Self-Aware Networks and QoS

EROL GELENBE, FELLOW, IEEE, RICARDO LENT, AND ARTURO NUNEZ

Invited Paper

Novel user-oriented networked systems will simultaneously exploit a variety of wired and wireless communication modalities to offer different levels of quality of service (QoS), including reliability and security to users, low economic cost, and performance. Within a single such user-oriented network, different connections themselves may differ from each other with respect to QoS needs. Similarly, the communication infrastructure used by such a network will, in general, be shared among many different networks and users so that the resources available will fluctuate over time, both on the long and short term. Such a user-oriented network will not usually have precise information about the infrastructure it is using at any given instant of time, so that its knowledge should be acquired from online observations. Thus, we suggest that user-oriented networks should exploit self-adaptiveness to try to obtain the best possible QoS for all their connections. In this paper we review experiments which illustrate how "self-awareness," through online self-monitoring and measurement, coupled with intelligent adaptive behavior in response to observations, can be used to offer user-oriented QoS. Our presentation is based on ongoing experimental work with several "cognitive packet network" testbeds that we have developed.

Keywords*—Adaptive networks, autonomic systems, Internet, IP protocols, quality of service (QoS).*

I. Introduction

At the periphery of the Internet, novel networked systems are emerging to offer user-oriented flexible services, using the Internet and LANs to reach different parts of the same systems, and to access other networks, users and services. Examples include enterprise networks, home networks (Domotics), sensor networks, and networks for military units or emergency services. The example of home networks is significant in that a family may be interconnected as a unit with PDAs for the parents and children, the health monitoring devices for the grandparents, the video cameras connected to the network in the infants' bedroom, with connections to smart home appliances, the home education server, the entertainment center, the security system, and so on.

A home network will simultaneously use different wired and wireless communication modalities, including wireless LANs (WLAN), 3G, wired Ethernet, etc. It will tunnel packets through the IP in and out of the Internet and try to satisfy the distinct quality-of-service (QoS) requirements of different connections. Such systems must allow for diverse QoS requirements, which can be implicit due to the nature of the application (e.g., alarm system connection to the security service, or video-on-demand or voice-over-IP) or which may be explicitly formulated by the end users of the network.

Networks of this kind raise interesting issues of intelligence and adaptation to user needs and to the networking environment, including routing requirements, possibly self-healing, security, and robustness. Thus, online adaptation, in response to QoS needs, available resources, perceived performance of the different network infrastructure elements, instantaneous traffic loads, and economic costs, would, therefore, be a very attractive feature of such networks. However, learning algorithms and adaptation have seldom been practically exploited in networks because of the lack of a practical framework for adaptive control, in particular for packet networks. There is also a conviction among many practitioners that feedback controls are too slow in the presence of the massive traffic peak rates which occur at the core of a communication network. However, the systems we are considering operate "on top" of network infrastructures and typically carry low traffic rates. Thus, they can potentially act to adaptively optimize their use of the communication infrastructure through judicious observation and fast decisions.

In this paper, we describe several experiments with a network architecture centered upon a QoS-driven routing protocol called a cognitive packet network (CPN). CPN is an experimental system which is implemented using PCs as routers. The CPN code is implemented on top of the Linux operating system. CPN nodes can be organized as a cloud within the IP world, and multiple CPN clouds can be interconnected via IP. The CPN nodes together route packets

Manuscript received November 21, 2003; revised April 2, 2004. This work was supported by the Engineering and Physical Sciences Research Council, U.K., under Grant GR/S52360/01.

E. Gelenbe and A. Nunez are with the Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, London SW7 2AZ, UK.

R. Lent is with the University of Central Florida, Orlando, FL 32816 USA.

Digital Object Identifier 10.1109/JPROC.2004.832952

across a mixed CPN/IP network to offer the best QoS, where QoS is defined by the users via QoS Goals.

CPN packets do *not* carry code, and CPN routers are *not* programmable by the network end user or application. This choice has been made to avoid creating excessive complexity in the network, and also to avoid increasing the risk of insecurity in the network. On the other hand, CPN uses smart packets (SPs) to collect measurements as they travel through the network. SPs carry QoS information related to the end users; they are routed using neural network algorithms which are resident in the nodes and which use both the SPs' QoS information and data which is locally resident at the nodes. This local data in the nodes is updated with information brought by acknowledgment (ACK) packets, which return to nodes on an SP's path, as a result of the SP's successful arrival at its destination. After briefly summarizing the principles and mechanisms of CPN routing, this paper illustrates how network self-awareness can be exploited in favor of user QoS by presenting three experiments we have conducted on the CPN testbeds that we have designed and implemented.

II. The CPN

In order to investigate the potential of using self-awareness and adaptiveness to offer QoS to users, we have developed a practical packet-switching architecture that allows a network with an arbitrary topology to observe its state in a distributed manner. These observations are then used by an online algorithm running autonomously at each node to make routing decisions based on an estimate of QoS. However, these routing decisions are restricted to certain "smart" packets, which then inform the source about the paths they have found which offer the best QoS. These paths are then used by the payload carrying packets until a better path is found by the SPs.

Thus, CPN [6], [7], [8], [10] is a packet routing protocol which addresses QoS using adaptive techniques based on online measurement. Although most of our work on CPN concerns wired networks, we have also developed a wireless extension of which can operate seamlessly with wired CPN or IP networks [9].

In CPN, users are allowed to declare QoS Goals such as "Get me the data object Ob via the path(s) of highest bandwidth which you can find," where Ob is the handle of some data object [2], or "Find the paths with least power consumption to the mobile user Mn," or "Get the video output Vi to my PDA as quickly as possible" (i.e., with minimum overall delay).

CPN is designed to *accept direction* by inputting Goals prescribed by users. It exploits *self-observation* with the help of SPs so as to be aware of the network state including connectivity of fixed or mobile nodes, power levels at mobile nodes, topology, paths, and path QoS. It performs *self-improvement*, and *learns from the experience of SPs* using neural networks and genetic algorithms [11] to determine routing schemes with better QoS. It will *deduce* hitherto unknown routes by combining or modifying paths which have been previously learned so as to improve QoS and robustness.

CPN makes use of three types of packets: SPs for discovery; source-routed dumb packets (DPs) to carry payload; and ACK packets to bring back information that has been discovered by SPs. Conventional IP packets tunnel through CPN to seamlessly operate mixed IP and CPN networks. SPs are generated by a user 1) requesting that a path having some QoS value be created to some CPN node or 2) requesting to discover parts of the network state, including location of certain fixed or mobile nodes, power levels at nodes, topology, paths, and their QoS.

SPs *exploit the experience of other packets* using random neural network (RNN)-based reinforcement learning (RL) [3], [7]. RL is carried out using a Goal which is specified by the user who generated a request for the connection. The decisional weights of an RNN are increased or decreased based on the observed success or failure of subsequent SPs to achieve the Goal. Thus, RL will tend to prefer better routing schemes, more reliable access paths to data objects, and better QoS. In an extended version of the CPN network, which is presented in [11], but which we do not discuss in this paper, the system *deduces* new paths by combining previously discovered paths, and using the estimated or measured QoS values of those new paths to select better paths. This is similar conceptually to a genetic algorithm which generates new entities by combination or mutation of existing entities, and then selects the best among them using a fitness function. These new paths can be tested by probes so that the actual QoS can be evaluated.

