Genetic Algorithms for Route Discovery

Erol Gelenbe, *Fellow, IEEE*, Peixiang Liu, and Jeremy Lainé

***Abstract*—Packet routing in networks requires knowledge about available paths, which can be either acquired dynamically while the traffic is being forwarded, or statically (in advance) based on prior information of a network's topology. This paper describes an experimental investigation of path discovery using genetic algorithms (GAs). We start with the quality-of-service (QoS)-driven routing protocol called "cognitive packet network" (CPN), which uses smart packets (SPs) to dynamically select routes in a distributed autonomic manner based on a user's QoS requirements. We extend it by introducing a GA at the source routers, which modifies and filters the paths discovered by the CPN. The GA can combine the paths that were previously discovered to create new untested but valid source-to-destination paths, which are then selected on the basis of their "fitness." We present an implementation of this approach, where the GA runs in background mode so as not to overload the ingress routers. Measurements conducted on a network test bed indicate that when the background-traffic load of the network is light to medium, the GA can result in improved QoS. When the background-traffic load is high, it appears that the use of the GA may be detrimental to the QoS experienced by users as compared to CPN routing because the GA uses less timely state information in its decision making.**

***Index Terms*—Cognitive packet networks (CPNs), genetic algorithm (GA), quality of service (QoS), routing.**

I. Introduction

ROUTES in networks are selected so that some criterion of performance can be satisfied, in addition to the most elementary need of conveying data from a specified source to a specified destination. For instance, Internet uses prior agreements between "autonomous systems" (AS) to carry each other's traffic based on economic considerations, while routing within each AS is generally based on the selection of a "shortest path" or the "smallest number of hops" from a source to a destination [1]. Asynchronous transfer mode (ATM) [2], [3] and multiprotocol label switching (MPLS) [1] networks establish a fixed path for a given "call" so as to satisfy the quality-of-service (QoS) requirements, such as delay or cell loss, within the overall traffic-engineering constraints imposed by the collection of calls that a network may be handling.

In earlier work [4], [5], we reported an autonomic distributed routing protocol called the "cognitive packet network" (CPN) [6], which dynamically selects paths through a store-and-forward packet network so as to route peer-to-peer user traffic based on QoS. CPN's ability to satisfy different user QoS metrics based on random neural networks (RNNs) and reinforcement learning (RL), distributed at each network node, was discussed in [4]. Its extension to a wireless *ad hoc* environment is covered in [7].

In this paper, we exploit an analogy between "genotypes" and network paths, where each path is viewed as the encoding of a genotype so that a genetic algorithm (GA) [8] can be used at the source node of each connection to discover new paths that may not have been discovered by the smart packets (SPs). The GA-daemon software runs in a background mode at the source node of a connection and uses the "crossover" operation to splice existing paths and create valid new paths. Through successive crossover and selection operations, the GA daemon enables the source node both to compose new routes from the existing ones and to select which routes to use based on their "fitness," which is estimated based on QoS. We discuss this approach in detail and present an implementation where the GA runs in background mode so as not to overload the ingress router. We also provide measurements on a network test bed. These measurements indicate that when the background-traffic load is light to medium, the GA can result in improved QoS. However, when the background-traffic load is heavy, the use of the GA may actually be detrimental to QoS as compared to the original CPN routing protocol.

This paper is organized as follows. Section II briefly describes the CPN routing algorithm. Section III presents a GA approach to packet routing, while Section IV describes its implementation on top of CPN. In Section V, we present measurements on CPN without the GA and, then, a set of experiments to evaluate the conditions under which the GA will, or will not provide useful QoS improvements. Section VI concludes the paper.

II. CPN

CPNs use SPs and acknowledgment packets (ACKs) to select routes from the user's source node to its destination based on the user's QoS requirement. At each CPN node, RNNs [9] with RL [10], [11] are used to make routing decisions. Each output link from a CPN node is represented by a neuron in the RNN. The arrival of an SP triggers the execution of the RNN algorithm, and the output link corresponding to the most excited neuron is chosen as the routing decision. The synaptic weights of each RNN are updated with the measured QoS values that are brought back by ACK packets and stored in the nodes' mailboxes, as described in detail in [4]. The path-discovery process continues throughout each user's session, and the most recently discovered best path is the one that is used by dumb packets (DPs) carrying the user's payload to the destination. The route for the DP is stored in the packet itself and is provided

Manuscript received June 18, 2005; revised January 13, 2006. This paper was recommended by Guest Editor U. Kaymak.

E. Gelenbe and P. Liu are with the Department of Electrical and Electronic Engineering, Intelligent Systems and Networks (ISN), Imperial College, SW7 2BT London, U.K. (e-mail: e.gelenbe@imperial.ac.uk).

J. Lainé is with the Ecole Polytechnique, 91000 Palaisean, France.

Digital Object Identifier 10.1109/TSMCB.2006.873213

by the source node using the routing information brought back by ACKs.

SPs behave as scouts that are sent out to look for routes to a destination, and the rate at which they are sent is a fraction of the rate at which DPs are sent so that they may sample events at a rate that is proportional to the frequency at which payload traffic travels through the network. As the SP moves from node to node, it collects measurements at the nodes (e.g., the time at which the visit took place). If the SP does not reach its destination node after a predetermined number of hops, which is a multiple of the network's diameter, the SP is destroyed. However, if the SP reaches its destination before that, an ACK packet is sent back to the source along the reverse of the path used by the SP; the reverse path is chosen, however, so that any loops in the SP's path are removed. ACK carries the measurements collected by the SP, and this information is used to update the synaptic weights of the RNNs at each of the nodes using the RL algorithm, which increases the excitatory value of weights, which have led to successful outcomes, and weakens the others based on a QoS "goal." The "goal" is the metric that characterizes the success of the outcome, such as packet delay, loss, jitter, and so on. Thus, the RL algorithm uses the observed outcome of a decision to "reward" or "punish" the corresponding decision of the routing algorithm so that its future decisions will more likely meet the desired QoS goal. The ACKs also bring route information, which has been discovered by SPs back to the source nodes, to be used by subsequent DPs.

