TESTING A COGNITIVE PACKET CONCEPT ON A LAN

X. Hu, A. N. Zincir-Heywood, M. I. Heywood
Faculty of Computer Science, Dalhousie University
{xhu@cs.dal.ca, zincir@cs.dal.ca, mheywood@cs.dal.ca}

Abstract

The concept of a Cognitive Packet Network (CPN) is modified in the light of advances in intelligent routing algorithms. To do so, Cognitive and Acknowledgement packets are used to update neural networks at each router with respect to quality of service information. Data is carried by a third class of packets, thus the density of data packets is higher than in the original CPN framework. The system is demonstrated on a LAN using standard TCP/ IP protocols.

Keywords: *Computer networks, neural networks, and reinforcement learning.*

1. INTRODUCTION

Research on flow controls and routing decisions of communication networks frequently focuses on improving the intelligence and power of the protocols, switches or intermediate control nodes. Packets in the networks are passively processed, whereas many attempts have recently been made to equip packets with routing intelligence. Specific examples include the: AntNet algorithm based on a social insect metaphor for control [1]; genetic algorithm routing controllers [2]; and active networks [3]. All methods naturally provide a different series of design tradeoffs. Both AntNet and genetic approaches make use of forms of global information. Active networks incur a processing overhead on every network packet. Here, we take the Cognitive Packet Network (CPN) method [4, 5] and make modifications in the light of approaches taken by the AntNet algorithm. In addition, the method is implemented on a LAN using typical TCP/IP technology.

The resulting routing algorithm is completely distributed in operation, making use of location information collected by a separate class of packets, denoted cognitive and acknowledgement packets. Neither packet carries data and has a purpose synonymous with forward and backward ants in the AntNet algorithm. Unlike the original CPN or the AntNet algorithm, quality of service (QoS) information collected by these packets is used to update neural networks residing at the routers using a reinforcement-learning framework. It is these neural networks that are responsible for making the routing decisions. To do so, the concept of mailboxes is introduced, where (routing) neural networks are responsible for specific (or sets of) destinations.

In the following, section 2 gives an overview of the CPN concept. Section 3 describes the design and implementation of this work. A description of the simulation testbed, and simulation results are given in section 4. Finally, discussion and conclusions are presented in section 5.

2. MODIFIED CPN CONCEPT

The CPN utilized here utilities three major types of packets: cognitive, data and acknowledgement. Cognitive packets represent the agent exploring potential source-destination paths. Upon arrival of a cognitive packet at its destination, the destination node creates an acknowledgement packet. This packet is responsible for initiating the distribution of path quality of service information. Data packets are delivered following the paths discovered by the cognitive packets. Among the three types of packets, cognitive packets and acknowledgement packets play key routing roles.

In this work, a cognitive packet (CP) is a data structure that interacts with routing information to continuously search for better paths. It provides one half of a distributed search mechanism (acknowledgement packets form the other half) such that local routing decisions are able to act on global information. Each CP is described in terms of two

fields: 1) a header that carries administrative information for handling the packets, e.g., the type of packet, the source and destination addresses, and any quality of service (QoS) requirements; and 2) a Cognitive Map (CM) that contains information about where the packet is currently located, a packet view of the state of the network, and a recommendation regarding the next node to visit.

On such a network, routers serve as "parking" areas where CPs stop to execute the 'code' contained in mailboxes. Mailboxes are identified by destination, thus CPs from different sources use the same mailbox as long as they have the same destination. As a result of executing mailbox code, the next link a CP takes is identified and the packet is placed in the respective output queue. The only exception to this is when a neighbouring node represents the destination, in which case the CP is merely placed on the corresponding output queue. In each case, the CP collects time stamp and node identifiers in the cognitive map field.

Acknowledgement packets (AP) are created once a CP reaches its destination. The purpose of this packet is to propagate the quality of service information collected by the CP back to the source node as soon as possible. Whilst retracing the path, an AP locates the mailbox with corresponding destination and updates the executable information with the objective of improving the route for future packets. In order to facilitate as timely an update as possible, the AP uses a high priority queue. APs are destroyed once they reach the original source node.

The mailbox code takes the form of a neural network, updated using reinforcement learning (section 3.2). This provides a very flexible interface for modifying mailbox operation using minimal amounts of information reported by APs, whilst also being very scalable in operation.

