Cognitive Packet Networks

Erol Gelenbe, *Fellow IEEE*, Zhiguang Xu and Esin Şeref
School of Computer Science
University of Central Florida
Orlando, FL 32816
erol@cs.ucf.edu

Abstract

We propose Cognitive Packet Networks (CPN) in which intelligent capabilities for routing and flow control are concentrated in the packets, rather than in the nodes and protocols. Cognitive packets within a CPN route themselves. They are assigned goals before entering the network and pursue these goals adaptively. Cognitive packets learn from their own observations about the network and from the experience of other packets with whom they exchange information via mailboxes. Cognitive packets rely minimally on routers. This paper describes CPN and shows how learning can support intelligent behavior of Cognitive Packets.

We propose packet switching networks in which intelligence is constructed into the packets, rather than at the nodes or in the protocols. Networks which contain such packets will be called *"Cognitive Packet Networks (CPN)"*. Cognitive packets in CPN route themselves, and learn to avoid congestion and to avoid being lost or destroyed. Cognitive packets learn from their own observations about the network and from the experience of other packets. They rely minimally on routers. Each cognitive packet progressively refines its own model of the network as it travels through the network, and uses the model to make routing decisions. In the most extreme case, a cognitive packet will "know" where it is in the network without asking for the identity of the switch where it is being currently stored, so that packets can be self-routed without relying on the routing algorithms provided by the network nodes. Cognitive packets rely minimally on routers, so that network nodes only serve as buffers, mailboxes and processors. We believe that in future networks, Cognitive Packets may inhabit a packet switching network at the same time as conventional packets. This may be needed, for instance, for certain applications which require a very high level of robustness, or for packets which would have to travel through a particuliarly unreliable part of the network. Cognitive Packet Networks (CPN) depart from traditional packet switching concepts embodied in the Internet, and empahize autonomous intelligent behavior of each individual packet.

Learning algorithms and adaptation have been suggested for telecommunication systems by various authors in the past [4, 7]. However these concepts have not been directly exploited in networks because of the lack of frameworks allowing the decentralized control of communications. CPs store information in their private Cognitive Map (CM) and update the CM and make their routing decisions using the code which is in each packet. This code will include neural networks or other adaptive algorithms which will be decribed below. Figure 1 presents the contents of a Cognitive Packet. The manner in which Cognitive Memory at a Node is updated by the node's processor is shown in Figure 2. In a CPN, the packets use nodes as "parking" or

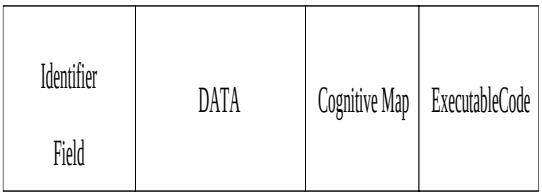

Figure 1. Representation of a CP

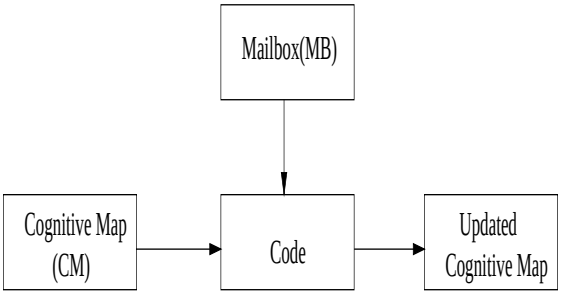

Figure 2. Update of a CP by a Node CPN

resting areas where they make decisions and route them-

selves. They also use nodes as places where they can read their mailboxes. Mailboxes may be filled by the node, or by other packets which pass through the node. Packets also use nodes as processors which execute their code to update their CM and then execute their routing decisions. As a result of code execution, certain information may be moved from the CP to certain mailboxes. The nodes may execute the code of CPs in some order of priority between classes of CPs , for instance as a function of QoS requirements which are contained in the identification field). A possible routing decision may be simply to remain at the current node until certain conditions in the network have changed. However, routing decisions will generally be result in the CP being placed in some output queue, in some order of priority, determined by the CP code execution. A CPN and a CPN node are schematically represented in Figures 3, 4.