III. GA Approach to Packet Routing

GA is a search strategy that is inspired by biological evolution [8] and which is based on the following features.

1) Population of individuals (or genotypes) where each individual represents a potential solution to the problem being considered. In this paper, an individual is a path from source S to destination D represented by the labels of each node on the path.
2) Fitness function that evaluates the utility of each individual's possible contribution to solving the problem. In our case, the fitness of a given path is the observed or estimated QoS of the path.
3) Selection function that selects individuals for reproduction based on their fitness. This means that we would choose paths with better QoS.
4) Genetic operators that are used to alter or combine individuals to create new individuals, namely crossover and mutation. If $SxNyD$ and $SuNvD$ are two valid paths from source S to destination D, which both use the intermediate node N, and x, y, u, and v are sequences of nodes on the two paths, then $SuNyD$ and $SxNvD$ are two paths that result from the crossover operation applied to the two previous paths.

In the approach that we propose, the GA runs as a background process at each source node to generate and select paths for DPs based on the QoS goal, and the paths are stored in a stack at the source node. The fitness of a path is determined from the QoS measurement data returned by an ACK, which is received in response to sending an SP or DP along that path. The choice of running the GA in background mode at the node is dictated at this point by the fact that it can be a slow process, and that it may be manipulating large data sets. Clearly, if we were able, in practice, to accelerate the GA's operation via, say, a hardware implementation at a router, then this choice, and also the resulting performance, would be different.

A. Paths and QoS

A path w is a variable-length sequence of nodes, which begins with the source node S and ends with the destination D. For any nodes a and b, ab is a subsequence in w only if there is a link (i.e., path of length one) going from a to b. Thus, w represents any viable path from S to D.

Each path w has a goal value $G(w)$ which is the observed QoS for that path. G describes how effective the path w is. Thus, a smaller value of $G(w)$ means that w is more desirable. A simple example is when $G(w)$ is the number of links on the path from S to D, i.e., the number of hops. Thus, a conventional shortest path routing [1] in the Internet protocol (IP) tries to minimize the function G if the goal function is the number of hops. However, $G(w)$ may also be the delay or loss of packets, which is observed over the path w, or the variance in packet delay, or the power consumed for processing and forwarding the packet in the hops of a wireless network [7], or the overall security level of the path, or combinations of many of these metrics.

We shall say that G is additive if for any path $w = \alpha\beta$, which is the concatenation of two paths α and β, we have $G(w) = G(\alpha) + G(\beta)$. Examples of additive goal functions include packet delay, path length, loss, variance of packet delay, and power consumption. Since the CPN algorithm [4], [6] measures the forward delay of SPs and DPs using the information brought back by ACKs, it, in fact, not only collects at the source, the source-to-destination delay, but the ACK also brings back the delay from any intermediate node on the path to the destination.

In our GA implementation, which uses the preexisting CPN system, new paths are generated in two different ways.

1) SPs discover routes, and ACK packets bring back valid routes to the source. These paths are stored into the stack so that CPN already provides a way of generating new paths w using SPs, which search their way through the network, using RNNs and RL, toward the destination. Conceptually, we may think that this is some kind of "mutation" operation that CPN already offers to the GA. The paths in the stack are organized in a list sorted in order of increasing $G(w)$ value. Therefore, the first path in the list is the fittest path.
2) At the source, we also generate additional paths using the path crossover operation that we described earlier using paths between the same source and destination nodes, which share the same intermediate node, and which are already in the stack. If some new path $SuNcD$, generated by a crossover from $SuNvD$ and $SaNcD$, was not yet in the stack, then it is placed there with the new goal value $G(SuNcD) = G(Su) + G(NcD)$, assuming that the goal is additive.

In summary, the GA operates as follows. Every path discovered by SPs and brought back by ACKs naturally becomes

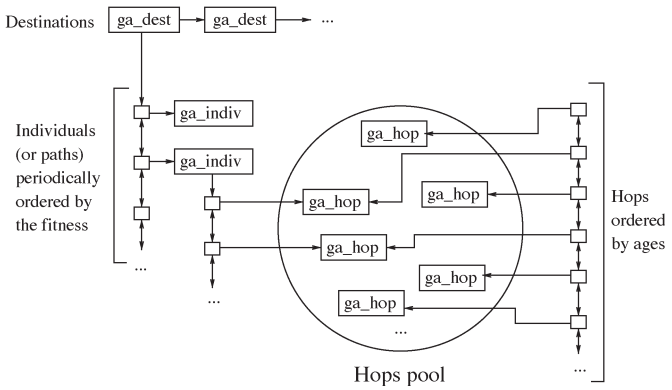


Fig. 1. Principal data structures of the GA daemon.

an individual in the GA population. The paths with the same source S and destination D form a GA route repository that we call the stack, which is limited to some maximum size. The paths in the same repository are combined to form new paths, provided they share the same intermediate node. Their QoS metric, or fitness, is synthesized from those of their constituents. Whenever a DP needs to be forwarded to the destination, the path at the top of the stack (i.e., the one with the smallest goal value) will be used as the source route.

IV. Implementation

Since the stack provides a ranking of paths for some source to destination pair, the ranking data need to be roughly consistent with respect to the time when the measurements were taken. Thus, we have chosen to store a pool of hops together with the relevant measurement data (e.g., hop, delay, possibly loss, etc.) and the time stamp indicating when this measurement was brought back to the source. The actual data structure representing the paths is then a set of pointers pointing to the collections of hops, as shown in Fig. 1. As newer measurements are sent over by the CPN module, hop data in this data structure is updated, and their time stamp is updated as well. The time stamp allows us to periodically get rid of obsolete hops from the hop pool so that the GA engine does not make decisions based on obsolete measurements.

