QoS and Routing in the Cognitive Packet Network*

Erol Gelenbe and Peixiang Liu
Department of Electrical & Electronic Engineering
Imperial College, London SW7 2BT
{e.gelenbe,p.liu}@imperial.ac.uk

Abstract

We present experimental results on an autonomic network test-bed, the Cognitive Packet Network (CPN), designed for research in adaptive Quality-of-Service (QoS) management. CPN is fully compatible at its edges with the IP protocol, while internally it offers dynamic routing based on on-line sensing and monitoring. CPN can implement distributed adaptive shortest-path routing, and we compare it with minimum-delay based routing and a composite approach.

1 Introduction

Distance Vector and Link State algorithms [2] are the usual methods for finding the shortest paths between source and destination in the Internet. In the Distance Vector routing, a router advertises a route as a vector of direction and distance. The direction refers to a port which leads to the next router along the path to the destination, and the distance is a metric that indicates the number of hops to the destination or other comparable metric. The routers create routing tables using the vector information exchanged. In the Link State routing, the routers gather the link-state information from neighbors and pass it on to the other neighbors. Eventually, all the routers have information about all the links on the network. Then each router runs the Dijkstra shortest path algorithm to calculate the best path to each network and build the routing table.

On the other hand, CPN [4] is a distributed protocol that provides QoS driven routing; possible applications include peer-to-peer connections, multi-homing for QoS or cost purposes, and overlay networks running on top of IP. It uses *Smart or Cognitive Packets (SP)* that discover routes using a reinforcement learning (RL) based on a QoS "goal" such as packet delay, loss, hop count, jitter, etc.. The "goal" may be defined by the user, or by the network itself. SPs find routes and collect measurements, they do not carry payload. The RL algorithm uses the observed outcome of a previous decision to "reward" or "punish" the previous choice, so that its future decisions are more likely to meet the QoS goal. When a SP arrives to its destination, an *acknowledgement (ACK) packet* is generated and it stores the "reverse route" and the measurement data collected by the SP. It travels along the "reverse route" which is computed from the SP's route, examining it from right (destination) to left (source), and removing any node sequences which begin and end in the same node due to multiple visits of the SP to the same node, e.g., the path $< a, b, c, d, a, f, g, h, c, l, m >$ will result in the reverse route $< m, l, c, b, a >$. Note that the reverse route is not necessarily the shortest reverse path, nor the one resulting in the best QoS. Finally, *Dumb Packets (DP)* carry payload and use dynamic source routing. The route brought back by an ACK is used as a source route by subsequent DPs of the same QoS class having the same destination, until a newer AND/OR better route is brought back by another ACK. A *Mailbox(MB)* in each node is used to store QoS information. Each MB is organized as a Least-Recently-Used (LRU) stack, with entries listed by QoS class and destination, which are updated when an ACK is received. The QoS information is then used to calculate the reward in the SP routing algorithm. We use recurrent random neural networks (RNN) [1] with reinforcement learning (RNNRL) in order to implement the SP routing algorithm. Each output link of a node is represented by a neuron in our RNN. The arrival of *Smart Packets(SPs)* triggers the execution of RNN and the output link corresponding to the most excited neuron is chosen as the routing decision. The weights of the RNN are updated so that decisions are reinforced or weakened depending on how they have been observed to contribute to the success of the QoS goal. The RNN is an analytically tractable spiked random neural network model whose mathematical structure is akin to that of queuing networks. It has "product form", like many useful queuing network models, although the RNN is based on nonlinear mathematics. The state q_i of ith neuron in the network is the probability that it is excited. The q_i, with

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

$1 \leq i \leq n$ satisfy the following system of nonlinear equations:

$$q_i = \lambda^+(i)/[r(i) + \lambda^-(i)] \quad (1)$$

where

$$\lambda^+(i) = \sum_j q_j w^+_{ji} + \Lambda_i,$$

$$\lambda^-(i) = \sum_j q_j w^-_{ji} + \lambda_i. \quad (2)$$

Here w^+_{ji} is the rate at which neuron j sends "excitation spikes" to neuron i when j is excited, w^-_{ji} is the rate at which neuron j sends "inhibition spikes" to neuron i when j is excited, and $r(i)$ is the total firing rate from the neuron i. For an n neuron network, the network parameters are these n by n "weight matrices" $W^+ = \{w^+(i,j)\}$ and $W^- = \{w^-(i,j)\}$ which need to be "learned" from input data.