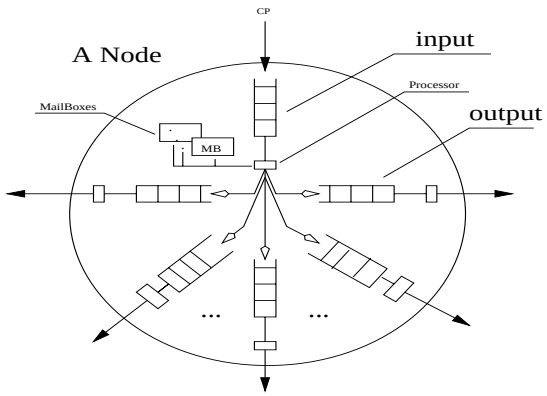

Figure 3. Schematic Representation of a CPN Node

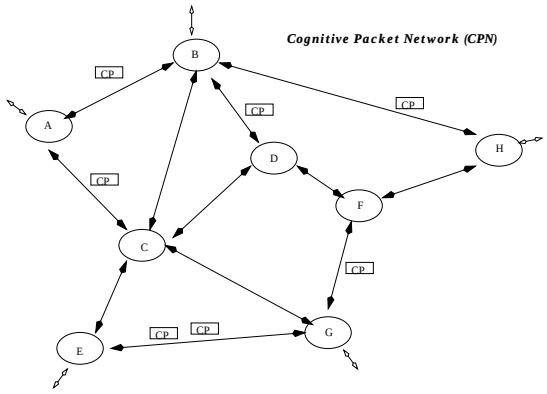

Figure 4. Schematic Representation of a CPN

CPs are grouped into "CP classes" which share similar characteristics such as quality of service requirements, sets of internal states, control rules, input and output signals, etc.. These "signals" are units of information which CPs use to communicate with each other via mailboxes in the nodes. These signals can also emanate from the environment (nodes, existing end-to-end protocols) toward the CPs. Cognitive packets (CP) contain the following fields:

1. The Identifier Field (IF) which provides a unique identifier for the CP, as well as information about the class of packets it may belong to, such as its quality of service (QoS) requirements.

2. The Data Field containing the ordinary data it is transporting.

3. A Cognitive Map (CM) which contains the usual Source and Destination (S-D) information, as well as a map showing where the packet currently "thinks" it is, the packet's view of the state of the network, and information about where it wants to go next; the S-D information may also be stored in the IF.

4. Executable code that the CP uses to update its CM. This code will contain learning algorithms for updating the CM, and decision algorithms which use the CM.

A node in the CPN acts as a storage area for CPs and for mailboxes which are used to exchange data between CPs , and between CPs and the node. It has an input buffer for CPs arriving from the input links, a set of mailboxes, and a set of output buffers which are associated with output links. Nodes in a CPN carry out the following functions:

1. A node receives packets via a finite set of ports and stores them in an input buffer.

2. It transmits packets to other nodes via a set of output buffers. Once a CP is placed in an output buffer, it is transmitted to another destination node with some priority indicated in the output buffer.

3. A node receives information from CPs which it stores in Mailboxes (MB's). MB's may be reserved for certain classes of CPs , or may be specialized by classes of CPs . For instance, there may be different MB's for packets identified by different Source-Destination (S-D) pairs.

4. A node executes the code for each CP in the input buffer. During the execution of the CPs code, the CP may ask the node to decline its identity, and to provide information about its local connectivity (i.e. "This is Node A, and I am connected to Nodes B, C, D via output buffers) while executing its code. In some cases, the CP may already have this information in its CM as a result of the initial information it received at its

source, and as a result of its own memory of the sequence of moves it has made. As a result of this execution:

- The CM's of the packets in the input buffer are updated,
- Certain information is moved from CPs to certain MB's,
- A CP which has made the decision to be moved to an output buffer is transfered there, with the priority it may have requested.

An important issue in the Internet and in future networks is security and dependability, in particular with respect to malicious threats such as viruses, worms, and other types of information warfare threats which may develop in the future. CPs themselves act as autonomous agents and therefore are robust to various forms of network degradation. However our current thinking is that nodes within a CPN could have the power to "clear, or encapsulate, or destroy" packets in a CPN. Each CP could be checked and cleared if it did not represent a threat. A packet which would seem to represent a threat would be "encapsulated" inside a secure packet and routed to its declared destination or to some specific receiving host. In the extreme case, a packet could be simply destroyed or eliminated by the node.

