# Cognitive Packet Networks: QoS and Performance

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

*Abstract*— **Reliability, security, scalability and QoS (Quality-of-Service) have become key issues as we envision the future Internet. This paper presents the** *"Cognitive Packet Network (CPN)"* **architecture in which intelligent peer-to-peer routing is carried out with the help of "smart packets" based on best-effort QoS goals. Since packetized voice has stringent QoS requirements, we then discuss the choice of a "goal" and "reward" function for this application and present experiments we have conducted for "voice over CPN". Its performance is detailed via several measurements, and the resulting QoS is compared with that of the IP routing protocol under identical conditions showing the gain resulting from the use of CPN.**

**Keywords:** Quality of Service, Reinforcement Learning, Cognitive Packet Networks, Voice over IP.

## I. Introduction

In recent papers [7], [9], [11], [12] we have proposed a new network architecture called "Cognitive Packet Networks (CPN)". CPN makes use of adaptive techniques to seek out routes based on user defined QoS criteria. For instance, packet loss and delay can be used as routing criteria to improve overall reliability for the users of the network, or delay and its variance can be used to find routes which provide the QoS requested by voice packets.

A CPN carries three types of packets: smart packets, dumb packets and acknowledgments (ACK). Smart or cognitive packets route themselves, they learn to avoid link and node failures and congestion and to avoid being lost. They learn from their own observations about the network and from the experience of other packets. They rely minimally on routers. Smart packets use reinforcement learning to discover routes, and the reinforcement learning "reward" function incorporates the QoS requested by a particular user. This reward is the inverse of a QoS "goal" which each user can provide before initiating a connection. When a smart packet arrives to a destination, an acknowledgment (ACK) packet is generated by the destination and the ACK heads back to the source of the smart packet along the inverse route. As it traverses successive routers, it updates mailboxes in the CPN routers; when it reaches the source node it provides source routing information for the dumb packets. Dumb packets of a specific QoS class use successful routes which have been selected in this manner by the smart packets of the same class.

The research was supported by U.S. Army Simulation and Training Command, by Giganet Technologies Inc. and Schwartz Electro-Optics Inc.

In this paper we first recall the principles underlying the CPN design, and present the CPN test-bed we have implemented.

Voice over the Internet is a very important application that has stringent QoS requirements; thus it is a very good example for illustrating the capabilities of CPN. We therefore present the manner in which CPN's QoS based routing algorithm can be tailored to voice packets. We discuss the "goal" and "reward" function that needs to be used in this case, and then present experimental results we have obtained in CPN for voice communications. Delay, jitter and packet desequencing measurements are reported, and our results are compared with measurements on the same network infrastructure running conventional IP. The QoS improvement offered by CPN over IP is very significant, particularly when the network is heavily loaded.

## II. Smart Packets

Learning algorithms and adaptation have been suggested for telecommunication systems in the past [3]. However these ideas have seldom been exploited in networks because of the lack of a practical framework for adaptive control in packet networks.

A node in the CPN acts as a storage area for packets and mailboxes (MBs). It also stores the code used to route smart packets. It has an input buffer for packets arriving from the input links, a set of mailboxes, and a set of output buffers which are associated with output links.

Smart packet routing is carried out using a reinforcement learning (RL) algorithm [4] based on Random Neural Networks (RNN) [5], [8], [10]. The algorithm code is stored in each router and its parameters are updated by the router. For each successive smart packet, the router computes the appropriate outgoing link based on the outcome of this computation.

A recurrent RNN. with as many "neurons" as there are possible outgoing links, is used in the computation. The weights of the RNN are updated so that decision outcomes are reinforced or weakened depending on how they have contributed to the success of the QoS goal. Earlier simulations of CPN [11], and our current test-bed implementation and experiments, have been used to validate this approach.

