Kocak, T., Seeber, J., & Terzioglu, H. (2003). Design and implementation of a random neural network routing engine. IEEE Transactions on Neural Networks, 14(5), 1128 - 1143. 10.1109/TNN.2003.816366

Link to published version (if available):
10.1109/TNN.2003.816366

Link to publication record in Explore Bristol Research
PDF-document

Design and Implementation of a Random Neural Network Routing Engine

Taskin Kocak, Jude Seeber, and Hakan Terzioglu

***Abstract*—Random neural network (RNN) is an analytically tractable spiked neural network model that has been implemented in software for a wide range of applications for over a decade. This paper presents the hardware implementation of the RNN model. Recently, cognitive packet networks (CPN) is proposed as an alternative packet network architecture where there is no routing table, instead RNN based reinforcement learning is used to route packets. Particularly, we describe implementation details for the RNN based routing engine of a CPN network processor chip: the smart packet processor (SPP). The SPP is a dual port device that stores, modifies, and interprets the defining characteristics of multiple RNN models. In addition to hardware design improvements over the software implementation such as dual access memory, output calculation step, reduced output calculation module, this paper introduces a major modification to the reinforcement learning algorithm used in the original CPN specification such that the number of weight terms are reduced from $2n^2$ to $2n$. This not only yields significant memory savings, but it also simplifies the calculations for the steady state probabilities (neuron outputs in RNN). Simulations have been conducted to confirm the proper functionality for the isolated SPP design as well as for the multiple SPP's in a networked environment.**

***Index Terms*—Network processors, neural network, packet switched networks, random neural networks.**

I. Introduction

As the Internet continues to expand in number of users, servers, IP addresses, and routers, the IP based network must evolve and change. There is a strong demand for novel routing architectures that can provide more efficient and robust service to the Internet constituents. The cognitive packet network (CPN) was proposed as an alternative to the IP based network architectures [1], [2]. CPN attempts to solve some of the problems associated with the legacy IP networks, such as QoS, the never-ending expansion of routing tables and their related maintenance issues. The rapid expansion of network applications and data traffic is also leading to new specialized processor designs that would keep up with the growing field of networking and communications. Network component design becomes more challenging as the performance and usage of communication networks increase. To meet the rapidly changing requirements such as performance, cost, flexibility, and interoperability; the networking industry has opted to build products around network processors (NPs) [9], [10]. The NP market grows rapidly as the driving force for faster, more powerful network products increase. This year the NP market is to reach $231 million but in four years it is estimated to reach $7.2 billion [3]. This is probably the reason that network processing is the fastest growing field in the networking industry.

The applicability of the CPN concept has been demonstrated through several software implementations [1], [2], [11]. However, higher data traffic and increasing packet processing demands require the imlementation of this new network architecture in hardware. The primary motivation for this study is the design and implementation of a prototype CPN router. The research presented within this paper is the initial work toward the realization of this goal. Specifically, this work identifies key functionalities of the CPN router and the elements that will implement them. The complete specifications for one component, called the smart packet processor (SPP), are derived and a design is implemented. The SPP is a dual port device that stores, modifies, and interprets the defining characteristics of multiple random neural network (RNN) models. RNN has been proven to be successful in a variety of applications [6] for more than a decade; however, it has not been implemented in hardware before. In addition to reporting the first RNN hardware implementation, this paper introduces a major modification to the reinforcement learning algorithm used in Gelenbe *et al.*'s paper [1] such that the number of weight terms are reduced from $2n^2$ to $2n$. This not only yields significant memory savings, but it also simplifies the calculations for the steady state probabilities (neuron outputs in RNN).

The rest of the paper is organized as follows: In the following subsections, we review the cognitive packet network and the random neural network. Section II introduces the initial design approaches for the CPN network processor. Smart packet processor design is presented in Section III. Circuit implementation details are also given in this section. Section IV discusses the system-level integration and verifies the operation of the SPP in a networked environment. The paper ends with some conclusions and directions for future work.

A. Cognitive Packet Network

The CPN is a store and forward architecture that achieves intelligent QoS based routing by employing "smart or cognitive" packets. The CPN uses three different types of packets: smart packets, dumb packets, and acknowledgment packets. The dumb packets carry payload (the user's data) and are source routed by applying routing information generated from the experiences of the smart packets. Smart packets are sent out continuously to search for routes to a destination. Before a particular flow of dumb packets can be transmitted, the route information for the QoS class must be available at the source. If it is unavail-

Manuscript received September 13, 2002; revised March 14, 2003 and April 30, 2003.

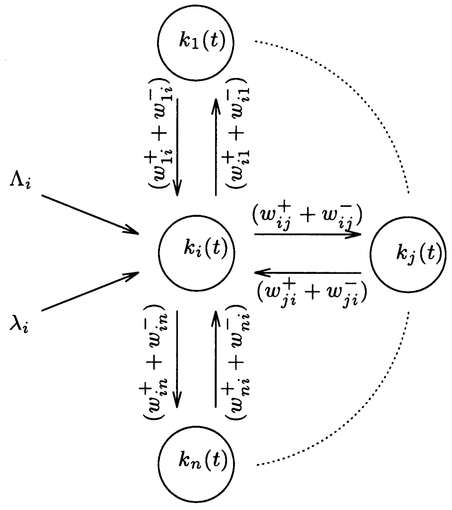
The authors are with the School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816 USA (e-mail: tkocak@cs.ucf.edu).

Digital Object Identifier 10.1109/TNN.2003.816366

able, the source node will create and dispatch smart packets to determine routes to the selected destination for the required QoS class. As the smart packets propagate through the network, they collect measurement data with regard to link quality. Acknowledgment packets carry back this measurement data, depositing it at the CPN routers as they travel the reverse route of the smart packets. The original source node uses the data to establish a route for the dumb packets. The CPN router acts as a buffer for packets, as a storage area for mailboxes (MBs) where acknowledgment packets deposit measurement data, and as a processor for packets. It receives packets via a finite set of ports and stores them in an input buffer, where sorting based upon QoS requirements may occur. It forward packets to other nodes via output buffers, and runs the algorithm used to make routing decisions concerning smart packets. Contrary to a conventional IP router, a CPN router does not maintain a routing table. The routing decisions in a CPN router rely upon a learning algorithm. Previous attempts to incorporate learning algorithms and adaptation into packet networks have been insufficiently researched due to the lack of practical mechanisms. CPN routers execute a reinforcement learning algorithm in order to select the output link for smart packets. Dumb and acknowledgment packets are source routed with their routes stored within the packets themselves. The reinforcement learning algorithm that smart packets rely on is based on a QoS "Goal". The term "Goal" is used to indicate that there is no QoS guarantee rather there is a best effort attempt to satisfy the QoS objective. The smart packets act as network explorers. They travel through the network, finding routes and collecting data. The measurements and the path traveled by the packet are stored in its Cognitive Map (CM). When the smart packet arrives at its destination router, the router generates a corresponding acknowledgment packet. The acknowledgment packet inherits the smart packet's source as its destination. In addition, the smart packet's CM is inverted and stored as the acknowledgment's CM. As the acknowledgment travels through the network, routers will reference its CM to find out where to send it next. Thus, the acknowledgment packet is source routed, following the inverse route of the smart packet that initiated it. Before the routers forward the acknowledgment to its next hop, they will read the relevant measurement data from its CM. In the reinforcement learning algorithm, the observed outcome of a decision is used to "reward" or "punish" the routing algorithm with respect to that decision. The "Goal" is the metric that characterizes the success of the outcome, such as packet travel time or transit delay. As an example, the QoS *goal* (G) that smart packets pursue may be formulated by as minimizing *transit delay* (W), *loss probability* (L), *jitter*, or some weighted combination, for instance

$$G = a * W + b * L \tag{1}$$

where a and b are constants selected by the application layer which signify the relative importance of the delay and loss for this particular application's QoS. The reinforcement learning algorithm for CPN routing uses a fully recurrent neural network model to ensure that all decision variables are mutually related. In order to ensure the convergence of the algorithm, the CPN model employs the random neural network. The internal state of the RNN is a unique solution for any set of weights and input variables.



Λ_i	arrival rate of exogenous excitatory signals
λ_i	arrival rate of exogenous inhibitory signals
w^+_{ji}	arrival rate of excitatory signals from neuron j
w^-_{ji}	arrival rate of inhibitory signals from neuron j
$k_i(t)$	instantaneous potential of the neuron

Fig. 1. Representation of a neuron in the RNN.

B. RNN Model

To establish intelligence based routing, the CPN employs the RNN model by Gelenbe [4], [5], [7]. The RNN is an analytically tractable spiked neural network model that has been implemented in a wide range of applications. The function of the RNN in the CPN model is to capture the effect of the unpredictable network parameters and convert it into a routing decision. In the random neural network model, signals in the form of impulses which have unit amplitude travel among the neurons. Positive signals represent excitation, whereas negative signals represent inhibition to the receiving neuron. Thus, an excitatory impulse is interpreted as a "$+1$" signal, while an inhibitory impulse is interpreted as a "-1" signal. Each neuron i has a state $k_i(t)$, which is its potential at time t, represented by a nonnegative integer.