Much attention has been devoted recently to networks which offer users the capability of adding network executable code to their packets. A recent survey article [8] is devoted to the *Active Network* concept; discrete (programmable switches) and integrated (capsules) approaches to the realization of active networks are discussed, and a summary of recent research on active networks is given. The potential impact of active network services on applications and how such services can be built and deployed, are discussed in [10]. It is argued that *Active Network Transport System (ANTS)* solves the problem of slow network service evolution by building programmability in the network infrastructure without sacrificing performance and security. Network services provided by ANTS are flexible in that besides providing IP-style routing and forwarding, applications can introduce new protocols. The packets are in the form of capsules as in integrated active networks. Capsule types that share information are grouped together into protocols, while we group Cognitive Packets which share goals and algorithms. Some specific nodes within the network execute the capsules of a protocol and maintain protocol state, similarly to the manner in which CPN nodes execute the code for each CP. In [11], Active Congestion Control (ACC), a system which uses Active Networking technology to reduce the control delay that feedback congestion control systems experience, is introduced. Every packet in the system includes the current state of the endpoint's feedback algorithm. Contrary to CPN where the packets change their behavior according to the state of the network, in ACC the nodes change their behavior. A practical framework which enables the addition of user code to the network element as part of the normal operation of the network is presented in [12], allowing new functionality to be rapidly introduced into the network. Advanced and conventional control architectures can exist together, solving the problem of retaining existing network solutions while at the same time creating innovative control systems for new services. In [13], the authors describe an object-oriented transport architecture that allows for dynamically binding a variety of protocol stacks on a per-call basis. In [14], a reference model is defined which separates control intelligence from control mechanisms. The IEEE P1520 standard aims to establish an open architecture in network control, and provide the capability to program the network through the programming interface. The basic concepts and nomenclature for talking about active networks, and various aspects of the architecture being developed in the Defense Advanced Research Agency (DARPA) program are described in [15]. The functionality of the active network node is divided between the execution environments (EEs) and the node operating system (NodeOS). The EE is responsible for implementing the network API, while the NodeOS manages access to local node resources by EEs. Protocol Booster in [16] is a novel methodology for protocol design aimed at overcoming the slow evolution and inefficiencies associated with general-purpose protocols. It incrementally constructs protocols from elements called *protocol boosters* on an as-needed basis. *Protocol boosters* are transparent to the protocol being boosted. They can reside anywhere in the network or end systems and are designed to improve the performance or features of an existing protocol. Safety and security are major concerns in [17]. The *Secure Active Network Environment (SANE)* architecture, which provides a means of controlling access to the functions provided by any programmable infrastructure, is illustrated in this article. The JAVA programming language and virtual machines combined with the Web as a platform for implementing and deploying protocols offers enticing features including portability, security and object-oriented capabilities [18]. The *Common Object Request Broker Architecture (CORBA)* and the *Distributed Component Object Model (DCOM)* [19] are chosen as two examples of Distributed Object Technology that facilitate open network interfaces. The benefits are abstraction, location independence, modularity, and software reusability. In [9], the impact of mobile agent technology on telecommunication service environments, influenced by the *Intelligent Network* architecture is discussed.

1 Adaptation by cognitive packets

Each cognitive packet starts with a given representation of the network from which it then progressively constructs its own Cognitive Map of network state and uses it to make routing decisions. Learning paradigms are used by CPs to update their CM and reach decisions using the packet's prior experience and the input provided via mailboxes. In the adaptive approach we propose for CPs, each packet entering the network is assigned a Goal before it enters the network, and the CP uses the goal to determine its course of action each time it has to make a decision. For instance if the CP contains part of a telephone conversation, a typical goal assigned to the packet might be: "Go from source S to destination D in minimum time". A more sophisticated goal in this case could be: "go from S to D in minimum time, but do not overtake any packets of the same sequence which left before you", since voice packets need to be used at the receiver in the sequence they were transmitted. On the other hand, if these were data packets, the goal may simply be: "go from S to D without getting lost or destroyed". In our work, these goals are translated into numerical quantities (e.g. delay values, loss probabilities, and weighted combinations of such numerical quantities) which are then used directly in the adaptation. A simple approach to adaptation is to respond in the sense of the most recently available data. Here the CPs cognitive memory contains data which is updated from the contents of the node's mailbox. After this update is made, the CP makes the decision which is most advantageous (lowest cost or highest reward) simply based on this information; we will call this approach the Bang-Bang algorithm. In this paper we will also use two learning paradigms for CPs.