The RNN [5] is an analytically tractable spiked random neural network model whose mathematical structure is akin to that of queuing networks. It has "product form" just like many useful queuing network models. The state $q_i$ of the $i-th$ neuron in the network represents the probability that the $i-th$ neuron is excited. Each neuron $i$ is associated with a distinct outgoing link at a node. The $q_i$, with $1 \leq i \leq n$ satisfy the following system of non-linear equations:

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

where

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

where $w^+_{ji}$ is the rate at which neuron $j$ sends "excitation spikes" to neuron $i$ when $j$ is excited, $w^-_{ji}$ is the rate at which neuron $j$ sends "inhibition spikes" to neuron $i$ when $j$ is excited, and $r(i)$ is the total firing rate from the neuron $i$. For an $n$ neuron network, the network parameters are these $n$ by $n$ "weight matrices" $\mathbf{W}^+ = \{w^+(i,j)\}$ and $\mathbf{W}^- = \{w^-(i,j)\}$ which need to be "learned" from input data. Various techniques for learning may be applied to the RNN. These include Hebbian learning, backpropagation learning [5], and Reinforcement Learning (RL) [8] which we have used in CPN.

RL is used in CPN as follows. Each node stores a specific RNN for each active source-destination pair, and each QoS class. The number of nodes of the RNN are specific to the router, since (as indicated earlier) each RNN node will represent the decision to choose a given output link for a smart packet. Decisions are taken by selecting the output link $j$ for which the corresponding neuron is the most excited, i.e. $q_i \leq q_j$ for all $i = 1, .., n$.

Each QoS class for each source-destination pair has a QoS Goal $G$, which expresses a function to be minimized, e.g., Transit Delay or Probability of Loss, or Jitter, or a weighted combination, and so on. The reward $R$ which is used in the RL algorithm is simply the inverse of the goal: $R = G^{-1}$. Successive measured values of $R$ are denoted by $R_l$, $l = 1, 2, ..$; These are first used to compute the current value of the decision threshold:

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

where $a$ is some constant $0 < a < 1$, typically close to 1. Suppose we have now taken the $l-th$ decision which corresponds to neuron $j$, and that we have measured the $l-th$ reward $R_l$. We first determine whether the most recent value of the reward is larger than the previous value of the threshold $T_{l-1}$. If that is the case, then we increase very significantly the excitatory weights going into the neuron that was the previous winner (in order to reward it for its new success), and make a small increase of the inhibitory weights leading to other neurons. If the new reward is not greater than the previous 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).

Let us denote by $r_i$ the firing rates of the neurons before the update takes place:

$$r_i = \sum_1^n [w^+(i,m) + w^-(i,m)], \quad (4)$$

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

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

Since the relative size of the weights of the RNN, rather than the actual values, determine the state of the neural network, we then re-normalize all the weights by carrying out the following operations. First for each $i$ we compute:

$$r^*_i = \sum_1^n [w^+(i,m) + w^-(i,m)], \quad (5)$$

and then re-normalize the weights with:

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

Finally, the probabilities $q_i$ are computed using the non-linear iterations (1), (2). The largest of the $q_i$'s is again chosen to select the new output link used to send the smart packet forward. This procedure is repeated for each smart packet for each QoS class and each source-destination pair.

## III. The CPN test-bed

In this section we describe the CPN protocol design and the test-bed implementation. The software we have developed has been integrated into the Linux kernel 2.2.x. with

minimal changes in the existing networking code, and it is independent of the physical transport layer. 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. The network interface is compatible with the popular BSD4.3 socket layer in Linux. It provides a single application program interface (API) for the programmer to access the CPN protocol.

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. All the nodes have been configured to use CPN and IP packets at the same time for comparison purposes. CPs are of variable size and consist basically of three areas: a header, a Cognitive map (CM) and a data portion. 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. We have tried out various topologies where each CPN router can be connected up to four other routers: Figure 1 depicts a topology used in our initial implementation. The current test-bed is shown in Figure 2.