When the potential of neuron i is positive, it is referred to as being 'excited', and it can transmit impulses (fire). The impulses will be sent out at a Poisson rate r_i, with independent, identical exponentially distributed interimpulse intervals. The impulses transmitted will arrive at neuron j as excitation signals with probability p^+_{ij}, and as inhibitory signals with probability p^-_{ij}. A neuron's transmitted impulse may also leave the network with probability d_i, therefore, $d_i + \sum_{j=1}^{n}[p^+_{ij} + p^-_{ij}] = 1$. To make these probabilities easier to work with, let $w^+_{ij} = r_i p^+_{ij}$, and $w^-_{ij} = r_i p^-_{ij}$; then firing rate of neuron i, r_i, is $\sum_{j=1}^{n}[w^+_{ij} + w^-_{ij}]$. The w matrices can be viewed as being analogous to the synaptic weights in connectionist models, though they specifically represent rates of excitatory and inhibitory impulse emission. Since

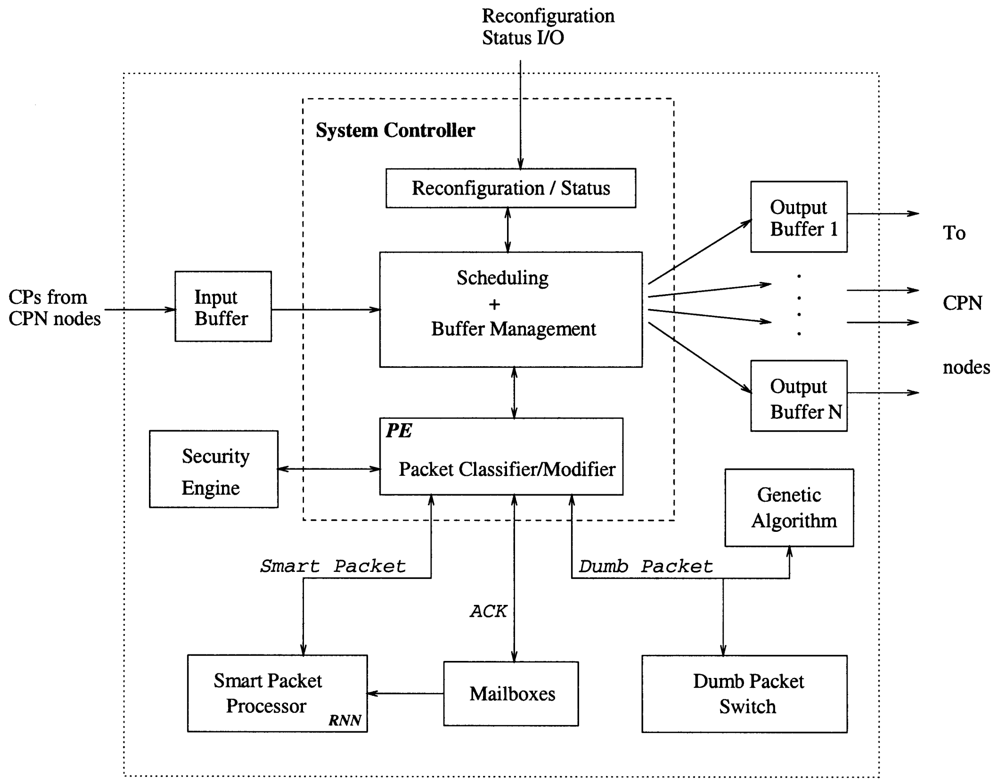


Fig. 2. CPN network processor architecture.

the w matrices are formed through a product of rates and probabilities, they are guaranteed to be nonnegative. Exogenous excitatory and inhibitory signals, meaning those arriving to the neuron from a source outside of the network, also arrive to neuron i at rates Λ_i and λ_i, respectively. These are analogous to the input received by the input neurons in a connectionist model; again however, they represent rates.

Fig. 1 shows the representation of a neuron in the RNN using the model parameters that have been defined above. In this figure, only the transitions to and from a single neuron i are considered in a recurrent fashion. All the other neurons can be interpreted as the replicates of neuron i.

At this point, it is necessary to consider the dynamics of the random neural network model by analyzing the possible state transitions. Within a time interval of Δt, several transitions can occur which change a neuron's state $k_i(t)$.

- The potential $k_i(t)$ of a neuron will decrease by one whenever it fires, regardless of the type of the signal emitted (excitation or inhibition). Also, when an exogenous inhibitory signal arrives from outside the network to neuron i, its potential drops to $k_i(t)-1$ at time $t+\Delta t$. Moreover, neuron i might receive an inhibitory impulse from another neuron j, whose effect will again be to decrement the value of k_i at time t by one.
- Arrival of an exogenous excitatory signal from outside, or an excitatory impulse from another neuron within the network will result in incrementing the neuron potential by one, yielding $k_i(t)+1$.
- Needless to say, the value of ith neuron's state remains unchanged when none of the events described above occur.

In the case when *self-inhibition* is allowed, the value of the neuron's state can drop by two units in a single time step, however this case will not be considered in the following expressions. Also in this model, *self-excitation* is not of interest because in its presence, the potential of the neuron may increase without bound which would lead to instability. There are also some boundary conditions which prevent some of the transitions from occurring. First of all, a neuron can fire only when it has a positive potential as explained above. Second, when the neuron has a potential of zero, the arrival of new inhibitory signals does not decrease its value further. All of these constraints will be unified in a single expression when the state transitions are expressed in mathematical form.

Let $\mathbf{k}(t)=k_1(t),\ldots,k_n(t)$ be the vector of signal potentials at time t, and $\mathbf{k}=k_1,\ldots,k_n$ be a particular value of the vector, and lets define the probability $p(\mathbf{k},t)=\Pr[\mathbf{k}(t)=\mathbf{k}]$. The behavior of the probability distribution of the network state can be derived through the following equations. Since $\mathbf{k}(t){:}\, t\geq 0$ is a continuous time Markov chain, it satisfies an infinite system of *Chapman-Kolmogorov* equations.

$$\begin{aligned}
p(\mathbf{k},t+\Delta t) \\
&= \sum_i [p(k_i^+,t)r(i)d(i)\Delta t + p(k_i^-,t)\Lambda(i)\Delta t\, \mathbf{1}[k_i(t)>0] \\
&\quad + p(k_i^+,t)\lambda(i)\Delta t + p(k_i,t)(1-\Lambda(i)\Delta t) \\
&\quad \times (1-\lambda(i)\Delta t)(1-r(i)\Delta t)\, \mathbf{1}[k_i(t)>0] \\
&\quad + \sum_j \{p(k_{ij}^{+-},t)r(i)p^+(i,j)\Delta t\, \mathbf{1}[k_j(t)>0] \\
&\quad + p(k_{ij}^{++},t)r(i)p^-(i,j)\Delta t \\
&\quad + p(k_i^+,t)r(i)p^-(i,j)\Delta t\, \mathbf{1}[k_j(t)=0]\}] + o(\Delta t) \qquad (2)
\end{aligned}$$

where

$$\mathbf{1}[x] = \begin{cases} 1 & \text{if x is true} \\ 0 & \text{otherwise} \end{cases}$$

For steady state analysis, let $p(\mathbf{k})$ denote the stationary probability distribution which is equal to $\lim_{t\to\infty} \Pr[\mathbf{k}(t) = \mathbf{k}]$ if it exists. Thus, in steady state, stationary probability distribution, $p(\mathbf{k})$, must satisfy the global balance equations

$$\begin{aligned} p(\mathbf{k}) & \sum_i [\Lambda(i) + [\lambda(i) + r(i)]\, \mathbf{1}[k_i > 0]] \\ &= \sum_i [p(k_i^+) r(i) d(i) + p(k_i^-)\Lambda(i)\, \mathbf{1}[k_i > 0] \\ &\quad + p(k_i^+)\lambda(i) + \sum_j \{ p(k_{ij}^{+-}) r(i) p^+(i,j)\, \mathbf{1}[k_j > 0] \\ &\quad + p(k_{ij}^{++}) r(i) p^-(i,j) + p(k_i^+) r(i) p^-(i,j)\, \mathbf{1}[k_j = 0]\}]. \end{aligned} \tag{3}$$

The stationary probability distribution associated with the model is the value which will be taken to be the output of the network, and is given by

$$q_i = \lim_{t\to\infty} \Pr[k_i(t) > 0], \quad i = 1, \ldots, n \tag{4}$$

which reduces to the form

$$q_i = \lambda^+(i)/[r(i) + \lambda^-(i)] \tag{5}$$

where the $\lambda^+(i), \lambda^-(i)$ for $i = 1, \ldots, n$ satisfy the system of nonlinear simultaneous equations

$$\lambda^+(i) = \sum_j q_j w_{ji}^+ + \Lambda(i), \quad \lambda^-(i) = \sum_j q_j w_{ji}^- + \lambda(i). \tag{6}$$

To put (5) into words, the steady state probability that the neuron i is excited is simply equal to the ratio of the sum of all the rates of arriving excitatory signals to the sum of the rates of arriving inhibitory signals together with the firing rate of that particular neuron.

C. Reinforcement Learning for the RNN

There are different learning algorithms that may be applied to the random neural network model. The gradient descent algorithm has been used with feed-forward topologies in many applications [6]. This algorithm has two distinct modes: offline training and online execution. For the gradient descent algorithm to be implemented, the RNN output vectors need to be known a priori and provided during the training mode. This requirement is not compatible with needs of the CPN, where the dynamic network parameters will preclude offline predictions. RNN-based Reinforcement Learning (RNNRL) applies the methodology described below [8]. Given some goal G that the SP has to achieve as a function to be minimized, we formulate a reward R that is simply $R = G^{-1}$. Successive measured values of the R are denoted by $R_l, l = 1, 2, \ldots$. These are first used to compute a decision threshold

$$T_l = a * T_{l-1} + (1-a) * R_l \tag{7}$$

where a is some constant $0 < a < 1$, typically close to 1. Now consider a closed RNN with as many neurons as decision outcomes. Let the neurons be numbered $1, \ldots, n$. Thus for any decision i, there is some neuron i. Decisions in this reinforcement learning algorithm are made by selecting the decision j for which the corresponding neuron is the most excited, i.e., the one with the largest q_j. Note that the lth decision may not have contributed directly to the lth reward because of time delays between cause and effect. Suppose that the lth decision corresponds to neuron j, and that we have measured the lth 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 that we call threshold T_{l-1}. If that is the case, then we increase very significantly the excitation 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 and then update the network weights as follows for all neurons:

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

$$\begin{aligned} w^+(i,j) &= w^+(i,j) + (R_l - T_{l-1}), \\ w^-(i,k) &= w^-(i,k) + (R_l - T_{l-1})/(n-1), \quad k \ne j \end{aligned} \tag{8}$$

else

$$\begin{aligned} w^+(i,k) &= w^+(i,k) + (T_{l-1} - R_l)/(n-1), \quad k \ne j, \\ w^-(i,j) &= w^-(i,j) + (T_{l-1} - R_l). \end{aligned} \tag{9}$$

Then we renormalize all the weights by carrying out the following operations, to avoid obtaining weights that indefinitely increase in size. First for each i, we compute

$$r_i^* = \sum [w^+(i,m) + w^-(i,m)] \tag{10}$$

and then renormalize the weights with

$$\begin{aligned} w^+(i,j) &\leftarrow w^+(i,j) * r_i / r_i^*, \\ w^-(i,j) &\leftarrow w^-(i,j) * r_i / r_i^*. \end{aligned} \tag{11}$$

Finally, the probabilities q_i are computed using the nonlinear iterations (5) and (6), leading to a new decision based on the neuron with the highest probability of being excited.

II. CPN Network Processor Design

All network processors require the ability to manage their input and output packet flows. Specifically, the CPN router, which houses the CPN network processor, will require the ability to classify packets by their type so they may be forwarded to their proper processing units (see Fig. 2). In addition, the buffer management will need to implement priority sorting based upon QoS requirements. Since dumb packets are source routed, the controller associated with forwarding these packets should be one of the simpler designs. The dumb packet switch will need to search the packet's CM and find the local router's address. The next address in the CM will also be the next hop for the dumb packet. With this information, the switch will then assign the packet to the corresponding output port.