Learning feedforward random neural networks (LFRNN) [6, 21]. These networks update their internal representation (the weights) using gradient based algorithms to improve either their predictive capabilities, or to improve their ability to reach decisions which elicit the maximum reward. *Random neural networks with reinforcement learning (RNNRL)* [6, 22]. In this case a recurrent network is used both for storing the CM and making decisions. The weights of the network are updated so that decisions are reinforced or weakened depending on whether they have been observed to contribute to increasing or decreasing the accomplishment of the declared goal. *Adaptive Stochastic Finite-State Machines (ASFSM)* [3, 4, 7] are another class of adaptive models which we will investigate in future work. It is known that by introducing randomness in a finite state machine's transitions, one obtains a computational model which is as powerful as a Turing machine [1]; such models can be easily implemented using deterministic finite-state machines and random number generators [2]. SFSM's can implement complex computational decision processes, and learning is achieved by updating the state transition probabilities so as to maximize expected rewards associated with decisions. State transitions are then carried out probabilistically using these updated transition rules. A description of the RNN and the related algorithms used in this paper are presented in Section 2.

2 The Random Neural Network and related algorithms

For the reinforcement learning approach to CP adaptation, as well the feed-forward neural network predictor, we have used the RNN [6] is an analytically tractable spiked random neural network model whose mathematical structure is akin to that of queueing networks. It has "product form" just like many useful queueing network models, although it is based on non-linear mathematics. The state q_i of the $i-th$ neuron in the network is the probability that it is excited. These quantities satisfy the following system of non-linear equations:

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

where

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

Here w^+_{ij} is the rate at which neuron i sends "excitation spikes" to neuron j when i is excited, and w^+_{ij} is the rate at which neuron i sends "inhibition spikes" to neuron j when i is excited. 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. Various techniques for learning may be applied to the RNN. These include Hebbian learning (which will not be discussed here since it is too slow and relatively ineffective with small networks), and Reinforcement Learning and gradient based learning which are used in this paper.

There are many different ways to introduce Reinforcement Learning in the RNN model. In this paper we have used what a simple approach [22] which was originally suggested for navigation in a maze. The simulations in the next section have revealed that this simple approach appears effective for autonomous routing of the CPs. Given some Goal G that the CP has to achieve as a function to be to be minimized (i.e. Transit Delay or Probability of Loss, or a weighted combination of the two), we formulate a reward R which is simply $R = G^{-1}$. Successive measured values of the R are denoted by $R_l, l = 1, 2, ..$ These are first used to compute a decision threshold:

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

where a is some constant $0 < a < 1$, typically close to 1. Now a RNN with (at least) as many nodes as the decision outcomes is constructed. Let the neurons be numbered $1, \ldots, n$. Thus for any decision i, there is some neuron i. Decisions in this RL algorithm with the RNN are taken by selecting the decision j for which the corresponding neuron is the most excited, i.e. the one with has the largest value of q_j. Note that the $l-th$ decision may not have contributed directly to the $l-th$ observed reward because of time delays between cause and effect. Suppose that we have now taken the $l-th$ decision which corresponds to neuron j, and that we have measured the $l-th$ reward R_l. Let us denote by r_i the firing rates of the neurons before the update takes place. We first determine whether the most recent value of the reward is larger than the previous "smoothed" value of the reward which we call the threshold T_{l-1}. If that is the case, then we increase very significantly the excitatory weights going into the neuron that was the previous winner (in order to reward it for its new success), and make a small increase of the inhibitory weights leading to other neurons. If the new reward is not better than the previously observed smoothed reward (the threshold), then we simply increase moderately all excitatory weights leading to all neurons, except for the previous winner, and increase significantly the inhibitory weights leading to the previous winning neuron (in order to punish it for not being very successful this time). This is detailed in the algorithm given below.

