Design and Performance of Cognitive Packet Networks

June 8, 2000

Abstract

Based on our earlier work [2], we discuss packet networks in which intelligent capabilities for routing and flow control are concentrated in the packets, rather than in the nodes and protocols. This paper describes a possible test-bed to test and evaluate their capabilities, and presents an analytical model for the worst and best case performance of such systems.

1 Introduction

In earlier work [2] we have proposed packet networks in which intelligence is constructed into the packets, rather than at the nodes or in the protocols. Such networks are called "Cognitive Packet Networks (CPN)". Cognitive packets route themselves, they learn to avoid congestion and to avoid being lost or destroyed. They learn from their own observations about the network and from the experience of other packets. They rely minimally on routers. CPNs carry three major types of packets: smart packets, dumb packets and acknowldegments (ACK). Smart packets serve as "explorers" for different source-destination pairs; they rely minimally on routers, so that network nodes only serve as buffers, mailboxes and processors. Smart packets can also be hierarchically structured so that a group of packets share the same goals, and make use of each other's experience. Upon arrival of a smart packet at its destination, the destination node creates an acknowledgement, which will follow the inverse of the route recorded by the smart packet on its way to the destination. The acknowledgement informs the source about the path that should be followed by dumb packets on their way to this specific destination. Dumb packets are given the path to follow to their destination by the source. We use the term Cognitive Packet (CP) to refer to smart packets.

Much attention has been devoted recently to networks which offer users the capability to add network executable code to their packets. Some of these ideas can be used to support a CPN. 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, as 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, similar to the manner in which CPN nodes execute the

code for each CP. The capsule processing routines are automatically and dynamically transferred to the nodes where they are needed. Contrary to CPNs where the code is a field of the CP, in ANTS this is done by a code distribution mechanism.

All CPs have the same packet format, which consist of four sections: a header carrying administrative information for handling the packet, such as quality of service (QoS) requirements, the type of packet and the source and destination addresses; a Cognitive Map (CM), which is used to compute CP routing based on the packets QoS needs; finally executable code, and a payload section. Smart packets make their routing decisions using the executable code which will include neural networks or other adaptive algorithms. The manner in which a CP's Cognitive Memory is updated by the node's processor is shown in Figure 1. Dumb packets follow the paths discovered by the CPs. In a CPN, the packets use nodes as "parking" areas where

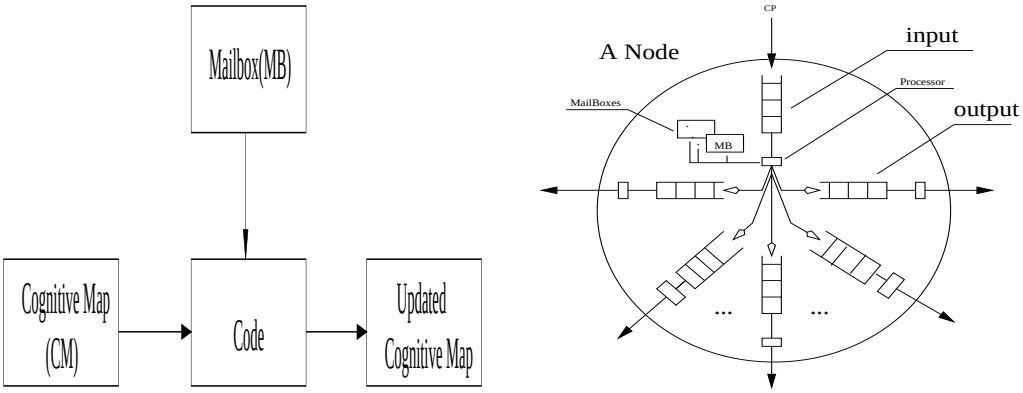

Figure 1: Update of a CP by a Node CPN (left) and contents of a CPN node (right)

they stop to make decisions and route themselves. They also use nodes as places where they can read their mailboxes. Mailboxes are updated by packets which pass through the node, and in particular by ACKs. Packets use nodes as processors which execute their code to update their CM and then execute their routing decisions. 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. Routing decisions will generally result in the CP being placed in some output queue, in some order of priority, determined by the CP's code Each CP 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. In [2] we use two learning paradigms for CPs :*Learning feedforward random neural networks (LFRNN)* [6]; 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, and *Random neural networks with reinforcement learning (RNNRL)* [6, 11]. In the latter 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 how they have been observed to contribute to increasing or decreasing the accomplishment of the declared goal.