As previously stated, the mailbox is where acknowledgment packets deposit the relevant data measurements they are carrying. In our hardware design, the mailbox is also responsible for calculating the reward value from the measurement parameters. Next, it must forward the reward value to a component that incorporates the RNN with reinforcement learning. In the meantime, the acknowledgment packet needs to be transmitted to its next hop in its path. Since the packet is source routed, the mailbox can feature a design similar to the dumb packet switch exclusively for use by the acknowledgment packets. The system controller has many responsibilities such as

- configuration control of CPN components;
- determining/verifying port connections with adjacent routers;
- classification of packets based upon type (i.e., smart, dumb);
- forwarding packets to appropriate routing component.

A. Smart Packet Processor Design Considerations

The function of the Smart Packet Processor (SPP) is to determine the outgoing port for arriving smart packets based upon its QoS, source and destination parameters (QSD). In accordance with the CPN model, the SPP needs to employ the RNN model with the reinforcement learning algorithm to make its decisions. The reinforcement learning algorithm requires the reward value calculated with the data from the acknowledgment packets. Therefore, the SPP, shown in Fig. 3, is a convergence point for the flow of both smart and acknowledgment packets. To meet its requirements, the SPP needs to interface with two external structures: the acknowledgment mailbox and the system controller. The ideal design allows simultaneous transactions between these components. The system controller waits for the SPP output to forward smart packets. On the other hand, the acknowledgment mailbox is merely a receptacle for the data measurements within the acknowledgment packets. Therefore, the priority of the SPP design must be the servicing of the smart packets.

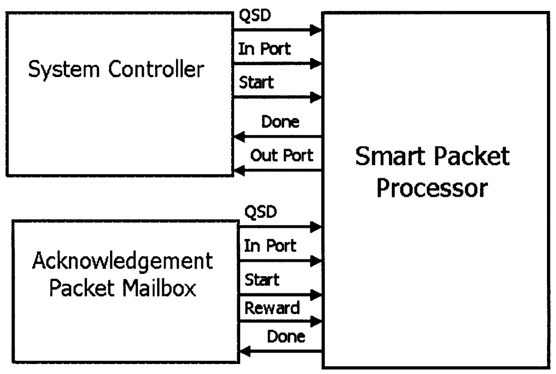


Fig. 3. Interfacing the smart packet processor with other components.

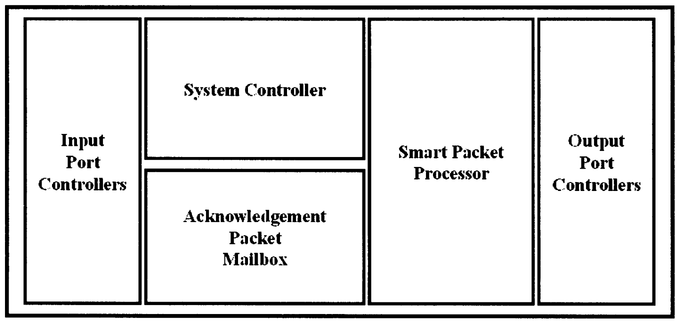


Fig. 4. Router architecture for system level integration.

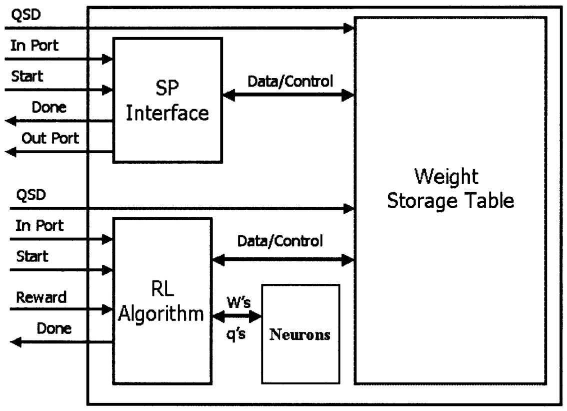


Fig. 5. Composition of the smart packet processor.

B. Preliminary Router Configuration

A basic CPN router, shown in Fig. 4, has been implemented to conduct preliminary system level simulations. The primary objective of the device is to enable the testing of the functionality of the SPP on a system level. Furthermore, the design allows for the future addition of other CPN router components. There are several requirements and assumptions for the architecture.

- The design must be modular and flexible to allow for various network topologies to be implemented and tested.
- The characteristics of the links between the routers must be configurable. There need to be methods for simulating the effects of congestion and broken links.
- The propagation and processing of acknowledgment packets is fundamental to analyzing and assessing the functionality of the SPP. Therefore, the design must generate and transmit the acknowledgment packets.
- Simplified network communication schemes will suffice for this initial testing phase. Since the link characteristics will be manipulated by external control, the inter-router communications are only required to be reliable and practical.
- Input and output queue management issues will not be tested or analyzed in this work.
- Dumb packet production and propagation will not be addressed during this initial testing evolution.

Several design options were considered to satisfy the requirements. The solution that has been implemented uses unidirectional data paths in its network communication scheme.

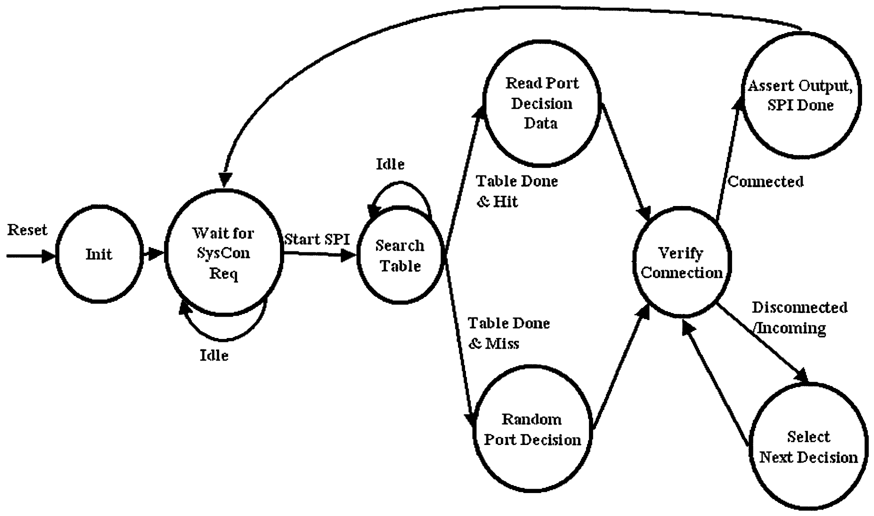


Fig. 6. SP interface state machine.

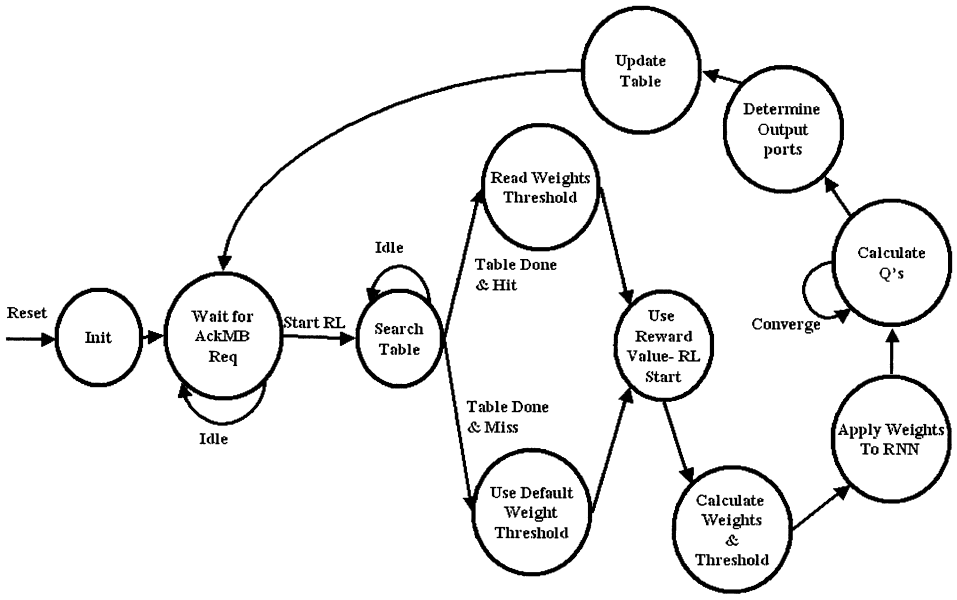


Fig. 7. RL algorithm state machine.

Each router is equipped with both input and output port controllers that enable duplex links to other routers. Each link is capable of being disconnected and reconnected during the simulation. During previous software simulations, link status was primarily defined by packet transit times. As a smart packet left router p, router p would place a timestamp in its cognitive map. When the smart packet reached its destination, the timestamp would be written into the cognitive map of the corresponding acknowledgment packet. Finally, as the acknowledgment returned through router p, the timestamp would be read and the packet transit time would be calculated. This time would be converted into a reward value in order to be submitted to the reinforcement learning algorithm. The block diagram of the completed smart packet processor design is shown in Fig. 5. It consists of four different components: the smart packet interface, the reinforcement learning algorithm, the neuron array, and the weight storage table. The SP interface is externally connected to the system controller while the RL algorithm receives control from the acknowledgment mailbox. As seen in the figure, both components have data and control paths to the weight storage table. The table is a complex dual port memory structure that stores the RNN weights, thresholds, outputs and QSD indexes. Lastly, the neurons are controlled by the RL algorithm and are used to calculate the steady state output of the RNN.

III. Smart Packet Processor Design

A. Smart Packet Interface

When requested, the SP interface provides an output port number to the system controller. A high level state diagram for the SP interface is shown in Fig. 6. Following a reset, the SP interface idles until it receives the start signal from the system controller. At the same time, the system controller is providing the QSD parameters to the weight storage table. The SP interface directs the table to perform a search upon these parameters.

