Autonomous Smart Routing for Network QoS [*]

Erol Gelenbe
Dennis Gabor Chair
Electrical & Electronic Eng'g. Dept.
Imperial College
London SW7 2BT, UK
e.gelenbe@imperial.ac.uk

Michael Gellman, Ricardo Lent, Peixiang Liu and Pu Su
School of Computer Science
University of Central Florida
Orlando FL 32816, USA
{michaelg,rlent,pliu,psu}@cs.ucf.edu

Abstract

We present an autonomous adaptive quality of service (QoS) driven network system called a "Cognitive Packet Network" (CPN), which adaptively selects paths so as to offer best effort QoS to the end users based on user defined QoS. CPN uses neural network based reinforcement learning to make routing decisions separately at each node. Measurements on an experimental test-bed are provided to show how the system responds to the choice of QoS goals. We also discuss and evaluate an extension of CPN that uses a genetic algorithm to generate and maintain paths from previously discovered information by matching their "fitness" with respect to the desired QoS.

1 Introduction

Broadly speaking, Quality-of-Service (QoS) is the level of performance that is experienced by a specific user (or a class of users) in a network, and it is an important consideration for many network applications such as real-time voice and video [4, 5] which may have stringent requirements for the amount of loss, delay, or jitter that they can tolerate. QoS is notoriously a requirement that is not well satisfied in existing packet networks which are based on the Internet Protocol (IP).

In this paper we describe an autonomic adaptive packet network architecture called the Cognitive Packet Network (CPN), which dynamically seeks out paths, and constantly updates paths, as a function of user specified QoS. In CPN, each connection in the network autonomously searches out a network path which provides it with the best QoS according to objectives that are defined by the connection itself.

The purpose of the CPN project is to experiment with self-awareness and autonomous control in networks and distributed systems [1]. CPN uses "smart packets" (SPs) belonging to a specific connection, to seek out network paths based on user specified QoS Goals. CPN routers store information which is relevant to QoS (such as delay and loss), which is gathered by SPs and carried back by acknowledgement packets (ACKs). Subsequent SPs will use this information to select better paths using a reinforcement learning (RL) algorithm running at each router, while payload packets will travel along paths that have been discovered by the SPs.

This paper describes CPN and its control algorithm, and discusses measurements on CPN test-beds. We also present an extension that uses genetic algorithms to generate new routing paths by combining paths which have been previously discovered. The QoS of these new paths can be estimated from the observed QoS of known paths, and is then employed to seek better routes. Additional experiments are described, where a Genetic Algorithm (GA) [7] is used in background mode, in conjunction with CPN running RL in the foreground. The experiments show the CPN without a GA provides the main QoS enhancement to users, and that the GA provides significant improvement only at light to intermediate loads.

1.1 CPN Routing

CPN includes *three* different types of packets which play different roles. *Smart or cognitive packets (SP)* are used to discover routes for connections; they are routed using a reinforcement learning (RL) algorithm [8] based on a QoS "goal". We use the term goal to indicate that there are no QoS guarantees; rather CPN provides best effort service to optimize the desired QoS metrics. The role of SPs is to find routes and collect measurements; they do not carry payload. CPN's RL algorithm uses the observed outcome of a decision to "reward" or "punish" the routing algorithm, so that its future decisions are more likely to meet the desired QoS

[*] We thank the Engineering and Physical Sciences Research Council (UK) for its support for this research under Grant GR/S52360/01.

Goal. The goal is the metric which characterizes the success of the outcome, such as packet delay, loss, jitter and so on. In CPN, the specific Goal used by a SP will depend on the user's QoS requirements.

When a SP arrives to its destination, an *acknowledgment (ACK) packet* is generated and the ACK stores the route followed by the original packet and the measurement data it collected. An ACK being returned as a result of a SP will be source-routed along the "reverse route" of the SP. The reverse route is established by taking a SP's route, examining it from right (destination) to left (source), and removing any sequences of nodes which begin and end in the same node. That is, 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 route, nor the one resulting in the best QoS. ACKs deposit QoS information in the mailboxes (MBs) of the nodes they visit.

Dumb packets carry payload and use source routing. Dumb packets also collect measurements at nodes. The route brought back to a source node by an ACK of a SP is used as a source route by subsequent dumb packets of the same QoS class having the same destination, until a new route is brought back by another ACK.