2 Implementation of a CPN test-bed

We are currently implementating a test-bed to show the effectiveness of the CPN concept in a realistic environment. The test-bed will consist of a network of CPN nodes where we may run current applications and further develop the capabilities of CPN. The CPN code is being

implemented in the latest Linux kernel (version 2.2.13). The Linux kernel support for low cost PCs and a growing number of platforms, the freely availability of its source code, and the module support in the kernel, makes Linux an attractive system for the development of a project of this nature. A clear separation between networking protocols in the implementation will make the CPN code independent of the physical layer and the data-link layer, providing flexibility in the interoperation with other existing communication protocols in the kernel. The goal is to produce a simple and compact code that can be easily ported over a wide variety of platforms, ranging from small single-board machines to supercomputers. The networking code in the Linux kernel, and many other operating systems, consists of three layers of software: device drivers, network interface, and protocol layer (see Figure 2). Device drivers perform the I/O operations in physical devices, providing a simplified interface to the protocol layer in the kernel. The network interface is compatible with the popular BSD4.3 socket layer in Linux, and provides a single application program interface (API) for the programmer to access all the protocols in the system. Sockets can be viewed as pipes where information that go into one end, come out at the other end. The socket concept support several types of services under the client-server model. The two principal are either connection-oriented, i.e. allowing the flow of data going orderly in both directions; or connectionless (also known as datagram sockets) allowing the transmission of only one message at a time. The protocol layer consists of several families of protocols: INET for TCP/IP, UNIX for Unix interprocess communications, etc. Data that arrive at this level either from a user application through the socket interface, or from the physical network via a device driver, have an identifier specifying which network protocol they carry. The core of the CPN protocol is being implemented in this layer, and it is able to receive, rewrite, discard and create new packets according to the CPN algorithm. Packets addressed to the local host are passed up to the socket interface. Packets addressed to a remote site are sent to the appropriate device driver for transmission. The communication between the CPN layer with the socket interface and the device drivers is performed using *skbuff* data structures. A *skbuff* structure contains pointers to a certain buffer area where packets are processed. Using this structure, the headers for each layer can be added or removed as needed while the packet goes up or down in the networking protocol stack. Incoming CPN packets are tagged with a unique identifier so that the receiving device driver can use this number to identify the CPN packets. Arriving CPN packets are delivered to the CPN receiving function where the routing decision is performed based on the information stored in the MBs and the CM. When a CPN node receives a new packet it checks the type of packet. If the packet is dumb and the destination local, then an upper layer protocol process the packet. If the destination if not local then the packet is delivered using the information stored in its CM. When a Smart packet is received with a destination address different to the local node, the code stored in the data section is executed. The Smart packet stores information about the current node in its private CM and then takes a path to another node. On the other hand if the current node is the final destination for the Smart packet, then an Ack packet is generated and sent back to the origin with all the information collected by the Smart packet. Each time a CPN node receives an Ack packet the internal MB is updated with the new information.

3 Modeling and Simulation of CPN behavior

The purpose of our modeling and simulation work is to compare and evaluate a variety of CP learning algorithms. Both very simple decision algorithms (such as the Bang-Bang algorithm described below), and more sophisticated algorithms using learning, were tested. CPs used three