TABLE I
WEIGHT MATRIX AFTER REWARD SCENARIO

Excitation to Neuron 1	Inhibition to Neuron 1	Excitation to Neuron 2	Inhibition to Neuron 2
$w_{11}^{+} = w_{11}^{+} + a$	$w_{11}^{-} = w_{11}^{-}$	$w_{12}^{+} = w_{12}^{+}$	$w_{12}^{-} = w_{12}^{-} + a/3$
$w_{21}^{+} = w_{21}^{+} + a$	$w_{21}^{-} = w_{21}^{-}$	$w_{22}^{+} = w_{22}^{+}$	$w_{22}^{-} = w_{22}^{-} + a/3$
$w_{31}^{+} = w_{31}^{+} + a$	$w_{31}^{-} = w_{31}^{-}$	$w_{32}^{+} = w_{32}^{+}$	$w_{32}^{-} = w_{12}^{-} + a/3$
$w_{41}^{+} = w_{41}^{+} + a$	$w_{41}^{-} = w_{41}^{-}$	$w_{42}^{+} = w_{42}^{+}$	$w_{42}^{-} = w_{42}^{-} + a/3$
Excitation to Neuron 3	Inhibition to Neuron 3	Excitation to Neuron 4	Inhibition to Neuron 4
$w_{13}^{+} = w_{13}^{+}$	$w_{13}^{-} = w_{13}^{-} + a/3$	$w_{14}^{+} = w_{14}^{+}$	$w_{14}^{-} = w_{14}^{-} + a/3$
$w_{23}^{+} = w_{23}^{+}$	$w_{23}^{-} = w_{23}^{-} + a/3$	$w_{24}^{+} = w_{24}^{+}$	$w_{24}^{-} = w_{24}^{-} + a/3$
$w_{33}^{+} = w_{33}^{+}$	$w_{33}^{-} = w_{33}^{-} + a/3$	$w_{34}^{+} = w_{34}^{+}$	$w_{34}^{-} = w_{34}^{-} + a/3$
$w_{43}^{+} = w_{43}^{+}$	$w_{43}^{-} = w_{43}^{-} + a/3$	$w_{44}^{+} = w_{44}^{+}$	$w_{44}^{-} = w_{44}^{-} + a/3$

TABLE II
WEIGHT MATRIX FOR RATE CALCULATION

Excitation from Neuron 1	Inhibition from Neuron 1	Excitation from Neuron 2	Inhibition from Neuron 2
$w_{11}^{+} = w_{11}^{+} + a$	$w_{11}^{-} = w_{11}^{-}$	$w_{21}^{+} = w_{21}^{+} + a$	$w_{21}^{-} = w_{21}^{-}$
$w_{12}^{+} = w_{12}^{+}$	$w_{12}^{-} = w_{12}^{-} + a/3$	$w_{22}^{+} = w_{22}^{+}$	$w_{22}^{-} = w_{22}^{-} + a/3$
$w_{13}^{+} = w_{13}^{+}$	$w_{13}^{-} = w_{13}^{-} + a/3$	$w_{23}^{+} = w_{23}^{+}$	$w_{23}^{-} = w_{23}^{-} + a/3$
$w_{14}^{+} = w_{14}^{+}$	$w_{14}^{-} = w_{14}^{-} + a/3$	$w_{24}^{+} = w_{24}^{+}$	$w_{24}^{-} = w_{24}^{-} + a/3$
Excitation from Neuron 3	Inhibition from Neuron 3	Excitation from Neuron 4	Inhibition from Neuron 4
$w_{31}^{+} = w_{31}^{+} + a$	$w_{31}^{-} = w_{31}^{-}$	$w_{41}^{+} = w_{41}^{+} + a$	$w_{41}^{-} = w_{41}^{-}$
$w_{32}^{+} = w_{32}^{+}$	$w_{32}^{-} = w_{12}^{-} + a/3$	$w_{42}^{+} = w_{42}^{+}$	$w_{42}^{-} = w_{42}^{-} + a/3$
$w_{33}^{+} = w_{33}^{+}$	$w_{33}^{-} = w_{33}^{-} + a/3$	$w_{43}^{+} = w_{43}^{+}$	$w_{43}^{-} = w_{43}^{-} + a/3$
$w_{34}^{+} = w_{34}^{+}$	$w_{34}^{-} = w_{34}^{-} + a/3$	$w_{44}^{+} = w_{44}^{+}$	$w_{44}^{-} = w_{44}^{-} + a/3$

If the table returns a hit, then the SP interface reads in the port numbers that are being provided by the table. In the event of a miss from the search, the SP interface randomly assigns the port numbers. If the primary port number is either disconnected or in the direction from which the smart packet came, then the SP interface will select the secondary port number and so on. If all other ports are disconnected, then the incoming port number will be selected. Finally, the port number is output to the system controller as the done signal is asserted.

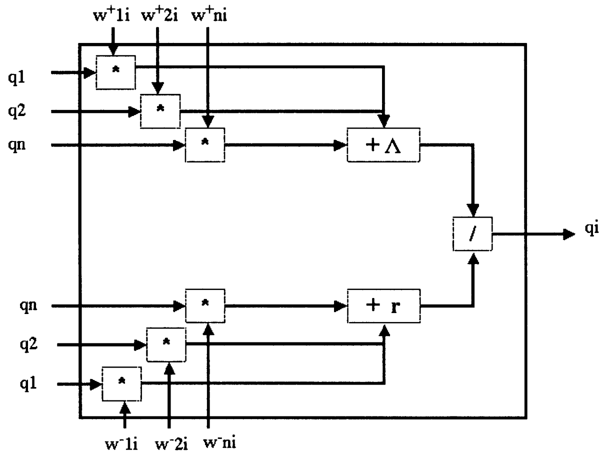
B. *Reinforcement Learning Algorithm Component*

The high level state diagram depicted in Fig. 7 represents the RL algorithm component. Similar to the SP interface, this component waits for a start signal generated by the acknowledgment mailbox. With it signals for a start, the MB also applies the QSD parameters to the search index of the weight storage table. The RL component initiates a search on the table and waits for a response. If present, the table will return the weights and threshold of the desired RNN model. If the RNN isn't found, then the RL algorithm will instantiate a new one with the default initialization values.

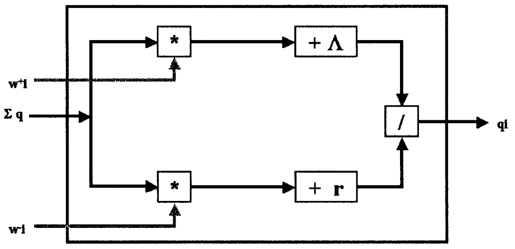
In accordance with Fig. 5, the MB also provides the RL component with the incoming port number of the associated acknowledgment and the reward value calculated from the measurement data. The RL component compares the reward value with the threshold to determine whether the previous decision, identified by the incoming port number, will be rewarded or punished. At this point, the new values for the RNN weights and threshold can be calculated. Next, the weights are delivered to the neurons, where the steady state probabilities are calculated over several iterations. Once they are obtained, the probabilities are sorted and the identification numbers (i.e., the corresponding port numbers) of the neurons with the highest probabilities are noted. Using the port number rather than the actual q value saves in both storage area and processing time for the SP interface. The final step is to update the weight storage table with the new weights, threshold, and output decisions.

C. *Neurons*

The original CPN model specifications required $2n^2$ weight terms for each RNN, where n is the number of output links/neurons. So, in a 32-port CPN router, each RNN would need to store and manipulate 2048 weight terms. Furthermore, multiple RNN models are needed to represent the different QSD pairs that are active in the router. The hardware design needs to be flexible and easily scalable, therefore, ways to minimize the number of weight terms are investigated. Consider a 4 neuron fully connected implementation with neurons 1 through 4. Suppose neuron 1 was the last decision made by the network. An acknowledgment packet is returned and the neural network will be



(a)



(b)

Fig. 8. (a) Neuron i. (b) Neuron i after weight term reduction.

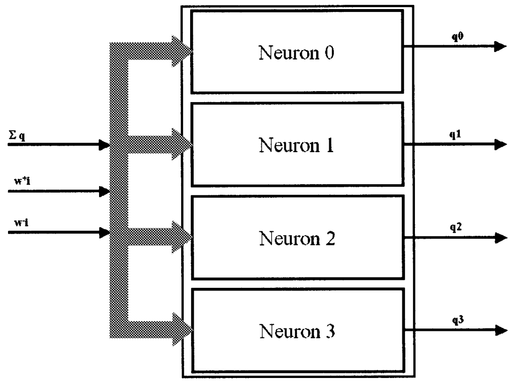


Fig. 9. Neuron array.

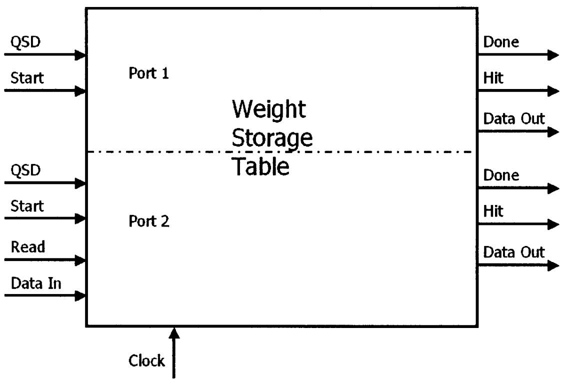


Fig. 10. Block diagram of weight storage table.

rewarded for making a good decision. If the weights are grouped together based upon destination (i.e., neuron 1) and type (excitation or inhibition), then it can be seen that all the weights of a group will have the same value. As shown by Table I, the excitation weights leading to neuron 1 are all incremented by the same amount a, where $a = (R - T)$. In addition, the inhibition weights leading to all other neurons are incremented by $a/(n-1)$ with n equal to the number of neurons in the network. The remaining weights are not modified during this stage.

The second part of the algorithm is the normalization process. Table II is a rearrangement of the previous table to facilitate the calculation of the interim rate of fire, $r*$. Summing up the weight terms, it can be seen that the value of $r*$ is $r_{\text{old}} + 2a$ for each neuron, where r_{old} was the previous rate of fire.

$$r_i^* = \sum[w^+(i,m) + w^-(i,m)] = r_{\text{old}} + 2a \qquad (12)$$

Per the algorithm, the next step is to multiply each weight by the normalization factor $r_{\text{old}}/r*$ which can be rewritten as $r_{\text{old}}/(r_{\text{old}} + 2a)$. Finally, the new rate of fire can be calculated. The substitutions shown below verify that the rate of fire for each neuron is constant.

$$r_{\text{new}} = \sum[w^+(i,m) * r_{\text{old}}/r * + w^-(i,m) * r_{\text{old}}/r*], \qquad (13)$$

$$r_{\text{new}} = \sum[w^+(i,m) + w^-(i,m)] * r_{\text{old}}/r*, \qquad (14)$$

$$r_{\text{new}} = (r_{\text{old}} + 2a) * r_{\text{old}}/(r_{\text{old}} + 2a), \qquad (15)$$

$$r_{\text{new}} = r_{\text{old}}. \qquad (16)$$

Realizing now that some of the weight terms are identical, the equation for the steady state probability of the neuron can be modified accordingly as shown below. The implication of this analysis is a neural network with $2n$ weight terms as opposed to $2n^2$. Fig. 8 shows the effect of this analysis. Note that the external inhibition signal, $\lambda(i)$, is zero in accordance with the CPN specifications and therefore not included in the figure.

$$q_i = \left[w^+ \sum_j q_j + \Lambda(i)\right] \Bigg/ \left[r(i) + w^- \sum_j q_j + \lambda(i)\right] \qquad (17)$$

An n port router has an SPP with n neurons. The neuron array requires only $2n$ weight terms and the summation of the q values as its inputs as seen in Fig. 9. The q values require a configurable variable in the design.