As mentioned earlier, the size of the data structures that we have described makes a kernel-level implementation of the GA daemon rather impractical. Thus, we have implemented the GA algorithm as a system daemon that runs in the background. In the sequel, we will present measurement results for CPN with and without the GA daemon. Note again that CPN is needed by the GA daemon so that it can receive new paths and obtain measurements that allow it to update the hops stored in the stack.

The GA-enabled system also differs from the CPN module in the way it handles ACKs. While CPN deals with the entire paths, the GA operates at a finer (hop) level; this means that caution needs to be taken when stripping a route of loops. Loops are no longer removed at the destination but, instead, as the ACK travels back, each node on the return path checks if it was present twice or more in the SP's forward path and, if that is the case, it removes the loop in the ACK and adjusts

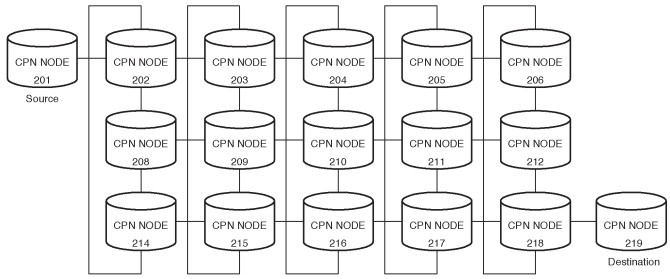


Fig. 2. Test-bed topology used in the experiments.

the dates associated with all nodes that were visited between the source and itself to eliminate the extra delay introduced by the loop. For example, if the ACK contains the return path $\langle a, b, c, d, e, c, l, m\rangle$, so that m is the source, the loop $\langle c, d, e, c\rangle$ will be removed at node c when it receives the ACK. The dates of nodes a and b are then reduced by the amount of the total forward delay in the loop time.

The GA daemon can be fired up at any time when the CPN module is loaded with no initial knowledge of the network. It consists of a main loop, which is in charge of polling the kernel for new paths, checking its internal data structures for size and consistency, selecting individuals for crossover and doing the actual crossover, and periodically updating the CPN module's DP route repository. The data-exchange operations between the CPN kernel module and the GA daemon are bidirectional as the module passes measurements to the daemon and, in return, the daemon periodically updates the module's route repository with the paths that have the best fitness at that time. The GA daemon is always the initiator of the data exchange so that the kernel module does not need to keep track of the presence or absence of the daemon to route CPN traffic. This way, we also eliminate lockups that would occur if the kernel were to probe the daemon while it is sleeping.

In our first implementation, the GA daemon would always select the best (i.e., fittest) path for a given destination. However, this led to two significant problems: 1) the same routes were being used too often leading to path overload, and 2) there was an insufficient generation of new data on different paths so that hop data related to unused paths became obsolete and had to be eliminated. Thus, we chose to modify the GA daemon so that it provides, in round-robin mode, the identity of each of the paths that are within 5% of the best QoS. The round-robin policy on the best routes then acts as a simple load-balancing mechanism to avoid saturation of any given path, and also offers the possibility to gather measurement data concerning a much bigger set of hops. Another way of increasing the measurement data available to all connections from a given source is to share the hop pool among several different connections emanating from the same source.

V. Performance Measurements

Measurements were conducted on the test bed shown in Fig. 2. Each node is a PC equipped with four Ethernet point-to-point ports, where the number of ports is related to the cost of the cards that we were able to use. Each port uses a 10-Mb/s Ethernet link connected by point-to-point copper cable to

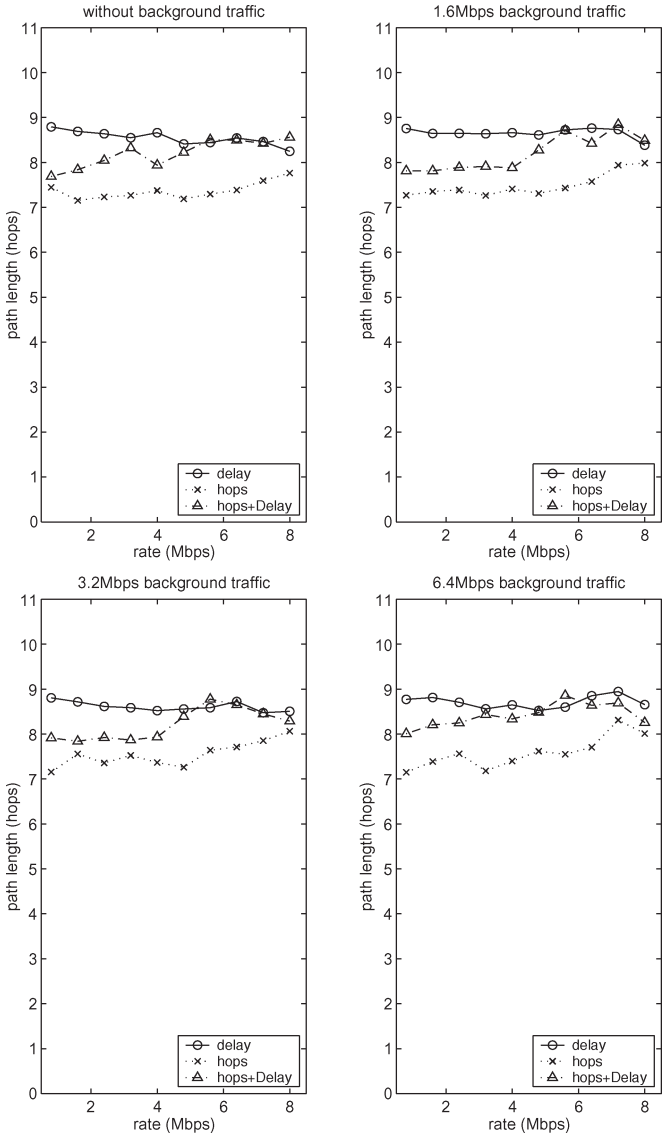


Fig. 3. Path-length comparison.