When an SP arrives to its destination, an ACK is generated and heads back to the source of the request. It updates *mailboxes* (MBs) in the CPN nodes it visits with information which has been discovered, and provides the source node with the successful path to the node. All packets have a lifetime constraint based on the number of nodes visited, to avoid overburdening the system with unsuccessful requests or packets which are in effect lost. A node in the CPN acts as a storage area for packets and MBs. It also stores and executes the code used to route SPs. It has an input buffer for packets arriving from the input links, a set of MBs, and a set of output buffers which are associated with output links. CPN software is integrated into the Linux kernel 2.2.x, providing a single application program interface (API) for the programmer to access CPN. CPN routing algorithms also run seamlessly on *ad hoc* wireless and wired connections [9], without specific dependence on the nature (wired or wireless) of the links, using QoS awareness to optimize behavior across different connection technologies and wireless protocols.

A. Routing Using SPs

The SP routing code's parameters are updated at the router using information collected by SPs and brought back to routers by the ACK packets. Since SPs can meander and get lost in the network, we destroy SPs which have visited more than a fixed number of nodes, and this number is set to 30 in the current testbeds. This number is selected based on the fact that it will be two to three times larger than the diameter of any very large network that one may consider in practice.

For each incoming SP, the router computes the appropriate outgoing link based on the outcome of this computation. A recurrent RNN [1] 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. In the RNN the state q_i of the ith neuron in the network is the probability that the ith neuron is excited. Each neuron i is associated with a distinct outgoing link at a node. The q_i satisfy the system of nonlinear equations

$$q_i = \lambda^+(i)/[r(i) + \lambda^-(i)] \tag{1}$$

where

$$\begin{aligned} \lambda^+(i) &= \sum_j q_j w_{ji}^+ + \Lambda_i \\ \lambda^-(i) &= \sum_j q_j w_{ji}^- + \lambda_i \end{aligned} \tag{2}$$

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.

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 an SP. 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 $i = 1, \ldots, n$. 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 $R_l, l = 1, 2, \ldots$; these are first used to compute the current value of the decision threshold

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

where $0 < a < 1$, typically close to one. Suppose that we have now taken the lth decision which corresponds to neuron j, and that we have measured the lth 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

$$r_i = \sum_1^n [w^+(i,m) + w^-(i,m)]. \tag{4}$$

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

- if $T_{l-1} \leq R_l$
  - — $w^+(i,j) \leftarrow w^+(i,j) + R_l$
  - — $w^-(i,k) \leftarrow w^-(i,k) + (R_l)/(n-2)$, if $k \neq j$;
- else
  - — $w^+(i,k) \leftarrow w^+(i,k) + (R_l)/(n-2), k \neq j$
  - — $w^-(i,j) \leftarrow w^-(i,j) + R_l$.

Since the relative size of the weights of the RNN, rather than the actual values, determines the state of the neural network, we then renormalize all the weights by carrying out the following operations. First, for each i we compute

$$r_i^* = \sum_1^n [w^+(i,m) + w^-(i,m)] \tag{5}$$

and then renormalize the weights with

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

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

B. Evaluation of the Percentage of SPS Needed During a Connection

An important question is whether the scheme we have proposed can only function if the number, or percentage, of SPs (and, hence, ACKs) used is very high. This is a question that we have examined attentively, both by examining the actual length of SPs, and with numerous experiments [6]. The results of one of these experiments for a heavily loaded network is summarized in Fig. 1, where we report the round-trip delay experienced by SPs and DPs and by all packets, when the percentage of SPs added on top of DP traffic was varied from 5% to 100% in steps of 5%. In these experiments, the user specified QoS Goal is "delay" so that what is being measured is the quantity that the user would like CPN to minimize. The top curve shows the round-trip delay for SPs, while the bottom curve is the round-trip delay for DPs, with the average delay of all packets being shown in the middle. As

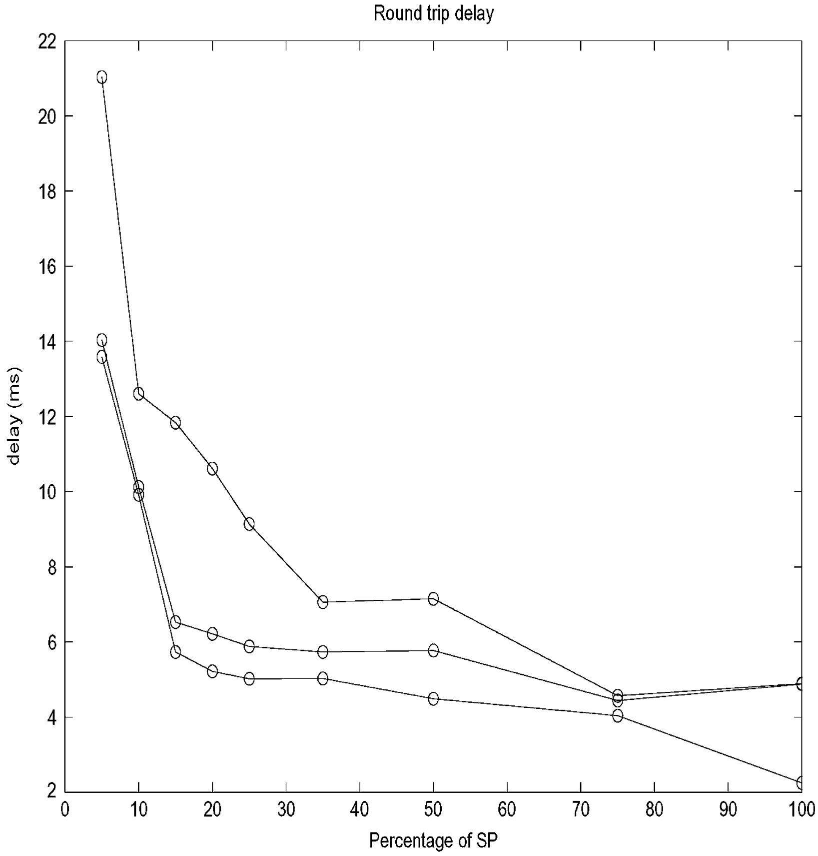

Fig. 1. Average round-trip delay for smart (top) and dumb (bottom) packets, and average delay for all packets (center) as a function of the percentage of SPs. These measurements were obtained for an end-to-end connection in the presence of obstructing traffic on several links of the network.

expected, when there are 100% of SPs, the average delay for SPs is the same as the average delay for all packets. The interesting result we observe is that as far as the DPs are concerned, when we have added some 15% or 20% of SPs, we have achieved the major gain in delay reduction. Going beyond those values does not significantly reduce the delay for DPs. In effect, DPs are typically full-sized Ethernet packets (as are IP packets in general), while SPs and ACKs are 10% of their size. Thus, if 20% of SP traffic is added, this will result in 14% traffic overhead, if ACKs are generated by both DPs and SPs, and only 4% of traffic overhead if ACKs are only generated in response to SPs. Note also that ACKs and DPs do not require a next hop to be computed at each node, contrary to IP packets. Both in CPN and IP we can of course reduce next-hop computations using appropriate caching and hardware acceleration.

III. Cold Start Setup Time Measurements

One of the major requirements of CPN is that it should be able to start itself with *no* initial information, by first randomly searching, then progressively improving its behavior through experience. Since the major function of a network is to transfer packets from some source S to some destination D, CPN must be able to establish a path from S to D even when there is no prior information available in the network. The network topology we have used in these experiments is shown in Fig. 2, with the source and destinations at

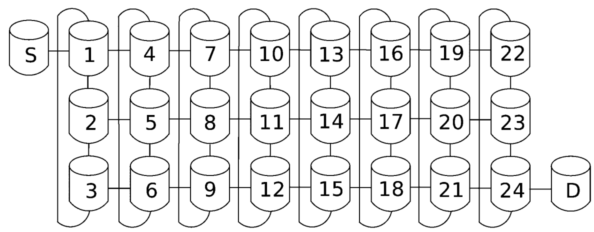

Fig. 2. CPN network topology for cold start experiments.

the top left and bottom right ends of the diagram. The network contains 24 nodes, and each node is connected to four neighbors. Because of the possibility of repeatedly visiting the same node on a path, the network contains an unlimited number of paths from S to D. However, the fact that SPs are destroyed after they visit 30 nodes, does limit this number though it still leaves a huge number of possible paths. In this set of experiments, the network is always started with empty MBs, i.e., with no prior information about which output link is to be used from a node, and with neural network weights set at identical values, so that the neural network decision algorithm at nodes initially will produce a random choice. Each point shown on the curves of Figs. 3 to 5 are a result of 100 repetitions of the experiment under identical starting conditions.