D. Weight Storage Table

The weight storage table is a dual port memory structure that maintains the characteristics of multiple RNN models. The table's dual port configuration allows for simultaneous read/read and read/write operations. Port 1 is for read operations only and is dedicated to smart packet processing. Port 2 is capable of read and write operations as required by the processing of the reinforcement learning algorithm. As illustrated in Fig. 10, the inputs for port 1 consist of a start signal and the QSD parameters. The outputs are a done and a hit signal as well as the requested data, if available. In addition to the interface incorporated by port 1, port 2 has a read/write signal and data inputs. Both ports are synchronized to an external clock source.

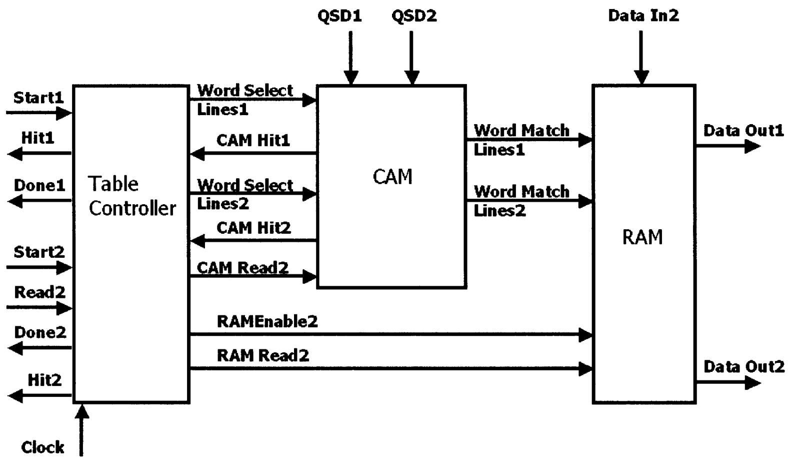


Fig. 11. Weight storage table components.

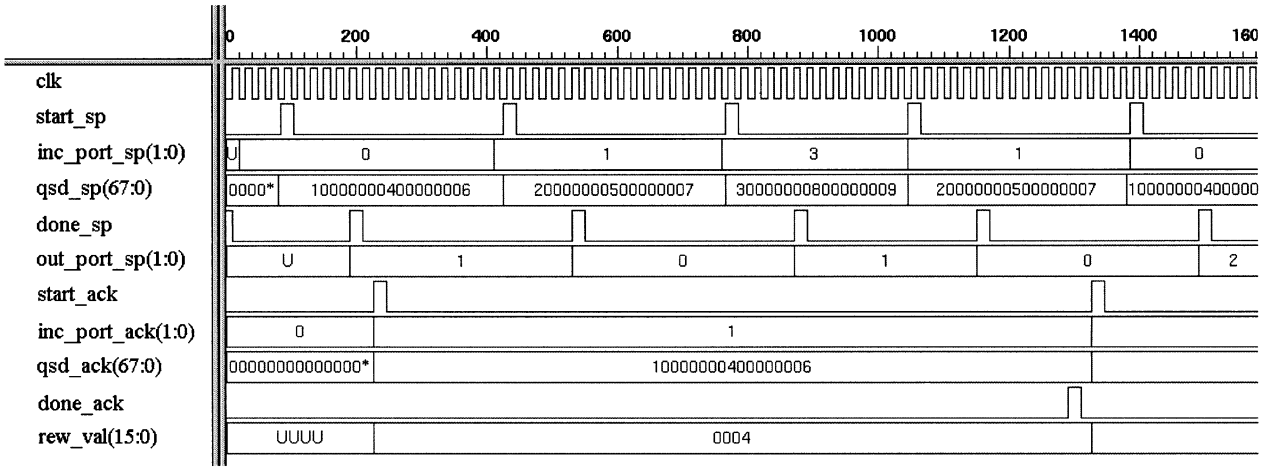


Fig. 12. SPP simulation.

The table consists of three components: a table controller, a content addressable memory, and a random access memory. Fig. 11 depicts the internal composition of the table. In the current implementation, the CAM size is 16 by 68 bits and the RAM size is 320 bytes.

E. Simulations

An SPP for a four-port CPN router has been implemented in VHDL. Simulated inputs from the system controller and acknowledgment mailbox have been applied to the design. Figs. 12 and 13 show some of the results from the simulations. In Fig. 12, the SPP has been initialized with no network information. The first smart packet arrives through port 0 (inc_port_sp) and the system controller queries the SPP for the next destination. The SP originated from node 4 with a destination of node 6 and a QoS requirement of 1 $(\text{qsd\_sp} = 10000000400000006)$. Since the SPP does not contain a relevant RNN model within its table, the device responds with a random outgoing port assignment (out_port_sp). The random value, in accordance with previous work, is cannot be the same direction from where the packet came. In this case, the packet is routed out through port 1. After the smart packet arrives at node 6, the acknowledgment packet is generated with a destination of node 4. When the ACK arrives at the intermediate router, it is processed in the acknowledgment mailbox and then forwarded accordingly. The mailbox supplies the SPP with the incoming port number of the ACK (inc_port_ack—which was the outgoing port number of the corresponding SP), the QSD parameters of the corresponding SP (qsd_ack—the ACK actually would have the source and destination reversed) and the calculated reward value. In this trial, the supplied reward value is less than the initial threshold resulting in a situation where the decision to use port 1 will be punished. Still in Fig. 12, a second smart packet with the same QSD parameters enters the router from port 0. In this instance, the SPP uses the experience of the previous smart packet to change its decision. This new smart packet is routed out through port 2. The simulation results in Fig. 13 show the low level execution of the reinforcement learning algorithm. Once the SPP receives the start signal (start_ack) from the acknowledgment mailbox, the RL component attempts to read the weights and threshold related to the given QSD from the table. In this case, the table responds with a miss. This resets the weight (w_p and w_m) and threshold terms to their default values. For this implementation, the weight, threshold and q terms each have 15 bits to the right of the ones position. Therefore, the hexadecimal value

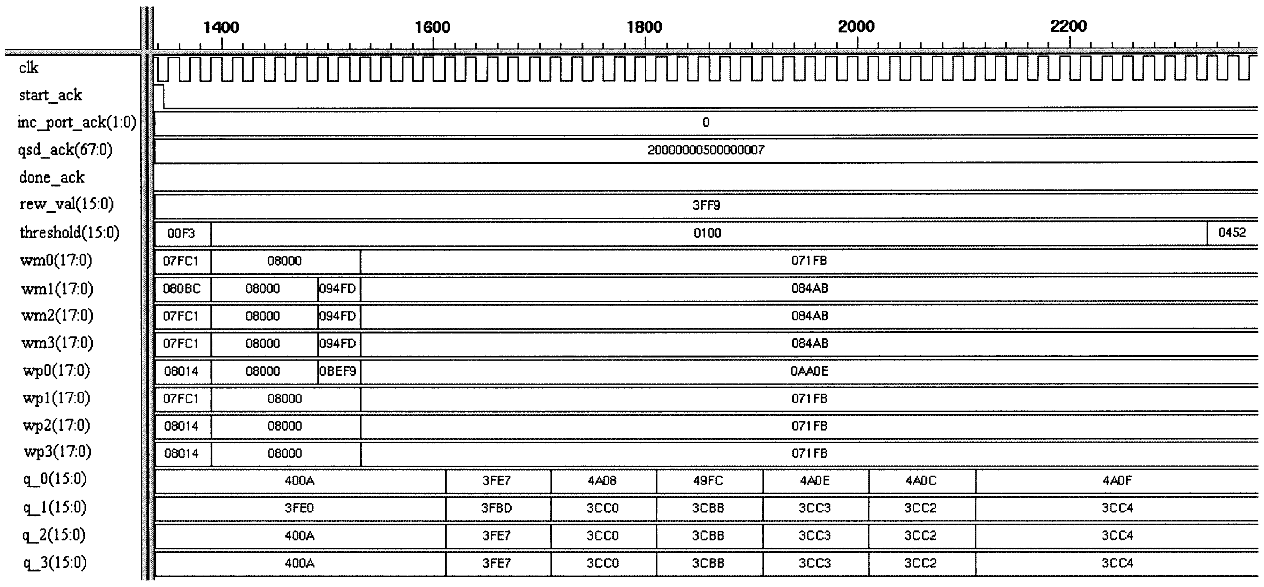


Fig. 13. Simulation results showing learning algorithm.

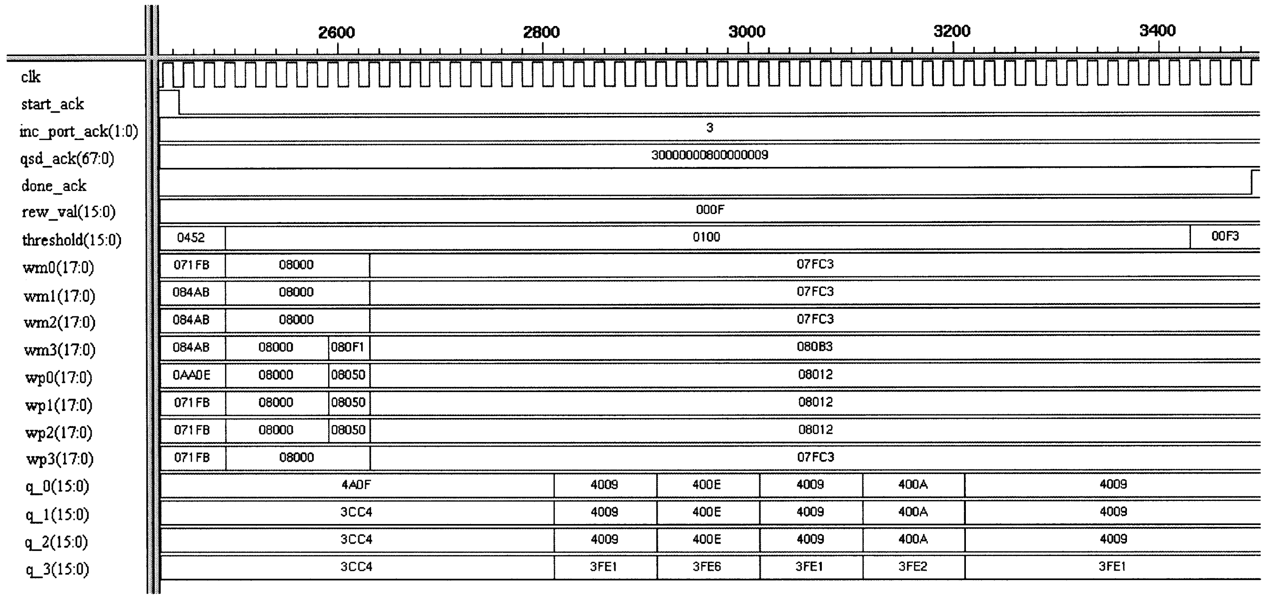


Fig. 14. Simulation showing punishment scenario.