The above methodology is rather different from that defined in the original CP framework [3, 4]. Specifically, CPs are no longer responsible for propagating data or executing 'programs'. By decoupling data from CPs the data density of the network traffic increases. Moreover, the only executable code exists in the mailboxes of each router – previously it was required that a neural network be part of each packet, along with the data packet information [4, 5]. In this case, the cognitive and acknowledgement packets act together to propagate the quality of service information back to routers. This means, we minimize the amount of additional information propagated across the network (unlike the active networking paradigm), whilst providing much more 'intelligence' than is available under current examples of the social insect metaphor.

3. IMPLEMENTATION OF THE CPN

This section details the various components of the CPN protocol, and discusses the adaptive learning rules used in this study.

3.1 Components of the Test-bed Platform

Figure 1 summarizes the organization of mailboxes and queues residing at each node in the network. As indicated above, there are three types of packets on this network. Both cognitive, and acknowledgement packets have the same packet format. This includes the header information, and the cognitive map. On the other hand, the format of a data packet includes only the header information, and data.

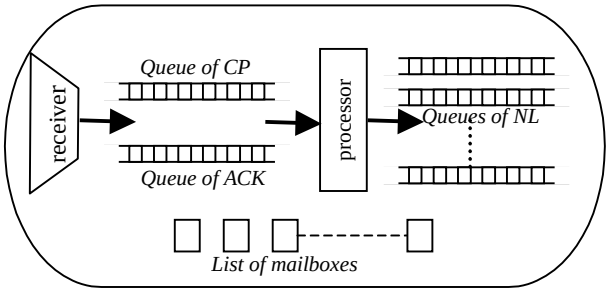

Fig. 1. A CPN node

In all packets, the header information is made up of the following fields: the type of the packet (cognitive, acknowledgement or data), the source address, and the destination address. Moreover, in the case of CPs (CP), the cognitive map stores the records of a CP's path. It has two fields used to identify the node visited by the CP and the respective trip time.

Each mailbox stores the current state of nodes on the CPN. The state is expressed in terms of a parameter set, accessed by CPs and updated by acknowledgement packets. Each mailbox includes an identifier that shows a destination node, and the weight values (w, w_{thresh}, v, v_{thresh}) required by the executable code, a neural network in this case (section 3.2).

Finally, each node has one queue for acknowledgement packets. That is, APs should affect updates to the mailbox routing algorithm as soon as possible. Whereas *n* queues contain cognitive, and data packets, where *n* corresponds to the number of links

that the current node has to its neighbours. That is, CPs collect feedback (trip time) from the environment, and therefore, should experience the same queuing properties as the data packets the system is attempting to route.

3.2 Cognitive Packet 'Code'

The algorithm used by mailboxes to make a routing decision is based on an associative reinforcement learning neural network from Gullapalli [6]. The basic idea behind this algorithm is that cognitive packets and data packets make next link 'decisions' using a probability density function (p.d.f) responsible for predicting payoff and receives feedback from the environment to evaluate those actions. Feedback is used to update p.d.f. such that the expectation of favourable evaluations in the future is increased. Ideally, a CP should be able to learn: 1) to associate with each input pattern, a link selection, for which the reinforcement signal it receives, indicates the highest degree of success; and 2) to improve its performance by using greater degrees of exploratory behaviour, when the expected reinforcement is low. The parameters governing the modification of the p.d.f. are summarized in Table 1.

Table 1. Parameters for the algorithm

Variable	Meaning	Initial value
$w_i(t)$	w weight of ith link at time t	Generated using a URNG
$v_i(t)$	v weight of ith link at time t	Generated using a URNG
w_{thresh}	Weight threshold	Generated using a URNG
v_{thresh}	Weight threshold	Generated using a URNG
$x_i(t)$	Input, the length of the ith queue at time t	A positive integer
α, β	Learning rate	The values are chosen for each test, from 0.0to 1.0
Note: URNG – uniform random number generator, it produces a value in the interval [0, 1].		

Such a model learns to produce real-valued outputs by estimating the mean, μ, and standard deviation, σ, of the Gaussian p.d.f., $\psi(\mu, \sigma)$, used to make a link selection. The reinforcement, $r(t)$, received from the environment is limited to the unit interval, with 1.0 denoting the maximum attainable reward. Expected reinforcement, $\hat{r}(t)$, is therefore employed to provide a more informative measure of performance.