Let us first discuss Fig. 3. An abscissa value of ten indicates that the number of SPs used was ten, and—assuming

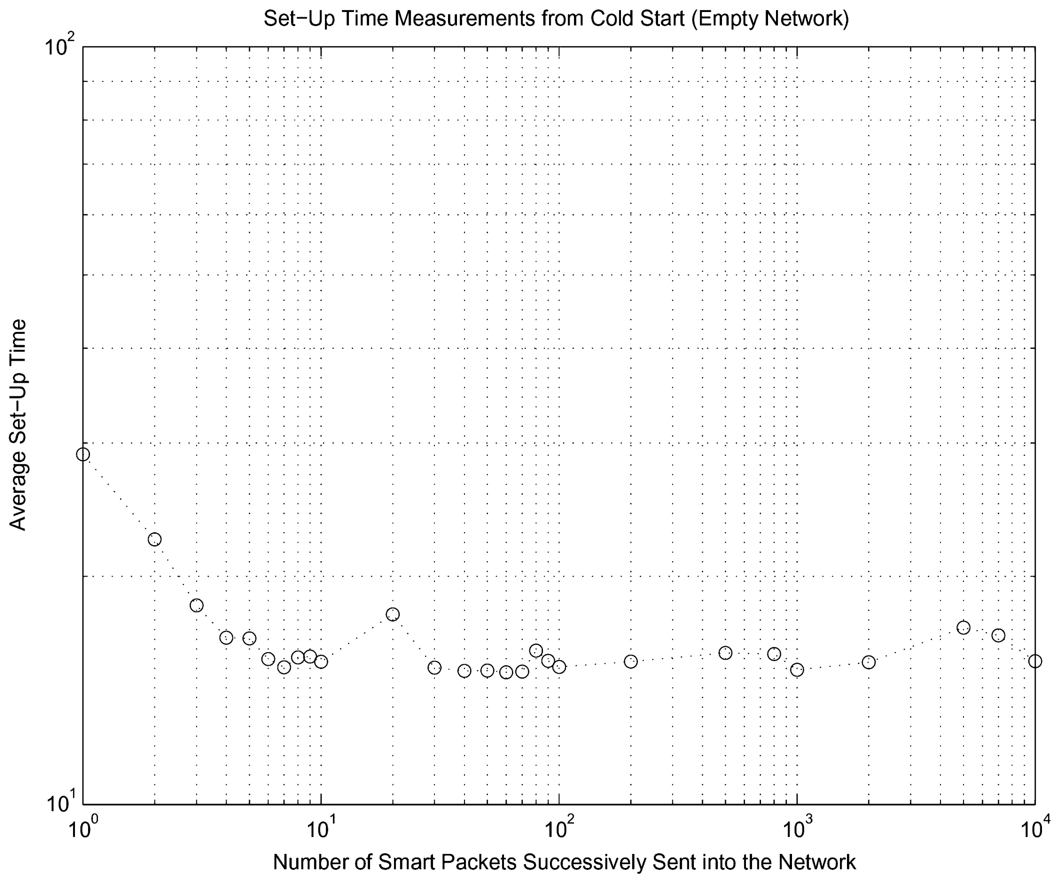

Fig. 3. Average network setup time from cold start, as a function of the initial number of SPs.

that the experiment resulted in an ACK packet coming back to the source—the ordinate gives the average time (over the 100 experiments) that it elapse between the instant that the first SP was sent out, and the first ACK comes back. Note that the first ACK will be coming back from the correct destination node and that it will be bringing back a valid forward path that can be used by the subsequent useful traffic. We notice that the average setup time decreases significantly when we go from a few SPs to about ten, and after that, the average setup time does not improve appreciably. Its value somewhere between 10 and 20 ms actually corresponds to the round-trip transit time through the hops. This does not mean that it suffices to have a small number of SPs at the beginning, simply because the average setup time is only being measured for the SPs which are *successful*; unsuccessful SPs are destroyed after 30 hops.

Thus, Fig. 4 gives a more complete understanding of what is happening. Again for an x-axis value of over ten packets, we see that the probability of successfully setting up a path is one, while with a very small number of packets this figure drops down to about 0.65. These probabilities must of course be understood as the empirically observed fraction of the 100 tests which result in a successful connection. The conclusion from these two data sets is that to be safe, starting with an empty system, a fairly small number of SPs, in the range of 20–100, will provide almost guaranteed setup of the connection, and the minimum average setup time. Fig. 4 provides some insight into the dynamics of the path being set up. Inserting SPs into the network is not instantaneous, and they are fed into the network sequentially by the source. The rate at which they are fed in is determined by the processing time per packet at the source, and also by the link speeds. Since here the link speed is 100 Mb/s and because SPs are only some 200 B long at most, the limiting factor appears to be the source node's processing time. Since the previous curves show that connections are almost always established with as few as ten SPs, and because the average round-trip connection establishment time is quite short, we would expect to see that a connection will generally be established before all the SPs are sent out by the source. This is exactly what we observe in Fig. 5. The x axis shows the number of SPs sent into the network, while the y axis shows the average number sent in (over the 100 experiments) before the first ACK is received. For small numbers of SPs sent out by the source, until the value ten or so, the relationship is linear. However, as the number of SPs being inserted into the network increases, we see that after (on the average) 13 packets or so have been sent out, the connection is already established (i.e., the first ACK has returned to the source). This again indicates that a fairly small number of SPs suffice to establish a connection. In addition to the experiments on the testbed, simulations have been conducted for a 1000-node network with results which are significantly similar.