08 000 is equivalent to 1.0000. In this examination, the exact values are not as important as being able to identify the correct trends in the simulation. After the terms are reset, the threshold is compared to the reward value from the mailbox. Since the reward value is greater, the RL component must now reward the previous decision ($\mathrm{inc\_port\_ack}$ represents the previous decision). To increase the probability of assigning port 0 again, the excitation weight w_{p0} is increased. In addition, the inhibition weights w_m associated with the other ports are incremented. This decreases the probability of one of the other ports being selected. Next, the weight terms are normalized to prevent them from growing unbounded. Once the weights are determined, they are used by the neuron array to calculate the steady state probabilities q. As seen in the figure, this iterative process enables the values to converge. Neuron 0 has the highest q at 4A0F (an actual probability of 0.578). After those calculations are complete, the RL component determines which two neurons had the highest q. The final calculation performed in this sequence is the adjustment of the threshold value. The new threshold, 017B, is significant increase over the previous value, 0100, due to the large reward that was received, 3FF9. The final step is to store the necessary RNN characteristics into the table. The threshold and weights terms are saved for future use by the RL component. Additionally, the numbers of the two neurons with the highest q's are stored. These numbers will be used by the SP interface when subsequent smart packets with the corresponding QSD need to be routed.

The simulation results for a punishment scenario are shown in Fig. 14. This time the QSD parameters of the packet are 30000000800000009 and the previous decision was port 3.

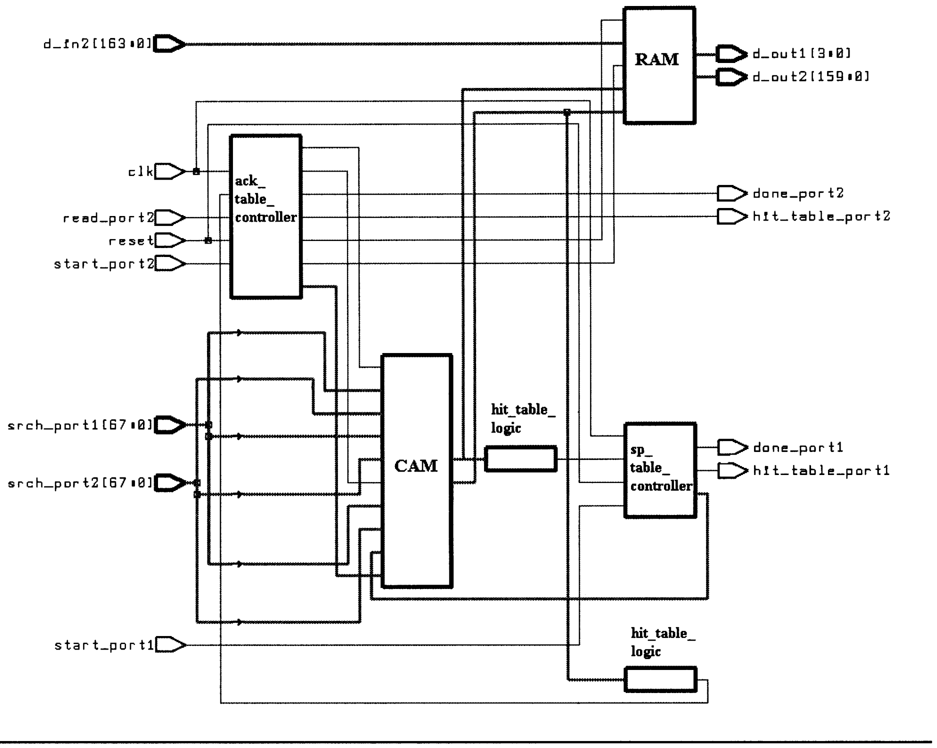


Fig. 15. RTL schematic of the "weight storage table" component.

Once again, this is a situation where there is no matching RNN model stored in the table. Therefore, the default weight and threshold values are loaded. Since the reward value is less than the threshold, the RL component must now punish the earlier decision. To decrease the probability of assigning port 3 again, the inhibition weight w_{m3} is increased. In addition, the excitation weights w_p assigned with the other ports are incremented. This will increase the chances of one of the other ports being selected. After the weight terms are normalized, they are used to calculate the new steady state probabilities q. This time neuron 3 has the lowest q at 4A0F (an actual probability of 0.4990) and the other neurons are 400A (0.5002). A new threshold value is calculated and stored in the table along with the weights and the port numbers 0 and 1. From the simulations, it can be seen that the smart packet processor only requires 6 clock cycles to service the smart packets. The combination of the table access, reinforcement learning, and steady state calculations need 55 clock cycles.

F. Circuit Implementation Details

The smart packet processor design is implemented in VHDL. Presynthesis simulations (provided in the previous subsection) are run to confirm the proper functionality for the design. The behavioral model for our design is synthesized with Synopsys tools using 0.6 μm CMOS library cells to obtain hardware circuit implementation. In synthesizing the design, we set some optimization constraints, such as maximum area, maximum delay and clock specifications. The operating frequency of the processor is set at 50 Mhz. The design is implemented in a hierarchical fashion and an example to this is shown in Fig. 15 for one of the major components in the design: "weight storage table" component. Compared to the architectural block diagram in Fig. 11, this RTL circuit depicts more implementation details such as the separate table controllers for ACK and SP packets

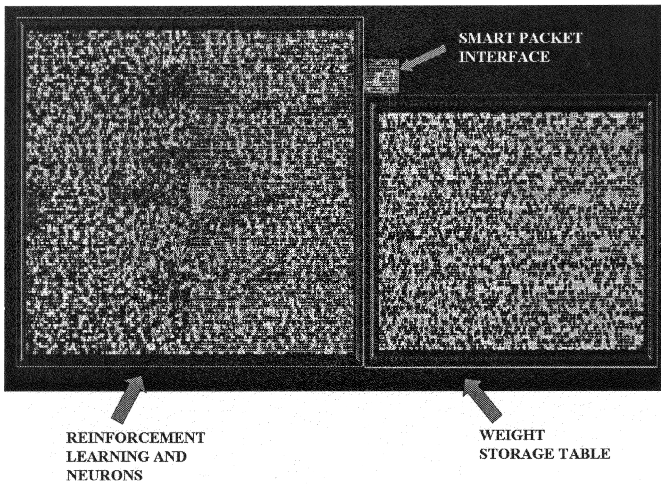


Fig. 16. SPP macro layout for 0.6-μm process.

TABLE III
OVERALL PERFORMANCE SUMMARY

Technology	0.6 μm CMOS
Supply voltage	3.3 V
Power consumption @ 50 MHz	57.7 mW
Number of gates	105,501
Core area	6.46 mm^2

TABLE IV
COMPONENTS SUMMARY

Component	# of gates	Power consumption @ 50 MHz	Critical path delay
Weight storage table	50,824	53.97 mW	9.59 ns
CAM	22,776	23.86 mW	2.76 ns
RAM	27,048	28.13 mW	5.93 ns
Reinforcement learning and neurons	54,376	3.59 mW	7.70 ns
SP interface	301	0.13 mW	4.60 ns

which ensure the dual access operation of this module. The gate-level netlist obtained after synthesis is imported to Cadence Silicon Ensemble, for floorplanning, placing and routing of the design. The layout for the design is shown in Fig. 16. The SPP core occupies 6.46 mm^2 in a 3-metal single-poly 0.6-μm digital CMOS process. Since this core is planned to be used as a macro in our network processor chip, the I/O pads are not drawn.

The performance summary for the implementation is given in Table III. The number of gates is noticably high, this is due the fact that the routing algorithm consists of many if-then and for-do statements and these statements map into a larger RTL design. This is also reflected on the total core area as being large. However, the area can be made further smaller by using a smaller process than 0.6 micron CMOS. In comparison, an average core area for a RISC CPU based processing module in a router is given as 5.2 mm^2 in 0.18-μm technology [12].

As far as the power consumption is concerned, the bulk of the measure is in the "weight storage table" component as shown in Table IV. Note that, this component includes the CAM and RAM blocks which comparably have higher switching activity than the rest of the design. Although, the "reinforcement learning" component has more number of gates, it consumes less power than the prior. However, this component shows a sizable path delay; this is mainly due to the loops used in the VHDL implementation of the learning algorithm. Based on the delay information given in Table IV, we can calculate rough estimates for the maximum number of searches per second the CAM component (16 words of 68 bits each) can provide: $(1\ \text{search}/2.76\ \text{ns}) = 362$ M searches/s. If compared to some of the few designs reported in this area, this shows a performance in between the one reported in [13] which has 9.4 M searches/s with a larger design: 128 words of 320 bits each, and the professional product reported in [14] which has 100 M searches/s with 32 Kwords of 288 bits each. The maximum number of smart packets that can be processed per second at one node by the SPP is given as follows: $(1\ \text{packet}/9.59\ \text{ns} + 4.60\ \text{ns}) = 70$ M packets/s. Here, it is assumed that smart packet routing involves only the SP interface and the weight storage table; and also the reinforcement learning algorihtm is run simultaneously. We are aware that without completing the dumb packet processing module and the rest of the CPN network processor, it would not be fair to compare this packet rate with other reported network processing hardware; however, an idea can be given such that our performance (based on smart packets with approximately 700 bits each) corresponds to 50 Gb/s wire-speed processing. This is better than the reported results given in the range of 2.4 to 10 Gb/s wire-speed processing rates [10], [15], [16].

IV. SYSTEM-LEVEL INTEGRATION

We have verified the operation of the smart packet processor in an isolated environment where the arrival of smart and acknowledgment packets is simulated. The next step is to create a basic CPN router model and use it to build a network simulation. This will provide us with the ability to test the interactions between multiple SPPs and to later add more of the functionalities found in the CPN model.

A. System Controller

Aside from the SPP, the system controller is the most complicated design of the CPN router. The system controller interacts with all of the other components. It receives packets from the input port controller (IPC) and dispatches them through the output port controllers (OPC). The system controller activates the SPP when it needs to route a smart packet and forward the locally generated acknowledgment packets to the mailbox. The state machine in Fig. 17 shows a high level representation of the system controller. The IPC requests service from the system controller upon the arrival of a smart packet. The system controller responds to the request and receives the packet for processing. The packet's destination is examined and used to determine which one of the three different routing strategies will be employed.