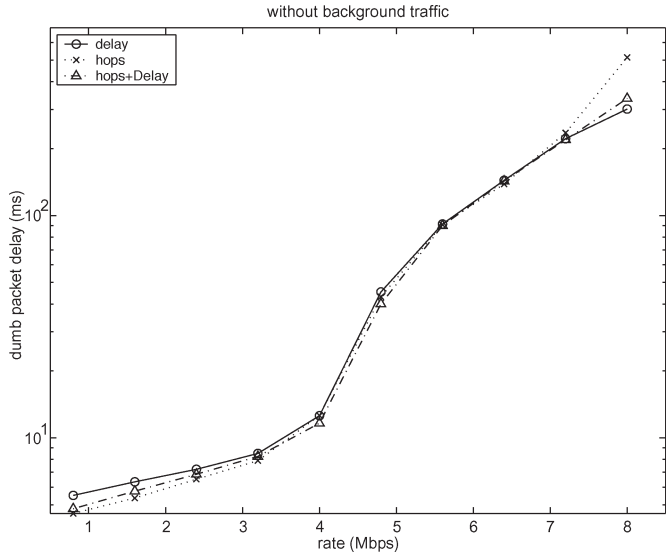


Fig. 4. Delay without background traffic.

another node. The CPN software is integrated into the Linux kernel 2.4.x. with minimal changes in the existing networking code, and is independent of the physical transport layer. The network interface is compatible with the popular BSD4.3 socket layer in Linux, and provides a single application program interface (API) for the programmer to access the CPN protocol. To maximize the length of the paths, and also have a large number of possible paths from the source to the destination, we chose the source node to be the one at the left edge of the test bed with the destination being the one at the right edge.

We will first report on some measurement results without having the GA in operation and then discuss the results with the GA.

A. Pure CPN: Experiments Without the GA Daemon

First, we ran several experiments for CPN, without the GA daemon running, and use different QoS goals for the user's connection, as indicated below. The user's connection is constituted by a flow of 1024-byte UDP packets at constant bitrate (CBR). Each measurement point is based on 10 000 packets that were sent from the source to the destination, and we inserted random background traffic into each link in the network with the possibility of fixing its rate at different levels including "no background traffic," 3.2-MB background traffic, 6.4-MB background traffic, and higher levels as well.

The CPN routing algorithm was used throughout the experiments using three different QoS goals: 1) delay [Algorithm-D]; 2) hop count [Algorithm-H]; and 3) the combination of hop count and forward delay [Algorithm-HD]. The measurements concern the average hop count and the forward delay under different background-traffic conditions.

Fig. 2 shows that the shortest path length from the source node (#201) to the destination node (#219) is seven, and that there are only five distinct shortest paths. For example, one of them is route $\langle 201 \rightarrow 202 \rightarrow 214 \rightarrow 215 \rightarrow 216 \rightarrow 217 \rightarrow 218 \rightarrow 219 \rangle$.

In Fig. 3, we report the measured average number of hops traversed from the source to the destination when different algorithms are used. When hop count is used as the QoS goal, we see that the average number of hops under different background-traffic conditions is close to the minimum of seven. In other words, the adaptive and distributed approximation to minimum hop routing, which is being offered by CPN, is actually working well. The forward delay is approximated as one half of the round-trip delay. From Fig. 4, we are surprised to observe that if the connection's traffic rate is less than 3.2 Mb/s, then Algorithm-H achieves the smallest delay, while Algorithm-D is the worst. However, when the connection's traffic rate is between 3.2 and 5.6 Mb/s, the performance of Algorithm-HD is better but Algorithm-H and Algorithm-D are almost the same. All algorithms are equivalent with respect to measured delay when the connection's traffic rate is between 5.6 and 7 Mb/s. When the connection's traffic rate is extremely high (> 7 Mb/s), Algorithm-D gives the smallest delay, and Algorithm-H is the worst. When we increase the background traffic per link to 3.2 Mb/s (Fig. 5), if the connection's traffic rate is less than

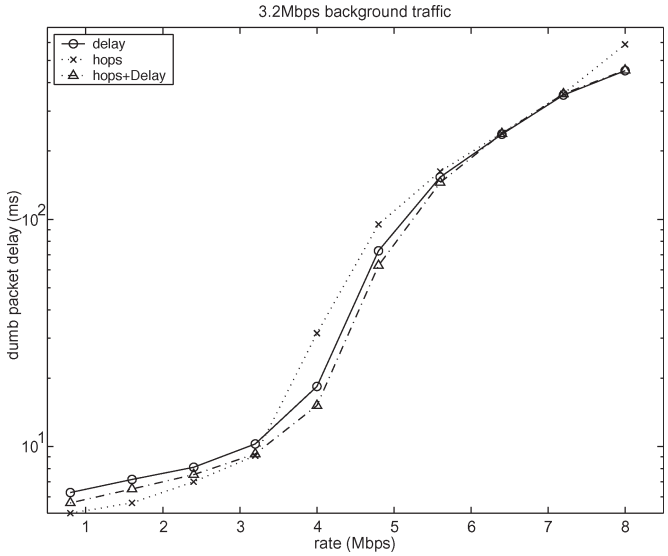


Fig. 5. Delay with 3.2-Mb/s background traffic.

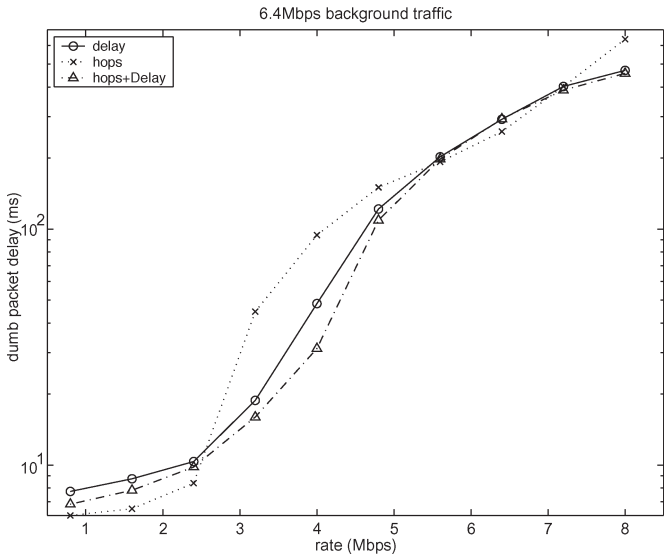


Fig. 6. Delay with 6.4-Mb/s background traffic.

