Machine Learning for Scalable Path Latency Monitoring in Overlay Networks

Martín Randall*
*Facultad de Ingeniería, Universidad de la República, Uruguay

***Abstract*—We consider a routing overlay in which the delay of a path can be obtained at some fixed cost by sending probe packets, and investigate the joint minimization of the probing cost and the routing delay. We use a Random Forest approach to estimate the best route to take at each time-slot while fixing periodical measurements, and compare its results to model-based approaches.**

I. A Random Forest approach to network monitoring

Network monitoring is an open problem for overlay networks. Since on the overlay we can only measure end-to-end delays, it is needed to send probe packets through every path to have full knowledge of the system's status. This can be seen as a possible cause for congestion, and as such we will consider there is a measuring cost to be taken into account. But what if by sending just enough probe packets we could have a good estimate of the system's delays? In this work we use Random Forests to estimate the system's delays (round time-trip, RTT) and to minimize measurements.

Other approaches use a hidden markovian chain modelling of the path delay, which results in closed form solutions by using montecarlo methods. As the number of possible states grows, these solutions quickly encounter the curse of dimensionality. It has been shown that approximations of such solutions, as receding horizon algorithms, can scalate better by avoiding (or delaying) the curse of dimensionality, we hereby present the exploration and comparison of other machine learning techniques known to be usefull when working with RTT estimation [1] [2].

Random Forests are machine learning algorithms, more precisely of supervised learning (we know the output when training the algorithm). It is a particular kind of bagging, where we use many tress built with random features, obtaining de-correlated trees, which helps to lower the variance of our prediction. As a quick reminder, trees are usually formed by dividing the feature space by binary splitting. Bagging techniques, such as Random Forest, allow us to use the simplicity and power of the trees while diminishing their high variance (due to their simplicity: a different first split will probably result in a completely different tree). Random Forest algorithms are very popular on network optimization [3], as they are scalable and robust, although they can be difficult to interpret.

We will consider a simple scenario where every measure is done simultaneously on all paths, every T time-slots. Let's denote by x_i the i-eth route from the N possible routes as $x_i \in \{x_1, x_2, \cdots, x_N\}$. We call $x_{i,t}$ the RTT value for path i at time-slot t. We will define our system as the last delay measured, and the time-slots transpired from the measure. The ouput will be y_t the path to choose that minimizes the delay at time-slot t. We will have then the training samples written as:

$$\begin{gathered}(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, 0, y_0)\\(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, 1, y_1)\\(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, 2, y_2)\\\vdots\\(x_{1,0}, x_{2,0}, \cdots, x_{N,0}, T-1, y_{T-1})\end{gathered}$$

when measuring in the first time-slot. As we move to the scenario measuring at the second time-slot, we get a new set of T training samples:

$$\begin{gathered}(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, 0, y_1)\\(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, 1, y_2)\\(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, 2, y_3)\\\vdots\\(x_{1,1}, x_{2,1}, \cdots, x_{N,1}, T-1, y_T)\end{gathered}$$

This is a simple setup, in which we use strong restrictions: both measuring all paths and doing so periodically. As a first approximation to supervised learning algorithms, a simple setup may provide us with good solutions, and justify a deeper approach. The algorithm is presented in 1. In this article we center in selecting the best T* amongst a set of possible T values, and comparing the results with the exact solution (when possible by policy iteration) and with a receding horizon approximation.

We will briefly describe the algorithm's settings. We chose a 10-tree random forest, as we didn't see a result improvement for a larger number of trees. We used no minimum/maximum number of leaves, and the gini criterion for impurity, as well as bootstrap samples for building each tree. This is a very simple usage of the **scikit** python library [4].

We will compare to the receding horizon solution proposed by [2], which is a finite approximation of the value policy based on the receding approach stated in [5]. After presenting

Algorithm 1 Random Forest

```
Divide training samples into training and validation
for $T \in T_i$, using cross-validation do
    Train Random Forest predictor
    Evaluate expected reward $\hat{D}[T_i]$ for validation set
end for
for $T_i = \operatorname{argmin} \hat{D}[T_i]$ do
    Apply random predictor $RF[T_i]$ to test samples.
end for
Find Expected Reward on test samples for routing decisions.
=0
```

results, a flexibilization of the "same T for all routes" restriction is proposed on the algorithm.