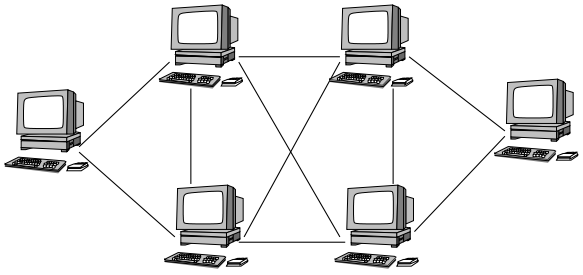

Fig. 1. Initial test-bed

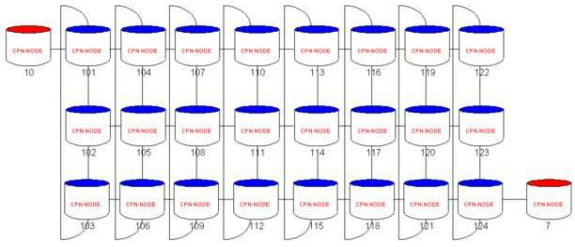

Fig. 2. The current test-bed topology

## IV. Voice Packets over CPN

Packet based voice transport in the Internet is a critical application which could dramatically increase the traffic carried over IP networks [2]. However, providing desired QoS for voice over IP is a major challenge. The QoS driven nature of CPN offers an opportunity to examine the manner in which network routing can actually serve the needs of voice transport over a packet network, and in this paper we show how CPN routing can be specifically adapted to address the needs of Real-Time voice stream delivery, by incorporating the requirements of RTCP (Real-Time Control Protocol) in the reward function used by the CPN "reinforcement learning" routing algorithm.

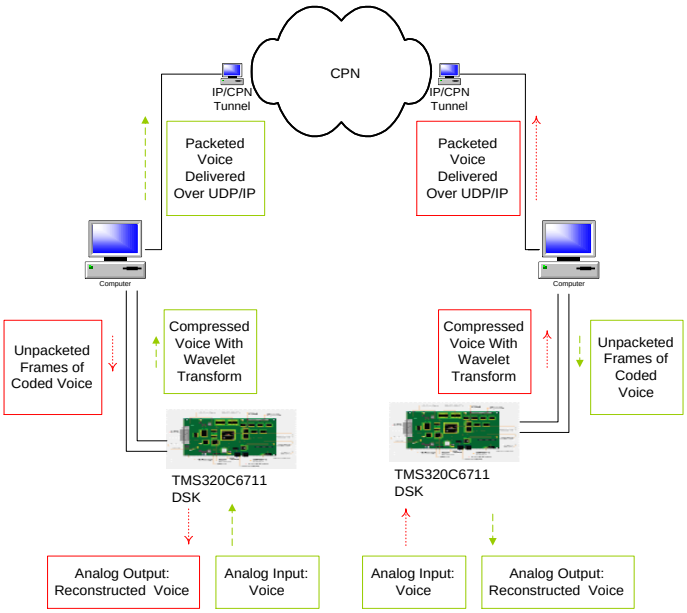


Fig. 3. Test-Bed Setup for Voice Transmission

Real-Time voice transport introduces tight constraints on QoS with respect to delay, jitter, loss and/or error, due to the limited tolerance of the human listener to both the average delay and the fluctuation of delay [2]. The QoS requirements can be divided into two categories: the end-to-end average network delivery time must be small; the end-to-end variation of the delivery time, as well as losses, must be small. If there is no buffering at the receiver, the overall delay is the sum of three components: the algorithmic delay of the codec, the processing delay of the codec and the network delay. The overall delay should not exceed 200-250 ms, but a delay of 200 ms to 800 ms is conditionally acceptable for a short portion of the conversation, when such delays are rare and far apart. Due to fluctuations of the network delay, buffering is needed at the receiver. If the buffer delay is much greater than mean network delay, the overall delay is dominated buffering period at the receiver and the algorithmic delay and processing delay of the codec.