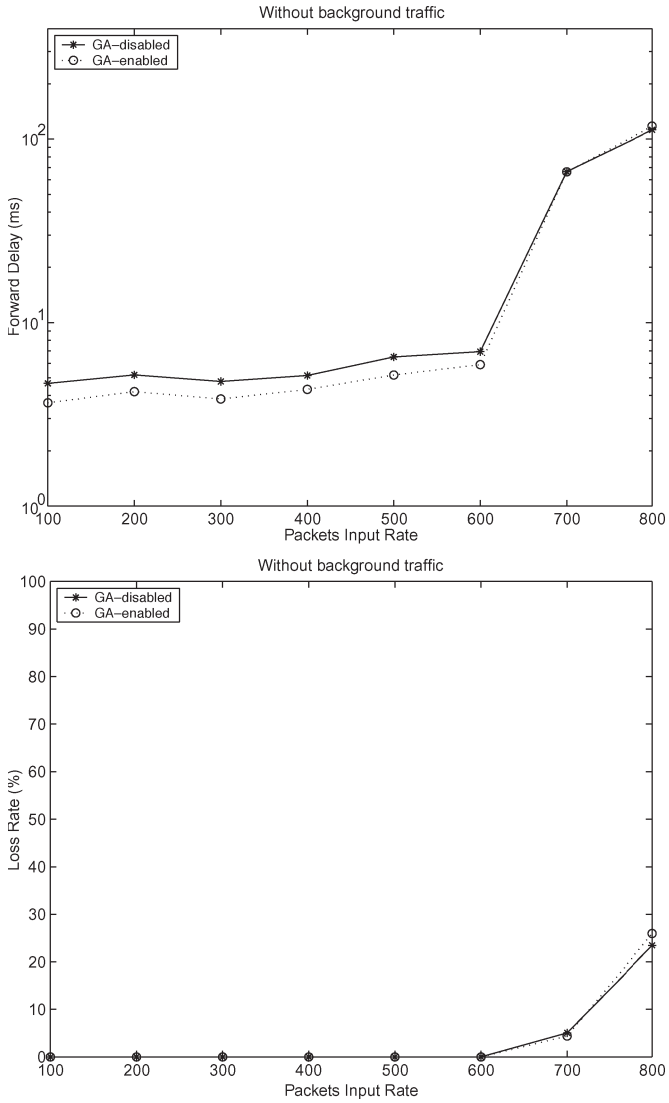


Fig. 7. CPN with and without the GA. Measured delay and loss without background traffic.

3.2 Mb/s, Algorithm-H is still the best, and, again, Algorithm-D is the worst. But if the connection's traffic rate is between 3.2 and 6.4 Mb/s, Algorithm-HD is the best and Algorithm-H is the worst. When the connection's traffic rate is higher than 6.4 Mb/s, Algorithm-D performs as well as Algorithm-HD, and both of them are better than Algorithm-H. Results are similar with the 6.4-Mb/s background traffic per link except that the threshold points change (Fig. 6). Thus, if the network topology is stable and the network is lightly loaded, the shortest path is the best choice as far as the forward delay is concerned. When the network is heavily loaded or saturated, sticking to the shortest path is a bad choice, and longer paths offer lower delay.

B. Measurements When the GA Daemon Was Running

When the network is initialized, all the nodes start with an empty route repository and empty GA pools. Then, the CPN path-finding algorithm is started at the source node, and SPs are sent out. Once the first ACK comes back from the destination carrying the first path that has been discovered, DPs from the source to the destination are sent. When the GA algorithm was not in operation, we simply disabled the GA daemon. On the other hand, if the GA daemon is enabled, when the first ACK arrives at the source node, the GA daemon is automatically started, and it will generate paths as described in the previous sections.

In a first set of experiments, the destination was not fixed, but chosen at random, so as to emulate an environment where the source is handling many different connections. As indicated previously, the GA daemon groups the paths into subpopulations based on the destination, and crossover is carried out for each given destination. However, the hop pool is common to the various subpopulations, which means that it shares measurements.

In all the experiments, we used delay only as the QoS goal for CPN and for the GA daemon, and we measured both the resulting loss and delay for the connections, but only for the payload traffic carried by the DPs, since the objective is to support the QoS needs of the users and to use SPs only for