MBs in nodes are used to store QoS information. MB entries in a given node are identified by QoS class and Destination. Each MB is organized as a "least-recently-used" stack: old information can be discarded from fast memory when the MB is full, and new information is rapidly accessible from the top. For each SP in the node's input buffer, the node will run the CPN routing code. Then the packet is placed in an output buffer which is selected by the CPN routing algorithm. If a DP or ACK enters a node, and the node number does not correspond to the node it should now be visiting, then the packet is discarded.

As an example of how QoS information is obtained, consider how delay is measured. When an ACK for a packet which was going from S to D and was of class K enters some node N from node M, the following operation will be carried out: the difference between the local time-stamp and the time-stamp stored in the ACK for this particular node is computed and divided by two. The resulting time is stored in the mailbox as the value $W(K, D, M)$ – it is an estimate of the forward delay for a packet of QoS class K going from node N to D and which exited node N via the port leading to M. Note that the identity of the local node N is obvious and need not be stored. The source node S is also not relevant since the $W(K, D, M)$ refers to the time to go from N to D using the next node M. The QoS class K is needed since the decision at each node, and the resulting observed delay, will depend on the requirements expressed by the QoS class K. The quantity $W(K, D, M)$ is inserted in the Goal function (see equation (4) of the RL learning algorithm for the delay value).

Different approaches to learning could in principle be used to discover good routes in a network, including Hebbian learning, back-propagation [6], and reinforcement learning (RL) [8, 3]. Hebbian learning is notoriously slow and was excluded from our consideration. Simulation experiments we conducted at the beginning of the project indicated that RL was the most effective approach to achieve fast adaptation to network conditions. Our RL implementation uses a *one step* update of weights based on the most recent information, so that new decisions are significantly impacted by the most recent outcome. In order to guarantee convergence of the RL algorithm to a single decision (i.e., selecting an output link for a given smart packet), CPN uses the random neural network (RNN) [2] which has an unique solution to its internal state for any set of "weights" and input variables. At each node we will have a separate RNN for each QoS class and destination. For a given QoS class, a specific neuron of the corresponding RNN is associated with a specific output link of the node.

In the RNN, the state q_i of the $i - th$ neuron denotes the probability that it is excited, and the q_i, $i = 1, \ldots n$ for an n neuron network satisfy the following system of non-linear equations:

$$q_i = \lambda^+(i)/[r(i) + \lambda^-(i)], \tag{1}$$

where

$$\lambda^+(i) = \sum_j q_j w_{ji}^+ + \Lambda_i, \quad \lambda^-(i) = \sum_j q_j w_{ji}^- + \lambda_i, \tag{2}$$

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" $\mathbf{W}^+ = \{w^+(i, j)\}$ and $\mathbf{W}^- = \{w^-(i, j)\}$ which need to be "learned" from input data. The state values q_i, $i = 1, \ldots n$, are used to make the decision of a SP to select the output link i at the node: the largest q_j designates the output link j which is selected.

As an example, the QoS Goal G that SPs pursue may be formulated as minimizing Transit Delay W, Loss Probability L, Jitter, or some weighted combination captured in the numerical Goal function G and the reward $R = 1/G$. Successive values of R, denoted by R_l, $l = 1, 2, \ldots$, are computed from the measured delay and loss data, and are used to update the neural network weights. The CPN RL algorithm first updates a threshold value:

$$T_l = aT_{l-1} + (1 - a)R_l, \tag{3}$$

where a is some constant $0 < a < 1$, typically close to 1 and R_l is the most recently measured value of the reward. T_l is a running value that is used by the RL algorithm to keep track of the historical value of the reward and is is used to determine whether a recent reward value is better or worse than the historical value. Suppose that the $l-th$ decision selected output link k, and that we received feedback from the network which measured the $l-th$ reward R_l. We first determine whether R_l is larger than, or equal to, the threshold T_{l-1}. If that is the case, then we conclude that the previous decision worked well; we increase the excitatory weights $w^+(i,k)$ to the neuron k that was the previous winner (in order to reward it for its success), and make a small increase of the inhibitory weights $w^-(i,j)$, $j \neq k$, leading to other neurons. On the other hand, if R_l is less than the threshold, we conclude that the previous decision was not good and moderately increase **all** excitatory weights leading to **other** neurons, and increase significantly the inhibitory weights leading to the previously selected neuron k in order to punish it for being unsuccessful:

- If $T_{l-1} \quad \leq \quad R_l$
  - $w^+(i,k) \leftarrow w^+(i,k) + R_l$,
  - $w^-(i,j) \leftarrow w^-(i,j) + \frac{R_l}{n-2},\ for\ all\ j \neq k.$
- Else
  - $w^+(i,j) \leftarrow w^+(i,j) + \frac{R_l}{n-2},\ for\ all\ j \neq k,$
  - $w^-(i,k) \leftarrow w^-(i,k) + R_l.$

Finally the node with the largest q_i is identified, and the smart packet is placed in the corresponding output buffer i.

2 Composite Goal Functions

For an application which has QoS needs that can include both loss and delay, the QoS goal that may be used to route packets will have to combine in one single Goal function both the loss and delay incurred from source to destination. In this case, we can form the goal function G as follows:

$$G = (1 - \overline{L_f})\overline{W} + \overline{L_f}(T + G) \tag{4}$$

where $\overline{W}$ is an estimate of the forward delay, $\overline{L_f}$ is an estimate of the forward packet loss ratio, and T is the additional time incurred by a packet which is retransmitted after a loss, including the time-out delay before a non-acknowledged (and presumably lost) packet is retransmitted, and any additional overhead resulting from the retransmission of the lost packet. The expression (4) is based on the idea that if a loss occurs, with probability $\overline{L_f}$, then the resulting cost is the delay T until the packet is retransmitted, and this will be followed by the same equivalent total delay G incurred by the freshly retransmitted packet. If on the other hand a packet is not lost with probability $[1 - \overline{L_f}]$, then the cost is simply the delay $\overline{W}$ that will be incurred by a packet as it traverses the network to reach its destination. Note that G appearing on both sides of (4) is written under the assumption that the subsequent packet sent out to replace the lost packet will on the average incur the same total cost G, since it too may be lost and could be retransmitted. This expression simplifies to yield the reward $R\ = 1/G$:

$$R = \frac{1}{T\frac{\overline{L_f}}{1-\overline{L_f}} + \overline{W}} \tag{5}$$

In order to use R we must obviously be able to estimate $\overline{W}$ and $\overline{L_f}$. In Section 1.1 we describe how ACK packets deposit an estimate of “delay to the destination” into the MBs of nodes that they visit. In order to select a particular path in the network based on composite path QoS metrics, CPN also needs to estimate path packet loss ratios defined as the number of packets sent but not received, divided by the number of packets which have been sent. We will discuss how to estimate link loss and path loss in the following.

2.1 Estimating Link Loss and Path Loss

Packet loss ratios are simply the ratio of number of packets correctly received to the number of packets sent. The link loss ratio refers to the corresponding quantity measured over a single link connecting two nodes. Path loss ratio on the other hand refers to the quantity measured over a path, from a source to a destination. We will use the terms “cumulative” or “path” loss interchangeably. In CPN, we estimate the link loss ratio by forwarding, over the link and back to the predecessor node, the number of packets that have been received by the next node on the link. This information is in fact stored or “piggy-backed” in ACK packets. If N packets have been sent over a link and R packets have been received at the next node, then the loss ratio is:

$$L = 1 - \frac{R}{N} \tag{6}$$

To estimate the path loss ratio, we use ACKs coming back from the destination node. The source is able to estimate the round-trip loss ratio by keeping track of the number of packets sent and the number of ACKs it receives. However, in addition the destination can keep track of the number of packets which are received at the destination, and this number can be piggy-backed inside ACK packets and returned to the source. The time stamp at the destination which is carried by the ACK, will allow the sender to estimate forward loss rates over a given period of time. Thus even if some ACK packets are lost, it is still possible to have a fairly accurate estimate at the source of forward (source to destination) path losses, and not of just round-trip losses. However, the loss ratio estimates at the source can be insensitive to *short term* changes which are important to QoS driven adaptive routing. We address this problem in CPN by using the following scheme:

- The sender maintains a smoothed average of the packet loss ratio: $\overline{L} \leftarrow (1-a)\overline{L} + aL$.
- The receiver modifies R as follows for some threshold value of R_{max}:
$$\text{if} \quad R > R_{max} \qquad \text{then} \quad R \leftarrow 0$$
- If R_l is $l - th$ value of R received at the sender, the sender carries out the following operation:
$$\text{if} \quad R_{i+1} < R_i \qquad \text{then} \quad N \leftarrow R_{i+1}.$$

As a result, large values of R are eliminated, while $\overline{L}$ preserves an accurate estimate of the loss ratio over the link from the sender's perspective.

Since it is impractical to have the destination nodes keep a count of the number of packets received for each possible route from every possible source, we need to find a scheme that will reduce the amount of data that is stored. This requires us to make a simplifying assumption based on the idea that forward and reverse routes generally use the same set of nodes and links. Thus, we assume that the DP loss ratio L_f from the source S to the destination D is proportional to the ACK loss ratio L_b in the opposite direction or $L_f = \alpha L_b$. Let N be the number of DPs sent from S to D, and A be the corresponding number of ACKs received by S. We can write:

$$\frac{A}{N} = 1 - L_f.L_b = 1 - \alpha(L_f)^2, \qquad (7)$$

so that

$$L_f = 1 - \sqrt{\frac{A}{\alpha N}}. \qquad (8)$$

The source S therefore stores separate (N, A) values for each of its destinations. Assuming that forward and reverse loss rates are identical, we set $\alpha = 1$ and $L_f = 1 - \sqrt{\frac{A}{N}}$.

If routing only selects paths which offer the lowest packet loss, there are several ways in which we can construct the reward R. One approach is to set $\overline{W} = 0$ in the expression (5), obtaining:

$$R = \frac{1 - \overline{L_f}}{T\overline{L_f}}, \qquad (9)$$

so that $1/T$ acts as a constant multiplier. In practice, since we do not want R to be infinite when $L_f = 0$, we set:

$$R = \frac{1 - \overline{L_f}}{T(\overline{L_f} + \epsilon)}, \qquad (10)$$

where ϵ is a constant representing some minimal value for the loss. A simpler approach is to use R of the form:

$$R = \frac{\beta}{\overline{L_f} + \epsilon}, \qquad (11)$$

which relates loss directly to the reward. This is the approach we have taken in our experiments when we just deal with loss (rather than loss and delay). In the experiments we report, we have used the following numerical values of the constants: $\epsilon = 10^{-5}$ and $\beta = 0.5$.

2.2 Measurement Results

We conducted measurements on the CPN testbed consisting of 26 nodes shown in Figure 2.2. The measurements that we report were performed under a variety of conditions. All tests were conducted using a flow of UDP packets entering the CPN network with constant bit rate (CBR) traffic and a packet size of 1024Kb. All CPN links used 10Mbps point-to-point Ethernet. The UDP packet stream with CBR, was sent into Node 10 as the source, for forwarding to Node 7 as the destination. No artificial packet losses were introduced, other than those that might result from congestion at the nodes.

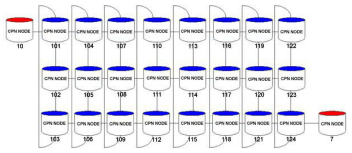

Figure 1. The 26 Node Test-Bed used in the CPN Measurements

As seen in the topmost curve of Figure 2, the reduction in packet loss rate when the QoS goal is "cumulative loss" appears to be very significant, compared to using delay, or loss and delay, as the QoS goal. Using the goal function based only on cumulative loss results in the lowest observed loss rates, and confirms the effectiveness of choosing a goal function identical to the desired QoS. The curves in the bottom of Figure 2 show that the lowest delay is obtained by using only *delay*, or *loss and delay* as the goal function. The linear scale used for delay (y-axis) in this figure clearly shows a peak for traffic in the range of 11 to 12Mb/s, with a reduction in delay above that value, when cumulative loss or only delay are used in the routing goal function. The drop in delay at higher traffic values is presumably due to the significant loss of packets which results in lower congestion.

