DOI:10.1145/1538788.1538809

Network software adapts to user needs and load variations and failures to provide reliable communications in largely unknown networks.

BY EROL GELENBE

Steps Toward Self-Aware Networks

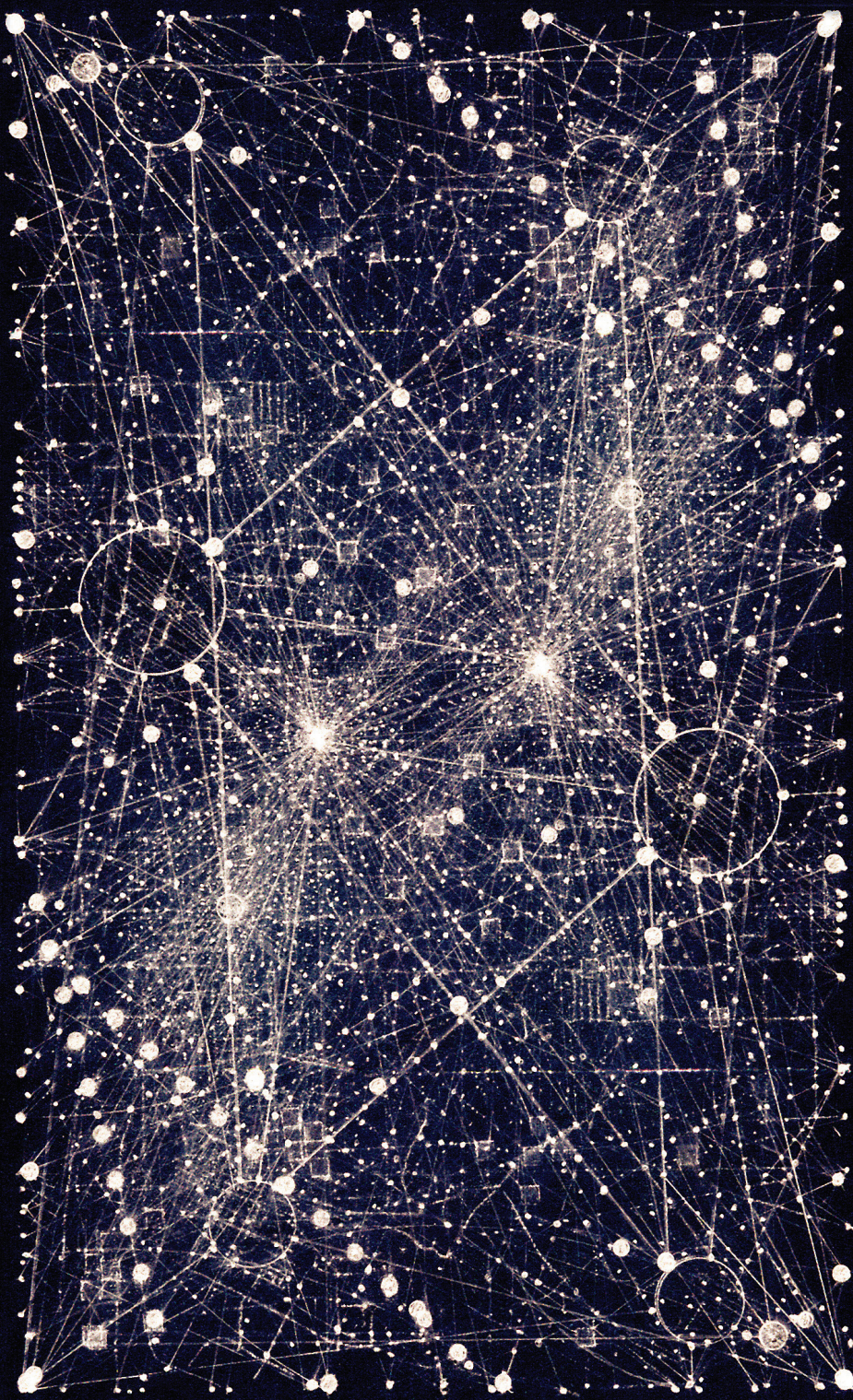
THE INFORMATION NEEDED to route packets in large networks and in networks in which nodes join and leave the network frequently or move in and out of wireless range of each other can change more frequently than the rate routing information is updated throughout the network. In such a system it becomes necessary to allow individual nodes to proactively discover the presence of other nodes, links, and paths (as needed and on demand), leading to the design of self-aware networks. Here, I focus on experimental and theoretical research concerning the technical steps leading to these networks.

The Internet Protocol offers an orderly update of its status based on the shortest-path algorithm,[18] Distance Vector,[22] and Link State[23] techniques so routing algorithms operate seamlessly, despite changes in network topology and conditions. However, as computer networks become extremely large, the information available concerning the network state becomes uncertain. Link state changes are more frequent in larger networks, increasing the overhead and delay due to updates throughout the network. Consequently, information about the network state, including connectivity, condition of nodes, traffic conditions, and quality of service (QoS), propagates more slowly than rate changes occur. The need to convey time-sensitive information (such as voice and media) also motivates investigation of routing techniques based on user requirements and the network's instantaneous state. Thus it is preferable that nodes discover the network state autonomously, without having to rely on an overall scheme that updates routing tables systematically throughout the network. Information updates can be initiated by the nodes that need this information at the time it is needed, rather than throughout the network and when changes occur.