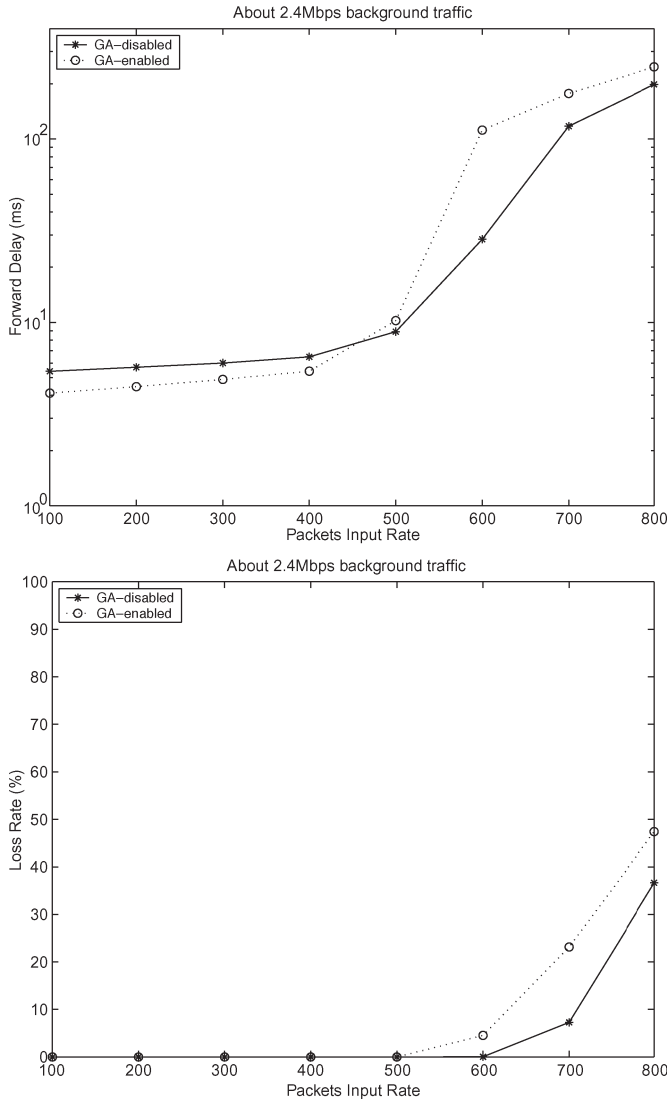


Fig. 8. CPN with and without the GA. Measured delay and loss for 2.4-Mb/s background traffic.

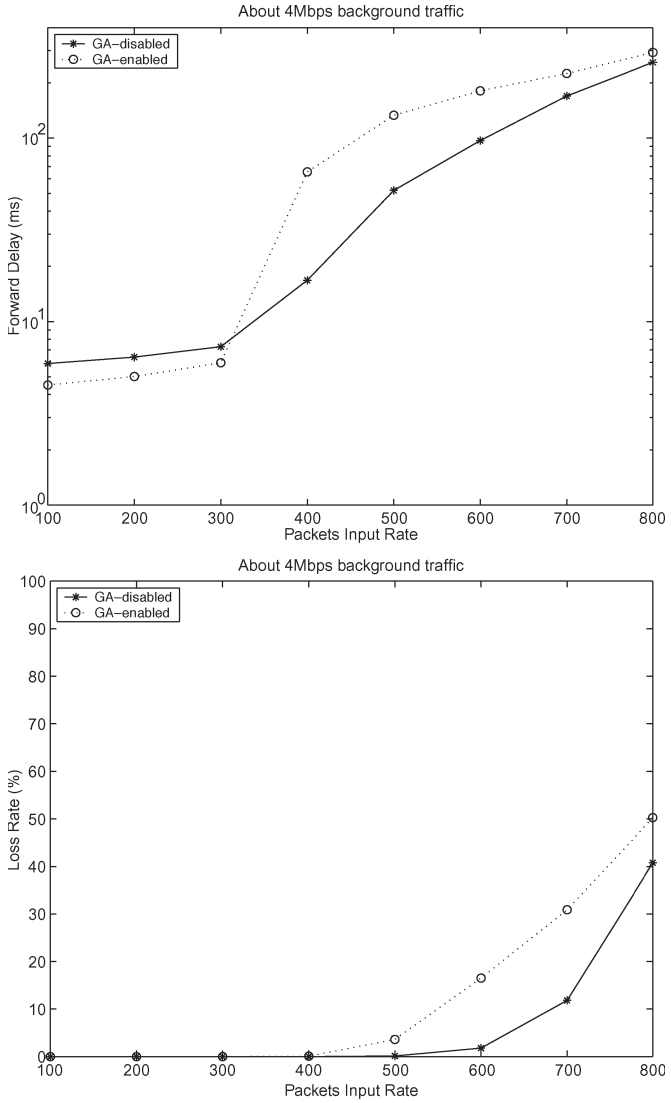


Fig. 9. CPN with and without the GA. Measured delay and loss for 4-Mb/s background traffic.

helping to find the best paths. Loss was measured simply by counting the number of ACKs r received for the DPs, and the number s of DPs that were transmitted so that the loss rate is $l = 1 - r/s$. Note that this value is a pessimistic estimate because some ACKs will also be lost so that we are in fact measuring the total loss of DPs plus ACKs.

The experiments were run with different rates of background traffic, composed of 512-byte packets traveling in both directions of each link. The size of all DPs was fixed to 1024 bytes. We varied the connections' DP rate between 100 and 800 packets/s. For each value, we ran ten experiments, and, in each experiment, the source node sent out 10 000 DPs to each destination. The average forward delay and loss rate were computed as an average for each DP transmission rate over all the ten experiments. The results without background traffic are shown in Fig. 7.

We observe that in the absence of the background traffic, CPN with the GA outperforms simple CPN. When the DP input rate is less than 600 packets/s, the delay with the GA is only 80% of that without the GA; loss rates are both small enough to be considered to be zero. When the DP input rate exceeds 600 packets/s, some packet loss is observed and the average delay increases significantly. The delay is still somewhat smaller with the GA, but the loss rates are indistinguishable. Experiments were also conducted with the 800-Kb/s background traffic on each link, and the results were almost identical.

We suggest that the positive effect of the GA at light background-traffic rates may be due to the fact that when its information is up to date, the GA focuses its selection on better overall performance. When background-traffic rates are high, state information changes very rapidly, and "fresher" measurement information is used by CPN, while the GA mixes this with "older" and, hence, less relevant data.

We increased the background-traffic level to 2.4 and 4 Mb/s on each link, respectively, and the resulting measurements are shown in Figs. 8 and 9. We observe that the QoS is still better with the GA than without it, provided that the input rate is low. When the DPs' input rate exceeds a certain threshold (e.g., 300 packets/s in the 4-Mb/s per link background-traffic scenario), the GA actually results in worse QoS. We may

