Design and Performance of Networks with Cognitive Packets

Erol Gelenbe, Ricardo Lent, Zhiguang Xu
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
{erol,rlent,zgxu}@cs.ucf.edu

October 12, 2000

Abstract

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

We propose 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 acknowledgments (ACK).

This paper reviews the basic concepts of CPNs, and proposes techniques for packet-based autonomous learning for routing and discuss flow control using adaptive finite-state machines and random neural networks to support these concepts. We describe a possible test-bed to test and evaluate their capabilities, and present analytical models for the worst and best case performance of these networks.

1.1 Networks with Packet-Based Processing Capabilities

Much attention has been devoted recently to networks which offer users the capability of adding 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, 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. 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.

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. When a router experiences congestion, it calculates the new window size and deletes the packets that the endpoint would not have sent and informs the endpoint about its new state. 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. The architecture, in which the atomic processing entity is based on the consumer/producer model, consists of the model's transport abstraction, called an engine, its control and management abstraction, called front-ends, and a set of controllers implementing network services. 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. The research concentrates on the discrete approach of active networking where service deployment and service processing are separately performed. In contrast to the traditional way of Intelligent Network service implementation in centralized service nodes which control the switching nodes via a dedicated outband telecom signaling network, here the Intelligent Network services are implemented by means of service agents which are software components performing specific tasks, and are divided into parts according to their functionality. Similar to the Cognitive Packets which make their own decision on their routing to choose the best way to reach their destination, agents try to find the best location inside the network to provide the service with minimum usage of the signaling network by moving from one system to the other or cooperating with other agents when they find it necessary.

2 Cognitive Packets and CPNs

Learning algorithms and adaptation have been suggested for telecommunication systems in the past [4, 7]. However these concepts have not been fully exploited in networks because of the lack of an adequate framework allowing decentralized control of communications.

In Cognitive Packet Networks (CPN) 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. We use the term Cognitive Packet (CP) an smart packets interchangeably. 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 acknowledgment, which will follow the inverse of the route recorded by the smart packet on its way to the destination. The acknowledgment 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.

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 described below. Figure 1 presents the contents of a Cognitive Packet and the manner in which Cognitive Memory at a Node is updated by the node's processor.

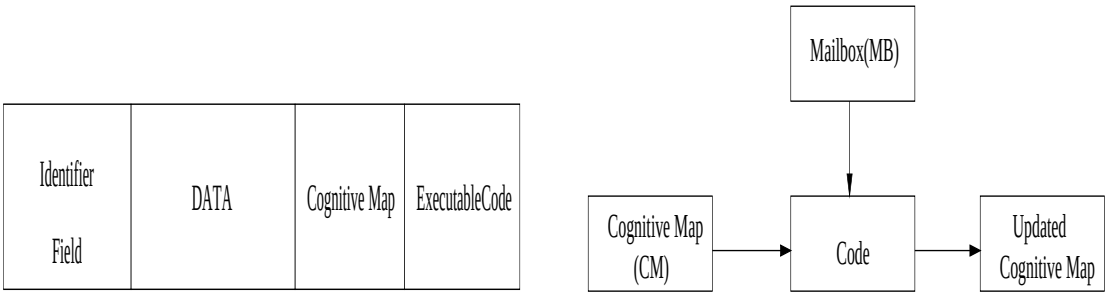

Figure 1: Representation of a CP (left), Update of a CP by a Node CPN (right)

In a CPN, the packets use nodes as "parking" or resting areas where they make decisions and route themselves. 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 Figure 2. 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