3 Genetic Algorithms and CPN

A GA is a learning algorithm which operates by simulating evolution [7]. Key features that distinguish a GA from other search methods include: a *population* of *individuals* where each individual represents a potential solution to the problem to be solved, a *fitness function* which evaluates the utility of each individual as a solution, a *selection function*

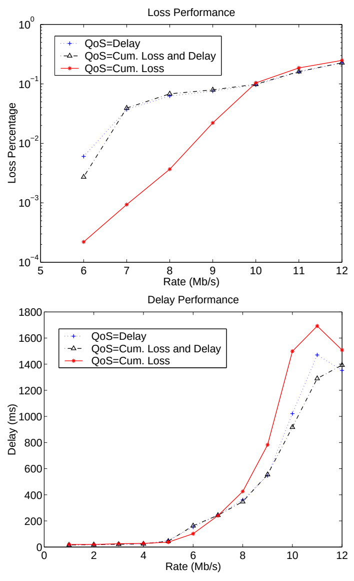

Figure 2. Variation of loss (top) and delay (bottom) measurements versus network load, for different QoS Goals

which selects individuals for reproduction based on their fitness, and finally the *genetic operators* which combine selected individuals to create new individuals via crossover and mutation.

In CPN, we have included a GA which runs as a background process at each source node, to generate and select paths for dumb packets based on the QoS goal. The GA population will consist of individuals which represent paths between the source node and potential destination nodes. We will use a variable length representation which is expected to allow the GA more flexibility to evolve in response to changes in the network [9]. The fitness of a path is determined from the measurement data returned by an ACK that is received in response to a dumb packet sent along the path. New paths are constructed by genetic operators, e.g. mutation constructs new paths via small modifications to existing paths while crossover constructs new paths from two existing paths that share a common intermediate node. The GA also receives input from the CPN as new paths are discovered by SPs. Exploitation and preservation of existing good paths (or partial paths) is accomplished through fitness-based selection. A path is selected if its fitness function is better than, or within a range of, the fitness function of the current population. A *routing word* or *word*, w, is a variable length sequence of nodes which begin with the source node S and end with the destination D, and w represents any viable path from S to D. Each routing word w has a goal value $G(w)$, and describes how effective the path described by the word w is. Thus, a smaller value of $G(w)$ means that w is more desirable. A Goal is *additive*, if for any word $w = \alpha\beta$ which may be expressed as the concatenation of two words α and β, we have $G(w) = G(\alpha) + G(\beta)$. Note that packet delay and loss are additive along paths. In our GA implementation, new words are generated in two different ways. Since SPs discover routes, and ACK packets bring back valid routes to the source, CPN already provides a way of generating new words w using reinforcement learning based search for routes. The source will keep the words which have been brought back by the ACKs in a list sorted in the order of increasing $G(w)$ values which we call the $Stack$. Secondly, at a source node we generate additional words using the following *path crossover* operation: Suppose two words w_1 and w_2 share some intermediate node, so that for some node a, we have $w_1 = u_1av_1$ and $w_2 = u_2av_2$. The crossover operation will then generate the strings $w_3 = u_2av_1$ and $w_4 = u_1av_2$. If any of these strings is not already in the $Stack$, then it is placed in the $Stack$ with the corresponding goal values $G(w_3) = G(u_2a) + G(v_1)$ and $G(w_4) = G(u_1a) + G(v_2)$. In this way, the $Stack$ is enriched both with new paths obtained by crossover, and by paths which are obtained via the CPN SP search process. Whenever a dumb packet needs to be forwarded to the destination, the word at the top of the $Stack$ (i.e. the one with the smallest goal value) will be used as the source route. Every complete route discovered by SPs and brought back by ACKs naturally becomes an individual in the GA population for enhancement of CPN routing. New routes will be evolved as offsprings. Collectively, the individuals with the same source S and destination D form a GA population repository $P(S, D)$ which is organized as an LRU stack with some predefined maximum size.

