Random Neural Networks for the Adaptive Control of Packet Networks

Michael Gellman and Peixiang Liu

Dept. of Electrical & Electronic Eng.,
Imperial College London
{m.gellman, p.liu}@imperial.ac.uk*

Abstract. The Random Neural Network (RNN) has been used in a wide variety of applications, including image compression, texture generation, pattern recognition, and so on. Our work focuses on the use of the RNN as a routing *decision maker* which uses Reinforcement Learning (RL) techniques to explore a search space (*i.e.* the set of all possible routes) to find the optimal route in terms of the Quality of Service metrics that are most important to the underlying traffic. We have termed this algorithm as the Cognitive Packet Network (CPN), and have shown in previous works its application to a variety of network domains. In this paper, we present a set of experiments which demonstrate how CPN performs in a realistic environment compared to *a priori*-computed optimal routes. We show that RNN with RL can autonomously *learn* the best route in the network simply through exploration in a very short time-frame. We also demonstrate the quickness with which our algorithm is able to adapt to a disruption along its current route, switching to the new optimal route in the network. These results serve as strong evidence for the benefits of the RNN Reinforcement Learning algorithm which we employ.

1 Introduction

The Cognitive Packet Network (CPN) approach has been used for routing in a number of different domains: from wire-line, to wireless Ad-hoc, and even overlay networks [1,2,3]. These have demonstrated that it is able to use the QoS goal of the user or application to selectively find the best path in the network that satisfies it.

At the core of the CPN algorithm is the Reinforcement Learning (RL) algorithm which uses a fully recurrent Random Neural Network (RNN) [4] as a decision-maker to route packets. Each output link from a CPN node (router) is represented by a neuron in the RNN, the weights of which are updated according to the quality of the routes explored by the Smart Packets (SPs) or used by the Dumb Packets (DPs). Therefore, the RL algorithm uses the observed outcome

* This work was supported by the UK Engineering and Physical Sciences Research Council under Grant GR/S52360/01.

S. Kollias et al. (Eds.): ICANN 2006, Part I, LNCS 4131, pp. 313–320, 2006.

of the current decision to either *reward* or *punish* the corresponding routing decision so that its future decisions are more likely meet the user's desired QoS goal.

In this paper, We begin by giving a brief overview of the mechanisms behind CPN, including one optimization which improves the Quality of Service given to data traffic. Following this, we describe our testbed and procedure that we have used to conduct our experiments. Finally, we present our results which demonstrate CPN's ability to *learn* the optimal route in a network, and also its ability to quickly react to a harmful change in that route.

2 CPN Overview

The CPN algorithm has already been described in detail in previous publications (*e.g.* [5]); thus, in this section, we provide a brief description of the mechanics of CPN, and focus more in-depth on an optimization that we have introduced in our experiments to improve the QoS of user traffic.

CPN is a *reactive* routing protocol – that is, it waits until a connection requests the establishment of a route to begin the routing process. For this purpose, there exists a division of labor between packets whose job it is to carry data (so-called *Dumb* packets or DPs), and those which are responsible for exploring the network and learning the best path (*Smart* packets or SPs).

Smart packets are sent at regular intervals during the connection, and are routed using a Reinforcement Learning (RL) algorithm at each hop in the network. At each router they visit, they may have some QoS information added into the packet, depending on the QoS metrics of the connection. For instance, when delay is the metric, we encode the current timestamp into the packet, in order to calculate a round-trip time estimate. When an SP reaches its destination, an acknowledgement packet (SAck) is generated which will travel along the reverse route of its SP, performing the RL algorithm at each hop to indicate the success or failure of the original SP in finding an optimal route.

In contrast to SPs, Dumb packets are source-routed using the routes brought back by SAcks. When they reach the destination, they also generate an acknowledgement (DAck) which also updates the RNN weights through the RL algorithm. This ensures that the reward information stored at each node is kept up-to-date. In the original algorithm, DPs would use the route brought back by the most recent SAck; however, in our experiments, we found that this was not always optimal, and thus we propose an optimization which we present in the following subsection.

2.1 Keeping Track of the Best Route

In the originally proposed algorithm, we proposed to use the route brought back by a SAck immediately, under the assumption that it contains the most

up-to-date information about the network, and would be the best route decided upon by the network. However, this does not take into account the fact that, in order to better explore the network, each router will (with a small probability) sometimes choose a random next hop for an SP, rather than the one decided upon by the RNN. This decision may or may not result in better QoS for the route, and thus we propose to compare the quality of routes brought back by a SAck against the current route before switching.