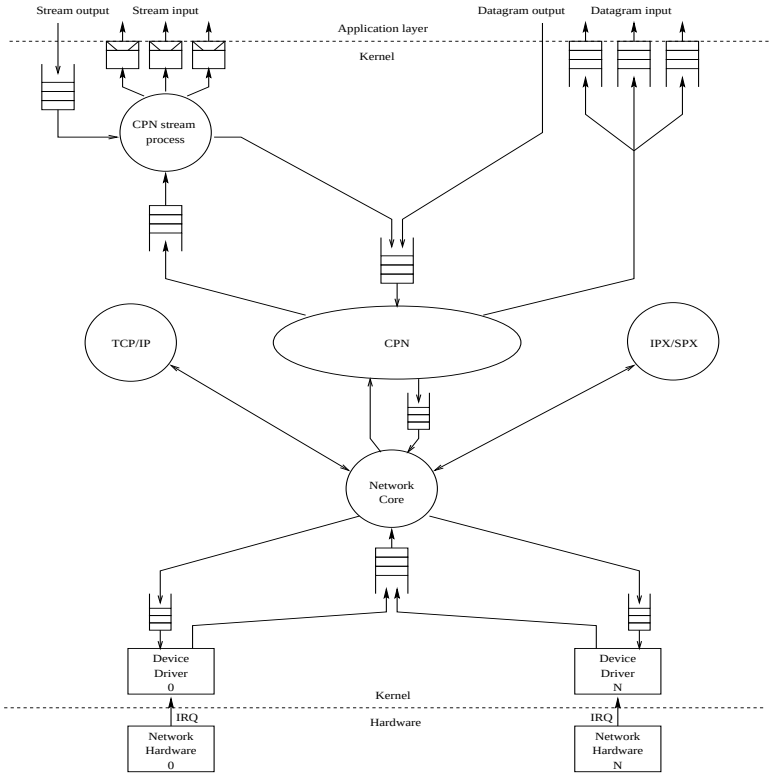

Figure 2: CPN code in the Linux Kernel

different paradigms for adaptation, under identical traffic conditions. A single network simulation program representing a rectangular 100 node network was simulated, and three different learning algorithms were used by the CPs. Throughout the simulations, 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. In our 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. Figure 3 compares 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.

3.1 Worst-Case and Best-Case Performance

The worst case performance is obtained by considering "smart" packets which simply try to find the route to their destination by moving at random, with two constraints related to the topology of the network which will be discussed below. The network topology that we use to evaluate the worst case is very similar to the one which we have used in our simulations. It has a cylindrical toplogy with "R" circles or rows, each containing "a" nodes, making up the cylinder. The only

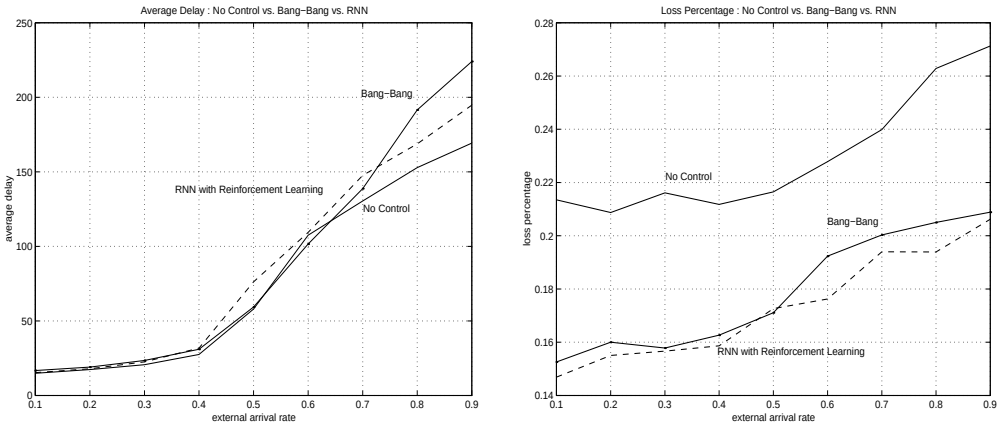

Figure 3: RL Based Control using Delay and Loss as the Goal. Comparison of Average Delay through the CPN with High Loss (left), Comparison of Average Loss through the CPN with High Loss (right).

difference with the network in the simulations is that because we take a cylindrical topology, there are no "edge" nodes in cylinder we use to construct the analytical model. Each node on the two (top and bottom) "end" circles serves both as a source and as a destination for packets. The two routing constraints for packets, which we mentioned above in relation to the topology of the network, are:

- A smart (or dumb) packet originating at the top row never heads upwards, and if it originates at the bottom row it never heads downwards.
- A smart packet which is one link away from its destination will directly go to its destination without any further random search. This is because in a real network, the outgoing link of a node will carry information concerning the identity of the node which is at the other end of the link.