We use the term "self-aware network," or SAN,[11] for a system consisting of nodes that can join and leave the network autonomously and discover paths when the need to communicate arises. The nodes in a SAN should sense the status of other nodes, links, and paths, including traffic level and congestion, so as to update their own relevant information about the paths they need to use, based on criteria specific to their own needs. Each connection may then use paths that optimize the connection's own QoS criteria, rather than a common criterion (such as the shortest path) for all connections. These needs might include user QoS requirements, or performance, reliability, security, defense against attacks,[9,24] and power utilization.[12] A SAN can be a wired, wireless, or a peer-to-peer system. A wireless ad hoc network is a practical example of a SAN that responds to time-varying conditions related to the mobility of nodes and changes in the conditions of wireless links (such as noise and physical obstructions). Networks that must operate autonomously and remotely (such as sensor networks) also benefit from self-aware capabilities.

Research on effective SAN architec-

DRAWING BY EMMA McNALLY



tures also motivates work on autonomic communications[5] and bio-inspired techniques for networking. Ideally, self-awareness is a desirable property of most networked systems. However, for SANs to be widely accepted, many fundamental questions must be answered affirmatively, including:

a. Assuming that in the worst case a node knows only its immediate neighbors (though the network is fully connected), can a node forward a packet successfully to any other node in the SAN in finite time without routing tables at each node?

b. What are practical means for gathering information about communication paths without flooding the network with requests for information and with replies to these requests? Is it possible to constantly improve the accuracy of the information being gathered (in the presence of time-varying network conditions) in a way that focuses on the information that is actually needed, rather than trying to gather information about all possible paths?

c. Can self-awareness be exploited for timely decision making without risking the consequences of constant "changes of mind"? For instance, distinct nodes could select the same path in an uncoordinated manner due to the fact it is momentary, then have to renege when all use it and hence overload it. What are the risks, costs, and mitigating factors associated with frequent "oscillations" regarding such decisions?

Figure 1: A testbed.

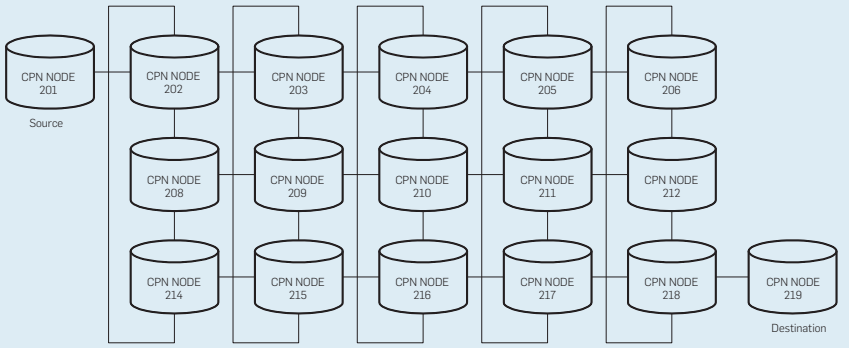


Figure 2: Path length compared.

Other relevant questions involve scaling, security, reliability, and mobility. Our work at Imperial College has shown that security[9,24] and reliability (discussed later) can also be enhanced in a SAN. However, the effect of malicious nodes and users and node mobility (often studied in mobile ad hoc networks[12]) need further work. Scalability of SANs can be improved through recursive routing[21] and hierarchical routing techniques that have long been used in the Internet.

Reliable Communication in Unreliable Networks

Travel time in unknown environments is of interest in networks and robotics; algorithms that minimize worst-case travel times in finite graphs were covered by Papadimitriou and Yannakakis.[25] The first question (a) raised earlier is answered by our result showing that average travel time is finite under worst-case conditions in an infinite graph,[8] as long as packet forwarding can be aborted when the packet is unable to reach the destination after a predetermined length of time, and the forwarding process is then restarted at the source, provided that the routing process is randomized. This proves that packets can be reliably forwarded to destinations with probability one, even when routing information is not available, provided that a randomized algorithm is used.

Consider some node U that wishes to forward a packet to a destination V to which there exists at least one valid path. However, we admit the possibility of errors in the routing information about how to reach V, allowing for approximate or erroneous routing. Since the network is infinite and nodes may not know the direction a packet needs to be forwarded, a packet can get lost and meander indefinitely from node to node without ever reaching its destination. Let us make things worse by also allowing packets to be dropped inadvertently. The system uses a time-out, so if a packet does not reach its destination before the time-out elapses, the packet is destroyed and retransmitted by the source. Since there is at least one path from U to V, the shortest path length D (expressed in number of hops) is finite. The mathematical model for such a system is a random walk, where the

"walker" is a packet being forwarded from U to V, starting at U at time $t = 0$. The packet's remaining distance X_t at time t is the length of the shortest path to the destination, and the travel time is the first instant T when $X_T = 0$. The key question—whether there exists a finite T such that $X_T = 0$—is answered by Gelenbe[8] showing that:

$$E[T] = 2D \frac{1 + \frac{\lambda}{r}}{-b + \sqrt{b^2 + 2c(\lambda + r)}}$$

Here b is the "drift" parameter, so if $b < 0$ then the packet is on the average making progress toward the destination, while if $b > 0$ then it is moving away from it, and c is the variance per unit time or fluctuation related to the packet's motion toward or away from the destination, so $c > 0$. $1/r$ is the average value of the time-out, and λ is the probability per unit time that the packet is lost. The expression above tells us, as expected, that if there is no time-out, that is, $r = 0$, and losses are possible, that is, $\lambda > 0$, then $E[T] = +\infty$, that is, a packet will never make it to its destination. If there are no packet losses, that is, $\lambda = 0$, and there is also no time-out, that is, $r = 0$, then $E[T] < \infty$ if $b < 0$, while if $b > 0$ then, as expected, $E[T] = +\infty$. The time-out is also thus a protection against packets that "lose their way" by traveling on and on through the infinite network without ever reaching their destination. Most interestingly, when $c > 0$, that is, there is randomness in the path, we have $E[T] < \infty$ as long as there are losses or a finite time-out. However, if the path is deterministic, that is, $c = 0$, then the travel time is infinite unless $b < 0$. Thus we establish that, even in the worst case of an infinitely large network in which individual nodes may lose packets and packets may lose their way by meandering indefinitely in the network, as long as there is randomness in the routing ($c > 0$) and a finite time-out is available ($r > 0$), the packet will reach its destination in finite time, even though no correct routing information is available at the nodes of the network. This model also covers the case of "wrong" routing information with $b > 0$, where packets are probabilistically sent away from the destination, and with uncertain or "partially correct" routing information with $b < 0$ where (on average) packets get closer to the destination at each step. The "ideal" case $b = -1$ is when the packet makes the fastest possible progress to the destination.

Interesting is that the criterion that combines delay with number of hops leads to the best results, though they are comparable to the results based on using just the delay as the QoS goal.

Self-Aware Routing

The second question (b) concerns the routing algorithms. Most routing techniques attempt to optimize one or more criteria in addition to the basic requirement of forwarding traffic from any source to any destination. The shortest-path routing algorithm is based on the premise that if a packet visits the smallest possible number of hops toward its destination, then the network overhead is minimized, as is most of the other criteria of interest (such as packet loss and packet delay). A SAN will attempt to optimize network performance through exploration, measurement, and adaptation, rather than through an a priori choice (such as the shortest path).

Much of the published work on SAN routing follows two approaches using reinforcement learning (RL), first proposed for packet routing by Boyan and Littman.[2] The Cognitive Packet Network (CPN) approach[11,13] uses "smart packets" (SP) for path discovery, together with RL and neural networks installed in each network node, adaptively selecting paths so as to offer "best effort" QoS to end users. SPs are sent out by nodes that are actively involved in forwarding packets to discover and assess paths that lead to destination nodes. The "Ant Colony"[3,4,19] paradigm searches for paths from source nodes to specific destination nodes by emulating the pheromone-based technique used by biological ants to mark their paths and communicate with fellow members of the same colony. Both CPN and Ant Colony algorithms include random search when information about suitable paths is unavailable, reinforcing the importance attributed to paths that appear to be best and using alternate paths when previously selected paths prove less desirable.

In CPN, SPs discover routes for connections to specific destinations. They are routed using RL based on a QoS "goal." We use the term "goal" to indicate that there is no guaranteed QoS and that CPN provides a best effort to satisfy the desired QoS. SPs do not carry payload, finding routes and col-

Figure 3: Total packet delay with 6.4Mbps background traffic carried out for SPs, that is, for a small fraction of the traffic, resulting in reduced router computation overhead.

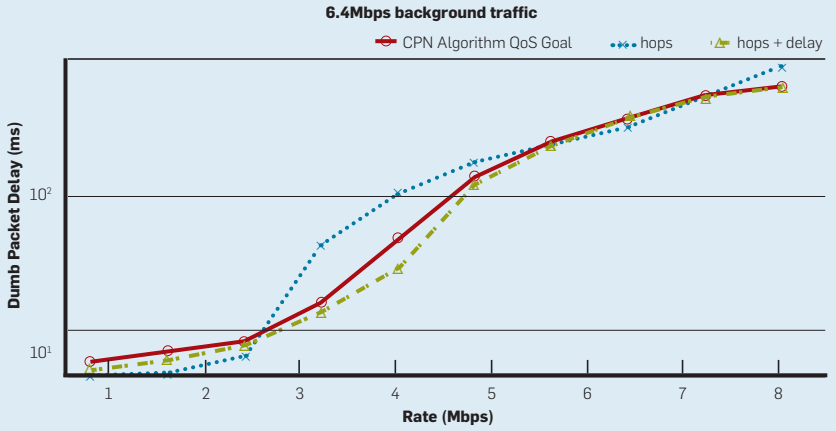
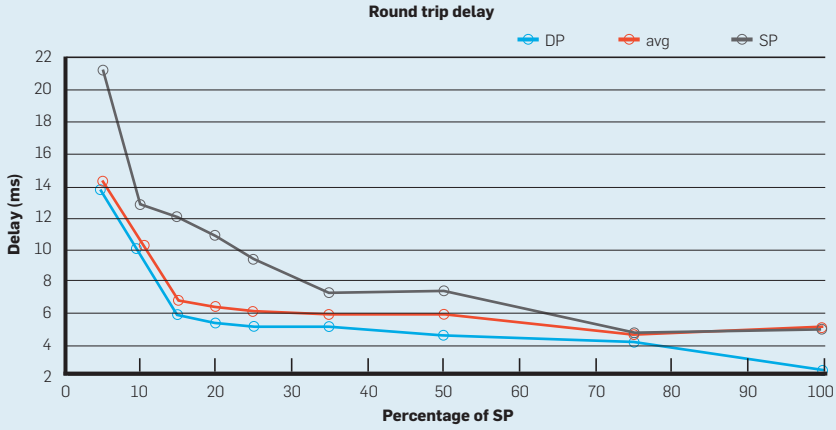