We compute T_{l-1} and then update the network weights as follows for all neurons $i \neq j$:

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

Then we re-normalize all the weights by carrying out the following operations, to avoid obtaining weights which indefinitely increase in size. First for each i we compute:

$$r_i^* = \sum_1^n [w^+(i,m) + w^-(i,m)], \tag{4}$$

and then re-normalize the weights with:

$$w^+(i,j) \leftarrow w^+(i,j) * \frac{r_i}{r_i^*},$$
$$w^-(i,j) \leftarrow w^-(i,j) * \frac{r_i}{r_i^*}\ .$$

Finally, the probabilities q_i are computed using the non-linear iterations (1), 2), leading to a new decision based on the neuron with the highest probability of being excited.

3 Simulation of a cognitive packet network

The purpose of our simulation was to see whether algorithms which can be implemented inside the CPs can result in delay and loss performance improvements. Both very simple decision algorithms (such as the Bang-Bang algorithm described below), and more sophisticated algorithms using learning, were tested. CPs used three different paradigms for adaptation, under identical traffic conditions. A single network simulation program was written, and three different learning algorithms were used by the CPs. The CPN was chosen to be quite large with 100 nodes, and we chose a locally interconnected rectangular grid topology as shown in Figure 5. All link speeds were normalized to 1, and packets were allowed to enter and leave the network either from the top ten nodes or the bottom ten nodes. Traffic arrival into the network was taken to be Poisson. Each packet's destination was one of the nodes at the opposite end of the network, and the destination node for each packet were chosen to be fixed when the packet enters the network, and drawn to be equally likely among all possible 10 destination nodes. Buffers in each node are of unlimited capacity so that blocking or loss is not tied to congestion. Packet loss was simulated probabilistically at all nodes with a small fixed probability of loss throughout all but a few specific nodes where there is a high packet loss probability; these latter nodes are not known in advance by the packets. Congestion at a node can be caused by the normal packet traffic and routing, or by packets which route to certain nodes because of congestion elsewhere in the network. It can also occur when packets remain in a "safe" node due to the risk of congestion or loss in other parts of the network. All the packets were assigned a common goal, which was to minimize a weighted combination of delay (W) and loss (L) which we write:

$$G = \alpha W + \beta L. \tag{5}$$

All the algorithms are allowed to use three items of information which are deposited in the nodes' mailboxes: (1) the length of the local queues in the node, (2) recent values of the downstream delays experienced by packets which have previously gone through the output links and reached their destinations, (3) the loss rate of packets which have passed through the same node and gone through the output links, (4) estimates made by the most recent CPs which have used the output links headed for some destination d of its estimated delay D_d and loss L_d from this node to its destination. The value D_d is updated by each successive CP passing through the node and whose destination is d as discussed below. Only a fraction of the packets to any destination are marked for monitoring so that we do not need to keep track of all packets; the departure time-stamps of these marked packets are stored in each node's mailboxes, and the arrival dates to destination for the same marked packets ar-

rive via acknowledgement (ACK) packets sent back by the destination nodes. These two times used to reconstruct the downstream delays for each node. In the Bang-Bang algorithm, the CPs read the mailboxes and for each destination compute an estimated running average delay of the form $W_d \leftarrow aW_d + (1-a)V_d$ where V_d is the most recently available downstream delay value to detination d based on the current decision. Similar information is collected and updated for packet loss l_d. The CP then makes an assessment of what a "reasonable" value of downstream Loss L_d and Delay D_d should be to its destination d, based on its knowledge of where it currently is in the network, of its distance to the destination node, and using the output queue lengths at the node. If either $L_d < l_d$ or $D_d < W_d$, then the CP selects at random any other output link which is on some path to the destination d.