When a SAck arrives at its destination, we compare the reward for this route to the currently active one. Only if the newly arrived SAck's reward is greater than our current route do we switch. DAcks update the reward stored with their route, but do not cause the active route to change. In the remainder of this paper we refer to this optimization as *CPN with Sorting*.

While at first this may seem to deprive us of some of the exploratory benefits of the CPN algorithm, we contend (and demonstrate in Section 3.3) that this is not the case because of DAcks. Because each DAck also triggers the RNN-RL algorithm at each router, a sudden decrease in quality along a route will punish the decision, and cause the next SP to choose an alternate next hop, which will enhance of discovering a new best route; thus, the network is still able to quickly respond to changes in network conditions, and it also ensures that data traffic always uses the best available path.

3 Experiments

We wanted to demonstrate CPN's ability to learn the optimal route in a network, as well as explore its performance for different levels of exploration overhead, and under network dynamics. Thus, we have used our network testbed facilities at Imperial College London to perform controlled experiments into each of these areas.

3.1 Testbed

Our network testbed consists of standard PCs which run Linux 2.6.15 and contain one or more quad-10/100 Ethernet cards. An important consideration for our experiments is the topology that we use; we would like it to be as realistic as possible so that we can generalize our results to other, larger networks. Towards that end, we contacted the Swiss Education and Research Network, who provided us details on their 46-router backbone, complete with bandwidth, OSPF costs, and link-level delays (Fig. 1). We have implemented this topology in its entirety in our lab[1]. We have configured IP routing using quagga 0.99.3 with the OSPF costs given to us; thus, the routes in our testbed should be exactly the same as those used in the real Swiss backbone.

[1] With the following differences: we have scaled the bandwidth down by a factor of 100, and have used artificial delays to replicate the physical layer delays in the data set (from 0-40ms).

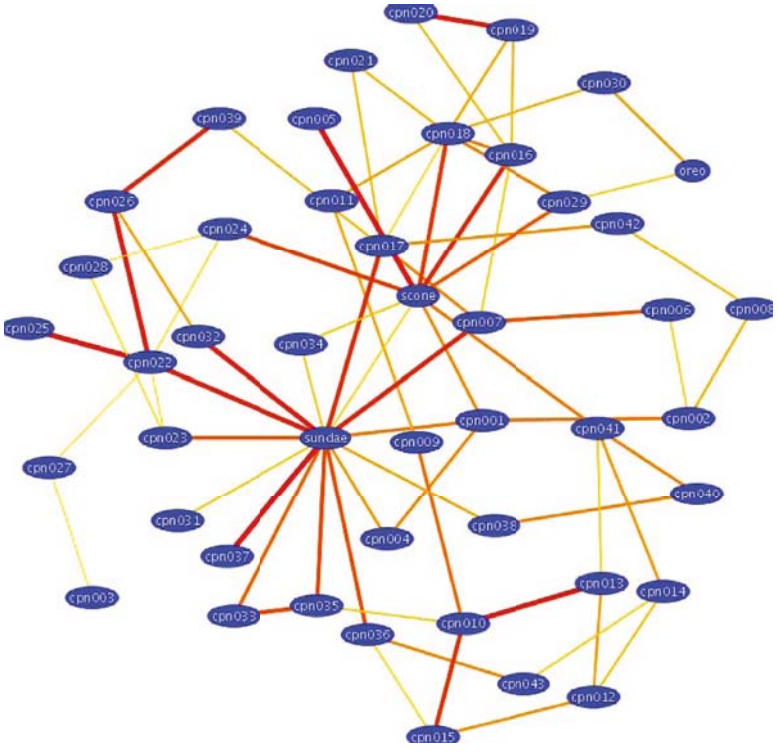

Fig. 1. The 46-node testbed topology. Links between nodes are colored and emphasized in terms of the delay of the link; lighter thinner lines are low-delay links, while darker, thicker ones denote higher delays.

3.2 Scenario and Implementation

As a traffic source, our experiments use the open-source Quake II multi-player game, which uses UDP as its transport protocol. Due to its open-source nature, we were able to modify the application to better suit it to network experimentation, including: removing all video, sound, and input code to allow clients to run without user input; and embedding *bot* code into the server so that our generated traffic resembles that of a real game complete with movement, player kills, and so forth. Because delay is critical to the performance of a first-person shooter, we configure CPN to optimize this metric, and also use it to compare our results.

Our experiments use a single server (cpn008 in the figure), and 10 clients. The delay results that we report are an average of the application-level *ping* results reported by the Quake II server, where each experiment lasts 15 minutes. Each experiment was repeated 20 times, and, unless otherwise noted, had minimal variance.