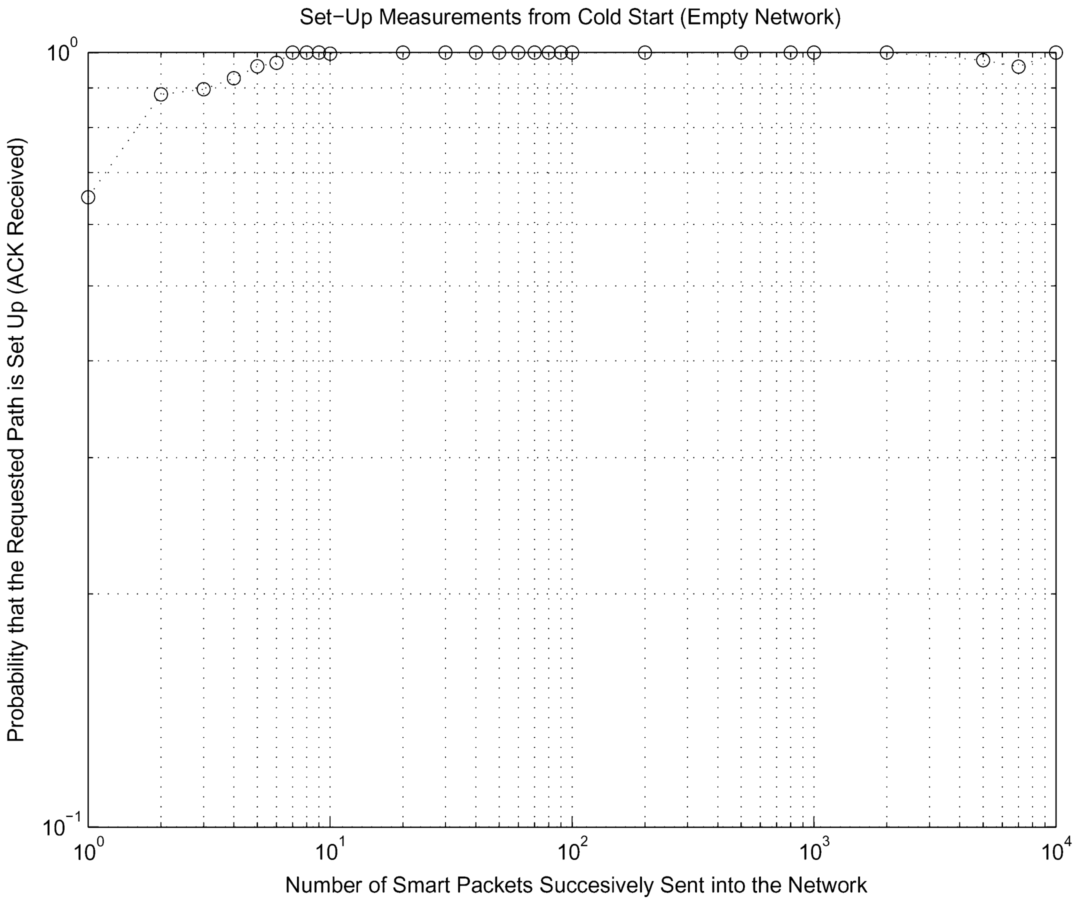

Fig. 4. Probability of successful connection from cold start, as a function of the initial number of SPs.

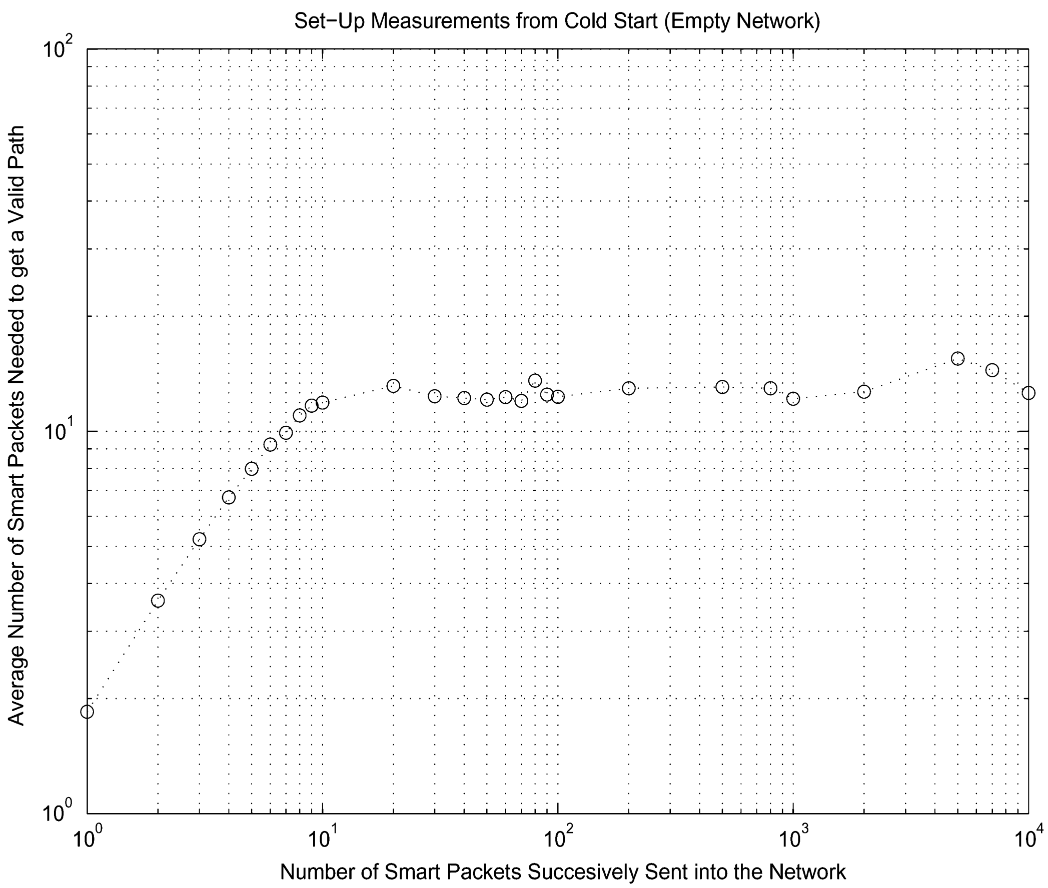

Fig. 5. Average number of SPs needed to obtain a path to the destination.

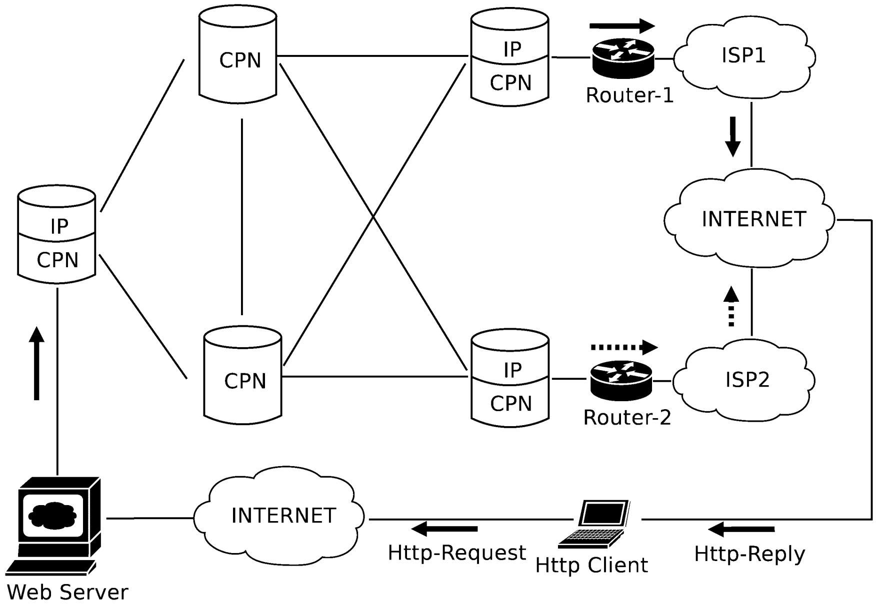

Fig. 6. Experimental setup for dynamic QoS control.

IV. Dynamic QoS-Based Traffic Routing From a Web Server Over the Internet

In this section, we present an experiment in which a CPN "cloud" or subnetwork, operating within the Internet, is used to dynamically route traffic which is being sent by a web server WS back to a web user W through different Internet ports so as to minimize the delay from the WS back to the user W.

In the system layout shown in Fig. 6, a web user at workstation W accesses to a web server WS via the Internet. The WS is connected to a CPN cloud which acts as an adaptive flow control system, and the CPN cloud is in turn connected to the Internet via two distinct ports which are implemented with conventional IP routers: A30, which connects into ISP ISP1, and A40, which connects into another provider, ISP2.

User W requests that transfers from WS back to W via the Internet arrive with *minimum delay*. Thus, in this experiment, "delay" is the QoS Goal that W has selected. Thus, in order to evaluate whether the CPN adaptive controller is indeed able to satisfy the user's QoS Goal, we artificially introduced additional delay values at the two alternate ports A30 and A40 so that the *difference in delay between the two* can be varied in a controlled manner.