To test the algorithm, we used on one hand synthetic scenarios where the paths are markovian built, and as such we can compute the exact solution. On the other hand, we used the RIPE-ATLAS dataset [6], which provides us with 72 different overlay configurations with each route measured every 2 minutes for a week. The overlay settings go from 2 possible paths to 7 different paths. This results in 3836 RTT samples for each path. This has proven to be a challenge, since it is not a great amount of samples, but has also allowed us to use the algorithm on real traces, in particular when the markovian analysis is non-realistic. This synthetic and real scenarios have been used in previous work [1] [2].

II. Results

A. Synthetic scenarios

We first use Random Forest to estimate the best route in a 2 paths scenario where each path has 2 possible RTT values, generated as a markovian process. This example is used in [1], and its values are result of the path Hong Kong - Kazakhstan, and Hong Kong - Latvia - Kazakhstan, and can be observed in table I. The transicion matrices are for each path:

$$P_1 = \begin{pmatrix} 0.995 & 0.005 \\ 0.05 & 0.95 \end{pmatrix}, \; P_2 = \begin{pmatrix} 0.995 & 0.005 \\ 0.005 & 0.995 \end{pmatrix}.$$

	RTT		Cost
	Path 1 [low, high]	Path 2 [low, high]	[Path 1, Path 2]
Scenario 1	[350, 440]	[320, 370]	[4, 4]
Scenario 2	[1, 3]	[0.5, 2]	[0.15, 0.05]
Rayleigh scenario	[350, 440]	[320, 370]	[4, 4]

TABLE I: Settings for the synthetic scenarios.

In this setup, we can find the optimal solution to our measuring/routing problem, by using dynamic programming. For that purpose we use the common policy iteration algorithm as described in [7]. We also compute the results of our Receding Horizon algorithm (with an horizon of 3), and the result of the Random Forest algorithm (10 tree forest). The T selection is done by using 5-fold cross-validation of an expected reward. We consider the expected reward $\hat{D}$ of policy π to be defined as follows:

$$\hat{D}_\pi = \Sigma_{i=0}^{k-1}([c + RTT] * \rho^k)$$

Results are shown in table II and figure 1.

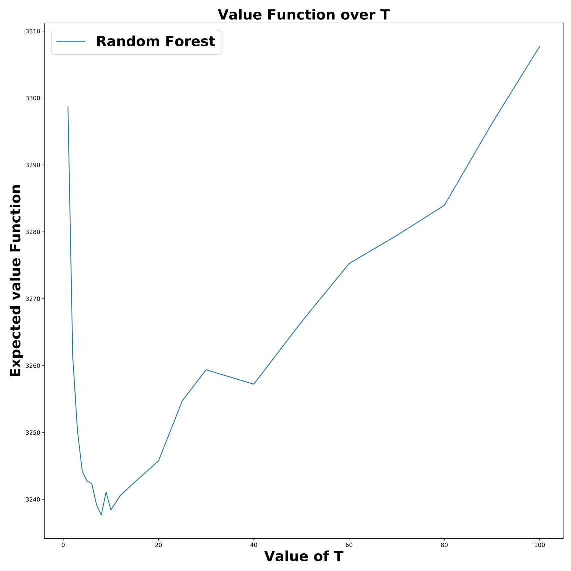

Fig. 1: Synthetic scenario number 1, based on paths from RIPE-ATLAS but processed as markov delays. Expected value for different T possible values. Minimum is reached for $T = 8$.

We then try another setup with a more variable route and a more stable one. This MDP's settings also comes from [1]. Results are shown in table II and figure 2. The transicion matrices are for each path:

$$P_1 = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}, \; P_2 = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}.$$

For both these settings the Random Forest performs really close to the dynamic programming algorithms. It's interesting to remark that the T-selection has similar behaviour: the optimal T is such that the cost isn't to high (with a low T the cost can have a huge impact on the expected reward), and the path measurement is enough to take close to optimal routing decisions. With a low-cost setting, a lower T will be always a preferred choice. At the same time, a very changing path will demand a smaller value of T. Usually there is a value of T from which the Expected Reward is stable, since the measuring cost has little impact on the reward estimation, and the knowledge on paths isn't enough to make great routing decisions (the algorithm will rather choose one path and stick to it). This is particularly clear in figure 2.