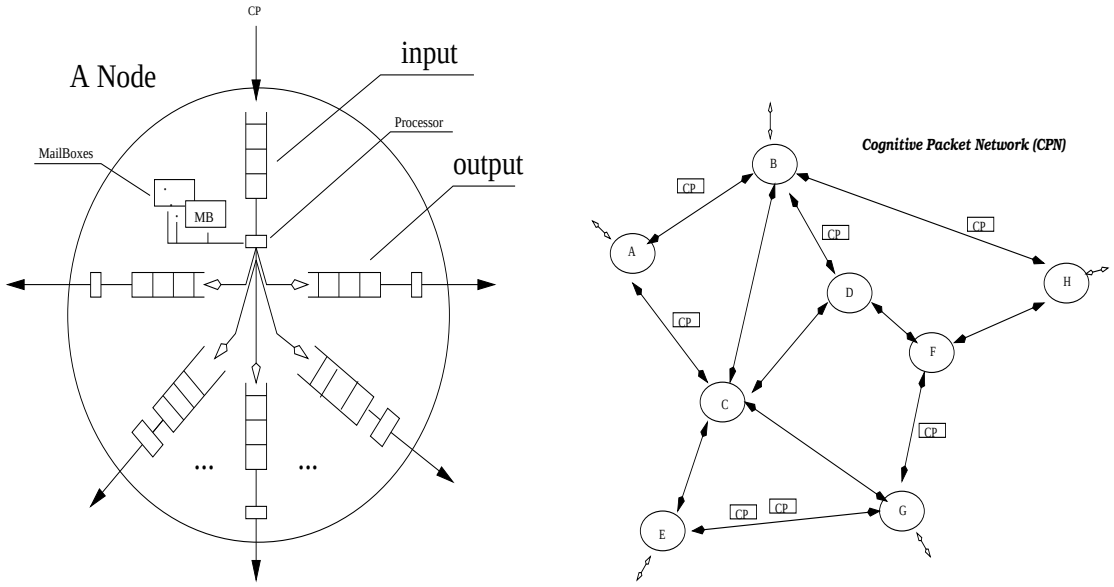

Figure 2: Schematic Representation of a CPN Node (left), Schematic Representation of a CPN (right)

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). Mailboxes 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. We plan to address these issues in relation to Cognitive Packet Networks in future work. 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 chacked 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.

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 3. Dumb packets follow the paths discovered by the CPs. In a CPN, the packets use nodes as

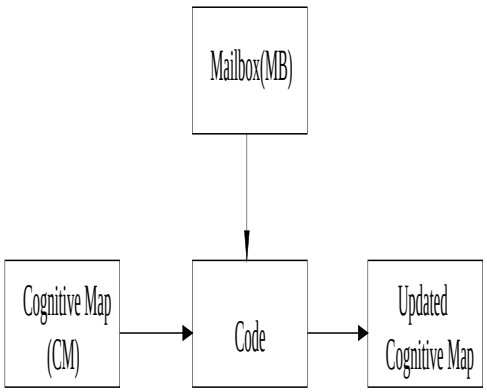

Figure 3: Update of a CP by a Node

"parking" areas where 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. We have tested 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, 22]. 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.

3 Test-bed CPN: software implementation and protocol design

In this section we describe aspects of the CPN protocol design and details of a Test-bed implementation. The software has been integrated into the Linux kernel 2.2.x. with minimal changes in the existing networking code and is independent of the physical transport technology. 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 the CPN protocol.

The Linux kernel support for low cost PCs and a growing number of platforms, and the freely availability of its source code, makes Linux an attractive system for the development of a project of this nature.

3.1 CPN Networking

A CPN provides a connectionless service to the application layer, and consists of a set of hosts interconnected by links of some kind, where each host can operate both as an end node of communication and/or as a router. The addressing scheme utilizes a single number of 32 bits to represent the CPN address of each node.

Cognitive Packets (CP) use the network to transport user data or routing information. We have considered three types of CPs addressing particular goals: Smart packets (SP), Dumb packets (DP) and Acknowledgment packets (AP). SPs and DPs transport user data whereas APs transport only routing information.

In addition to transporting user data, SPs principal goal is to find the best route to a given destination. SPs are sent to the network with the objective of discover new destinations for the source, update existing information, or seek better routes to known destinations. DPs use the information acquired by SPs to transport user data using the best possible route.

3.2 Physical description of the Test-bed

The current version of the test-bed is based on PCs with Linux as the Operating System. We are experimenting with various topologies where each CPN router can be connected up to four other routers (Figure 4 depicts a typical topology). Each port of a CPN node uses a 10 Mbps Ethernet link connected with another CPN node. The physical connection between routers uses a crossover twisted pair copper cable.

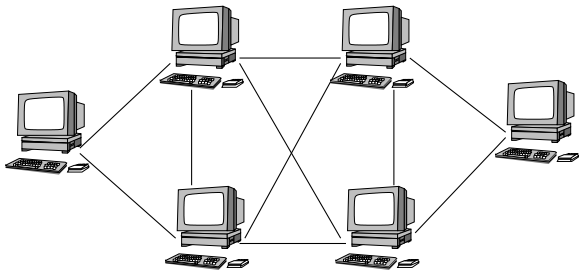

Figure 4: Typical Test-bed topology

Details concerning each node are as follows:

- 800 MHz PC
- 32 Mb Ram
- Four 10/100 Mbps Ethernet cards

All the nodes have been configured to use CPN and IP packets at the same time for comparison purposes.

3.3 CP format

CPs are of variable size and consist basically of three areas: a header, a Cognitive map (CM) and a data portion. Figure 5 depicts the format for each type of CP.