The above description conforms to the ACTION-CRITIC reinforcement model in which the ACTION network models the mean response, and the CRITIC models the variance. In the case of the ACTION network, the mean, $\mu(t)$, is modeled as a weighted sum of n queue lengths, x_i, at update/node t,

$$\mu(t) = \sum_{i=1}^{n} w_i(t)x_i(t) + w_{thresh}(t) \qquad (1)$$

The expected reinforcement, $\hat{r}(t)$, as modeled by the CRITIC network is then computed as a weighted sum of the inputs using a different set of weights v_i,

$$\hat{r}(t) = \sum_{i=1}^{n} v_i(t)x_i(t) + v_{thresh}(t) \qquad (2)$$

Moreover, for a given input, the standard deviation, σ(t), depends on how close the current output is to the expected reward/reinforcement, $\hat{r}(t)$. If the expected reinforcement is high, the packet is performing well for that input, hence $\sigma(t)$ should be small. On the other hand, if the expected reinforcement is low, $\sigma(t)$ should be larger so that the packet explores a wider interval in its output range. The standard deviation is computed as,

$$\sigma(t) = max((\frac{1.0 - \hat{r}(t)}{5.0}), 0.0) \qquad (3)$$

The recommendation, $a(t)$, is computed based on $\mu(t)$ and $\sigma(t)$, defining a Gaussian distribution [6],

$$a(t) = \psi(\mu(t), \sigma(t))$$
$$= \sigma(t) * GaussianFunction + \mu(t) \qquad (4)$$

The output, $y(t)$, merely maps the recommended action to the unit interval for the purposes of interpreting it as a link selection.

$$y(t) = (1 - \exp(-a(t))^{-1} \qquad (5)$$

Table 2 gives an example of a mapping case. Finally, in the above algorithm, the input, $x_i(t)$, is a positive integer, so it also needs to be mapped to a value between 0 to 1 c.f. (5).

Table 2. Mapping to a link

Outputs	Link 0	Link 1	Link 2
Node with 1 neighbour	[0.0, 1.0]		
Node with 2 neighbours	[0.0, 0.76]	(0.76, 1.0]	
Node with 3 neighbours	[0.0, 0.52)	[0.52, 0.54)	[0.54, 1.0]

Once a CP reaches its destination, *external* reinforcement (reinforcement from the environment - QoS) is calculated in terms of trip time, $r(t)$, where this is also mapped to the unit interval. Moreover, each node along a route has a unique reinforcement value following the information propagated by the AP. Thus,

$$r(t) = (1 - \exp(-log(trip\ time))^{-1} \quad (6)$$

An AP provides for weight updating in the action network of each node as follows:

$$w_i(t+1) = w_i(t) + \alpha \bullet_w(t)\, x_i(t) \quad (7)$$

$$w_{\text{thresh}}(t+1) = w_{thresh}(t) + \alpha \bullet_w(t) \quad (8)$$

where α is a learning rate and [6],

$$\bullet_w(t) = (r(t) - \hat{r}(t))\left(\frac{a(t) - \mu(t)}{\sigma(t)}\right) \quad (9)$$

The updating of the weights for the expected reinforcement has a similar form,

$$v_i(t+1) = v_i(t) + \beta \bullet_v(t) x_i(t) \quad (10)$$

$$\bullet_v(t) = r(t) - \hat{r}(t) \quad (11)$$

where β is the learning rate parameter[6].

In summary, CPs select a destination using (1) to (5), i.e., $y(t)$ is mapped to the set of integers representing outgoing links at the current node. On the way to a destination, the CP records the intermediate node identity and trip time. When a CP reaches the overall destination, an acknowledgement packet is created that retraces the explored path backwards. On the way back to the source node, the acknowledgement packet updates the weights, defining neural networks, at each mailbox. To do so, the real-time millisecond delay value is transformed into the unit interval (6), and weights updated using (7) to (11).

4. EXPERIMENTAL SETTINGS AND SIMULATION RESULTS

Figure 2 shows the topology of the local area network (LAN) that is used as a testbed in this study. On this testbed, the main hardware configuration of each node is an AMD K6-266MHz CPU with 32 Mbytes RAM. The operating systems on these nodes are all Windows 95. The network communication protocols used are standard TCP/IP. Multiple network cards of 10M/100M are installed on each computer. All the computers on the testbed are grouped into sub-networks, and assigned different IP addresses according to the sub-network divisions. Each network card is configured to work in full duplex mode.

Furthermore, *AboutTime* [7], time synchronization software is installed on each computer to enable CPs to collect the trip time information in order to calculate the real environmental feedback. *AboutTime* uses Simple Network Time Protocol (SNTP) to synchronize time on the Internet or on a local area network. The Simple Network Time Protocol (SNTP) is described in RFC 1769 [8]. It is an adapted version of the Network Time Protocol (NTP), which is used to synchronize computer clocks on the Internet. *AboutTime*, achieves typical synchronization accuracies of ± 50 milliseconds on a local area network [7].