Figure 4: Total average delay for SPs (top) and DPs (bottom) and average delay for all packets (center) as a function of the percentage of SPs.



lecting measurements based on three complementary elements:

▸ Each node engaged in forwarding packets to some destination sends out SPs that search for paths to the destination(s) and gather measurement data about these paths. This data is not limited to delay and packet loss but may also include measurable information about power utilization by nodes on those paths, the volume of traffic on the paths, and the security of the nodes and links on the paths. SPs do not carry the actual traffic payload but are used just for measurement and exploration;

▸ Each node maintains a neural network to compute the next node an SP from this node must go to. The weights of the neural network are updated using an RL algorithm that uses data collected by the SPs. In CPN, the role of the neural network is just to route the SPs, and the "dumb packets" (DP) that carry the payload are routed differently; and

▸ Each source node maintains an ordered list of paths to the destination(s) they are concerned with. This list includes paths that are discovered by the SPs and is updated using the QoS information collected by the SPs. The list is ordered with the best paths at the top, so the payload or DB is forwarded along the complete path (that is, they are source-routed), and intermediate nodes do not normally interfere with them other than providing a store-and-forward capability.

When (and if) an SP arrives at its destination, the destination generates an acknowledgment (ACK) packet, and the ACK stores the "reverse route," as well as the measurement data collected by the SP. An SP that does not reach its destination after a predetermined number of hops (typically set as a multiple of the network's diameter, here 30) is destroyed. The ACK being returned as a result of an SP will travel along the "reverse route" obtained from the SP's route, examining it from right (destination) to left (source), removing any sequences of nodes that begin and end in the same node. For instance, the path $< a, b, c, d, a, f, g, h, c, l, m >$ will result in the reverse route $< m, l, c, b, a >$. Note that the reverse route is not necessarily the shortest reverse path nor the one resulting in the best QoS. Also ACKs deposit QoS measurement data in mailboxes (MBs) at the nodes they visit as they move toward the SP's source node. On the other hand, DPs carry payload and use dynamic source routing. The route brought back by an ACK is used as a source route by subsequent DPs of the same QoS class with the same destination until a new route is brought back by another ACK. An MB in each node stores QoS information.

Each MB is organized as a least-recently-used (LRU) stack. The entries in an MB are identified with the QoS class and the destination. Since SPs are routed at each node using RL, they concentrate their search on the most promising paths for a given destination. Each node contains one or more random neural networks (RNNs),[15] where each RNN corresponds to a QoS class and a destination. In the RNN, the choice of the output link of a node is represented by a neuron, and the link corresponding to the "most excited" neuron is used to forward a given SP. The RL algorithm operates as follows:

A "goal function" G is used to characterize the objective one wishes to optimize for a given source to destination connection; this objective may be hop count (if one wants to minimize path length), delay, packet loss rate, energy utilization, and more, or a combination of these factors. The reward R, which is defined as $R = 1 \div G$, and successive values of R obtained from measurements carried back by the ACK packets, are denoted by R_l, $l = 1, 2, \cdots$, and used to compute a "historical value" of R:

$$T_l = \alpha T_{l-1} + (1 - \alpha) R_l$$

where α is some constant ($0 < \alpha < 1$) that determines the algorithm's mem-

ory, and R_l is the most recently measured value of the reward. Suppose we have made the lth decision that chooses the output link (neuron) j, where the lth reward calculated for the QoS information received from the network is R_l. We first determine whether R_l is larger than or equal to the threshold T_{l-1}. If it is, then to reward this success, we increase (significantly) the excitatory weights going into neuron j and make a small increase in the inhibitory weights leading to other neurons. If the R_l is less than T_{l-1}, then we moderately increase the excitatory weights leading to all neurons other than j to open up different decision options and increase significantly the inhibitory weight leading to neuron j in order to punish it for not having provided a useful prediction.

Finally, the excitation probabilities of each neuron in the RNN are computed, and the SP is forwarded to the output link corresponding to the neuron that is the most "excited." The arrival of an ACK to a node triggers the update of the weights of the RNN, while the arrival of an SP to the node triggers the execution of the RNN algorithm to make the routing decision. Thus several weight updates can occur between two successive updates of a routing decision. Similarly, if no ACKs arrive at a given node between two successive arrivals of an SP, the successive SPs will use the same routing decision.

Numerous experiments have been run with CPN with both simulation and actual network testbeds;[11,13,14] here, we report on three with real networks with 17, 25, and 46 nodes and different topologies. All measurements we report used testbeds built with off-the-shelf components running CPN. The routers were Pentium IV-class machines with four-port Ethernet interfaces running Linux 2.6.15, where CPN was implemented as a loadable kernel module. All links were full-duplex at 10MB/sec or 100MB/sec, depending on the experiment.

We start with measurements made in a wired testbed consisting of 17 nodes (see Figure 1) chosen because it offers a large number of alternate paths within a relatively small network; adjacent nodes are connected with 10Mbps Ethernet links. All tests use a flow of UDP packets entering the CPN network with constant bit rate (CBR) traffic and packet size of 1,024KB. For each experiment, 10,000 packets were sent out from source to the destination, and each measurement point provided averages or statistics for the 10,000 packets when background traffic is introduced to each link. The average hop count, forward delay, and packets loss rate under different background traffic were reported. We used Algorithm-H, Algorithm-D, and Algorithm-HD to denote the RNN routing algorithms using hop, delay, and the combination

Figure 5: Use of routes with low traffic rate; delay is the QoS goal.