Throughout the simulation results, we vary the arrival rates of packets to each input node between 0.1 and 1. All simulations compare the CPs with controls of different kinds, against a static routing policy (marked "No Control" on the figures) where the packet is routed along a static shortest path to the output layer of nodes, and then horizontally to its destination. Clearly, this static algorithm could also be implemented in a CPN. However the adaptive algorithms we have simulated allow each individual CP to carry out a separate computation and make its own individual decisions; thus these adaptive algorithms are a better illustration of the capabilities which can be implemented in a CPN. In these first simulations, lost packets are not retransmitted by the source nodes.Loss rates at most of the the network nodes are set to a value of 10%, while the rate is 50% at two specific areas which are unkown to the CPs. These "high loss" areas are in four continiguous nodes given in (x,y) coordinates as (2,0) to (2,3), and also in four other nodes (7,7) to (7,10). These values are very high in practical terms, but are selected at these high values simply to be able to illustrate to be able to observe the effect of the control algorithms. Simulations results which compare the Bang-Bang approach to a fixed shortest-path type routing of all packets are shown in Figures 6, 7, 8, 9. We see that when the goal is to minimize the delay (Figures 6, 7), the average delay of packets with control is lower than without control (Figure 6), while we have the opposite effect on loss (Figure 7) as would be expected. When we use a combination of loss and delay, the particular values chosen of α and β lead to the results observed in Figures 8, 9 where average delay is just a little worse than if there were no control, while there is close to $70-80\%$ reduction in packet loss showing that the CPs are autonomously avoiding regions of high packet loss. Figures 10 and 11 pursue the simulation results under the same conditions as in the four previous figures. Here we compare the RNN with Reinforcement Learning (RL) and Bang-Bang when the Goal includes both Loss and Delay. We see that RL and the Bang-Bang algorithm provide essentially equivalent performance. This has also been noticed in many other simulations which we do not report here. We then examine the use of a RNN which is used to predict the Delay and Loss quantities. Recent samples of these quantities are used to train a feedforward RNN, and the network is then used to make decisions for each CP. The network is trained to predict the numerical value of the Goal, and the decision is made by choosing the outcome which has the smallest numerical value of the Goal. This method is "indirect" in that it trains and uses the RNN as a predictor, while the RL algorithm trains the RNN directly as the decision making element. Figures 12 and 13 present results of the simulation runs when CPs use this approach to control routing. We clearly see that this approach does improve performance over the simulations with "No Control", but that it is not as effective as RL or the Bang-Bang algorithm which both bypass the construction of a predictor for decision making.

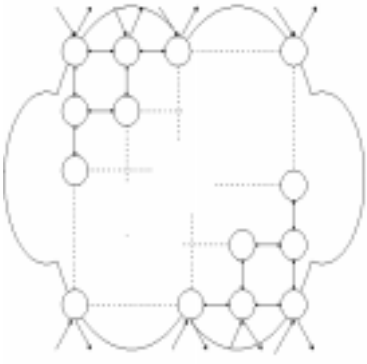

Figure 5. The Simulated CPN Topology

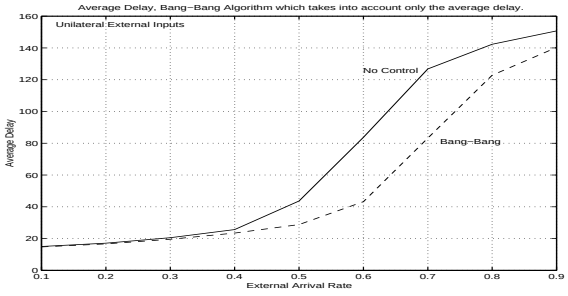

Figure 6. Bang-Bang Control based on Estimated Delay. Comparison of Average Delay through the CPN with High Loss

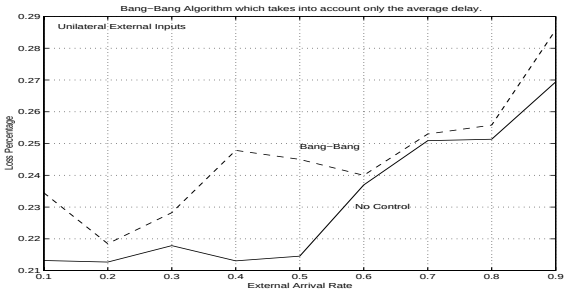

Figure 7. Bang-Bang Control based on Estimated Delay. Comparison of Average Loss through the CPN with High Loss

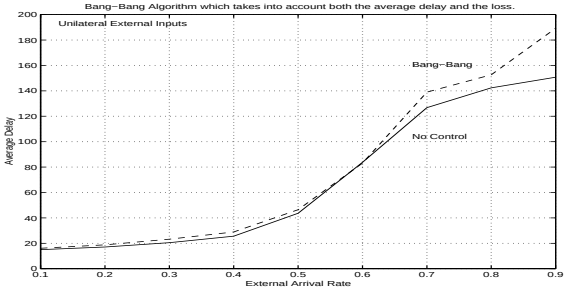

Figure 8. Bang-Bang Control based on combined Estimated Delay and Estimated Loss. Comparison of Average Delay through the CPN with High Loss