The size of the data structures that the GA uses make it unlikely that a kernel-level implementation can be used. Thus the GA algorithm has been implemented as a system daemon. The GA-enabled module works in a similar fashion to the regular CPN module, with the main difference being the handling of the ACKs when they reach the source. When the ACK reaches the source, we have a complete path to the destination and the measurement data for each hop. This data is put into a FIFO buffer for the GA daemon to process whenever it becomes available, as shown in Fig.3. Unlike the regular CPN module, the GA-enabled module does not update the dumb packet route repository unless there is no route for the current destination or if the GA daemon is dead (defined as “if the GA daemon has not talked to the module for the past second”). The GA daemon is started when the CPN module is loaded with no initial knowledge of the network. It consists of a main loop which polls the kernel for new paths, checking its internal data structures for size and consistency, selecting individuals for crossover and doing the actual crossover and periodically updating the

CPN module's dumb packet route repository.

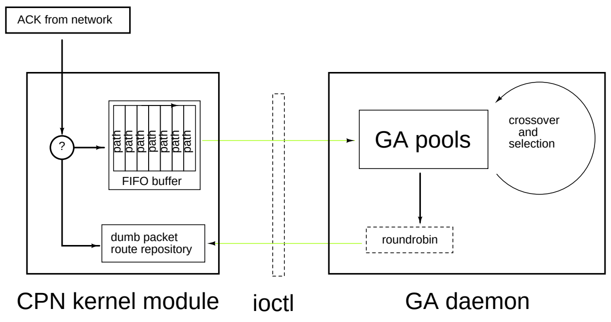

Figure 3. Interaction between module and daemon

The role of the path selection process is to return a pair of individuals which are suitable for crossover. Depending on the situation, the GA daemon either requests a "match" for a specific individual or just asks for a pair of individuals to crossover. The data exchange between the CPN kernel module and the GA daemon is bidirectional as the module passes measurements to the daemon and in return the daemon periodically updates the module's route repository with the paths who have the best fitness at that time. The GA daemon is always the initiator of the data exchange so that the CPN kernel module does not need to keep track of the presence or absence of the daemon to route CPN traffic. This way we also eliminate lockups which would occur if the kernel were to probe the daemon while it is sleeping.

3.1 Measurements of CPN with the GA

To evaluate the performance of CPN with and without the GA, measurements were conducted on the testbed shown in Figure 4. Initially, all the machines in the testbed start with an empty dumb packet repository and empty GA pools. For this particular set of experiments, the source node is the node at the left edge of the testbed as shown in Figure 4. The CPN path finding algorithm is started at the source node using SPs and ACKs. Once the first ACK comes back from the destination carrying the first path that has been discovered, DPs from source to destination are sent over the CPN network at a constant rate. Note that the GA daemon only runs at the source node.

For the experiments in which the GA algorithm is *not* in operation, we simply disabled the GA daemon. On the other hand if the GA daemon is enabled, when the first ACK arrives at the source, the GA daemon is automatically started up and will generate paths as described in the previous section. If a node sends packets to several destinations, the GA daemon divides the paths into subpopulations based on the destination and crossover is done for each given destination. However, the hop pool is common to the all subpopulations, so that when a source sends packets to several destinations, the GA hop pool will contain more data as well as potentially more *recent* data about hops, than if only one destination is used. However in this case the GA still creates independent paths for each destination from the hop pool.

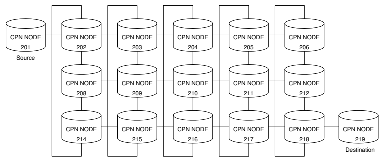

Figure 4. The CPN-GA testbed topology

Throughout the experiments, the QoS Goal that is used for selecting paths is to minimize source to destination delay. However we measured the QoS both in terms of observed average delay and observed average packet loss rate for *DPs*, i.e. for the payload packets. both when the GA was disabled and when it was enabled. Note that SPs' routing is not affected by the GA in either case. We obtain the delay and loss measurements for DPs as follows. At the source, we count the number of ACKs received for DPs, say N_{AD} and the number of DPs transmitted, N_D. We can evaluate the DP loss rate as $L_{DP} = 1 - N_{AD}/N_D$. Note that this value is pessimistic because some ACKs will be lost and we are in fact measuring the total loss of DPs plus ACKs.