Let us describe the manner in which CPN *adaptively* implements shortest path routing with all nodes in the network being able to contribute to decision in a distributed manner. The QoS goal G will in this case will be hop count, and the reward function R is:

$$R = \frac{1}{\beta \cdot G} \quad (3)$$

where $\beta = 0.75$ was used in the experiments. G can be readily measured by SPs by incrementing a counter that increases each time the SP visits another node. The value of G related to a SP's path from a given node to the destination is brought back to that node by the SP's corresponding ACK, and is then stored in the node's MB. Successive values of R denoted by $R_l, l = 1, 2, \cdots$ are used first to compute a decision threshold:

$$T_l = \alpha T_{l-1} + (1 - \alpha) R_l \quad (4)$$

where α is some constant($0 < \alpha < 1$) that is used to tune the responsiveness of the algorithm: for instance $\alpha = 0.8$ means that on the average five past values of R are being taken into account. Suppose that the RL algorithm's the lth decision was to select output link (neuron) j, and that the lth reward obtained via an incoming ACK is R_l. If R_l is larger than, or equal to, the threshold T_{l-1}, then we increase significantly the excitatory weights going into neuron j and make a small increase of the inhibitory weights leading to other neurons. If R_l is less than T_{l-1}, then we increase moderately the excitatory weights leading to all neurons other than j and increase significantly the inhibitory weight leading to neuron j, in order to punish it for not being successful:

- If $T_{l-1} \leq R_l$

$$\begin{aligned} w^+(i,j) &\leftarrow w^+(i,j) + R_l \\ w^-(i,k) &\leftarrow w^-(i,k) + R_l/(n-1), \text{ for } k \neq j. \end{aligned}$$

- Else

$$\begin{aligned} w^+(i,k) &\leftarrow w^+(i,k) + R_l/(n-1), \text{ for } k \neq j \\ w^-(i,j) &\leftarrow w^-(i,j) + R_l \end{aligned} \quad (5)$$

The probabilities q_i are then computed using (1) and (2); the next SP will be forwarded to the output link which corresponds to the neuron having the largest excitation probability. We can also combine the hop count metric with the forward delay, so that the goal takes into account both the length H and the delay D of the path:

$$G = H + \gamma D \quad (6)$$

We take $\gamma = 10^{-3}$ so as to bring D (μs) and H into a mutually comparable range of values.

2 Experimental results

We report results of numerous experiments that were run on a CPN test-bed consisting of 17 nodes shown in Figure 1. Each pair of nodes is connected by point-to-point

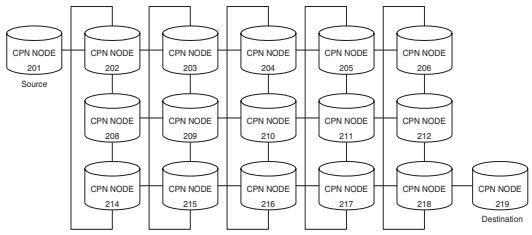

Figure 1. Test-bed used in the experiments

10Mbps Ethernet links. All tests were performed using a flow of UDP packets entering the network at constant bit rate (CBR) with 1024 bytes packets and three traffic rates: Low Traffic Rate (LTR) corresponds to 100 packets/second (p/s), Medium Traffic Rate to 500p/s and High Traffic Rate (HTR) to 1000p/s. Each measurement point is based on $10,000$ packets that were sent from the source to the destination, and we inserted random background traffic into each link in the network with the possibility of varying its rate. The CPN routing algorithm is used throughout the experiments using three different QoS goals: (a) delay [*Algorithm-D*], (b) hop count [*Algorithm-H*] and (c) the combination of hop count and forward delay [*Algorithm-HD*]. Our measurements concern average hop count, the forward delay and packet loss rate under different background traffic conditions.