Unlike the delay, silence periods due to delay fluctuations can compromise seriously the intelligibility of the

speech and they must be carefully controlled. The gap structure perceived by the listener will be not only be a function of network load fluctuations but also of the network policy or protocol at the receiver for dealing with these gaps. The RTCP defines a procedure to measure the network delay fluctuation, i.e. the "interarrival jitter". It is the mean deviation (smoothed absolute value) of the difference $D$ in packet spacing at the receiver compared to the sender for a pair of packets. That is: If $S_i$ is the RTCP sender timestamp for packet $i$, and $R_i$ is the timestamp of the same packet's arrival at destination (measured in RTCP timestamp units), then for two packets $i$ and $j$, $D$ is:

$$D(i,j) = (R_j - R_i) - (S_j - S_i)$$

The interrarrival jitter is calculated continuously as each data packet is received using $D$ measured for successive packet *in order of arrival*:

$$J_i = J_{i-1} + (|D(i-1,i)| - J_{i-1})/16$$

Another important measurement for Real-Time applications is the probability of packet desequencing detected at the receiver, since the receiving application should receive packets in the same order as they were sent. The procedure we use to measure the number of desequenced packets $Q$is: 1. Set $Q = 0$ at the receiver; 2. At the sender, start sending packets with sequence numbers $1, 2, 3...$; 3. At the receiver, call $S_1$ the sequence number of the first arrived packet, $S_2$ the sequence number of the second arrived packet, etc. At the arrival of the j-th packet, increment $Q \leftarrow 1$ if $S_j < max\{S_1, S_2, ...S_{j-1}\}$, else leave $Q$ unchanged; 4. Each time packet loss is detected increment $Q$ by the number of lost packets. Note that in speech transmission, packet losses as high as 50 percent can be tolerated with marginal degradation if such losses occur for very short time intervals (¡20 ms).

In order to fulfill the requirements of RTCP, we need to incorporate the variance of packet delay in the *Reward* function which is used in our Reinforcement Learning routing algorithm. In CPN, each smart or dumb packet measures the date at which it enters a node, and provides this date information to the ACK packet which heads back from the destination to the source of the corresponding smart or dumb packet. In the sequel we will drop indexing of symbols used with respect to some specific destination; it is assumed that the quantities we discuss are all indexeed separately for each separate destination. Furthermore, these quantities are specific to RTCP QoS class, so that traffic from other QoS classes in the same Cognitive Packet Network may not need them if they do not belong to the RTCP QoS class.

When an ACK (say $ACK_i$) in CPN reaches some node on its way back to the source, it estimates the forward delay $Delay_i$ from this node to the destination by simply taking the difference between the current time at the node and the time at which the corresponding SP or DP visited the same node, and dividing it by two. This rough estimate is used by CPN to avoid the need for clock synchronization between nodes. The node maintains a smoothed exponential average of these estimated forward delays for each active destination, and we denote it here by $\overline{Delay_i}$. To approximate $D(i,j)$ defined in RTCP we have used:

$$V_i \leftarrow Delay_i - \overline{Delay},$$

and jitter is approximated by

$$J_i \leftarrow J_i + (|V_i| - J_i)/16,$$

and deposited in the node's (router) mailbox, to be used by subsequent smart packets going to the same destination. Now when a smart packet for the RTCP QoS class enters a node, it uses information from the mailbox to computes the reward $R$ for the most recent routing decision:

$$R_i \leftarrow \frac{1}{\overline{Delay_i} + J_i}$$

and executes the CPN Reinforcement Learning routing algorithm to select an outgoing link from the node.

### A. Experimental Results

In Figure 3 we show the system configuration which has been used while Figure 4 shows the CPN test-bed which was used as part of the experiment. The test-bed shown in Figure 4 can also be used as an IP network, by simply replacing the CPN protocol software in each of the routers, by the IP protocol stack. Voice traffic originates at machine IP1 as IP packet traffic and its destination is machine IP2. Voice IP traffic from IP1 enters the CPN test-bed at machine CPN10, and leaves the CPN test-bed at machine CPN5 to go to the destination IP2.