CPN is implemented as a Linux kernel module that runs at the networking layer. It receives Quake II packets through an IP-over-CPN tunnelling mechanism that hides from the application that it is using CPN; thus, no re-writing of the application is necessary. The only downside of this approach is that there is a small amount of additional packet overhead (an extra IP header), although, as we show in Section 3.3, this is minimal.

3.3 Results

In this section, we present the results from three different sets of experiments that we conducted on our testbed. The first set compares the performance of CPN with the "sorting" optimization that we described in section 2.1. Following this experiment, we investigated the impact of varying the amount of exploration overhead on CPN's ability to quickly discover the optimal path in the network. Our final set of experiments demonstrates the quickness with which CPN is able to react to a performance degradation along its optimal route.

Sorting vs No Sorting. When we first experimented with CPN's ability to find the optimal route in the network, we noticed that it would sometimes route Dumb packet traffic along non-optimal paths for short periods of time before returning to the best path (Fig. 2). Upon closer investigation, we discovered that this was due to Smart packets who, with a small probability[2] at each hop, would randomly choose a next as opposed to using the RNN-RL algorithm. While this is an integral component of the CPN algorithm because it enhances our ability to discover new paths, it was also harming the performance of data traffic that would use these new routes without considering how they compared to their current route. Thus, we implemented the optimization described in section 2.1.

The outcome of the optimization is shown in Fig. 2, where we can see it leads to a significant improvement. Once CPN has discovered the optimal route in the early stages of the experiment, it continues to use it whereas without the optimization, CPN subjects the data traffic to a number of other routes with worse delay performance. Thus, throughout the remainder of this paper, all experiments take place with this optimization enabled.

Discovering Optimal Routes. In order to demonstrate CPN's ability to quickly find the optimal route in the network, we compared its performance with that of IP using OSPF to determine its routes. Because the cost of each link is proportional to its delay, OSPF routing converges to the minimal delay path, giving us a baseline for comparison.

We conducted experiments (according to the procedure above) for different percentages of Smart packets[3], which controls the overhead of network exploration. The results can be seen in Fig. 3.

[2] In our experiments, the probability of randomly choosing a next hop is 0.05.

[3] While we refer to it as a percentage, it is more accurately described as the *probability* that, when sending a DP, that we also send an SP.

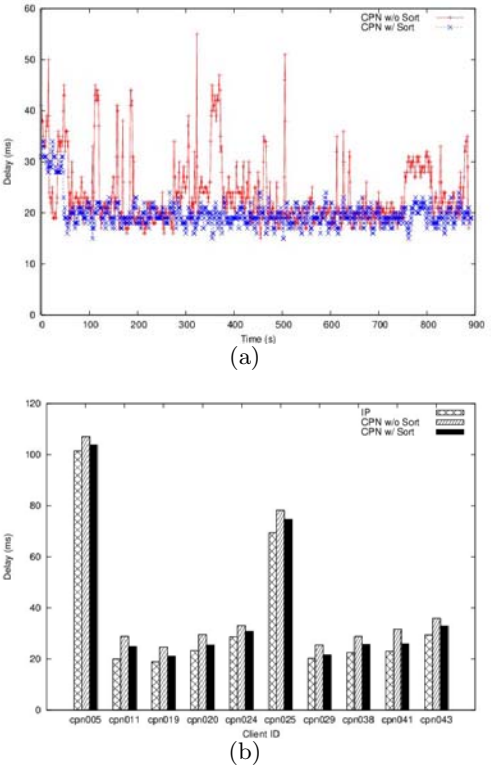

Fig. 2. Comparing the effect of sorting. In (a), where we have plotted the measured delay during the experiment for one selected client, we can see the variance in the non-sorting case is much higher, as some of the Dumb packets use a route with higher delay before switching back to the optimal one. The results for every client are summarized in (b), where we see that the average delay is lower when using the sorting optimization.

Looking at these results, we can see that CPN can always find the optimal route in the network; it just takes a longer time when the exploration overhead is low (Fig. 3.b). We can also see that the difference in delay performance between 10% and 20% SPs is minimal for many of the clients, leading us to decide to use the former for the remainder of the paper.

CPN's Response to Network Dynamics. In our final experiment, we wanted to see how quickly CPN could adapt to a sudden change along the optimal route. We decided to test this by manually increasing the delay along one of the links in the network (the link between cpn017 and cpn042 in the Fig. 1) midway through the experiment (at time $t = 450s$). The results of this experiment are shown in Fig. 4, where CPN was able to react to the change within 1 second, and find a route around the hotspot.