Experiments were conducted with different levels of background traffic, where the background traffic is defined as additional traffic which is added locally at some fixed rate on *each link* in the network. Background traffic is composed of fixed size packets of 1024 bytes travelling from one end of a link (between two adjacent nodes) to the other. We first ran experiments without any background traffic, meaning that the whole testbed only contained the DPs, SPs and ACKs for a single source to destination connection. Experiments were run without the GA daemon being enabled, and then with the GA daemon. The size of all DPs at the source was fixed at 1024 bytes. We varied the DP rate for a connection between 100 packets/sec to 800 packets/sec. for each traffic rate we ran 10 experiments and in each experiment the source node sent out 10,000 DPs to the destination. The average forward delay and loss rate were computed as an average for each DP transmission rate over all the 10 experiments, and the results (without background traffic) are shown on Fig.5.

We observe that in the absence of background traffic, CPN with the GA outperforms the "regular" CPN. However the improvement is not considerable. When the connection's DP rate is less than 600 packets/sec, the delay with the GA enabled is only 80% of that without GA and the loss rate are both small enough to be considered to be

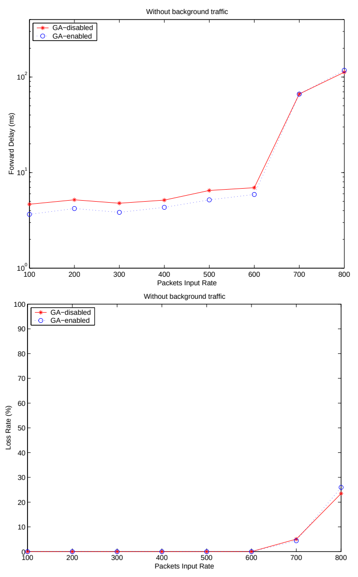

Figure 5. QoS comparison of the system with and without the GA, without background traffic

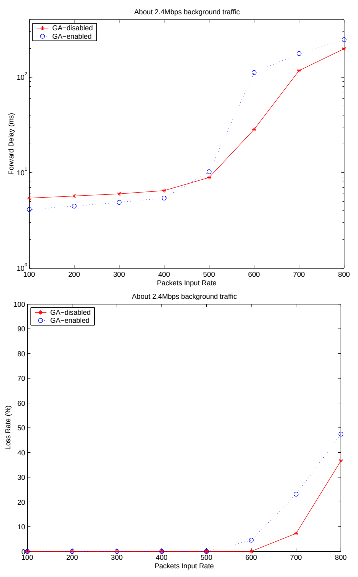

Figure 6. Delay and loss rate for DPs with 2.4 Mbps background traffic on links

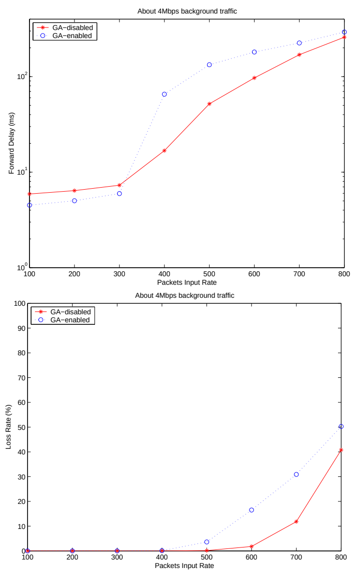

Figure 7. Delay and loss rate with about 4 Mbps background traffic on links

zero. When the DP rate is great than 600 packets/sec, some packet losses are observed and the average delay increases dramatically in both cases. If we only consider the delay, the performance was improved by about 20% due to the introduction of GA. When we increased the background traffic level to 2.4Mbps and to 4Mbps on each link, the results were somewhat disappointing as shown in Figures 6 and 7. The performance of the system with the GA was again better than that of regular CPN when the DP traffic rate was low. When the DP traffic rate exceeded a certain value (e.g. 300 packets/sec in the 4Mbps excess link traffic scenario), the performance of regular CPN was better than that of CPN with the GA. We may explain this as follows. When the load of the network becomes heavier, since the DP traffic has an additive effect on the existing link traffic, the DP traffic will significantly increase the observed delay. SPs will try other routes which do not have DP traffic, and the corresponding ACK will bring back information about paths which are momentarily less loaded. These paths will be immediately used by CPN when the GA is not enabled, leading to path switching and the distribution of the traffic on multiple paths. On the other hand, CPN with the GA will be operating with information which is always "old" because round-trip delays are high, and will therefore recommend paths which may have worse QoS by the time the decision is taken.