Speech is sampled at 8 kHz. The frame length of each packet is 32 ms. The mean value of the bitrate is 12 kbit/s. Voice is digitized by TI's TLC320AD535 16bit data converter on a TMS320C6711 board. The digitized voice is compressed by means of a wavelet packet transform running on the FP C6711 Digital Signal Processor (DSP). The code is written in C and compiled by the Code Compiler Studio optimization software included in the board's toolkit. The compressed voice is sent to the host computer IP1 through the host port interface in real time. The host computer packetizes the compressed voice with the UDP/IP protocol and sends it to the CPN test-bed, where

a program called “Tunnelling” running on every CPN interface/edge router does the IP/CPN conversion. At IP2 a jitter buffer of about 200ms is used to avoid data overflow and/or underflow. This buffer permits the reordering of desequenced packets. A server application at IP2 then unpacks speech packets and sends the compressed speech to a DSP board, where it is reconstructed by means of the decoding algorithm and on-board Digital Analog Conversion (DAC). Our experiments compare the QoS for voice traffic over the CPN test-bed, with the same voice traffic carried over an IP network route on exactly the same test-bed.

In the first experiment we send voice from host machine IP1 to machine IP2, connected to the two edge routers CPN10, CPN5, of the CPN network; “silence” is sent from IP2 to IP1 to simulate a two-way conversation.From IP1 to IP2 the voice appears to be transitting through an IP/UDP network, while in fact it either goes through IP or it goes through CPN. Two alternatives (IP and CPN) are compared under identical traffic conditions are created in the network:

- UDP/IP voice packets travel from IP1 to IP2 with IP routing along the path {IP1, CPN10, CPN1, CPN3, CPN5, IP2}.
- UDP voice packets travel from IP1 to CPN10, then are forwarded by CPN QoS driven routing to CPN5, and then become IP packets from CPN5 to IP2IP2.

and In both cases we introduce additional bi-directional data traffic of varying intensity to observe the impact on the voice traffic. In Figure 5 (dashed lines) we show how the obstructing data traffic introduced in various network links at different traffic rates. The obstructing traffic bitrates are set at values greater then 10Mbit/s so as to obtain heavy load conditions. In Figure 6 the logarithmic value of the percentage of lost packets is plotted. For IP routing (the upper curve) we observe that the percentage of lost packets grows exponentially as the number of saturated links increases. For QoS driven CPN adaptive routing (the lower two curves), we observe that the percentage of lost packets remains small and quasi-constant. These logrithmic curves provide a clear view of the significant differences in packet loss between CPN and IP.

In another experiment, we saturate other links using data traffic (cf. Figure 7), so that only one path remains unsaturated. The percentage of lost packets (cd. Table I) remains close to zero, while values of jitter and delay are slightly greater than those from the previous experiment (see Figures 4 and 5), because CPN now has to select a route with at least 4 hops. The percentage of desequenced packets is under 5%, allowing for fast reordering of voice packets at the receiver.