WS responds to these requests by generating standard Internet IP packets, which enter into the CPN shown at the top of the figure; these packets then tunnel through the CPN, which dynamically directs them back into the Internet via two alternate ISP ports A30 and A40 shown at the right side of the figure. From there they merge into the Internet and return to the W as shown.

Figs. 7 and 8 show the fraction of traffic taking A30 (L) and A40 (R) as this delay is varied, and demonstrate that the CPN subsystem is indeed dynamically directing traffic quite sharply in response to the user's QoS goal. Figs. 9 and 10 show that when the delay through either port is identical, the instantaneous traffic via both ports is very similar. On the other hand, Figs. 11 and 12 clearly show that the instantaneous traffic strongly differs depending on which port has a higher measured delay, as do Figs. 13 and 14, but for the opposite imbalance in delay. Figs. 11–14 also clearly show that the dynamic control provided by CPN requires an adaptation time (seen at the left of the figures, and roughly of the order of 100 ms) before most of the traffic actually takes the best output port in each case.

V. Conclusion

In this paper, we have discussed the concept of a network that uses online measurement and probing as a means to estimate the QoS that may be expected from different routing choices, and then uses the outcome to forward payload along the resulting paths. The system we propose carries out probing with SPs continuously during a connection. ACK packets coming back from the destination nodes to the intermediate and source nodes bring back the results of the probing, and provide information about the paths which have been selected based on user QoS Goals. We have summarized experimental data showing that a relatively that a comparatively small fraction of SPs and ACKs, compared to total user traffic, is needed to serve the users' QoS Goals, and that a small number of SPs can suffice to initially set up paths. We have also shown how a CPN subsystem can be inserted into the Internet to carry out traffic engineering functions based on QoS. Other results we have not reported on here have discussed critical applications such as voice-over-CPN [10]. Future work will consider the experimental insertion of multiple CPN clouds into the Internet so as to address the QoS needs of selected groups. We will also study the use of mechanisms derived from CPN to provide protection against denial-of-service attacks to network nodes.

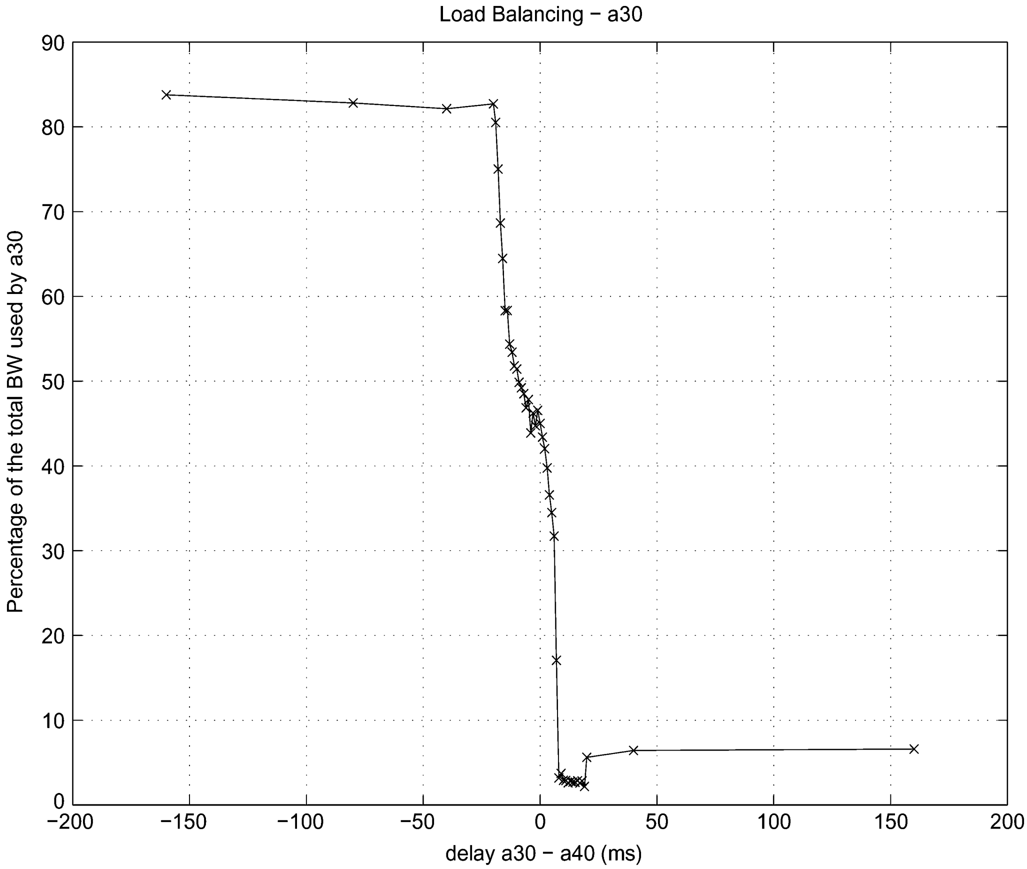

Fig. 7. Percentage of traffic flowing through port A30 as a function of the difference in delay between the two ports.

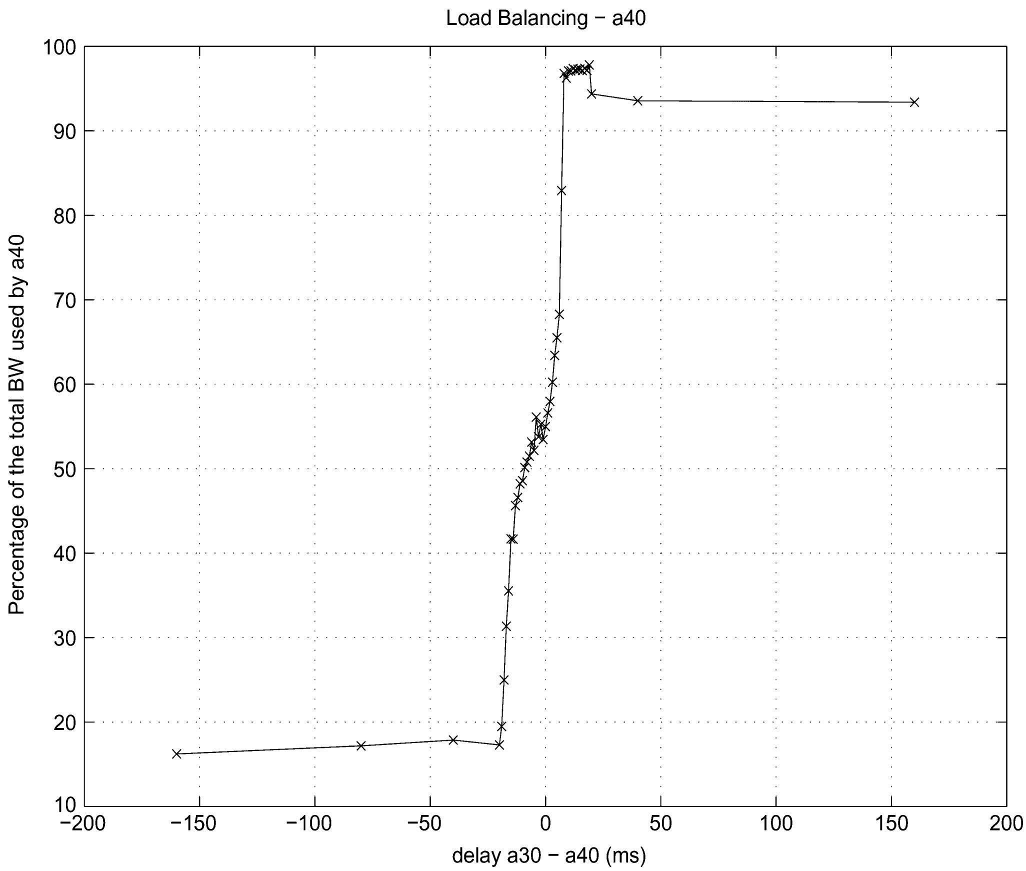

Fig. 8. Percentage of traffic flowing through port A40 as a function of the difference in delay between the two ports.