1. *ID* (4 bits): Identify the packet type:

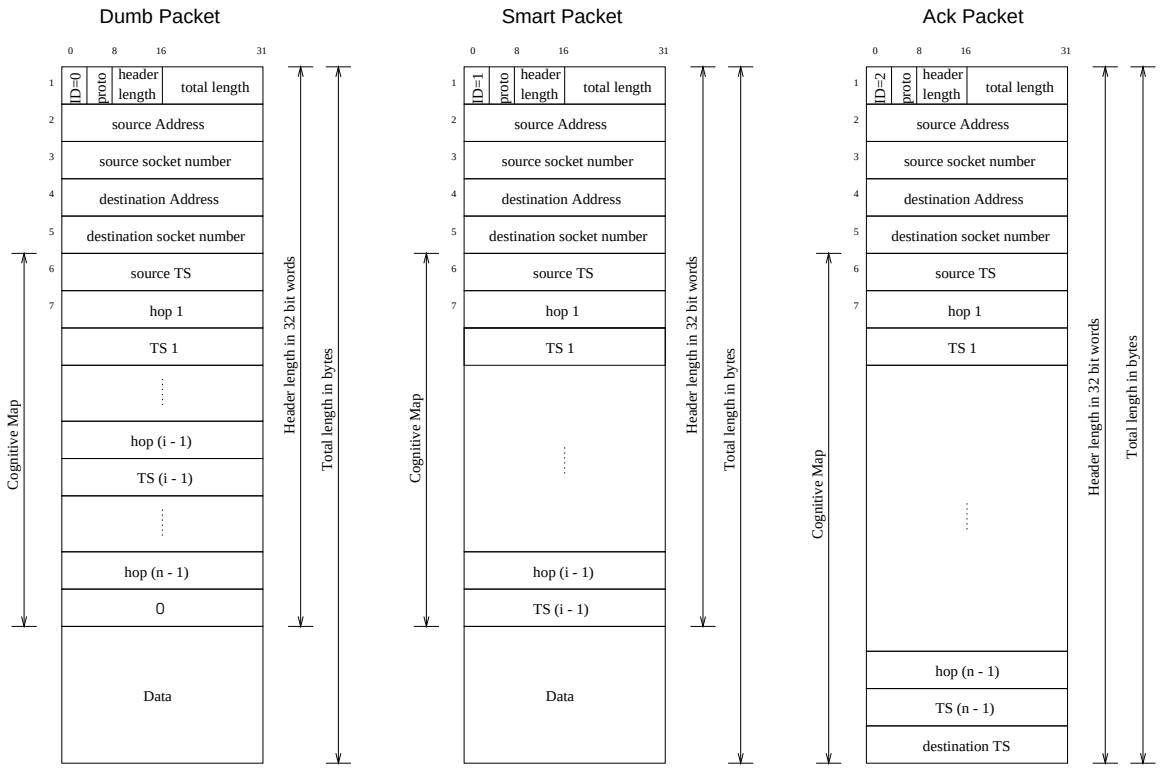

Figure 5: Cognitive Packet Format

type	use
0	DP
1	SP
2	AP
3-15	reserved

2. *Proto* (4 bits): This field indicates the next level protocol (transport layer) used in the data portion of the CPN packet.

3. *Header length* (8 bits): Indicates the length of the packet header plus the length of the CM in words of 32 bits.

4. *Total length* (16 bits): Indicates the length of the CPN packet in bytes. The field allows a length of packet to be up to 65,535 bytes.

5. *Source address* and *destination address* (32 bits): CPN address are 32 bits long. The total address space is of 4,294,967,296 hosts.

6. *Source socket* number and *destination socket* number indicates the socket identifiers of the connection to which the packet belongs.

7. *Cognitive Map* (variable): The CM transports information about the time and path taken by the packet. The meaning of the CM depends on the type of packet. See below for a detailed description.

8. *Data* (variable): The user data and transport header (if available) is carried in this field. For AP the field is empty.

3.4 Internal tables

There are three basic data structures in each CPN node related to the adaptive routing of packets.

Each CPN node keeps a mailbox (MB) and a weights table, which are updated by APs. Each of them maintains one entry for each source-destination pair that can be completed through this node. For each entry, the mailbox defines the next hop along with information about the average delay and average loss of that link. The weights table keeps useful information for the operation of the neural net responsible of taking the routing decisions.

For each known destination, each CPN node keeps a detailed path in a DP routes table. DPs uses this table to acquire the path to a given destination and create their CM.