We then try our algorithms in a 2 routes 2 levels situation, but now the routes don't follow a markovian process, but

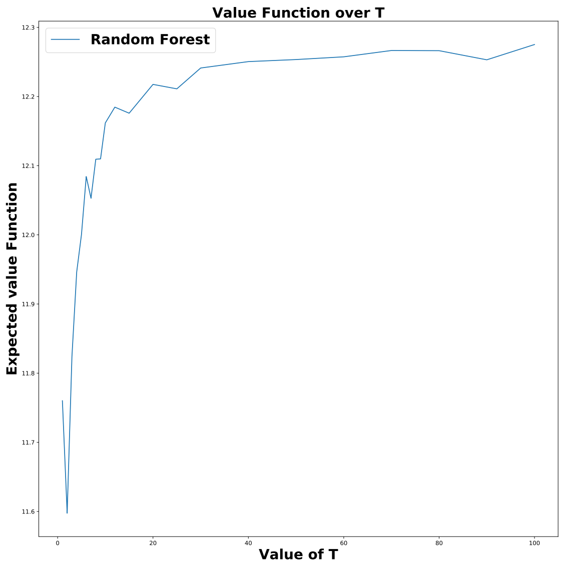

Fig. 2: Synthetic scenario number 2. Expected value for different T possible values. Minimum is achieved for $T = 2$.

are constructed using Rayleigh's distribution. In this scenario, all algorithms agree on using a low-measuring policy, and just selecting which appears to be the best path. The Policy Iteration and Receding Horizon don't even measure once, and the T selection for the Random Forest shows that the larger the T, the better, which means that we do not need to measure often to make a path choice. In overall, even if selecting a "shortest path", the total cost (once added the measuring cost) is larger for the Random Forest, but they all are really close one to another.

	Expected Reward			Total Delay + Cost (in e3 ms)			T
	PI	RH	RF	PI	RH	RF	RF
Scenario 1	3197.7	3201.7	3200.6	16.8	16.8	16.7	8
Scenario 2	10.71	10.75	11.27	5.26	5.29	5.89	2
Rayleigh scenario	3291	3291	3292	687.1	687.1	687.4	25

TABLE II: Summary of the results achieved for the 3 compared algorithms .

B. *Real traces*

Finally, we tried our Random Forest for some of the real traces used in [2]. Results are summarized in table III and figures 4 and 5 . For both settings, the Random Forest has proven comparable to the Receding Horizon algorithm. In one case, performing better than the receding horizon, and at a much lower processing time/cost, as seen in table III. The T selection for the Haifa-Santiago selection was done using the simple (T, T) algorithm, and $T = 4$. For the Paris-Tokyo scenario, the T search modification was used, and the result is a $(T_1, T_2) = (13, 15)$. The expected reward for combinations around $(14, 14)$ is shown in figure 3, relative to the minimum expected reward (just for visualization).

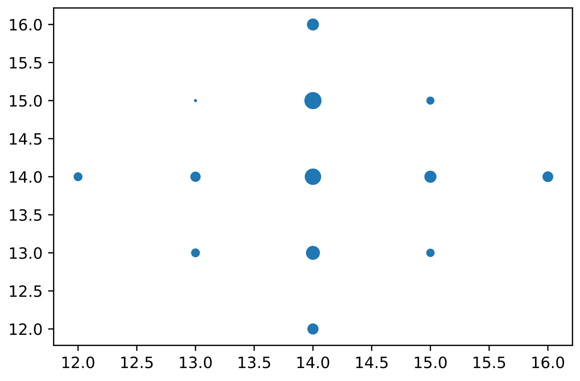

Fig. 3: Expected rewards for Random Forest algorithm trained with different (T_1, T_2) values around $(14, 14)$. For visualization, expected rewards are plotted in relation to the minimum achieved, in this case, for $(13, 15)$.