From Figure 1, we see that the shortest path length from the source node $(\#201)$ to destination node $(\#219)$ is 7, and that there are five distinct shortest paths, one being route $\langle 201 \rightarrow 202 \rightarrow 214 \rightarrow 215 \rightarrow 216 \rightarrow 217 \rightarrow$

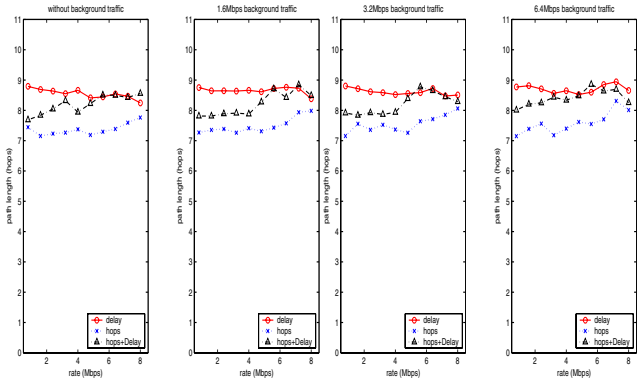

Figure 2. Shortest Path length with CPN

$218 \rightarrow 219\rangle$. Figure 2 reports the average number of hops traversed from source to destination when different algorithms are used. When hop count is used as the QoS goal, we see that CPN's average number of hops under different background traffic levels is close to the minimum of 7. We also trace the routes used by each packet without background traffic. Figure 3 shows the routes used under LTR. 25 different routes are used in total and one shortest path is discovered: 1805 of 2000 packets use route $\langle 201 \rightarrow 202 \rightarrow 203 \rightarrow 204 \rightarrow 205 \rightarrow 206 \rightarrow 218 \rightarrow 219\rangle$, which is one of the shortest routes. At MTR (Figure 4), 20 routes are used and 4 of them are shortest paths. 1711 of the 2000 packets use one of the shortest paths, route #3. Under HTR, 12 routes are used and 4 of them are the shortest paths. In this case, only 1021 packets are using the shortest paths. We can conclude that when *Algorithm-H* is used, the shortest paths are indeed discovered by the the SPs and used by most of the DPs; most of the DPs keep on using the same shortest path even if more than one of them has been discovered. Since the topology of out test-bed does not change, once a shortest path is discovered and used, the positive feedback brought back by the ACKs will reward the previous choice so that the RNNs keep recommending that the same path be chosen. When the traffic rate is very high, it appears that the nodes do not receive enough QoS updates from ACKs as SPs are sent out because of congestion. That is why only 12 routes were discovered and there was no usage preference on the routes under HTR. When forward delay is used as the QoS goal, the average number of hops is not minimised any more (here it is close to 9) as seen in Figure 2). Figures 6,7 and 8 show the routes are being used for different levels of connection trafic rate. Compared to *Algorithm-H*, more routes are being discovered and used. The number of routes used is 40, 35 and 15 under LTR, MTR and HTR, respectively. The delay will decrease when traffic is spread over more routes, so that the network is using more alternate paths under HTR. Figures 9,10 and 11 show the routes with *Algorithm-HD*. The number of routes discovered by the smart packets are 43, 45 and 12 under LTR, MTR and HTR, respectively. From Figure 2, we can see that the average path length is close to 8 when the connection's traffic rate is low or medium; when it is high, the average path length is close to 9. The experiments confirm our expectations with respect to path length: *Algorithm-H* is the best, and *Algorithm-HD* is better than *Algorithm-D*. We also measure the forward delay and the packet loss rate. The forward delay is approximated as one half of the round trip delay. To measure the loss rate, we count the number of