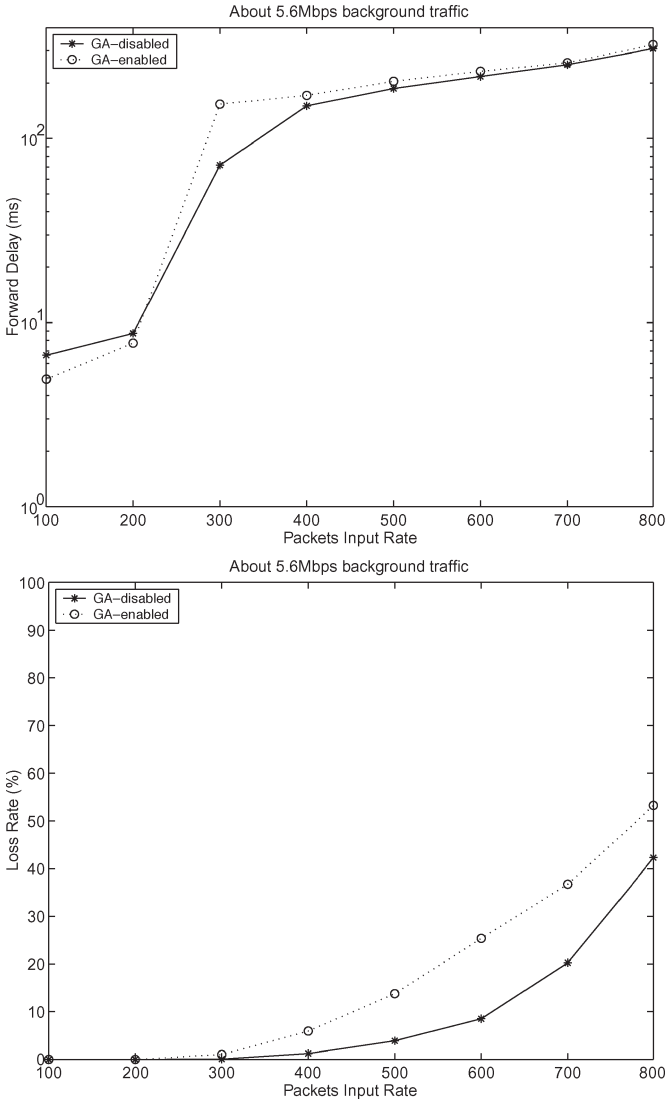


Fig. 10. CPN with and without the GA. Measured delay and loss for 5.6-Mb/s background traffic.

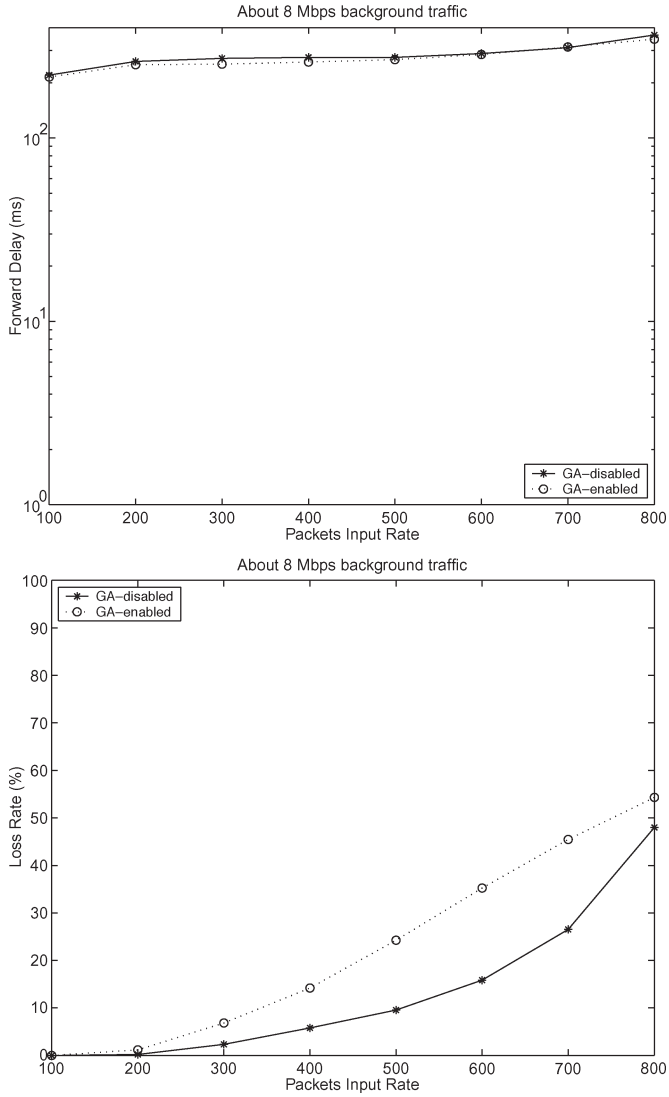


Fig. 11. CPN with and without the GA. Measured delay and loss for 8-Mb/s background traffic.

explain this as follows. At heavier network load, since the DP traffic has an additive effect on the existing link traffic, the DP traffic will significantly increase the observed delay. SPs will try other routes, and the corresponding ACKs will bring back information about the paths that are momentarily less loaded. These paths will be immediately used by CPN when the GA is not enabled, which leads to path switching and distribution of the traffic on multiple paths. On the other hand, when the GA daemon is enabled, less recent information is necessarily being used overall to select paths, and the resulting QoS will also be worse.

When we increase the background traffic further to 5.6 Mb/s on each link, the previous results are accentuated, as shown in Figs. 10 and 11. At 5.6-Mb/s background traffic per link, the threshold input rate at which the GA is underperforming, as compared to the system without the GA, is even smaller at 200 packets/s. However, at a very heavy background traffic of 8 Mb/s per link, we notice that since the network is saturated, whether we use or not use the GA makes a very little difference.

VI. Conclusion

This paper describes an innovative use of GA at the ingress level for dynamic adaptive routing in packet networks. We have presented the basic concept, as well as its implementation, in our existing CPN test bed. We have also reported measurements obtained under different background-traffic and connection-traffic levels.

We have implemented the GA daemon to run in background mode at the source nodes of connections so that through crossover and selection operations, the GA enables the source node both to compose new routes from the existing ones and to select routes based on their QoS. Furthermore, we have introduced a round-robin policy on the best routes to act as a simple load-balancing mechanism and as a way to collect measurement data for decision making from a wider variety of network hops. When the network is idle, the GA daemon uses a total of 376 kB of memory, of which 316 kB is shared memory used by standard libraries. As for processor usage, it is difficult to give an exact figure, although when we ran the

GA for an hour with a connection input rate of 800 packets/s, the cumulated processor time used by the GA daemon was in total below a second. Therefore, running a GA daemon in the background at the source node consumes very few system resources.