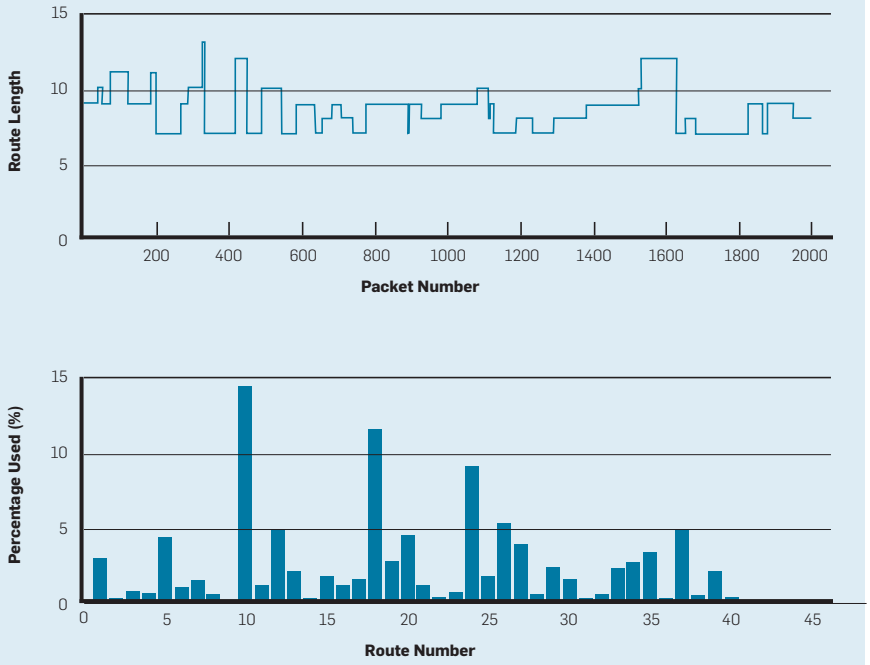
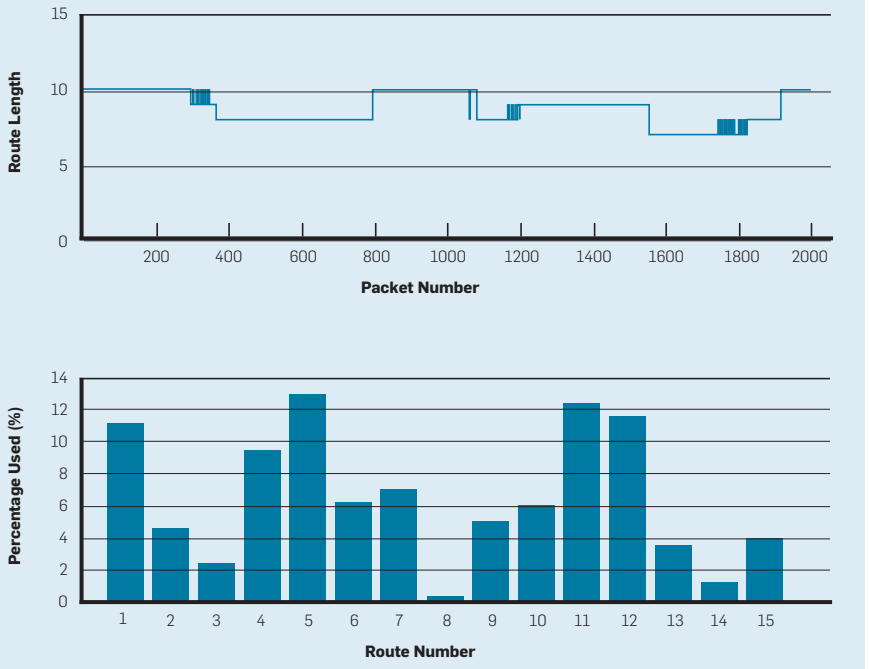


Figure 6: Use of routes with high traffic rate; delay is the QoS goal.



of hop count and delay as the QoS goal, respectively. The length of the shortest path between the source (201) to the destination (219) was seven hops; there were five different shortest paths.

Figure 2 shows that CPN indeed provides the "self-aware" capability being sought since its measured behavior corresponds to the stated QoS goal. The curves give the average number of hops of the routes CPN selects when different QoS goals are used with different levels of background traffic (distinct figures), while end-to-end traffic is varied along the *x*-axis. When hop count is used as the QoS goal (cross or "hop" in the figure), the average number of hops under different background traffic rates are close to the minimum value of seven hops. When delay is used as the QoS goal (circle or "delay" in the figure), the average path length is longer, so CPN adapts to the guidelines it has received and chooses shortest-delay paths rather than shorter hop paths. Note that when the source-to-destination traffic is high (right-hand side of the figure) there is little difference in path lengths for the different QoS goals, due to the fact that performance is equally poor for all possible criteria in heavy traffic.

The average packet-forwarding delay for each of the goal functions (see Figure 3) is also measured as a function of the amount of traffic from source to destination for different levels of background traffic; the results confirm those in Figure 2. The blue curve in Figure 3 corresponds to using the number of hops as the QoS goal to be minimized; as expected, it leads to the longest delay. Interesting is that the criterion that combines delay with number of hops leads to the best results, though they are comparable to the results based on using just the delay as the QoS goal. Confirming the results in Figure 2, these results show that the CPN algorithm is indeed self-aware in that it is able to translate its overall objectives into effective adaptive decisions taken in real time.