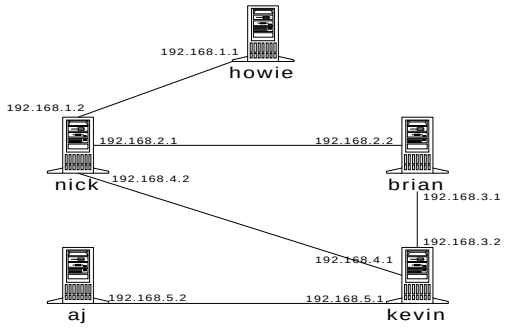

Fig. 2. The network topology of the testbed

In this study, two major scenarios are tested: 1) CPN has no data packets; and 2) CPN has data packets. Indeed, different learning parameters are used for these scenarios, table 3.

Table 3. The learning parameters that are used

Parameters	**1st run**	**2nd run**	**3rd run**
α	0.71	0.71	0.71
β	0.91	0.71	0.97
CPCR	2	5	5
DPCR-TL1	No DP	No DP	No DP
DPCR-TL2	20	20	20
CPCR: CP creating rate (packets/second) DPCR: data packet generating rate (packets/second) TL: Traffic Load			

Since the aim of our experiments is to explore how the CPs find routes to their destinations, and adapt to the changes in the environment, the percentage of arrived packets (to their target destinations) are collected during these tests to explore the performance of the CPN. Therefore, below, all the figures' results report measurements from node *Brian* for the packets targeting to go to *Howie*. As depicted in figure 4, when learning rates (α, β) are decreased, even though the CP

generating rate is increased, the system cannot learn the routes and its performance decreases compared to figure 3. On the other hand, when the learning rate is increased as well as the CP generating rate, figure 5, the performance of the system stays the same – approximately 84% - compared to the 1st run, figure 3.

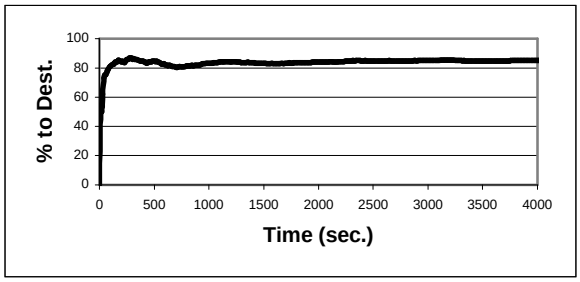

Fig. 3. Percentage of arrived packets from Brian to Howie during the 1st run without any data packets

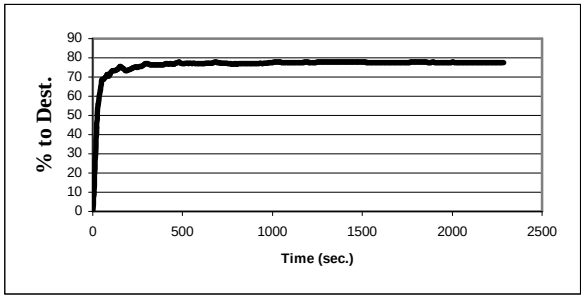

Fig. 4. Percentage of arrived packets from Brian to Howie during the 2nd run without any data packets

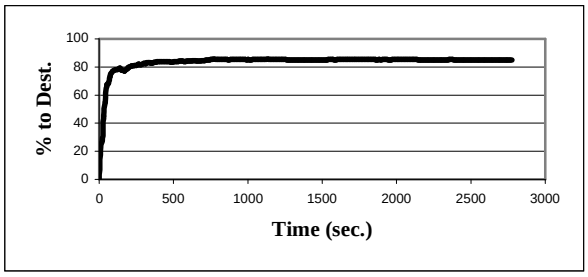

Fig. 5. Percentage of arrived packets from Brian to Howie during the 3rd run without any data packets

When data packets are introduced to the system, the performance of the system is compatible in the first run, figure 6, although it takes longer for it to converge. However, it decreases by approximately 4% in the second, figure 7, and third, figure 8, runs compared to the respective results (figures 4 and 5) of the first scenario.