Our analysis provides the following results under the assumption that packets at any source may head to any destination, and that smart packets do not know the direction they should head in, except that they need to go from the top row to the bottom row (or vice versa).

- Average Number of Nodes Visited in Row i by a Smart Packet When There Are No Losses

$$A(i) = \begin{cases} \frac{1}{f_i} & i \leq i \leq R-2 \\ \frac{a}{(1+(a-1)f_i)} & i = R-1 \\ \frac{a}{2} & i = R \end{cases}$$

  where f_i is the probability that a smart packet leaves a node in row i to move to the next row.

- Average Number of Nodes Visited in Row i by a Smart Packet with Losses at each Node

$$A_l(i) = \begin{cases} \frac{1}{l_i+f_i(1-l_i)} & i \leq i \leq R-2 \\ \frac{1}{1-(1-f_i)(1-l_i)(1-\frac{1}{a})} & i = R-1 \\ \frac{a}{2+(a-2)l_i} & i = R \end{cases}$$

  where l_i is the probability that a loss can occur at some node in row i of the network.

- The Probability that a Smart Pcket Is Lost as It Traverses Row i

$$\pi(i) = 1 - \frac{A_l(i)}{A(i)},\ 1 \leq i \leq R$$

- The Probability that a smart packet eventually enters row i

$$P_e(i) = \begin{cases} 1 & i = 1 \\ (1 - \pi(i-1)) \cdot P_e(i-1) & 2 \leq i \leq R \end{cases}$$

- The Average Number of Times that a Randomly Selected Smart Packet Visits a Node In the row i

$$e_l(i) = P_e(i) \cdot \frac{A_l(i)}{a}, \quad 1 \leq i \leq R$$

  under the assumption that the traffic in the network is homogeneous.

- The Probability that a Smart Packet Is Lost as It Traverses the Network

$$P_l^s = 1 - \Pi_{i=1}^{R}(1 - \pi(i))$$

- The Effective Traffice from S to D

$$\lambda(S, D) = \frac{\lambda^0(S, D)}{1 - P_l^s}$$

  where $\lambda^0(S, D)$ is the offered S to D traffic.

- The Average Traffic of Smart Packets Entering a Node in Row i
  - For unilateral traffic going just from the top to the bottom of the network

$$\lambda_s(i) = \sum_S \sum_D e_l(i) \cdot \lambda(S, D)$$

  - For symmetric bilateral traffic going both from top to bottom and vice versa

$$\lambda_s(i) = \sum_S \sum_D (e_l(i) + e_l(R - i + 1)) \cdot \lambda(S, D)$$

- The Average Number of Smart Packets at a Node in Row i

$$R(i) = \frac{\lambda_s(i)}{\gamma - \lambda_s(i)}$$

  where γ is the average service rate at each Node.

- The Overall Average Delay of Smart Packets
  - For unilateral traffic going just from the top to the bottom of the network

$$r = \frac{\sum_i a \cdot R(i)}{\sum_S \sum_D \lambda^o(S, D)}$$

– For symmetric bilateral traffic going both from top to bottom and vice versa

$$r = \frac{\sum_i a \cdot R(i)}{\sum_S \sum_D \lambda^o(S,D) \cdot 2}$$

To illustrate these results, we have varied f_i, the probability a smart packet leaves a node in row i to move to the next row. The numerical results of the leftmost graph in Figure 4 show that the larger the value of f_i, the smarter the packets are, and consequently, the smaller the average response time R.

The *best case* performance can be achieved if the following three conditions are satisfied:

- The traffic load is evenly distributed among all the nodes in the network;
- Each packet takes the shortest path from its source to its destination under the assumption that the number of nodes visited by a packet is the dominant factor among those that determine its delay;
- There is no packet loss.