We first provided experimental data from measurements with "pure" CPN, i.e., when the GA daemon was turned off, and used different QoS goals to route packets for the user's connection. These measurements show on one hand that CPN actually works, in the sense that, using the hop count as the QoS goal actually does come close to minimizing the number of hops. However, CPN, actually, is also useful in under medium to heavy traffic conditions; trying to minimize the number of hops will not in itself minimize the packet delay and loss. Thus, the use of composite QoS goals, including delay, for instance, does provide an improvement to the QoS that the user will perceive.

We, then, present experimental results when the GA daemon is activated on top of CPN. The experiments now indicate that when the overall network load is light to medium, or when the user's packet input rate is low, the GA clearly improves the observed QoS over the levels observed when only CPN is used. On the other hand, when the background traffic becomes heavier, and also when the packet input rates are high for the connection itself, the user end-to-end delay measured for the DPs with the GA in operation is actually somewhat worse than when the GA is turned off. This suggests that overall improvement could be achieved by adaptively following, or not, the advice provided by the GA. This particular point will be examined in future work.

The main conclusion of this paper may be that smart techniques, such as the GA approach, which operate at the ingress nodes of a network, or at the peer points of a peer-to-peer network, of connections cannot beat techniques where smart decision making is taking place in a distributed manner throughout the network, such as "pure" CPN. Thus, one possible direction of future research would be to investigate the use of GA-type decision mechanisms in a staged manner, at intermediate nodes in the network, to move the decision making closer to the locations where the actual traffic-related events are taking place.

REFERENCES

[1] C. Huitema, *Routing in the Internet*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1999.

[2] V. Srinivasan, A. Ghanwani, and E. Gelenbe, "Block cell loss reduction in ATM systems," *Comput. Commun.*, vol. 19, no. 13, pp. 1077–1091, Nov. 1996.

[3] E. Gelenbe, X. Mang, and R. Önvural, "Diffusion based call admission control in ATM," *Perform. Eval.*, vol. 27–28, pp. 411–436, Oct. 1996.

[4] E. Gelenbe, M. Gellman, R. Lent, P. Liu, and P. Su, "Autonomous smart routing for network QoS," in *Proc. 1st Int. Conf. Auton. Comput.*, May 2004, pp. 232–239.

[5] E. Gelenbe, R. Lent, and A. Nunez, "Self-aware networks and QoS," *Proc. IEEE*, vol. 92, no. 9, pp. 1478–1489, Sep. 2004.

[6] E. Gelenbe, "Cognitive packet network," U.S. Patent 6 804 201, Oct. 12, 2004.

[7] E. Gelenbe and R. Lent, "Power-aware ad hoc cognitive packet networks," *Ad Hoc Netw.*, vol. 2, no. 3, pp. 205–216, Jul. 2004.

[8] J. H. Holland, *Adaptation in Natural and Artificial System*. Ann Arbor, MI: Univ. of Michigan Press, 1975.

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

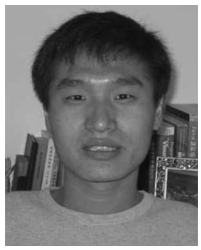
[10] R. S. Sutton, "Learning to predict the methods of temporal difference," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988.

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

Erol Gelenbe (F'86) received the Ph.D. degree from the Polytechnic Institute, New York, in 1970.

He holds the Dennis Gabor Chair at Imperial College, London. His research covers three areas that contribute to the work described in this paper: design of quality-of-service-based packet network protocols, performance-evaluation methods for computer systems and networks, and the study of computational models inspired by biological behavior. He has previously published four books, which have appeared in French, English, Japanese, and Korean. He has published over 115 papers in professional journals and is a frequent Keynote Speaker at conferences on the technical aspects of the Internet. He has graduated over 50 Ph.D. students. He is currently on the editorial board of the *Proceedings Royal Society, Series A*, *Acta Informatica*, *International Journal of Computer Communications*, *Telecommunication Systems*, *Recherche Opérationnelle*, *Computational Management Science*, and *Performance Evaluation*.

Dr. Gelenbe, as a Founder of International Federation of Information Processing Societies (IFIP) WG7.3 (Computer System Modeling and Performance Evaluation), was twice its Vice Chair and Chair, and is a member of IFIP WG6.4 (Computer Network Performance). His honors include the honorary doctorate from the Universities of Rome Tor Vergata (Italy), Bogazici (Istanbul), and Liege (Belgium). He has received several distinctions and scientific awards from IFIP, the French Academy of Sciences, and the Parlar Foundation of Turkey and is a Commander of Merit of Italy and an Officer of the Order of Merit of France. He has served as an editor for PROCEEDINGS OF THE IEEE and the IEEE TRANSACTIONS ON SOFTWARE ENGINEERING. He is a member of Academia Europaea and is also a Fellow of ACM and IEE, London.



Peixiang Liu received the B.E. degree in mechanical engineering from the China Institute of Metrology, Hangzhou, in 1996, the M.S. degree in computer science from Peking University, Beijing, China, in 2001, and the M.S. degree in computer science from the University of Central Florida, Orlando, in 2004. He is currently working toward the Ph.D. degree at Imperial College, London, U.K.

He is a Research Assistant of Prof. Gelenbe. His research interests include routing protocols, quality-of-service routing, network-performance evaluation, and genetic algorithms.

Jeremy Lainé received the degree in applied mathematics and electronics from Ecole Polytechnique, Paris, France, in 2002, and the M.S. degree in engineering physics from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2004.

His research interests include QoS routing, GAs, digital communications, and wireless networks. He is currently working on third-generation mobile terminals as a Consultant for a mobile telecommunications operator in Paris, France.