In addition to these tables, when the CPN operates over Ethernet, each CPN node keeps a mapping table between CPN addresses and MAC addresses. This table lists each CPN neighbor address along with their MAC address and the name of the local port used to reach that neighbor. The information kept in this table is used to fill the physical layer destination address of the CPs. The content of the table can be introduced manually or can be constructed dynamically using a protocol similar to ARP.

3.5 Operation

The CPN code is divided in three areas: the first area provides support to the BSD socket system to use the CPN protocol. Appropriate functions have been written to allow socket manipulations from the application layer under coexistence of multiple protocols (i.e. TCP/IP, IPX, CPN, etc). The second area is related to the transmission and reception of packets. This area is in charge of encoding and decoding the user data in/from SPs or DPs, and the CM in/from APs. The transmission and reception of packets is performed using the networking support of the kernel where device drivers control the I/O operations of physical devices.

The third area is the routing decision zone. Next hop decisions are taken here based on the information carried by the packets and the local information stored in the nodes. After a SP or a DP reaches its destination, the routing function forwards the data to an upper layer (Figure 6) and activates an acknowledgment mechanism.

For a new communication to take place, both end nodes use the *socket()* system call to create a new socket identifier using a special number to select the underlying CPN protocol family (defined as AF_CPN). Upon reception of this request by the BSD socket system, the CPN socket creation function is activated to register the new socket in the internal socket list.

User data is sent to the network using the socket identifier and the *sendto()* system call, which activates the dispatcher function in the CPN implementation. The dispatcher determines whether a SP or a DP will carry the data. When a destination is not known by the source, the dispatcher forward the data to the SP routing function. Otherwise, the data is forwarded to the DP routing function with probability $(1 - P)$, where P is the probability of SPs in the network that has been pre-fixed in the code.

SPs are sent by the source node with a CM carrying only the source timestamp. As the packet travels through the network, each visited node is recorded in the CM along with the arrival time of the packet.

At each node, the SP heads to a new destination selecting randomly a new neighbor if the local node does not have information about the source-destination pair of the packet. Otherwise, an internal neural network previously created for the given connection provides the next hop for the SP.

Once a SP reaches its final destination, the CM is processed to eliminate loops and the resulting table is sent to the source node using an AP. The AP visit each node recorded in the new CM in reverse order updating the mailbox information at each visited node.

Upon reception of an AP, the source node updates the internal mailbox and creates or updates the corresponding source-destination pair in the DP routes table. This new information can be used later by DPs, which check this table for a given destination before going to the network. DPs follow the route dictated by the DP routes table. At each visited node, they record the arrival time. After one DP reaches its destination, an AP is sent to the source following the same procedure for SPs.

At the receiving end, the user data is stored in a queue. The application can then recover the data

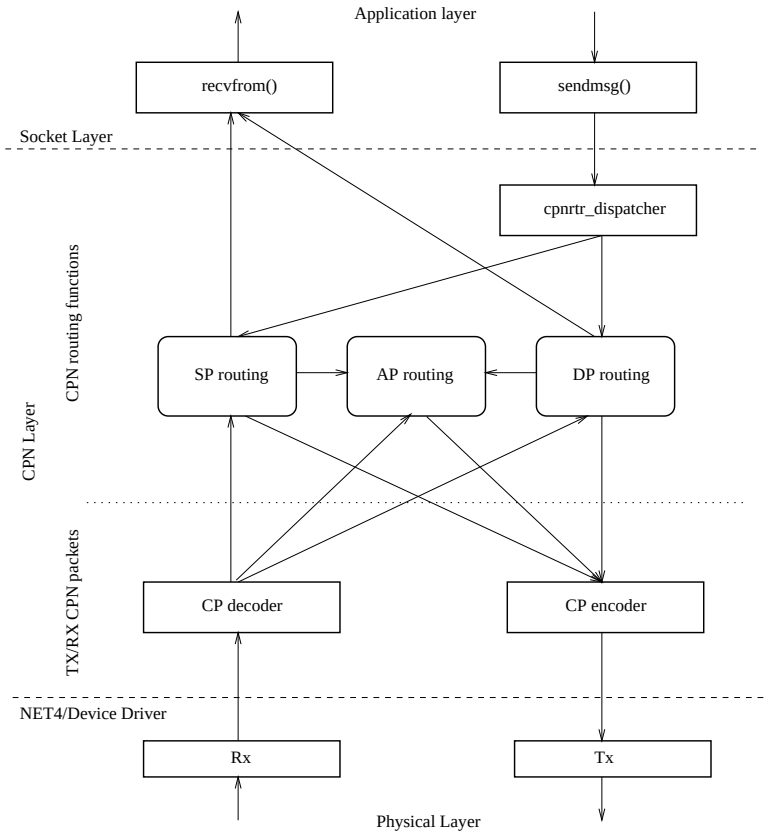

Figure 6: General organization of the CPN code

using the *recvfrom()* system call providing the adequate socket identifier.