If the smart packet has arrived at an intermediate node in its path then the SPP will be referenced to determine the next hop. The QSD parameters and the port that the smart packet arrived through are applied to the smart packet interface of the SPP with a signal to start processing. The system controller will remain idle until the interface responds with an outgoing port number. Next, the CM of the smart packet is updated with the address of the next hop (determined by the outgoing port number) and a reward value. In this implementation, the reward value is a variable that can be controlled by the user throughout the simulation. Finally, the smart packet is forwarded to the correct outgoing port. Service from the OPC is requested. After the OPC responds, the system controller waits for the next IPC request. Routers in the CPN know the addresses of their adjacent neighbors. In the second case, if the smart packet has arrived at a router connected to the destination router, then the system controller will immediately attempt to forward the packet to its destination. The user can control the status of the port connections.

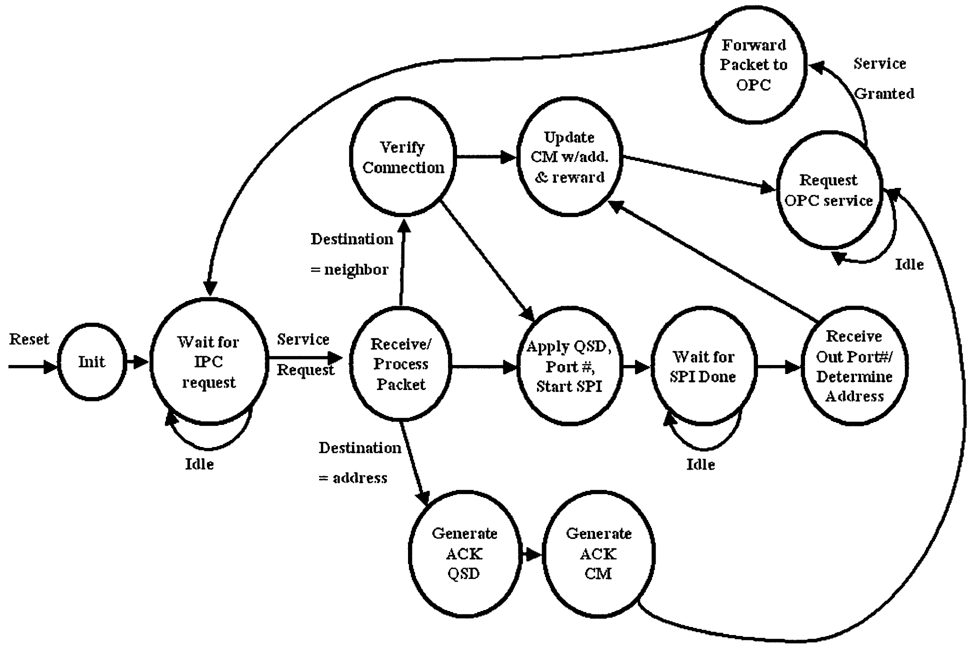


Fig. 17. System controller state machine.

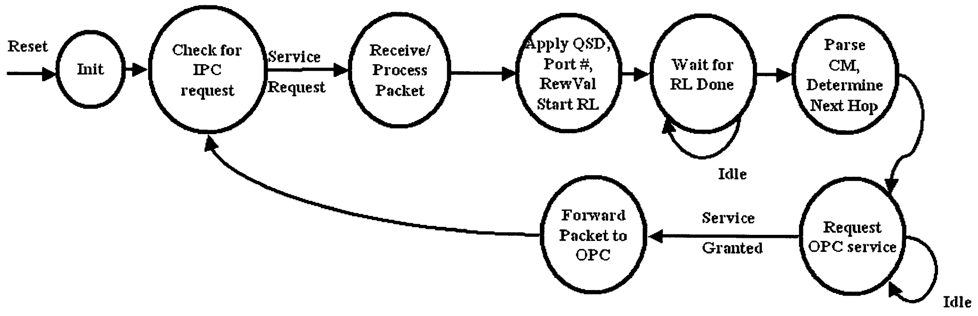


Fig. 18. Acknowledgment mailbox state machine.

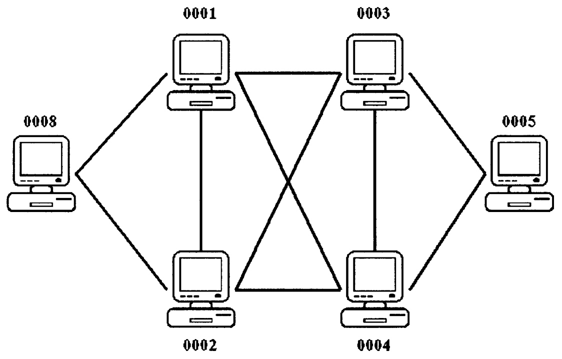


Fig. 19. Simulation configuration.

If the port has been disconnected, then the system controller, as above, will request service from the SPP to determine the outgoing port number. In the final case, if the smart packet has arrived at its destination router, then it has successfully completed its journey. An acknowledgment packet must now be generated to return the measurement data that has been collected. First, switching the source and destination of the smart packet generates the packet's QSD. Next, the CM from the smart packet is written in reverse into the acknowledgment's CM.

B. Acknowledgment Mailbox

The final component of the CPN router is the acknowledgment mailbox. The state machine for the design is shown in Fig. 18. When the IPC makes a request, the mailbox will read in the packet and its incoming port number. The mailbox must search the CM for its address. It can then extract the correct reward value from the CM. The incoming port number and the reward value can be immediately applied to the SPP's RL component. The source and destination of the acknowledgment packet must be switched so that the QSD parameter is consistent with the QSD of the corresponding smart packet. Then, the RL algorithm of the SPP can be activated. In this design, the mailbox waits for the SPP to complete its calculation. After it is complete, the mailbox reads the next address in the CM and determines the outgoing port number. Service from the OPC is requested and, once granted, the acknowledgment packet is forwarded.

C. Simulations

The topology of the network that is used for system level simulation is shown in Fig. 19. The configuration is similar to the one used in the CPN software test bed. As discussed, each

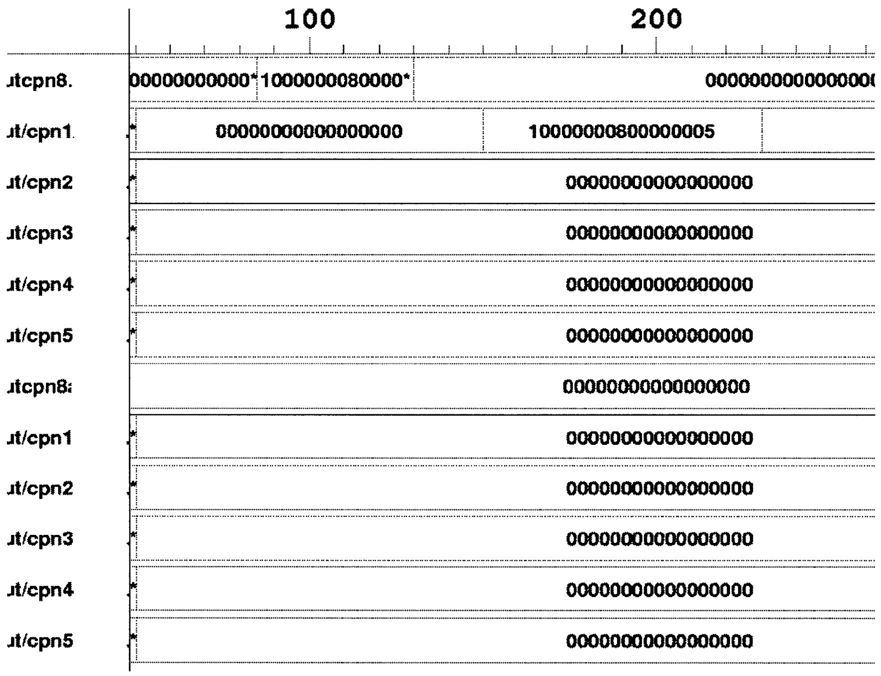


Fig. 20. Network simulation aid.

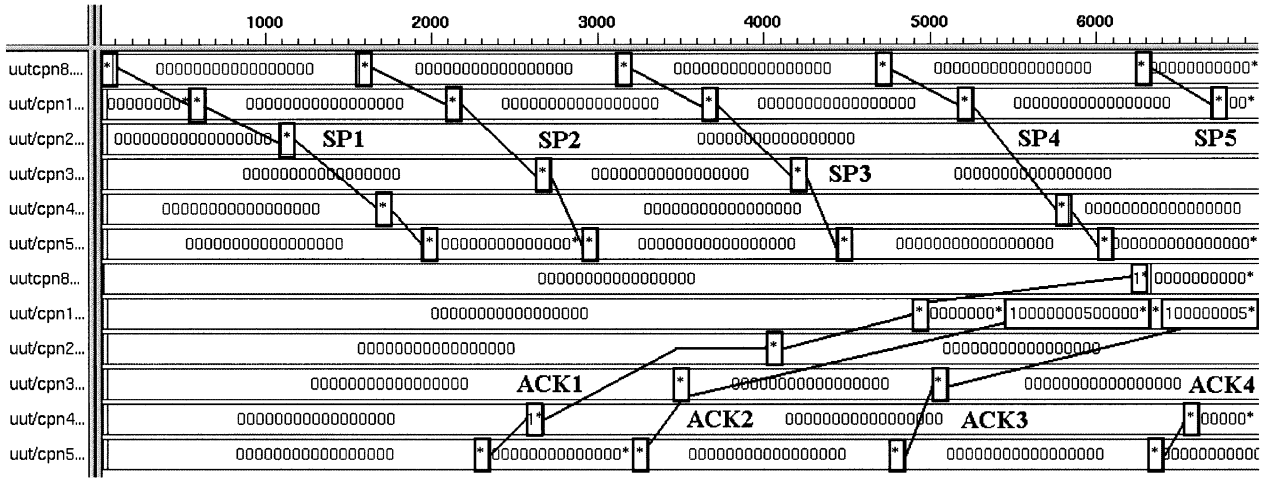


Fig. 21. Network simulation.

link in the network is configurable in terms of reward value and connection status (i.e., disconnected). In the simulation, smart packets are generated at router address 00000008 with a destination of address 00000005 and a QoS of 1. Fig. 20 is an aid for viewing subsequent figures. The top six lines show the QSD of the packets as they pass through routers. The bottom six lines show the acknowledgment packet flow. Thus, in Fig. 20, we can see a smart packet, QSD $= 10000000800000005$, moving from router 8 to router 5. For this particular simulation, we have designed the links so that the 8-1-3-5 route will be rewarded while other paths are punished. We will verify the correct behavior of the system by analyzing the results.