Measuring total delay, including queueing and forwarding delay, experienced by SPs and DPs, one sees that (see Figure 4) when the SPs are increased (expressed as a percentage of the DPs being forwarded), the overall QoS improves, but most of the improvement is achieved at a relatively low 20% of SPs with respect to DPs. As one would hope, the DPs experience better QoS; the SPs "pay the price" of the search activity by experiencing less-favorable QoS. Since SPs and ACKs are each approximately 10% of the length of a full Ethernet packet, for 20% of SPs over DPs, the total additional traffic generated by CPN over and above the payload traffic is 0.04%. Note that route computations in CPN are carried out only for SPs, that is, for a small fraction of the traffic, resulting in reduced router-computation overhead.

To measure whether CPN spreads traffic among many paths as the traffic rate increases from 100 packets/sec in Figure 5 to 1,000 packets/sec in Figure 6, there is a more even distribution of traffic over a smaller number of paths having better QoS.

Adverse effect of slower decisions. One obvious trade-off in any decision process is whether it is better to "optimize more and decide later" or provide decisions as soon as they can be formulated and hope for the best. Thus the experiments described earlier refer to a situation where decisions were taken

Figure 7: CPN 46-node testbed subject to failures.

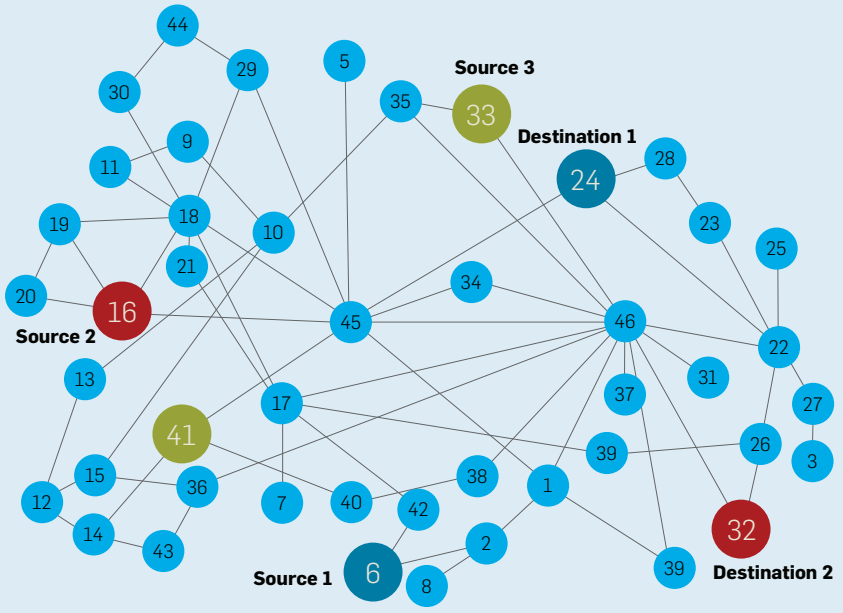
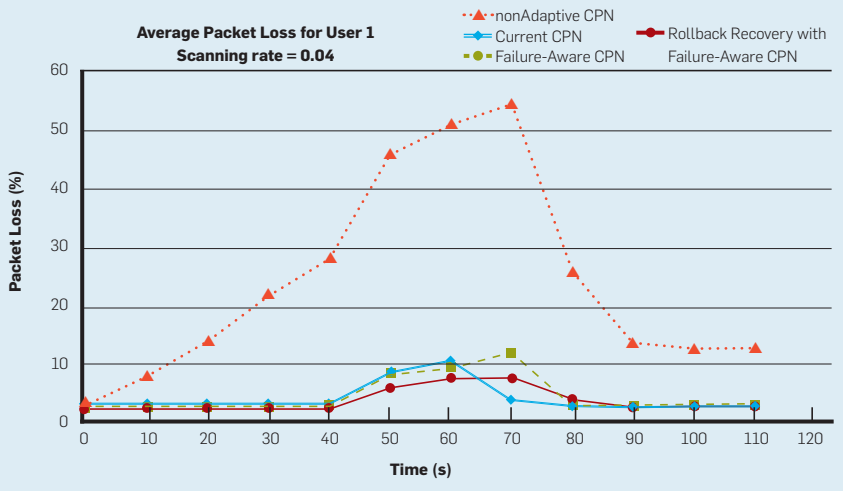


Figure 8: Adaptation reduces loss when failures occur.