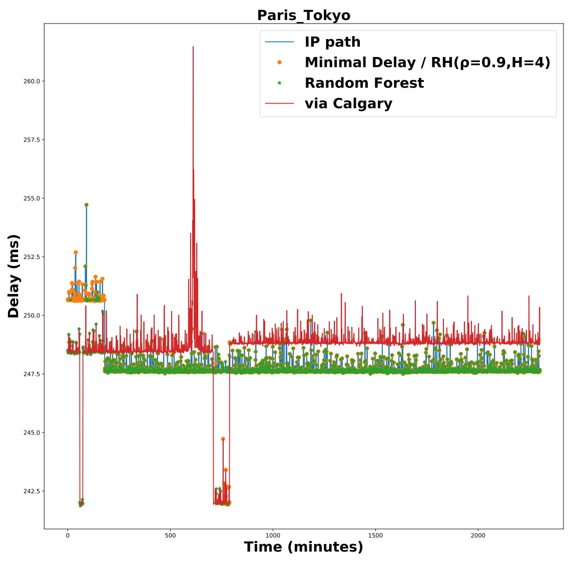

Fig. 4: Paris-Tokyo delay, with the RF and RH (H=4) routing-decisions. The average delay of the IP static path is 247.97 ms, the one achieved with the RH is of 247.89 ms and the best path selection is performed by the RF, with an average delay of 247.75 ms.

O-D pair	Avg number of measures		Avg Processing time (ms)		Gap to min Delay	
	RH	RF	RH	RF	RH	RF
P-T	0.026	0.072	180	1.96	13.4e-4	7.7e-4
H-S	0.124	0.250	8.42	1.71	0.002	0.019

TABLE III: Summary of the results achieved for the considered OD pairs, where RH stands for receding horizon, RF for random forest. Averages are calculated per time-slot. We used an horizon of H=4 for the receding policy for the Paris-Tokyo problem, and of H=1 for the Haifa-Santiago scenario (myopic policy).

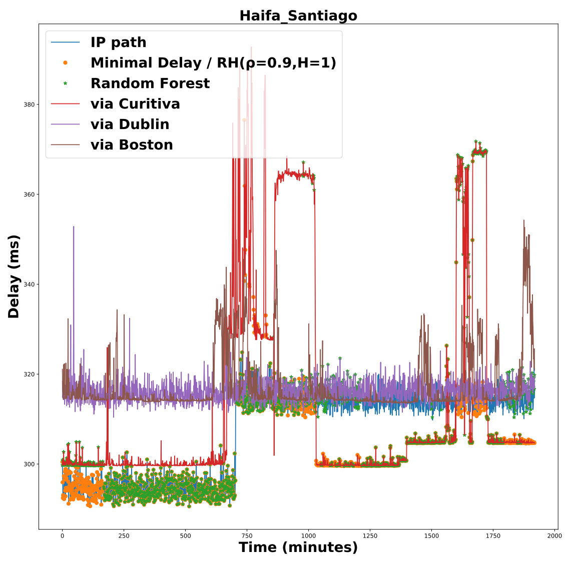

Fig. 5: Haifa-Santiago delay, with the RF and RH routing-decisions. In this case, as there are four possible paths, we compared with the myopic policy, which means an horizon of 1 time-step. The average delay of the IP static path is 307 ms, the one achieved with the RH is of 303 ms and the RF got an average delay of 308 ms.

An interesting observation is that as the number of paths grows, the Random Forest algorithm seems to perform more poorly. This may be due to the need of a larger training set in order to have an accurate prediction. Also, there is a noticeable increase in processing time, even if it is much lower than the processing time of the Receding Horizon algorithm, and even the myopic policy, which can be viewed as a Receding Horizon of horizon 1.

