can only improve these results by taking into account time out expiration when designing the loss classifier.

A second potential limitation is that our database was generated from simulated networks (topologies and traffics) which may differ to a certain extent from actual ones. However, the learning sample was generated by randomizing all networks conditions. Hence, there is no bias in the classifier. We have also carried out experiments with a more realistic topology generator (BRITE [13]) that have not shown significant differences in terms of performance of the classifiers with respect to the results presented in this paper. Furthermore, restraining the learning to actual topologies and flows can only improve the accuracy of the classification.

References

1. A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for Mobile Hosts. *15th International Conference on Distributed Computing Systems*, 1995.
2. S. Biaz and N. H. Vaidya. Distinguishing Congestion Losses from Wireless Transmission Losses: A Negative Result. *Proc. IC3N, New Orleans*, 1998.
3. L. Breiman, J.H. Friedman, R.A. Olsen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth International (California), 1984.
4. S. Floyd. A report on some recent developments in TCP congestion control. *IEEE Communication Magazine*, 39(84-90), April 2001.
5. S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM 2000*, pages 43–56, 2000.
6. Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proc. of the 2th European Conference on Computational Learning Theory*, pages 23–27, 1995.
7. C. P. Fu and Soung C. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE (JSAC)*, February 2003.
8. A. Gurtov and S. Floyd. Modeling wireless links for transport protocols. *SIGCOMM Comput. Commun. Rev.*, 34(2):85–96, 2004.
9. T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction.* Springer, 2001.
10. S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone. In *Proc. of IEEE INFOCOM*, Mar 2003.
11. I. EL Khayat, P. Geurts, and G. Leduc. http://www.run.montefiore.ulg.ac.be/ ˜elkhayat/ Boosting-DT/, 2004.
12. J. Liu, I. Matta, and M. Crovella. End-to-End Inference of Loss Nature in a Hybrid Wired/Wireless Environment. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.
13. A. Medina, I. Matta, and J. Byers. BRITE: A Flexible Generator of Internet Topologies. Technical report, 2000.
14. J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Reno performance: a simple model and its empirical validation. *IEEE/ACM Transactions on Networking*, 8(2):133–145, 2000.
15. R. Wang, M. Valla, M.Y. Sanadidi, B.K.F Ng, and M. Gerla. Efficiency/Friendliness Tradeoffs in TCP Westwood. In *In Proc. of the 7th IEEE Symposium on Computers and Communications, Taormina, July 2002.*