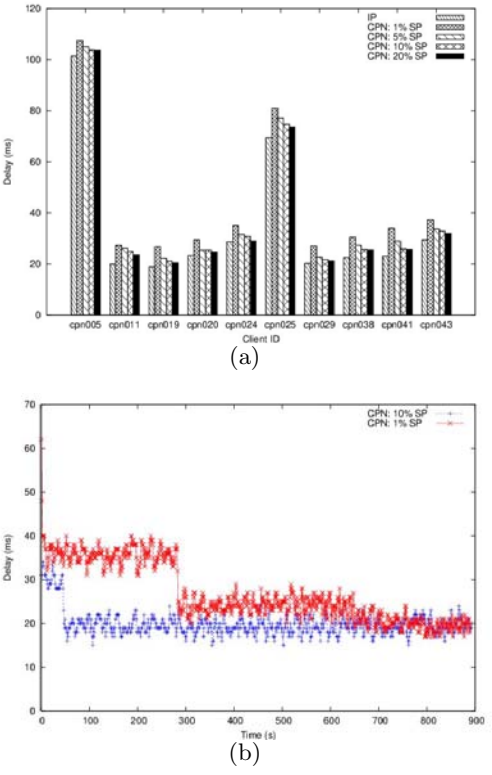

Fig. 3. IP vs. CPN for different percentages of Smart packets. In (a), we can see that the average delay for CPN is very close to optimal. Also, as the number of SPs increases, the delay decreases. The reason the average is a bit higher can be seen in (b) where at the beginning of the experiment, CPN has not yet found the optimal route, increasing the overall average. In addition, the impact of the SP percentage on the rate of discovering the optimal route is highlighted.

4 Conclusion and Future Work

In this paper, we have presented a number of experiments which demonstrate CPN's ability to find the optimal route in the network, and the speed with which it is able to do so. We showed that even with minimal exploration overhead (1% Smart Packets), CPN is still able to find the optimal route in the network, and that increasing the overhead simply accelerates the convergence process. We have also described a small optimization to the original algorithm where we only re-route data traffic when we find a better route, and have shown that this performs better than the previous method. These all point to the strength of the RNN as a decision-maker in a dynamic, on-line scenario.

Looking to the future, we would like to extend this work to a larger, more heterogeneous environment, and are currently exploring how PlanetLab can be

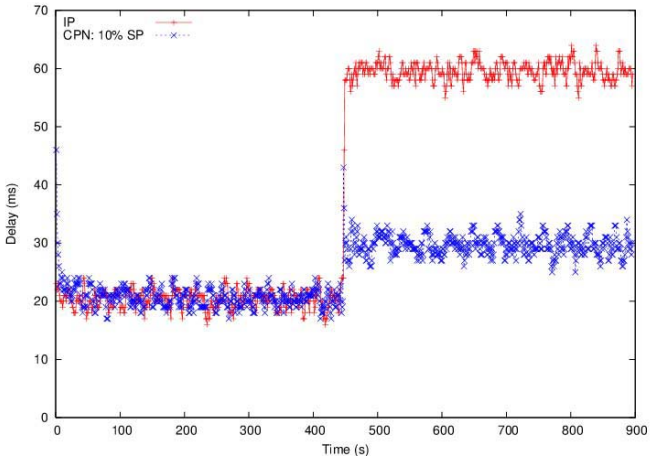

Fig. 4. How CPN reacts to a disturbance along the optimal route. At time $t = 450s$, the delay along one link in the optimal route is increased by 40ms. Within 1 second, CPN has switched to another route.

used for this purpose. We also plan to diversify the traffic types that we include in our evaluation, including TCP flows, and real-time multimedia streams.

Acknowledgements

We would like to thank our supervisor Prof. Erol Gelenbe for his support and guidance in this work. We would also like to thank the Swiss Education and Research Network for sharing their network topology data.

References

1. Gelenbe, E., Xu, Z., Seref, E.: Cognitive packet networks. In: Proceedings of the 11th International Conference on Tools with Artificial Intelligence. (1999) 47–54
2. Gelenbe, E., Gellman, M., Lent, R., Liu, P., Su, P.: Autonomous smart routing for network QoS. In: Proceedings of the First International Conference on Autonomic Computing, New York, NY (2004) 232–239
3. Gelenbe, E., Lent, R.: Power-aware ad hoc cognitive packet networks. Ad Hoc Networks Journal **2**(3) (2004) 205–216
4. Gelenbe, E.: Learning in the recurrent random neural network. Neural Computation **5** (1993) 154–164
5. Gelenbe, E., Lent, R., Xu, Z.: Measurement and performance of a cognitive packet network. Journal of Computer Networks **37**(6) (2001) 691–701