Take an arbitrary packet, let "s" and "d" represents its source and destination respectively. The length of the shortest path that it can possibly take is $M = |d - s| + R$ with expected value $\overline{M} = \frac{a}{2} + R$. Since all nodes are equally loaded, the packet arrival rate to each node is

$$\lambda_n = \frac{2a\lambda^0(S,D)\overline{M}}{aR} = (\frac{a}{R} + 2)\lambda^0(S,D)$$

where $\lambda^0(S,D)$ is the offered S to D traffic. Similar to the previous worst case analysis, now we can adopt the queueing theory to obtain the best case average packet delay:

$$r = \frac{aR\frac{\lambda_n}{\gamma - \lambda_n}}{2a\lambda^0(S,D)} = \frac{\frac{a}{2} + R}{\gamma - (\frac{a}{R} + 2)\lambda^0(S,D)}$$

The following two figures conclude our smart packet analysis. The one on the left shows the best case average packet delay and the one on the right illustrates the performance comparison among the best case, the worst case and the simulation (RNN with reinforcement learning) in terms of the average packet delay.

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, E. Seref, Z. Xu " Towards networks with intelligent packets, "*Proceedings of the Fourteenth International Symposium on Computer and Information Sciences*, pp. 1-11, Oct. 1999.

[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.

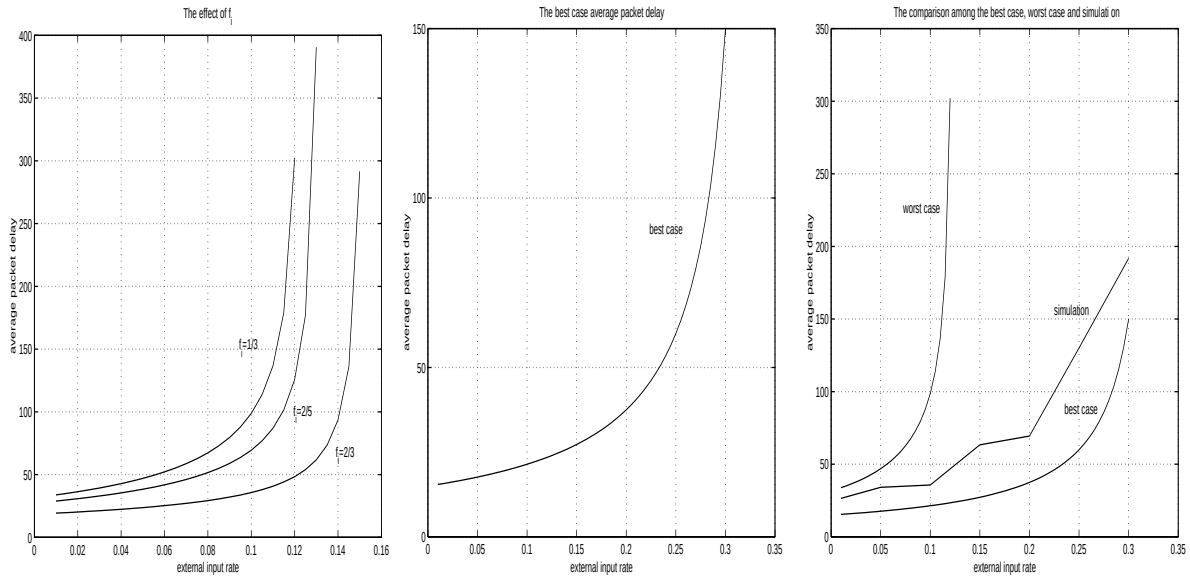

Figure 4: Comparison between the Worst Case (left), Best Case (middle), and the Analytical Worst and Best Cases Compared to RNN-RL Learning Based Simulation (right)

[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] K. L. Calvert, S. Bhattacharjee, E. Zegura, J. Sterbenz, "Directions in Active Networks," *IEEE Communications*, Vol. 36, pp. 64-72, No. 10, Oct. 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] U. Halici, "Reinforcement learning with internal expectation for the random neural network," *European Journal of Operations Research* (in press).

[12] E. Gelenbe "Queueing networks with negative and positive customers", *Journal of Applied Probability*, Vol. 28, 656-663 (1991).

[13] E. Gelenbe, A. Labed "G-networks with multiple classes of signals and positive customers", *European Journal of Operations Research*, Vol. 108 (2), pp. 293-305, July 1998.