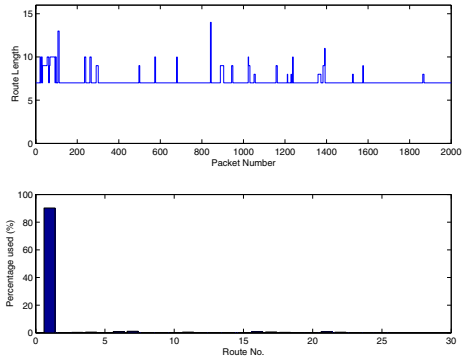

Figure 3. H-Algorithm route usage with LTR

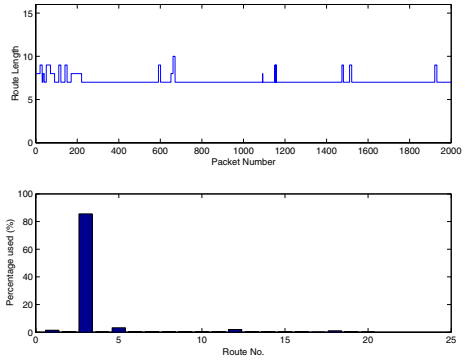

Figure 4. H-Algorithm route usage with MTR

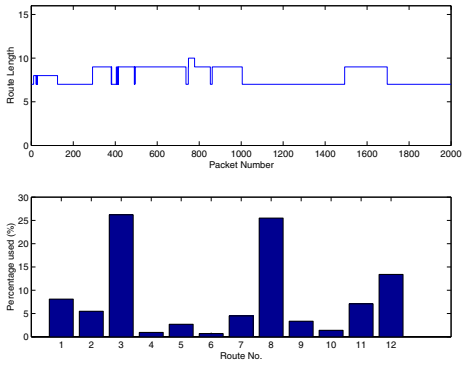

Figure 5. H-Algorithm route usage with HTR

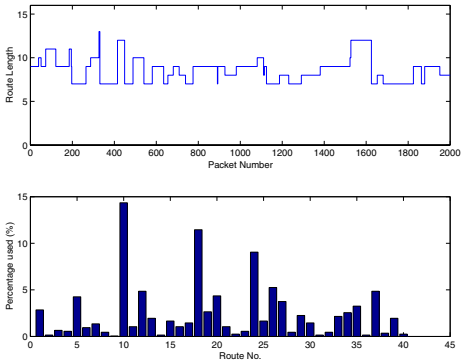

Figure 6. D-Algorithm route usage with LTR

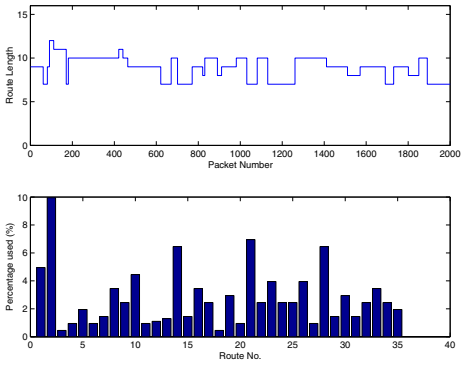

Figure 7. D-Algorithm route usage with MTR

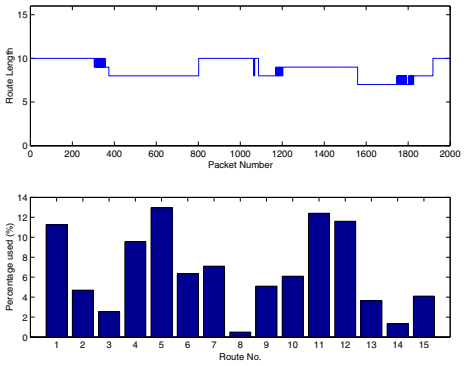

Figure 8. D-Algorithm route usage with HTR

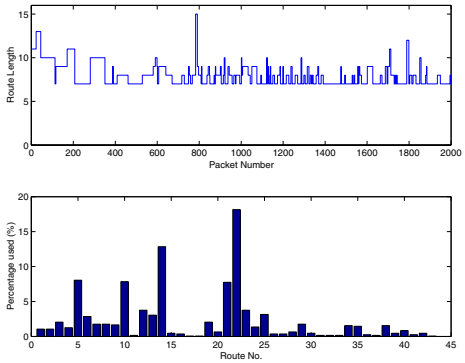

Figure 9. HD-Algorithm route usage with LTR

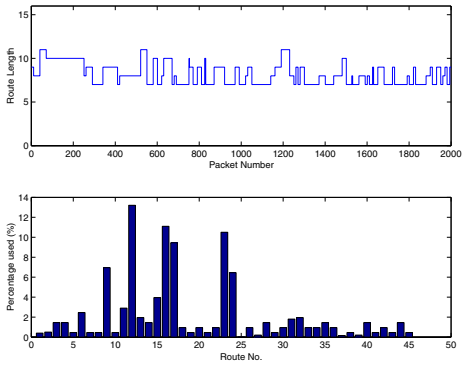

Figure 10. HD-Algorithm route usage with MTR

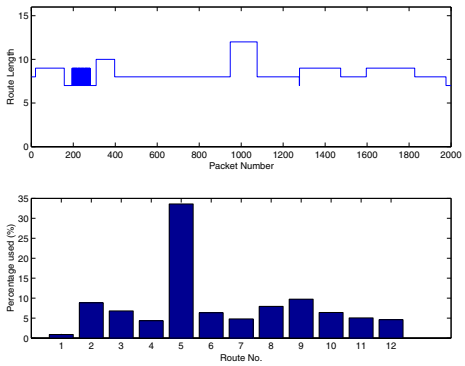

Figure 11. HD-Algorithm route usage with HTR

packets s sent out from the source node and the number of packets r which are received at the destination. The packet loss rate is then $L = 1 - r/s$. From Figure 12, we observe that if the connection's traffic rate is less than 3.2Mbps, then *Algorithm-H* achieves the smallest delay, while *Algorithm-D* is the worst. However, when the connection's traffic rate is between 3.2Mbps and 5.6Mbps, the performance of *Algorithm-HD* is better but *Algorithm-H* and *Algorithm-D* are almost the same. All algorithms are equivalent with respect to measured delay when the connection's traffic rate is between 5.6Mpbs and 7Mbps. When the connection's traffic rate is veryhigh (>7Mbps) *Algorithm-D* gives the smallest delay, the and *Algorithm-H* is the worst. When the background traffic per link is 3.2Mbps (Figure 13) and the connection traffic rate is less than 3.2Mbps, *Algorithm-H* is still the best; again *Algorithm-D* is the worst. But if the