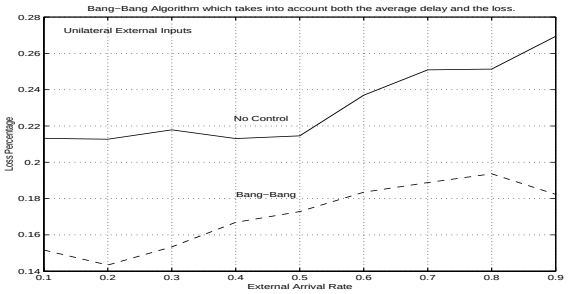

Figure 9. Bang-Bang Control based on Delay and Loss. Comparison of Average Loss through the CPN with High Loss

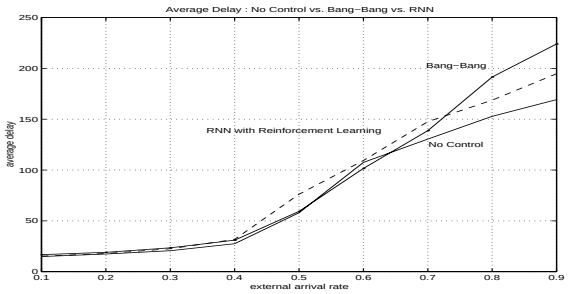

Figure 10. Reinforcement Learning Based Control using Delay and Loss as the Goal. Comparison of Average Delay through the CPN with High Loss

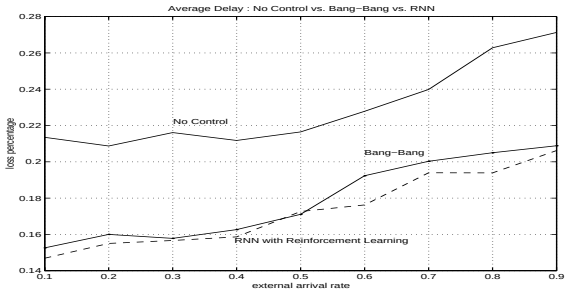

Figure 11. Reinforcement Learning Based Control using Delay and Loss as the Goal. Comparison of Average Loss through the CPN with High Loss

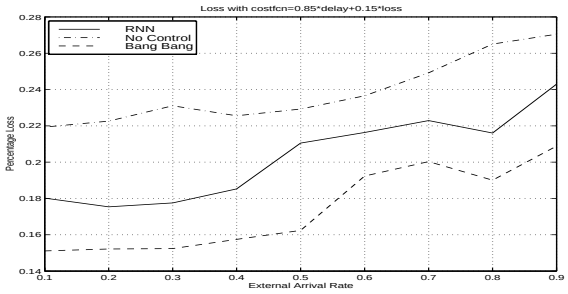

Figure 12. Feedforward Neural Network Learning Based Control using Delay and Loss as the Goal. Comparison of Average Delay through the CPN with High Loss

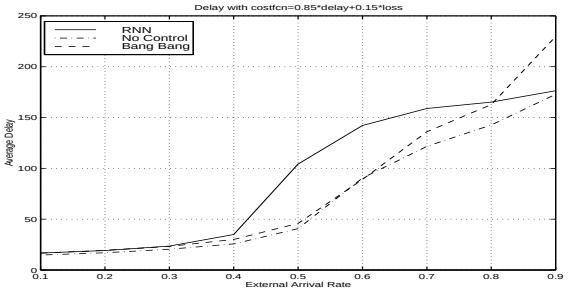

Figure 13. Feedforward Neural Network Learning Based Control using Delay and Loss as the Goal. Comparison of Average Loss through the CPN

References

[1] E. Gelenbe "Probabilistic automata with structural restrictions", SWAT 1969 (IEEE Symp. on Switching and Automata Theory), also appeared as "On languages defined by linear probabilistic automata," *Information and Control*, Vol. 18, February 1971.