All CPN packets are tagged with a unique identifier so that the receiving device driver can use this number to recognize the CPN packets. Arriving CPN packets are delivered to the CPN receiving function where the routing decision is performed as described before.

4 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 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 unknown 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. Figures 7 and 8 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.

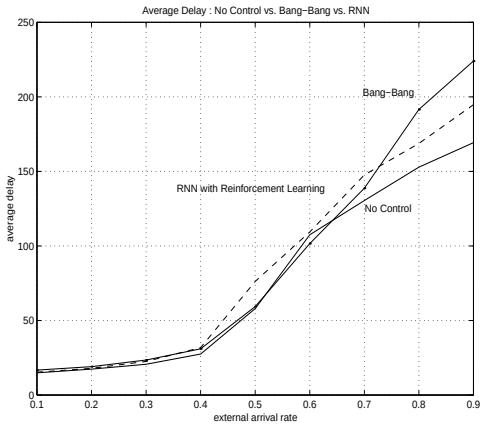

Figure 7: RL Based Control using Delay and Loss as the Goal: Comparison of Average Delay through the CPN with High Loss

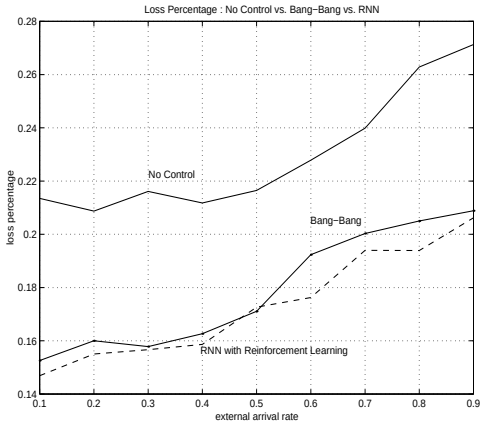

Figure 8: RL Based Control using Delay and Loss as the Goal: Comparison of Average Loss through the CPN with High Loss

4.1 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 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 9 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 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.

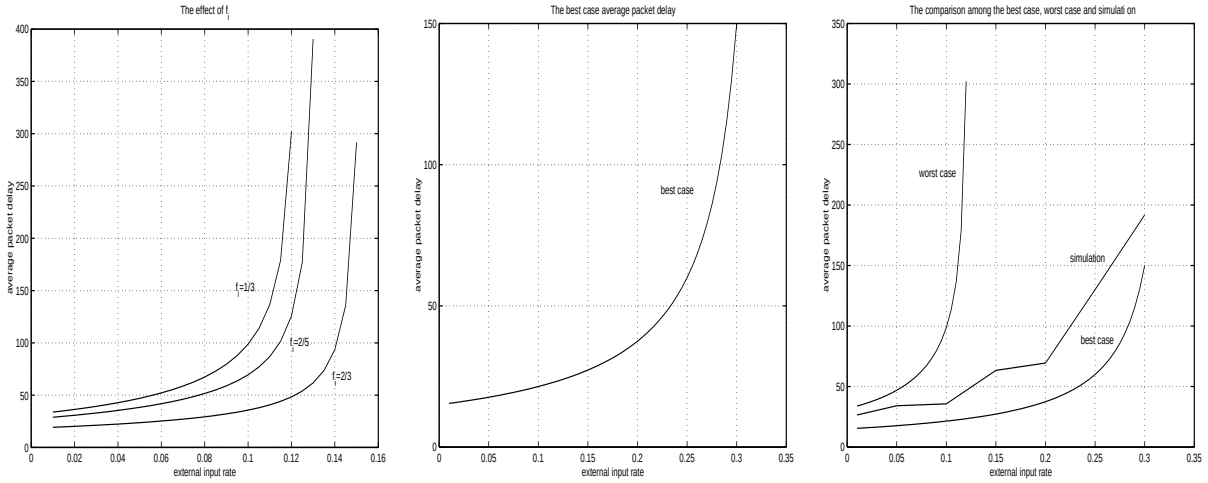

Figure 9: Comparison between the Worst Case (left), Best Case (center), and the Analytical Worst and Best Cases Compared to RNN-RL Learning Based Simulation (right)

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).