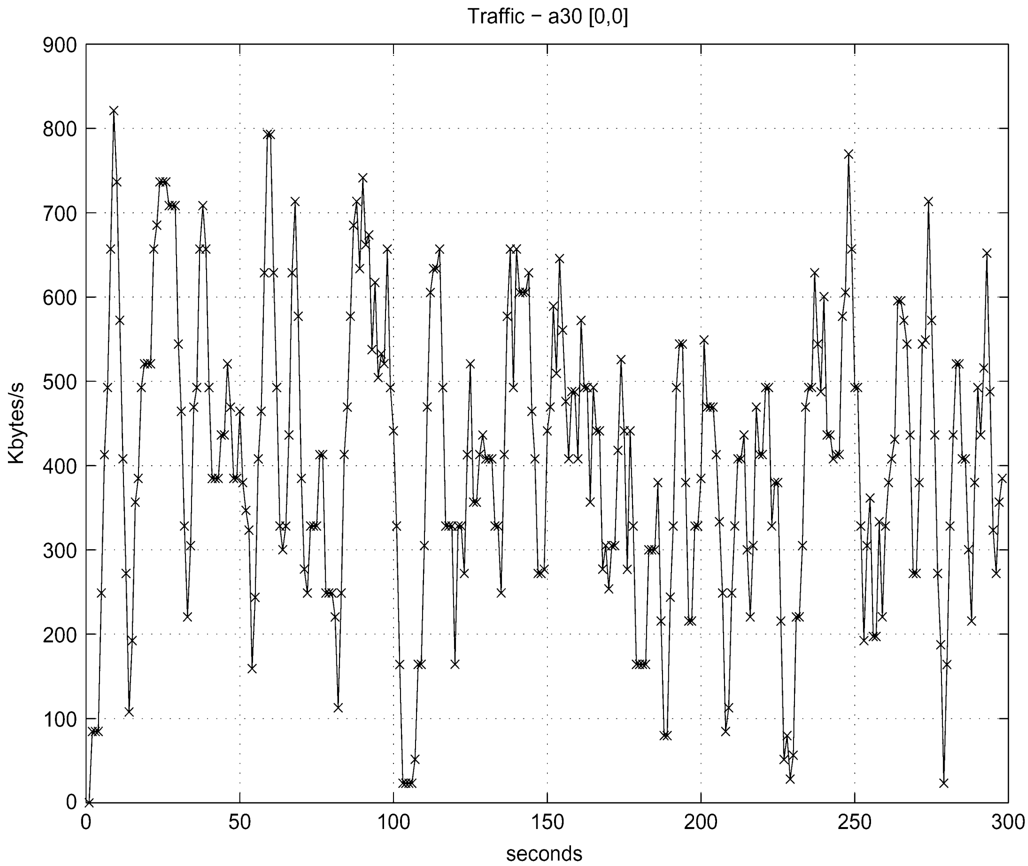

Fig. 9. Instantaneous traffic flow through port A30 when delays are identical.

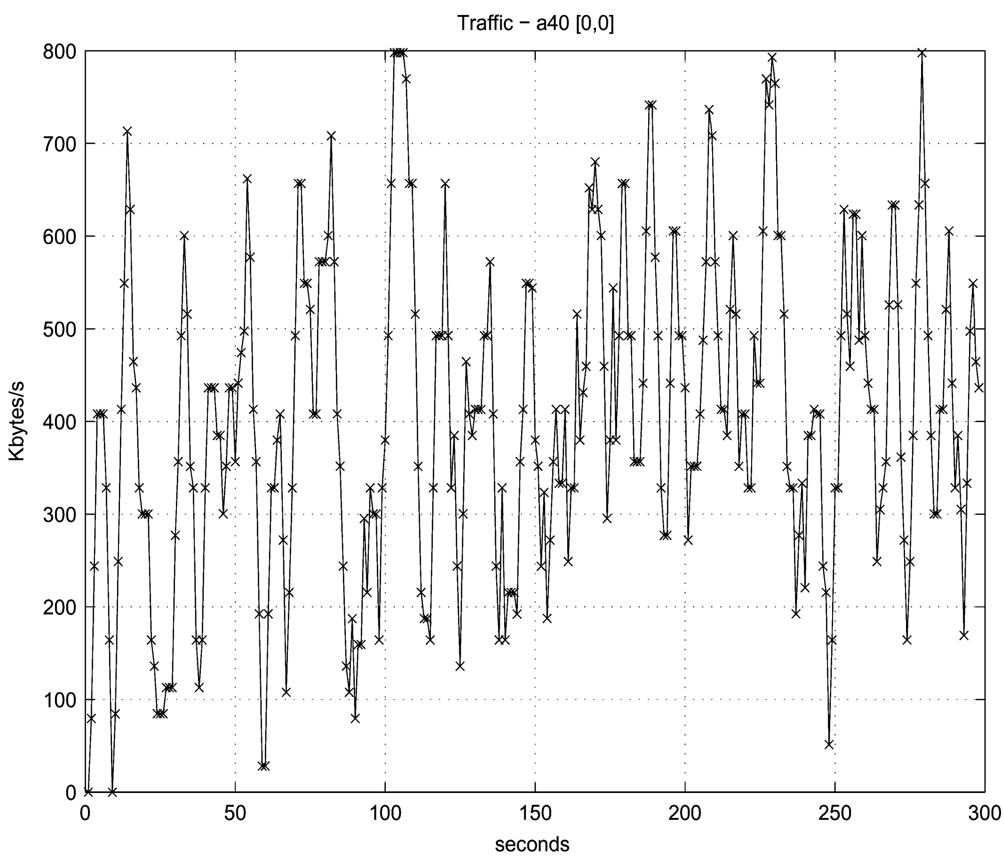

Fig. 10. Instantaneous traffic flow through port A40 when delays are identical.