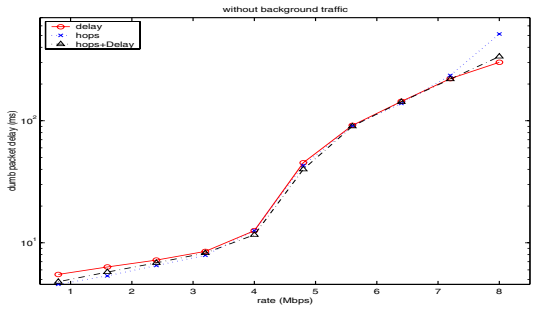

Figure 12. Delay without background traffic

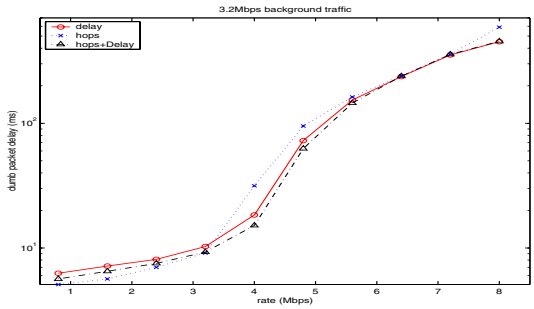

Figure 13. Delay with 3.2Mbps background traffic

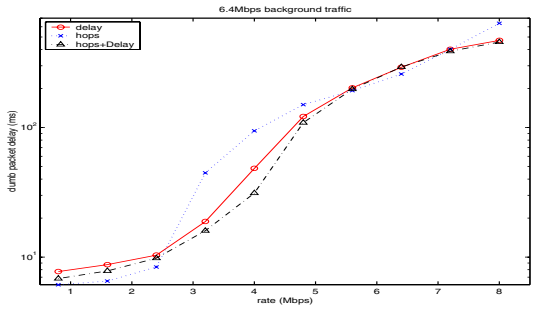

Figure 14. Delay with 6.4Mbps background traffic

connection traffic rate is between 3.2Mbps and 6.4Mpbs, *Algorithm-HD* is the best and *Algorithm-H* is the worst. When the connection traffic rate is higher than 6.4Mbps, *Algorithm-D* performs as well as *Algorithm-HD* and both of them are better than *Algorithm-H*. These results are similar with 6.4Mbps background traffic per link, except that the threshold points change (Figure 14). Thus, if the network topology does not vary and the network is lightly loaded, *H-Algorithm* is best as far as the forward delay is concerned. When the network is heavily loaded, sticking to the shortest path is a bad choice, and longer paths offer lower delay.

We have measured packet loss under different background traffic conditions, but we only present results for high traffic rates in Figure 15 for lack of space. We observe that *Algorithm-H* has the smallest loss rate, while the loss rate with *Algorithm-HD* and *Algorithm-D* are almost the same: when the network load is high, all links suffer high losses, but losses on the shortest paths are smaller since they sum losses over the smallest number of links.

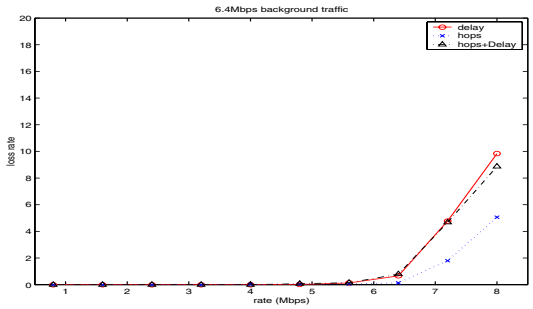

Figure 15. Loss with 6.4Mbps background traffic

3 Conclusions

In this paper we present experiments on an autonomic network test-bed, the Cognitive Packet Network, that offers QoS based routing. It uses a distributed reinforcement learning algorithm that takes decisions at each node, based on locally implemented neural networks whose weights are updated with data collected and brought to each node by SPs ACKs. Experiments show the CPN can approximately find shortest paths, as well as offer more complex criteria for routing including packet delay. The use of criteria more complex than the shortest number of hops, can provide better overall quality of service.

References

[1] E. Gelenbe. Learning in the recurrent random neural network. *Neural Computation*, 5(1), pp. 154–164, 1993.

[2] D. Williams and G. Apostolopoulos. QoS Routing Mechanisms and OSPF Extensions. RFC 2676, Aug. 1999.

[3] K. Kowalik and M. Collier. Should QoS routing algorithms prefer shortest paths? In *Proc. IEEE Int'l. Conf. on Comms.*, pp. 213-217, 2003.

[4] E. Gelenbe. Cognitive Packet Network. *U.S. Patent No. 6,804,201 B1*, Oct. 12, 2004.