CPN can in principle allow both SPs and DPs to carry

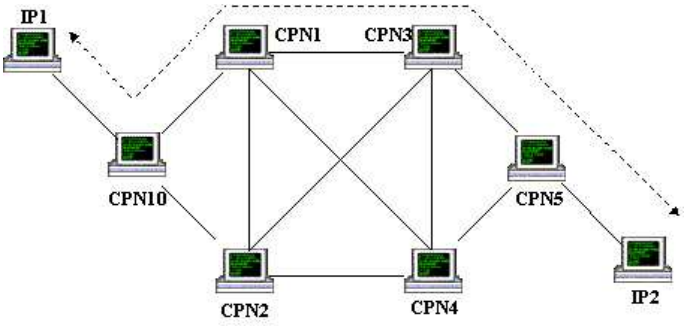


Fig. 4. CPN Test-Bed topology and IP (dashed line) route

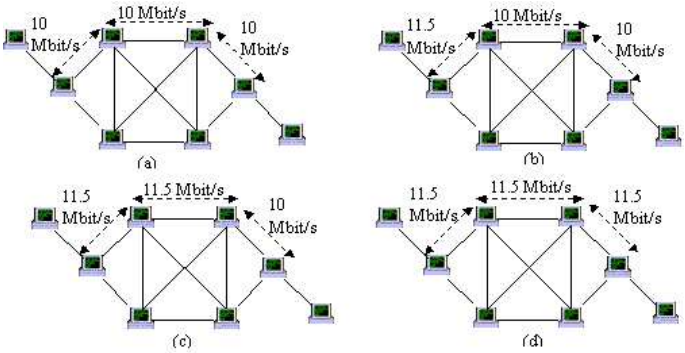


Fig. 5. The links marked by dashed lines are progressively saturated

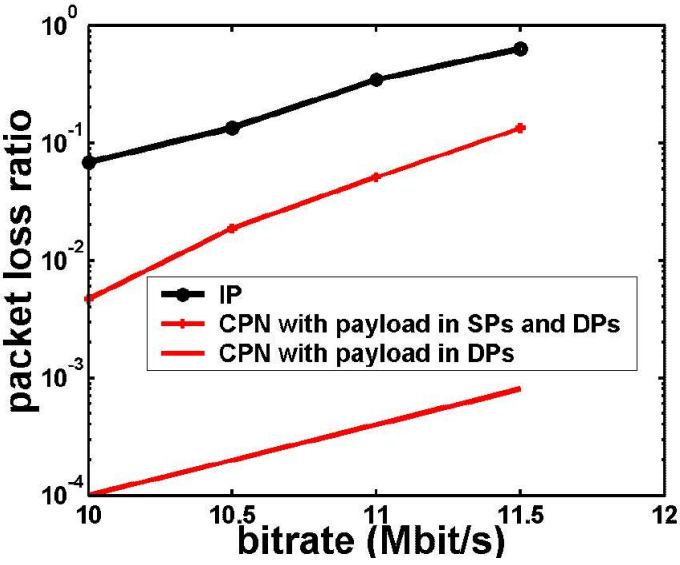


Fig. 6. Logarithmic value of the percentage of lost packets plotted as a function of link load

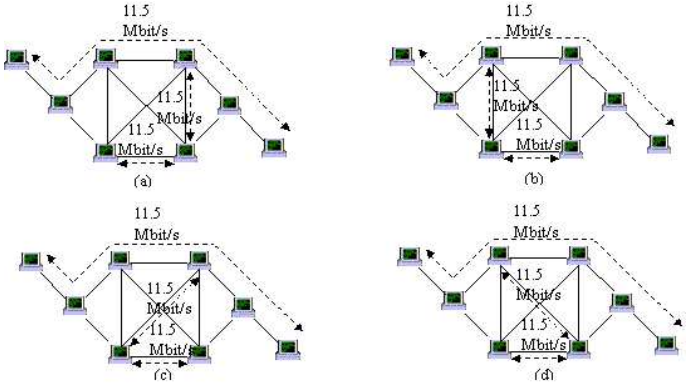


Fig. 7. More links become saturated

| | Delay | Jitter | Desequencing |
|---|---|---|---|
| Fig. 3 | 9ms | 25ms | 3.3% |
| Fig. 5 (a) | 14ms | 34ms | 3.8% |
| Fig. 5 (b) | 14ms | 36ms | 4.0% |
| Fig. 5 (c) | 13ms | 32ms | 4.3% |
| Fig. 5 (d) | 16ms | 36ms | 4.1% |

TABLE I
CPN QOS IS ROBUST TO INCREASES IN OBSTRUCTING TRAFFIC

payload, but our standard CPN implementation only places payload in DPs. Thus SPs "take risks" to search for routes providing better QoS, and DPs take advantage of the results of SPs' efforts. The next experiment shows the advantage of this latter approach by showing the performance difference between the case where both SPs and DPs carry payload, and the case where only DPs carry payload. In the following figures the $x$-axis represents the inter-packet arrival time of the flow of packets from node CPN10 to node CPN5 (refer to Figure 4). There is also a flow of obstructing traffic from CPN10 to CPN5 as illustrated in Figure 5. In the lower curves of Figures 8 and 9, we see that that if only DPs carry payload, both average delay, its standard deviation (jitter), and the packet desequencing probability remain (relatively) constant when input traffic rate varies widely. In contrast, when both SPs and DPs are allowed to carry payload (curves above), both delay, jitter and the packet desequencing probability increase dramatically with increasing input rate.