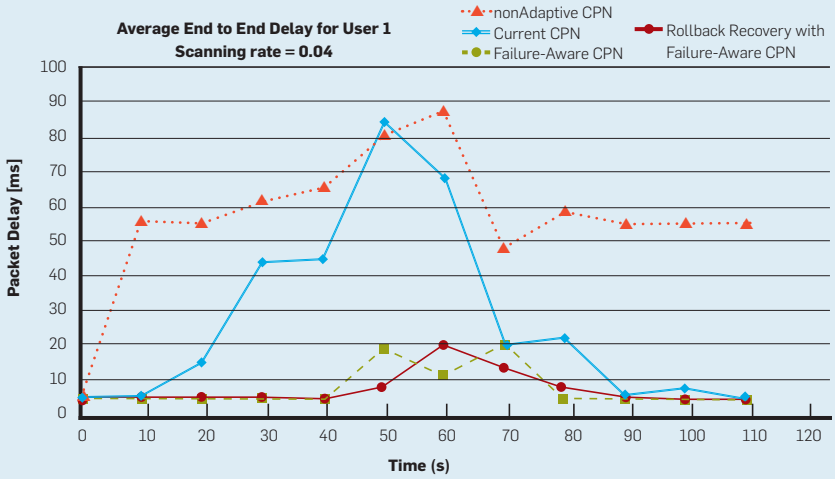


in real time based on either the current state of the RNN in each node or on the most recent RL updates that have been made. A more sophisticated way of selecting paths could use the underlying RNN and RL but also make a further optimization decision based on the fact that at each source node, CPN maintains a set of paths with the most recent measured QoS metric for each path, as well as for each of the destinations with which the source communicates. Now suppose that two distinct paths *SxIyD* and *SuIvD* connect source *S* to destination *D* through the same intermediate node *I*, and these paths have been discovered by SPs and provide QoS metrics *G*(*SxIyD*) and *G*(*SuIvD*). Obviously *SuIyD* and *SxIvD* are also valid paths. If they were not previously discovered by SPs, one can still infer their estimated (not measured) QoS assuming the QoS data is additive. We denote the inferred QoS values by *g*(*SuIyD*) and *g*(*SxIvD*). Now suppose that one of the two inferred QoS values, say, *g*(*SuIyD*) is the "best" one (such as smallest delay, smallest loss, or best value of some other metric of interest). One can then use the hitherto untested path *SuIyD* to forward DPs, rather than the best of the two paths actually tested. Note that this operation of selecting new paths by combining prefixes and suffixes of previously discovered paths resembles the "crossover operation" in a genetic algorithm.[10] Experiments run with 1,024B DPs and varying the DP rate of 100 to 800 packets/sec showed that this additional optimization provided a small improvement in both packet loss rate and average packet delay.

However, when a significant amount of background traffic was added to each link, even with relatively low DP rate exceeding a certain value (300 packets/sec), the original CPN algorithm performed significantly better, showing that the slower optimization process using older data introduced on top of CPN by the genetic algorithm-like approach is unable to respond quickly enough to changing network conditions.

Self-aware adaptation to failures. During experiments conducted in a 46-node testbed (see Figure 7), we observed that CPN can also protect a network against worm-like failures. In these experiments, failures begin at a given node that then randomly causes other nodes to fail. A node that fails is unable to forward traffic, causing neighbors to fail. Node failure is followed by recovery, representing cleaning and patching, at a constant rate of 1 node/sec. Figures 8 and 9 report the measured average packet loss and delay for 10 experiments where User 1 sent 7MB/sec CBR traffic from Node 6 to Node 24. When CPN is operating, the QoS is significantly better than when CPN is stopped after paths are established (top curves). Moreover, as further adaptive measures are taken (other curves),[27] performance and QoS improve further.

Figure 9: Adaptation reduces delay in the presence of failures.



Ant colony routing. Ant colony routing algorithms[3,4,19] differ from CPN in the way they use RL, as well as in other respects. Inspired by the way ants use pheromones to mark their paths and communicate about sources of food, packets represent ants, nodes and links represent locations, and packets move toward their destinations based on paths with strong markings. When a packet reaches its destination, a corresponding "marking" packet heads back to the source by following the path in reverse or quasi-reverse order (or following the "strongly marked trail" and strengthening the marking at each link and node it visits). The markings degrade over time ("forgetfulness") if not reinforced by the passage of other packets. The algorithm is initiated by a random search until the destination node is found and the discovered path(s) is reinforced by the returning packets. The returning packet from the destination is like the ACK packet in CPN, but ant colony routing algorithms use payload packets for both search and data delivery, while CPN separates the search role via SPs from the payload role via DPs; the resulting QoS is better when DPs specialize in payload conveyance and SPs are restricted to search. Thus CPN uses more packets, since SPs are constantly being sent forward to accomplish the search function, representing a constant fraction (such as 10%) of total traffic. Moreover, ant colony algorithms do not use a neural network (as in CPN) to store RL information inside a given node. CPN routers carry out route computation only for SPs and represent a small fraction of total traffic, but ACKs and DPs are source-routed, while ant colony algorithms typically require route computation for all packets. Clearly, ant colony algorithms are better adapted to networks that experience frequent disruption, since all packets are in a sense autonomous. In CPN, if a DP's path is disrupted, the packet must be retransmitted at the source, with information brought back by a subsequent SP that finds another path. Thus CPN will be better at forwarding packets but slower in responding to changes in topology.

Route Oscillations

Since the days of the ARPANET, it has been observed that route oscillations[18] can cause performance to suffer under

Figure 10: CPN testbed emulating the Swiss Education and Research Network. The square node is the sink; the 24 round nodes are sources.