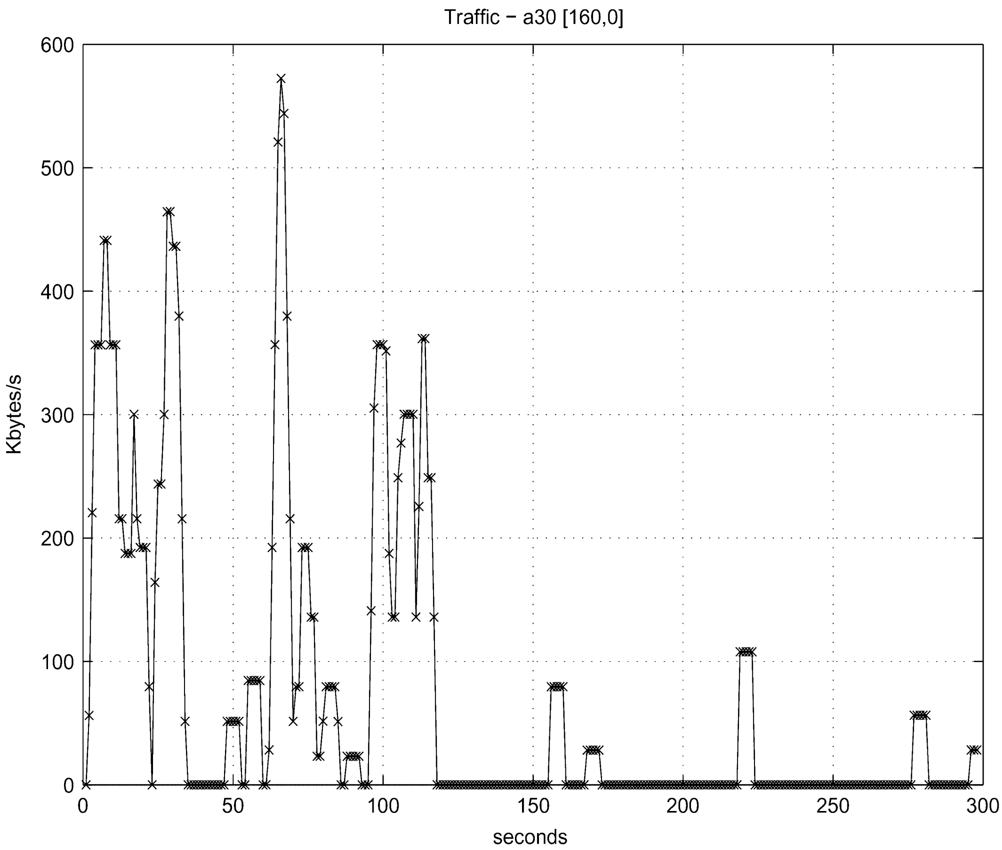

Fig. 11. Instantaneous traffic flow at port A30 when DA30−DA40 = 160 ms.

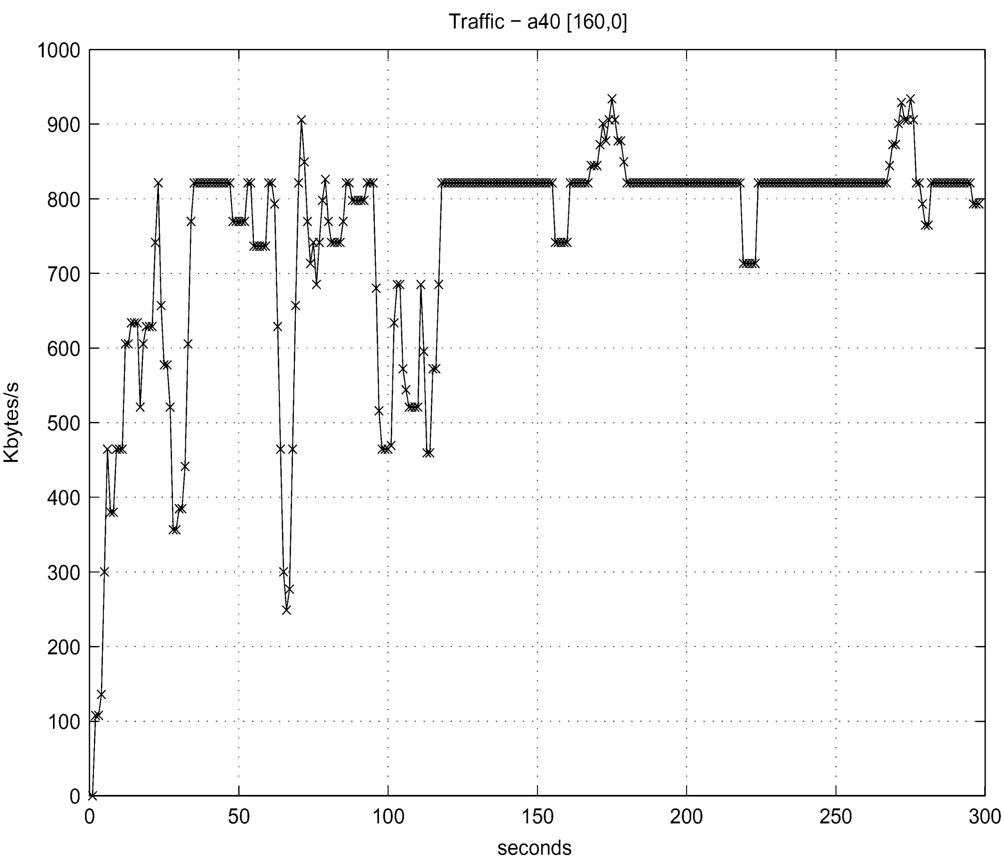

Fig. 12. Instantaneous traffic flow at port A40 when DA30−DA40 = 160 ms.

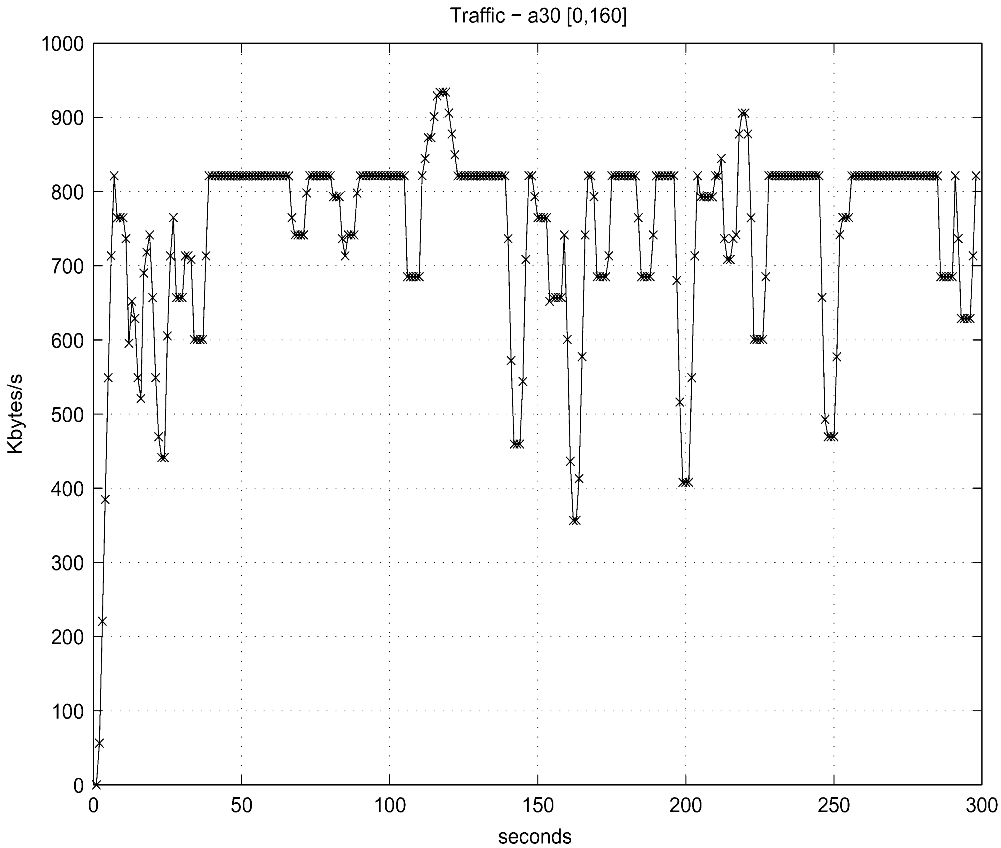

Fig. 13. Instantaneous traffic flow at port A30 when $\mathrm{DA}30-\mathrm{DA}40 = -160$ ms.