When we increase the background link traffic to 8 Mbps, as shown in Figure 8, the network is saturated so that there is little difference that can be made by using or not using the GA: we always have high delay no matter which route the DPs use.

When the network load is light and the packet input rate is low, the Genetic Algorithm helps to reduce the delay of dumb packets by about 20%. When the background traffic becomes heavier, and at high packet input rate, the delay for DPs without the GA is somewhat lower than when the GA is used, and if the network is saturated or when the user traffic saturates its own path, there is almost no difference

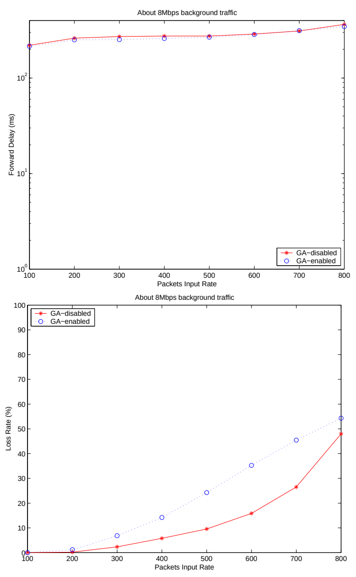

Figure 8. Delay and loss with 8 Mbps background traffic on links

in QoS when the GA is used or not used.

4 Conclusions

In this paper we have described the design of an autonomous and dynamically adaptive network which uses self-measurement and a simple reinforcement learning algorithm to carry out routing decisions. The system uses SPs to search for better paths, ACK packets which bring back information to intermediate routers and to the sources of traffic, and dumb packets which simply carry payload along paths which have been discovered. SPs make decisions at nodes to select paths which provide QoS that users prescribe through “QoS Goals”. We have presented measurements indicating that this scheme can actually respond to different QoS goals. In addition, we have considered the use of a Genetic Algorithm (GA) to create new routes from the information discovered by SPs. A GA daemon at the source node composes new routes from existing routes and selects those that offer a good “fitness” with respect to the desired QoS for the path. Our results with the GA are mixed. The GA daemon significantly improves QoS under light network traffic but not under high traffic conditions. An intuitive explanation is that the GA tends to delay decision making, since it stores more information and makes recommendations based on longer term trends. Thus the GA seems to have the shortcomings of many slower adaptation schemes.

Our ongoing research includes further experimentation with larger, geographically separated network testbeds, interoperating with conventional Internet Protocol (IP) routing algorithms. We are also investigating the use of CPN based techniques to protect sub-networks against malicious attacks. An extension of the CPN architecture to wireless, and mixed wired-wireless networks has been designed and implemented, and is currently being evaluated.

References

[1] E. Gelenbe, R. Lent, Z. Xu, “Towards networks with cognitive packets”, in K. Goto, T. Hasegawa, H. Takagi and Y. Takahashi (eds), “Performance and QoS of next Generation Networking”, Springer Verlag, London, 2001.

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

[3] U. Halici, Reinforcement learning with internal expectation for the random neural network, European Journal of Operations Research, Vol. 126 (2),288–307, 2000.

[4] F. Hao, E.W. Zegura, M.H. Ammar, QoS routing for anycast communications: motivation and an architecture for Diffserv networks, IEEE Communications Magazine, vol. 46, No. 2, 48–56, June 2002.

[5] J.C.S. Lui, X.Q. Wang, Providing QoS guarantee for individual video stream via stochastic admission control, in K. Goto, T. Hasegawa, H. Takagi and Y. Takahashi (eds.), Performance and QoS of Next Generation Networking, 263–279, Springer Verlag, London, 2001.

[6] D.E. Rumelhart, J.L. McClelland Parallel distributed processing vols. I and II, Bradford Books and MIT Press, 1986.

[7] J.H. Holland, Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[8] R.S. Sutton, Learning to predict the methods of temporal difference, Machine Learning, vol. 3, 9–44, 1988.

[9] D.S. Burke, K.A. De Jong, J.J. Grefenstette, C.L. Ramsey, A.S. Wu, Putting more genetics in genetic algorithms, Evolutionary Computation, vol. 6, No. 1, 387–410, 1998.