C. T search modification

In order to find a better subset of values for when to measure, and mainly, to dissociate measuring in all paths at the same time, we propose a modification. Let's assume that we are only working on a 2 paths situation. We are selecting the best (T, T) for a periodical measurement of both paths. An improvement could be to do a T selection for any possible (T_1, T_2), where T_i is the T-value for which we periodically measure route i. This search can grow as the number of paths grow, and we will fall back into the curse of dimensionality. A simpler approach is to only look for a better (T_1, T_2) in a small interval around the selected (T, T). This can enable one of the paths to be measured more often than the other, but not by a large difference.

We tried this modification, attaining very similar results to the original algorithm. In many cases, a different couple of T-values is preferred, but the results end up being practically the same. A best-case scenario for this application would be a very stable path and a very changing one, so to have a lower T-value for the fast changing path, and a larger value for the stable one. Still, the first selection of a (T, T) couple may be already restrictive of the possibilities of such an improvement. We mean that after selecting the T-value, we already used a compromise between paths, and an interval around that T-value can be non-optimally for both paths.

We depict the (T_1, T_2) selection for the real trace Paris-Tokyo. Selection offers a higher expected reward, but does not imply in a diminished measuring algorithm, since for one path we increase measurements while lowering this value for the other.

Another idea, one that we must look into, is to find somehow a relationship between T_i and some statistical of our paths. The goal would be to use to our advantage the knowledge of path's variations, looking for a higher T for more stable routes, and for a smaller T for the more variable ones. This has not been explored further, and we fear that we might fall back in an analysis of the expected reward over T, which can take us back to a markovian framework.

III. Conclusion

As a first approach to a supervised machine learning algorithm for our measure+routing problem, the Random Forest algorithm has proven to be more than capable to compare in results to state-of-the-art algorithms. Even though this is a very simple setup, where little features are at play and strong restrictions are imposed (same periodical T for all paths), the algorithm has proven its worth both in results and simplicity, and hence, on scalability, which was a constraint for previous studied solutions [2].

An important disclaimer to be made is that in fact we are comparing 2 different optimization problems. In one (receding horizon), we optimize the expected reward, including cost of measures, and future routing decisions. In the random forest, the routing decision is built on a previous state of the routes and the lag of time from that measurement. This means that the cost is nowhere to be found, and neither is the future taken into account. This may point a future work direction, on a better formulation of the random forest goals and features. Actually, in the T_i selection, the expected reward is minimized, but not in the random forest training.

A strong relationship exists between the value of T, the interval between measurements in the Random Forest algorithm and the cost c of measuring the paths. The costs could be different for different paths, the same for all paths, etc. In the random forest scenario, as we measure all routes it's the sum of all costs that interests us. This direct bond fixes the number of measures, for both algorithms considered. Depending on the value of the cost one algorithm may yield better results or the other.

Finally, the proposed solution does not imply a markovian analysis. It could be that the receding horizon algorithm is more suited to markovian paths, while a random forest approach would yield better results for non-markovian traces. This points another line of work, in finding more scalable solutions amongst machine learning algorithms, or exploring (and pushing) the limits of the ones at hand.

References

[1] S. Vaton, O. Brun, M. Mouchet, P. Belzarena, I. Amigo, B. Prabhu, and T. Chonavel, "Joint Minimization of Monitoring Cost and Delay in Overlay Networks: Optimal Policies with a Markovian Approach," *Journal of Network and Systems Management*, vol. 27, no. 1, pp. 188–232, Jan. 2019.

[2] M. Mouchet, M. Randall, M. Ségneré, I. Amigo, P. Belzarena, O. Brun, B. Prabhu, and S. Vaton, "Scalable monitoring heuristics for improving network latency," in *IEEE/IFIP Network Operations and Management Symposium*, 2020.

[3] S. Wassermann, P. Casas, T. Cuvelier, and B. Donnet, "Netperftrace: Predicting internet path dynamics and performance with machine learning," in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 2017, pp. 31–36.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] G. C. Goodwin, M. M. Seron, and J. A. D. Doná, *Constrained Control and Estimation. An Optimisation Approach*. Springer-Verlag London, 2005.

[6] "RIPE Atlas," https://atlas.ripe.net/, accessed: 2019-11-26.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning, An Introduction*, second edition ed. The MIT Press, 2018.