## V. CONCLUSIONS

CPN is a new packet network paradigm which addresses some of the needs of peer-to-peer networking. CPN transfers the control of QoS based best-effort routing to the connections, which use smart packets for route discovery. Routing tables are replaced by reinforcement algorithm based routing functions. A CPN carries three distinct

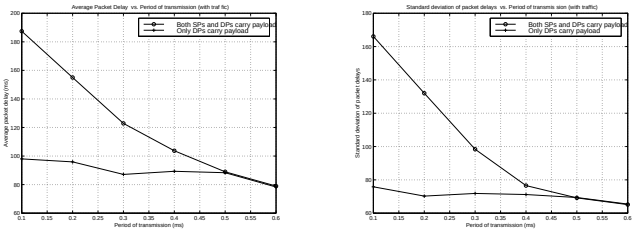


Fig. 8. Delay (left) and Jitter (right) measurements for Voice over CPN. Only DPs carry payload (curves below), and both DPs and SPs carry payload (curves above)

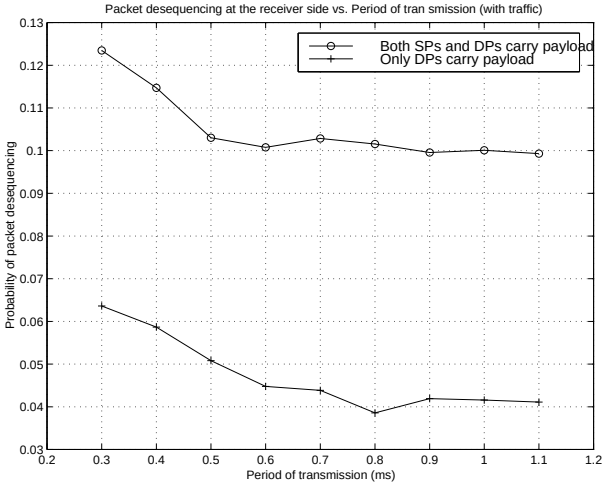


Fig. 9. Probability that a packet arrives out of sequence at the receiver: Only DPs carry payload (curve below), and both DPs and SPs carry payload (curve above)

types of packets: smart or cognitive packets which search for routes based on a QoS driven reinforcement learning algorithm, ACK packets which bring back route information and measurement data from successful smart packets, and Dumb packets which do source routing.

In this paper we have summarized the basic principles of CPN. We have presented the Reinforcement Learning (RL) algorithm which taylors smart packet routing to the QoS needs of each peer-to-peer connection. We have then described the design and implementation of our current test-bed network which uses ordinary PC-based workstations as routers. CPN software has been implemented in a Linux environment and is portable to a wide range of platforms. We have shown how the QoS driven routing algorithm used in our novel Cognitive Packet Network can be specifically designed to address the needs of store-and-forward routed packetized voice and also to optimize end-to-end delay for web traffic.

Our ongoing work includes the deployment of a large test-bed, and the inclusion of wireless links and an ad-hoc mobile extension to CPN [12], and the design of single-card routers leading to low cost router design.

## REFERENCES

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

[2] D. Minoli, E. Minoli, "*Delivering Voice over IP Network*", John Wiley & Sons, New York, 1998.

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

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

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

[6] 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.

[7] E. Gelenbe, E. Seref, Z. Xu "Towards networks with intelligent packets", *Proc. IEEE-ICTAI Conference on Tools for Artificial Intelligence*, Chicago, November 9-11, 1999.