[2] E. Gelenbe "A realizable model for stochastic sequential machines," *IEEE Trans. Computers*, Vol. C-20, No. 2, pp. 199-204, February 1971.

[3] R. Viswanathan and K.S. Narendra "Comparison of expedient and optimal reinforcement schemes for learning systems," *J. Cybernetics*, Vol. 2, pp 21-37, 1972.

[4] K.S. Narendra and P. Mars, "The use of learning algorithms in telephone traffic routing - a methodology," *Automatica*, Vol. 19, pp. 495-502, 1983.

[5] R.S. Sutton "Learning to predict the methods of temporal difference", *Machine Learning*, Vol. 3, pp. 9-44, 1988.

[6] E. Gelenbe (1993) "Learning in the recurrent random neural network", *Neural Computation*, Vol. 5, No. 1, pp. 154-164, 1993.

[7] P. Mars, J.R. Chen, and R. Nambiar, *Learning Algorithms: Theory and Applications in Signal Processing, Control and Communications* , CRC Press, Boca Raton, 1996.

[8] D. L. Tennenhouse, J. M. Smith, D. W. Sincoskie, D. J. Wetherall, G. J. Minden, "A survey of Active Network research," *IEEE Comm. Magn.*, Vol. 35, no. 1, pp. 80-86, January 1997.

[9] M. Bregust, T. Magedanz, "Mobile agents-enabling technology for Active Intelligent Network implementation," *IEEE Network Magn.*, Vol . 12, no. 3, pp. 53-60, May/June 1998.

[10] D. Wetherall, U. Legedza, J. Guttag, "Introducing new Internet services: Why and How," *IEEE Network Magn.*, Vol . 12, no. 3, pp. 12-19, May/June 1998.

[11] T. Faber, "ACC: Using Active Networking to enhance feedback congestion control mechanisms," *IEEE Network Magn.*, Vol . 12, no. 3, pp. 61-65, May/June 1998.

[12] S. Rooney, Jacobus E. van der Merwe, S. A. Crosby, I. M. Leslie, "The Tempest: a framework for safe, resource-assured, programmable networks," *IEEE Communications*, Vol. 36, No. 10, Oct. 1998.

[13] J.-F. Huard, A. A. Lazar, "A programmable transport architecture with QoS guarantee," *IEEE Communications*, Vol. 36, No. 10, pp. 54-63, Oct. 1998.

[14] J. Biswas, A. A. Lazar, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Wang, S. Weinstein, "The IEEE P1520 standards initiative for programmable network interface," *IEEE Communications*, Vol. 36, No. 10, pp. 64-72, Oct. 1998.

[15] K. L. Calvert, S. Bhattacharjee, E. Zegura, J. Sterbenz, "Directions in Active Networks," *IEEE Communications*, Vol. 36, pp. 64-72, No. 10, Oct. 1998.

[16] W. Marcus, Ilija Hadzic, Anthony J. McAuley, J. M. Smith, "Protocol boosters: applying programmability to network infrastructures," *IEEE Communications*, Vol. 36, No. 10, pp. 79-83, Oct. 1998.

[17] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, J. M. Smith, "Safety and security of programmable networks infrastructures," *IEEE Communications*, Vol. 36, No. 10, pp. 84-92, Oct. 1998.

[18] B. Krupczak, K. L. Calvert, M. H. Ammar, "Implementing communication protocols in Java," *IEEE Communications*, Vol. 36, No. 10, pp. 93-99, Oct. 1998.

[19] J.-P. Redlich, M. Suzuki, S. Weinstein, "Distributed object technology for networking," *IEEE Communications*, Vol. 36, No. 10, pp. 100-111, Oct. 1998.

[20] M. Falousos, A. Banerjea, R. Pankaj, "QoSMIC : Quality of Service sensitive multicast Internet protocol," *Computer Communications Review*, SIGCOMM'98, Section 4, Quality of Service, pp. 281-190, 1998.

[21] E. Gelenbe, Zhi-Hong Mao, Y. Da-Li (1999) "Function approximation with spiked random networks" *IEEE Trans. on Neural Networks*, Vol. 10, No. 1, pp. 3–9, 1999.

[22] U. Halici, "Reinforcement learning with internal expectation for the random neural network," *European Journal of Operations Research* (in press).