In Fig. 21, we can see the complete path of a smart packet, SP1. In router 1, there is currently no route information stored for the QSD combination in any of the routers. In this case the smart packet is arbitrarily forwarded from router to router until it arrives at router 4. In router 4, the packet's destination is an adjacent router. The system controller verifies the connection and the packet is immediately directed to router 5. Note that propagations involving random assignments take 540 ns (27 clock cycles). In contrast, the forwarding a packet to a direct connection (from router 3 to router 5) requires only 260 ns (13 clock cycles). The difference in time occurs because the SPP needs to be accessed before the random assignment can be made.

As seen in Fig. 21, the next smart packet, SP2, follows a different path than its predecessor, traveling through routers 8-1-3-5. Again, the assignments are arbitrarily made. This figure also shows the generation of the acknowledgment packets in router 5. The creation of ACK1 is initiated by the arrival of SP1. Since ACK1 is source routed and follows the reverse course of SP1, its first hop is router 4. Similarly, ACK2 is manufactured as a response to the receipt of SP2. ACK2's

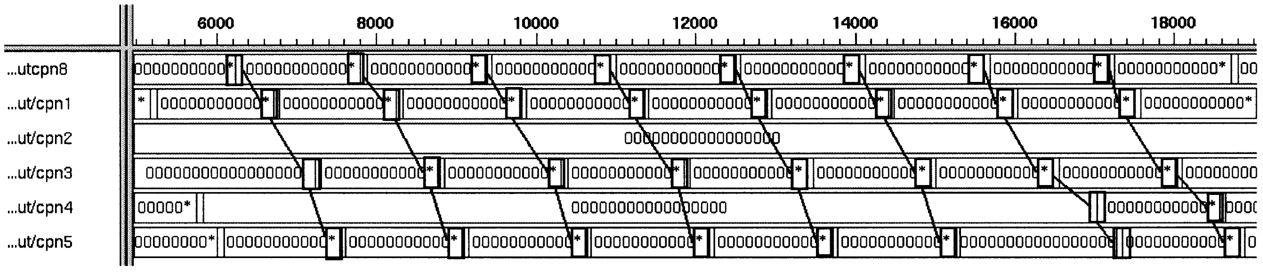


Fig. 22. Link disconnected.

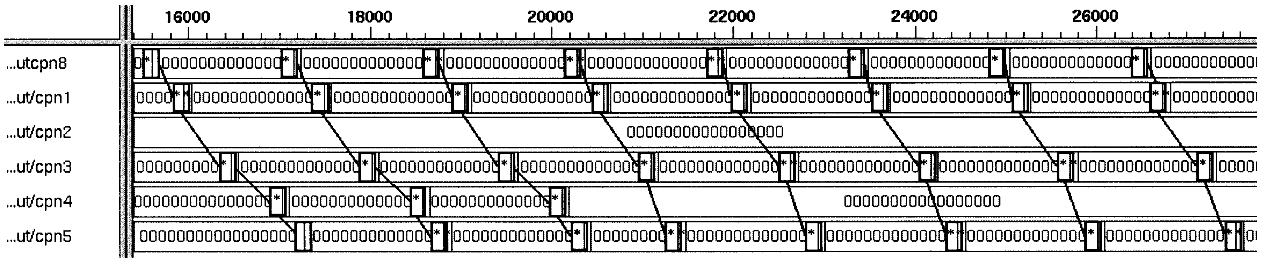


Fig. 23. Link reconnected.

first hop is router 3, as seen in the figure. Fig. 21 also shows the complete path of ACK1 and the production of more ACKs and SPs. Notice that for all the acknowledgment packets shown, the time between leaving router 5 and appearing in the next router is significantly less than the time between the subsequent hops. Since router 5 generates these acknowledgment packets, they do not invoke the reinforcement learning component of the SPP in router 5. In all other routers, the arrival of these packets will activate the RL component, which delays the transmission of the packet. Also, notice the pileup of acknowledgment packets in router 1 occurring after 5000 ns. This is due to acknowledgment packets arriving from different directions and then waiting for service. Both of these situations will be addressed in future implementations of the mailbox by using more efficient technique to extract and buffer the measurement data. Still in Fig. 21, we can see the propagation of two more smart packets, SP3 and SP4. SP3 takes the same route as SP2, 8-1-3-5, while SP4 uses a different route, 8-1-4-5. The decisions made in router 1 are still arbitrary and will remain so until the data from ACK1 is integrated into router 1's SPP. This occurs at approximately 6000 ns into the simulation.

In Figs. 22 and 23, the acknowledgment packets are not shown so that we may focus on the smart packet propagation. Fig. 22 shows the smart packets after they have learned from the experiences of previous smart packets. They have correctly chosen route 8-1-3-5. The smart packets would continue to choose this path as long as it is available and the reward for taking it is large enough. In this simulation, we decided to force them to alter their path by disconnecting the link between routers 3 and 5 at 15 000 ns. As a result, the SPP in router 3 selects router 4 as the next hop and the packet eventually completes its journey to router 5. At 21 000 ns (seen in Fig. 23), we reconnect routers 3 and 5. Once again, the system adapts and selects the correct link for packet transmission.

V. Conclusion

The implementation of a neural network routing engine for the CPN has been verified through simulations. The SPP design simultaneously services smart packets while integrating the reinforcement learning algorithm. To accomplish this, the SPP incorporates several interacting state machines with a dual port memory structure for storing and accessing the parameters of multiple RNN models. As a major improvement over the CPN software implementation, the RNN models have been reduced in size from $2n^2$ weight terms to $2n$ weight terms. The behavioral model for the design is synthesized using 0.6 μm CMOS library cells to obtain hardware circuit implementation. The current digital implementation of the learning algorithm and neurons consumes a significant amount of space. As a future direction for this work, an analog/mixed-signal RNN implementation will be considered. A basic CPN router has also been developed in VHDL to test the SPP on the system level. In addition to the SPP, its subcomponents are the input port controller, output port controllers, the system controller, and the acknowledgment mailbox. The next step in the development of the CPN router is to enhance the functionalities of the other components. Both the system controller and the acknowledgment mailbox need to be modified to handle the calculations of rewards and goals. The dumb packet switch and security controller need to be designed and integrated.

Acknowledgment

The authors would like to thank A. Ejnioui for the fruitful discussions and the critical review of the paper, and D. Harper for his CAD tools assistance.

References

[1] E. Gelenbe, R. Lent, and Z. Xu, "Design and analysis of cognitive packet networks," *Perform. Eval.*, vol. 46, pp. 155–176, 2001.

[2] Z. Xu, “Design and Analysis of Adaptive Routing in Cognitive Packet Networks,” Ph.D. Dissertation, University of Central Florida, Orlando, FL, 2001.

[3] J. Caruso, “Network processor market to grow,” *Network World High Speed LAN’s Newsletter*, Sept. 9, 2001.

[4] E. Gelenbe, “Learning in the recurrent random neural network,” *Neural Comput.*, vol. 5, no. 1, pp. 154–164, 1993.

[5] E. Gelenbe, Z. Mao, and Y. Li, “Function approximation with spiked random networks,” *IEEE Trans. Neural Networks*, vol. 10, no. 1, pp. 3–9, 1999.

[6] H. Bakircioglu and T. Kocak, “Survey of random neural network applications,” *Europ. J. Oper. Res.*, vol. 126, pp. 319–330, Oct. 2000.

[7] E. Gelenbe and K. Hussain, “Learning in the multiple class random neural network,” *IEEE Trans. Neural Networks*, vol. 13, no. 6, pp. 1257–1267, 2002.

[8] U. Halici, “Reinforcement learning algorithm with internal expectations for the random neural network,” *Europ. J. Oper. Res.*, vol. 126, pp. 288–307, Oct. 2000.

[9] N. Shah, “Understanding network processors,” M.S. thesis, University of California at Berkeley, Berkeley, CA, 2001.

[10] T. Kocak and J. Engel, “A survey of network processors,” School of EECS, Univ. of Central Florida, Sept. 2002. Available http://www.cs.ucf.edu/~tkocak/TR/NPsurvey.ps.

[11] R. Lent, “On the design and performance of cognitive packets over wired networks and mobile ad hoc networks,” Ph.D. dissertation, University of Central Florida, Orlando, FL, 2003.

[12] T. Wolf and J. S. Turner, “Design issues for high-performance active routers,” *IEEE J. Select. Areas Commun.*, vol. 19, pp. 404–409, 2001.

[13] J. Dirmar, K. Torkelsson, and A. Jantsch, “A dynamically reconfigurable FPGA-based CAM for internet protocol characterization,” in *Proc. 10th Int’l Conf. Field-Programmable Logic and Applications*, 2000, pp. 19–28.

[14] SiberCore Technologies, Inc. *SiberCAM Ultra-2M SCT2000*, 2002. Product Brief.

[15] H. Shimonishi and T. Murase, “A network processor for flexible QoS control in very high-speed line interfaces,” in *Proc. IEEE Workshop on High Performance Switching and Routing*, 2001, pp. 402–406.

[16] N. Shalaby, L. Peterson, A. Bavier, Y. Gottlieb, S. Karlin, A. Nakao, X. Qie, T. Spalink, and M. Wawrzoniak, “Extensible routers for active networks,” in *Proc. DARPA Active Networks Conf.*, 2002, pp. 92–106.

Taskin Kocak received the B.S. degree in physics and the B.S. degree in electrical and electronics engineering from Bogazici University, Istanbul, Turkey, in 1996. He received the M.S. and Ph.D. degrees in electrical and computer engineering from Duke University, Durham, NC, in 1998 and 2001, respectively.

Currently, he is an Assistant Professor and graduate program coordinator of computer engineering in the school of EECS at University of Central Florida, Orlando, FL. From June 1998 to September 2000, he worked as a mixed-signal VLSI design engineer at Semiconductor Division of Mitsubishi Electric Corporation in Research Triangle Park, NC. His current research interests are VLSI design, networking and wireless communications.

Jude Seeber received the B.S. and M.S. degrees in computer engineering from the University of Central Florida, Orlando, in 2000 and 2002, respectively.

He is currently an ASIC Engineer with 3Dlabs in Huntsville, AL. His primary responsibility is the development of the test strategy for cutting edge graphic chips.

Hakan Terzioglu received the B.S. degree in electrical engineering from Bogazici University, Turkey, in 2002. He is currently working toward the Ph.D. degree in the Department of Computer Engineering at University of Central Florida, Orlando. His areas of interest are network processors and analog circuit design.