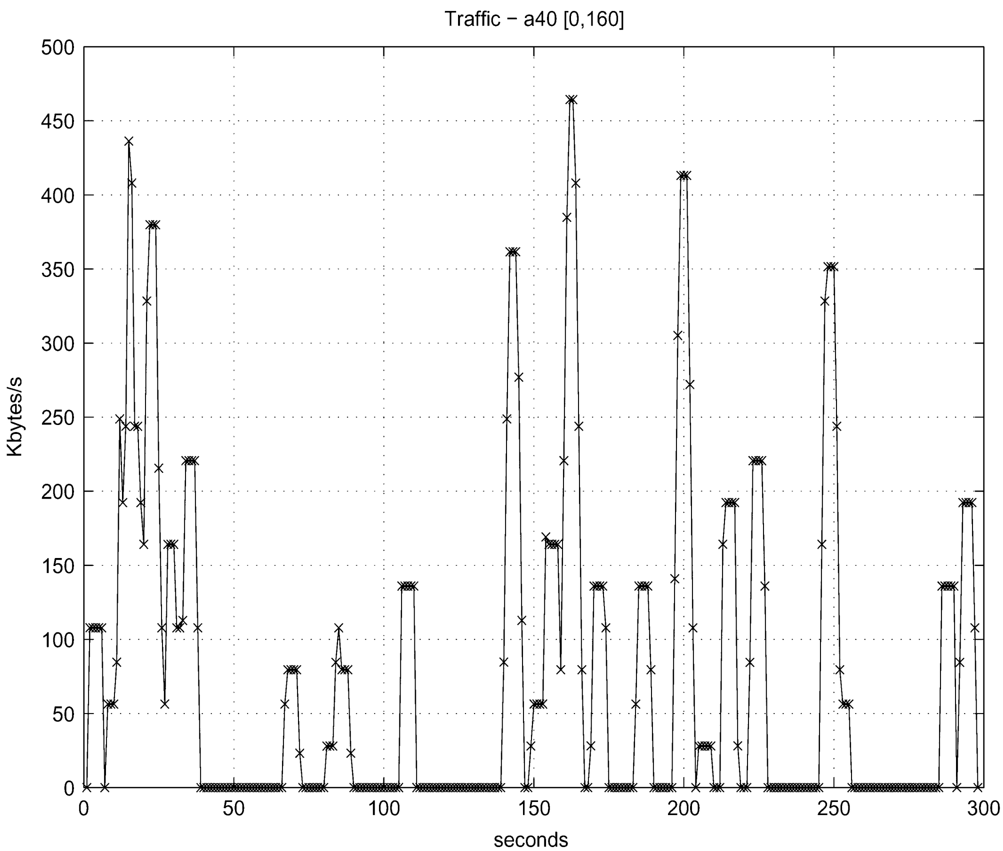

Fig. 14. Instantaneous traffic flow at port A40 when $\mathrm{DA}30-\mathrm{DA}40 = -160$ ms.

References

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

[2] R. E. Kahn and R. Wilensky. A framework for digital object services. [Online]. Available: http://www.cnri.reston.va.us/cstr/arch/k-w.html

[3] U. Halici, "Reinforcement learning with internal expectation for the random neural network," *Eur. J. Oper. Res.*, vol. 126, no. 2, pp. 288–307, 2000.

[4] E. Gelenbe, E. Şeref, and Z. Xu, "Simulation with learning agents," *Proc. IEEE*, vol. 89, pp. 148–157, Feb. 2001.

[5] E. Gelenbe, R. Lent, and Z. Xu, "Toward networks with cognitive packets," in *Proc. Int. Conf. Performance and QoS of Next Generation Networking*, K. Goto, T. Hasegawa, H. Takagi, and Y. Takahashi, Eds., 2001, pp. 3–17.

[6] ——, "Design and performance of cognitive packet networks," *Perform. Eval.*, vol. 46, pp. 155–176, 2001.

[7] ——, "Measurement and performance of cognitive packet networks," *J. Comput. Netw.*, vol. 37, pp. 691–701, 2001.

[8] ——, "Networking with cognitive packets," presented at the Int. Conf. Artificial Neural Networks, Madrid, Spain, 2002.

[9] E. Gelenbe and R. Lent, "Mobile ad-hoc cognitive packet networks," in *Workshop Applications and Services in Wireless Networks*, Paris, France, 2002.

[10] E. Gelenbe *et al.*, "Cognitive packet networks: QoS and performance," presented at the IEEE Conf. Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), San Antonio, TX, 2002. Keynote Paper.

[11] E. Gelenbe, P. Liu, and J. Lain, "Genetic algorithms for route discovery," presented at the Int. Symp. Performance Evaluation of Computer and Telecommunication Systems, Montreal, QC, Canada, 2003.

[12] E. Gelenbe and K. Hussain, "Learning in the multiple class random neural network," *IEEE Trans. Neural Networks*, vol. 13, pp. 1257–1267, Nov. 2002.

Erol Gelenbe (Fellow, IEEE) received the B.S.E.E. degree (with high honors) from Middle East Technical University, Ankara, Turkey, in 1966, the M.S. and Ph.D. degrees from Polytechnic University, New York, in 1969, and the D.Sc. degree from the University of Paris, Paris, France, in 1973.

He was the Nello L. Teer Professor of Electrical and Computer Engineering at Duke University. From 1998 to 2003, he was the University Chair Professor and Director of the School of Electrical Engineering and Computer Science at the University of Central Florida. He currently holds the Dennis Gabor Chair at Imperial College, London, U.K. His research, currently funded by the Engineering and Physical Sciences Research Council (EPSRC) and the Ministry of Defence in the U.K., covers three areas which contribute to the work described in this paper: design of packet networks, performance evaluation methods for computer systems and networks, and the study of computational models inspired by biological behavior. He has published four books which have appeared in French, English, Japanese, and Korean. He has published over 100 papers in a variety of professional journals. In 2000, 2001, 2002, and 2003, he was the Keynote Speaker at several U.S., European, and Asian conferences. He has graduated over 60 Ph.D. students. He is currently on the Editorial Boards of *Acta Informatica*, the *International Journal of Computer Communications*, *Telecommunications Systems*, *Recherche Opérationnelle*, *Computational Management Science*, and *Performance Evaluation*.

He is a Fellow of the Association for Computing Machinery and the Institution of Electrical Engineers. In 1991, he was awarded the Honor of *Chevalier, Ordre National du Mérite*, by the President of France and promoted to *Officier* in 2001 for his research contributions. In 2003, he was awarded the Honor of *Chevalier, Palmes Académiques*. He has received the distinction of *Dott. Ing. Honoris Causa* from the University of Rome–Tor Vergata, and scientific awards from IFIP, the French Academy of Sciences, and the Parlar Foundation of Turkey. A founder of International Federation for Information Processing (IFIP) WG7.3 (Computer System Modeling and Performance Evaluation), he was twice its Vice-Chair and Chair, and is a member of IFIP WG6.4 (Computer Network Performance). He has served on many editorial boards including the PROCEEDINGS OF THE IEEE and the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

Ricardo Lent received the B.S. and Diploma degrees in electrical engineering from Universidad Ricardo Palma, Lima, Peru, in 1993 and 1994, respectively, the M.S. degree in electrical engineering from Universidad Nacional de Ingeniería, Lima, in 1998, and the M.S. and Ph.D. degrees in computer science from the University of Central Florida, Orlando, in 2001 and 2003, respectively.

From 1993 to 1999, he was a Member of the Technical Staff with Red Cientifica Peruana. From 1998 to 1999, he was also a Lecturer at Universidad Nacional de Ingeniería. He is currently a Visiting Assistant Professor of Computer Science at the University of Central Florida.

Arturo Núñez received the engineering degree from Universidad Ricardo Palma, Lima, Peru, in 1994 and the M.S. degree in computer science from the University of Central Florida, Orlando, in 2003. He is currently working toward the Ph.D. degree in the Electrical and Electronic Engineering Department, Imperial College, London, U.K.

He was with the Peruvian Telecommunications Company Telefónica del Perú. His research interests include computer network design, routing protocols, performance evaluation of computer systems, and operating systems.