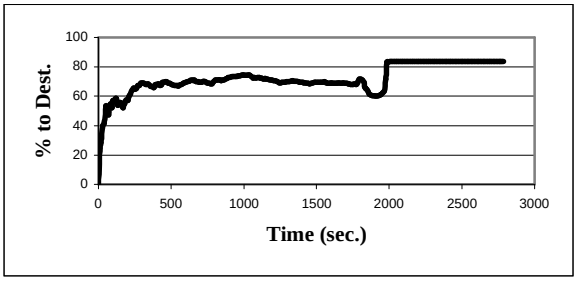

Fig. 6. Percentage of arrived packets from Brian to Howie during the 1st run with data packets

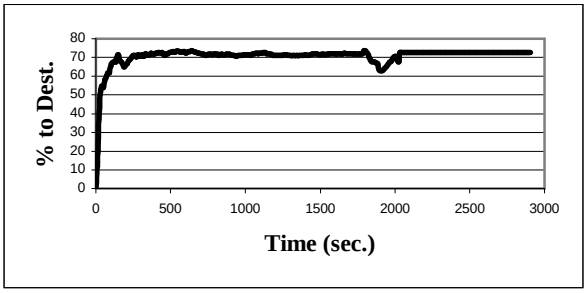

Fig. 7. Percentage of arrived packets from Brian to Howie during the 2nd run with data packets

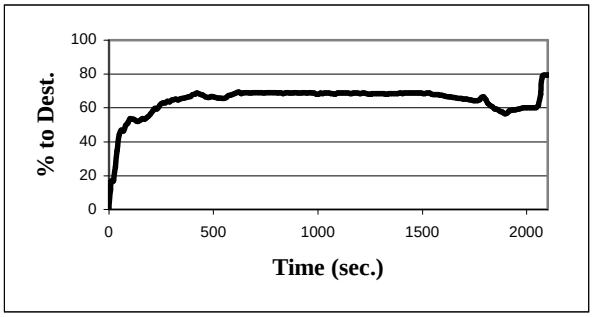

Fig. 8. Percentage of arrived packets from Brian to Howie during the 3rd run with data packets

These results indicate that the CPs are able to find the correct paths with approximately 84% accuracy (where there is 2 CPs for every 20 DPs – 1st run), and to adapt to the traffic load conditions over time under different scenarios.

5. CONCLUSIONS AND FUTURE WORK

In this study, we designed and implemented a CP Network (CPN) on a LAN environment, and modeled three types of packets: CPs, APs and data packets. CPs and APs work together to propagate QoS information into 'mailboxes' responsible for making routing decisions.

Two sets of test scenarios are investigated, where one contains data packets and the other does not. For each of these scenarios, three runs of data are collected, each with different learning parameters. It is observed that the learning rate plays a critical role in the routing of packets. The higher the rate is, the better the traffic load distributions are. Moreover, with CPs and APs working together to explore paths, data packets can be delivered to their destinations with high performance.

The studies reported are naturally of a preliminary nature. Future work is expected to test the system for its reliability and robustness. Tests on different network topologies and bigger networks as well as using different synchronization methods need to be performed. In effect, the algorithm is sensitive to different learning parameters and quality of service requirements; hence different tests under different conditions and learning algorithms are of future interest.

Acknowledgements

We gratefully acknowledge the funding provided by NSERC individual research grants for Drs. Zincir-Heywood, and Heywood.

References

[1] G. Di Caro, M. Dorigo, "AntNet: Distributed Stigergetic Control for Communication Networks". *Artificial Intelligence.* 9, 317-365, 1998.

[2] Munetomo M., Takai Y., Sato Y., "An Adaptive Network Routing Algorithm Employing Path Genetic Operators," Proceedings of the 7th International Conference on Genetic Algorithms, pp 643-649, 1997.

[3] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, G. Minden, "A Survey of Active Network Research", *IEEE Communications*, 35(1), pp 80-86, Jan 1997.

[4] E. Gelenbe, Z. Xu and E. Seref, "Cognitive packet networks", Proceedings *of the IEEE*, pp. 47-54, 1999.

[5] E. Gelenbe, E. Seref and Z. Xu, "Simulation with learning agents", *Proceedings of the IEEE*, vol. 89, no. 2, pp. 148-157, February 2001.

[6] V. Gullapalli, "A stochastic reinforcement learning algorithm for learning real-valued functions", *Neural Networks*, vol. 3, pp. 671-692, 1990.

[7] The web site for the *AboutTime* Software, http://www.arachnoid.com/abouttime/

[8] D.Mills, "Simple Network Time Protocol (SNTP)", *RFC 1769*, University of Delaware, 1995.