[8] U. Halici, "Reinforcement learning with internal expectation for the random neural network" *Eur. J. of Opns. Res.*, Vol. 126, no. 2, pp. 288-307, 2000.

[9] E. Gelenbe, R. Lent, Z. Xu "Towards networks with cognitive packets," *Proc. IEEE MASCOTS Conference*, ISBN 0-7695-0728-X, pp. 3-12, San Francisco, CA, Aug. 29-Sep. 1, 2000.

[10] E. Gelenbe, E. Şeref, Z. Xu, "Simulation with learning agents", *Proceedings of the IEEE*, Vol. 89 (2), pp. 148–157, Feb. 2001.

[11] E. Gelenbe, R. Lent, Z. Xu Measurement and performance of Cognitive Packet Networks, *J. Comp. Networks*, Vol. 37, 691–701, 2001.

[12] E. Gelenbe and R. Lent "Mobile Ad-Hoc Cognitive Packet Networks", *Proc. IEEE ASWN*, Paris, July 2-4, 2002.


**Brief Biographies of the Authors:**

**Erol Gelenbe** is the University Chair Professor of Electrical Engineering and Computer Science, and Director of the School of EECS at the University of Central Florida. He is a Fellow of the IEEE and of ACM. Author of over 100 papers in the leading journals of Computer Science, Electrical Enngineering, and Applied Probability, his current work covers programmable network design, advanced simulation methods as well as queueing theory. His research is funded by NSF, the U.S. Army and by several companies. Erol is the co-author or author of four books published by Academic Press, J. Wiley and Elsevier, and some of these books have also been published in French, Japanese and Korean. He has been awarded several scientific prizes and international distinctions including the Parlar Award of Turkey (1994), the "Grand Prix France Telecom" of the French Academy of Science, the Honorary Doctorate of the University of Rome "Tor Vergata" (1996), and the "Chevalier" and "Officier" Medals of the National Order of Merit of France (1992, 2001).

**Ricardo Lent**, was born in Chiclayo, Peru. He received the B.S. degree and the "Ingeniero Electronico" title from Universidad Ricardo Palma, Lima, Peru, in 1992, and the M.S. degree in Electrical Engineering from the Universidad Nacional de Ingenieria, Lima, Peru in 1997. From 1992 to 1999 he made major contributions toward the establishment of the first Internet network in Peru. He has also been a Lecturer at the Universidad Nacional de Ingenieria and at the Universidad Nacional Pedro Ruiz Gallo in Peru. He has co-authored several papers in IEEE and other international conferences and is also co-author of a paper in the journal *Performance Evaluation*. He is currently pursuing a doctoral degree in Computer Science at the University of Central Florida, Orlando, FL.

**Alfonso Montuori** received both his Dott. Ing. and Ph.D. degrees from the Politecnico di Torino, Turin, Italy where he is currently a researcher as well as a lecturer in the Department of Electronics Engineering. His interests include packetized audio transmission, specialized microprocessor design, and computer networks. During 2001 he was a post-doctoral researcher at the University of Central Florida.

**Zhiguang Xu** received the undergraduate degree in Computer Engineering from Beijing University of Posts and Telecommunications, China and the M.S. degree in Computer Science from University of Central Florida. He received the Ph.D. degree in Computer Science at the University of Central Florida in the Fall of 2001. He was with Nortel Networks for four years before coming to UCF. His research covers neural-network applications in modelling and simulation and in the design of novel packet switching architectures. He has co-authored several papers which have appeared in the journals *Proceedings of the IEEE* and *Performance Evaluation*, and at conferences such as *IEEE Conference on Tools for Artificial Intelligence* in 1999, *International Symposium on Computer and Information Sciences* in 1999, and *IEEE MASCOTS Workshop 2000*. He is currently on the faculty of the Computer Science Department, Valdosta State University, Valdosta, Georgia.