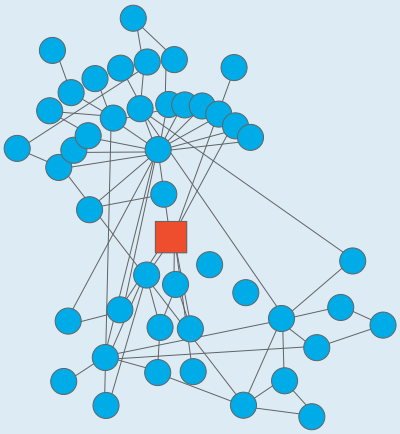
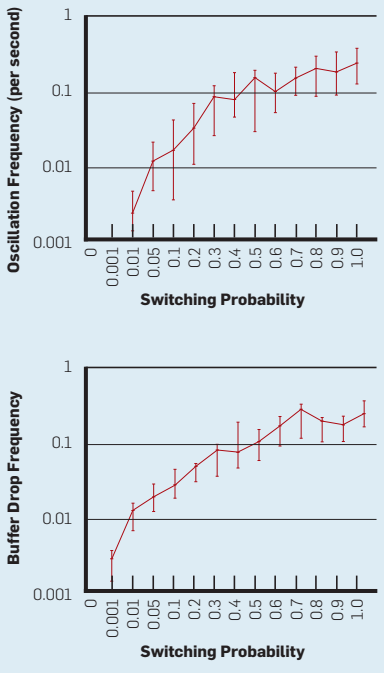


Figure 11: Oscillation frequency (top) and packet-drop rate (bottom) vs. switching probability.



medium to high load conditions. Oscillations occur if load-sensitive metrics are used to select routes, becoming more frequent at higher loads[28] due to the transfer of flows to lightly loaded paths that then become overloaded; the result is that flows are transferred to other paths that in turn get overloaded. The overlap of different routers' measurement windows also lead to oscillations when flows interfere and prevent the network from stabilizing. Frequent route switching can reduce performance by slowing the network's convergence to best paths,[6] increasing node overhead. Route oscillations also affect TCP[26] due to TCP response to asymmetric paths (when a data packet's path is different from that of its ACKs) and to out-of-order packet delivery when packets take different paths and reach their destinations in a different order. The output node must then reassemble packets into the right order,[1] causing additional delay and loss of packets due to the finite capacity of the buffers used for reassembly; QoS is thus degraded for real-time applications (such as voice and media). Routing oscillations are also studied in overlay networks.[17]

One must therefore examine whether (i) frequent oscillations can occur in a SAN, (ii) whether they can be easily mitigated or reduced, and (iii) whether they are necessarily detrimental to performance. Concerning (iii), each time a path is selected, CPN forwards the traffic along that path until the path is changed, and as long as the path is being used, useful work is done and packets are delivered to the destination. When a path switch occurs, packets already engaged in the path continue flowing to the destination along the previous path, since each DP stores the path it is following, and the change in path decided at the source affects only subsequent packets, not those already engaged in the path. Thus CPN does limit the effect of path switching in a SAN.

Concerning (ii), various ways are available to mitigate or reduce oscillations. For instance, the source node can allow switching only if the QoS gain exceeds a significant threshold. Another approach is to require that each time a path is used, that usage must exceed a certain number of packets before switching can be considered again.

Here, we report on a testbed with full-duplex links at 10Mb/sec running CPN (see Figure 10) that emulates the topology of the Swiss Education and Research Network (as of 2007)[16]; it included 24 constant bit-rate flows at 1.66Mb/sec generated to create DP traffic of just over 40Mb/sec. Combined with SP and DP traffic, the result was a slightly overloaded system. SP traffic was set at 10% of DPs, and each source that sent traffic to the same destination had four inputs for a total of 40Mb/s of available incoming bandwidth. The QoS goal used was the minimization of delay. All the experimental curves we report include 95% bars for the measurement values.

Figure 11 shows the effect of introducing a simple rule that limits the frequency of path switching; each time a source selects a new path identified as causing the smallest delay, the decision is accepted only with probability P (the

Figure 12: Average packet delay vs. switching probability.

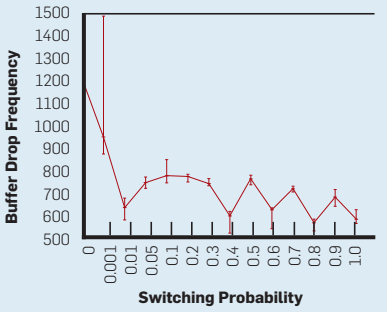


Figure 13: Average packet delay vs. threshold.

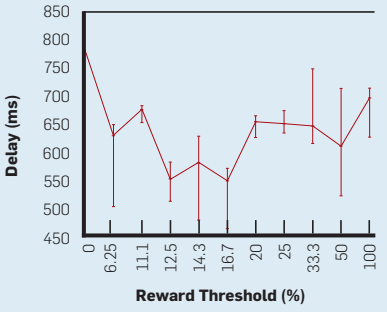


Figure 14: Rate of path oscillations vs. threshold.

switching probability). Thus if $P = 1$ all recommended path switches occur; when $P = 0.001$ only one of every 1,000 recommended path switches actually takes place. Thus the top curve shows how the switching probability affects path oscillations, starting with a given path and returning to it again; as the switching probability increases so does the rate at which paths oscillate. The effect of P on the packet drop rate at the output resequencing buffer is shown in the bottom chart in the figure.

Figure 12 indicates that improvement in QoS (delay in this case) can be achieved with a small switching probability ($P = 0.01$ or a little higher). Path switching improves average delay (and is why CPN attempts to switch paths), though it comes at the cost of packet loss. However, one can mitigate this loss by probabilistically limiting the switching while retaining the benefit of improved QoS by lowering packet delay.

The SAN programmer can also limit oscillations by setting a threshold that allows a path switch only when the projected QoS improvement exceeds the threshold. A small threshold allows more frequent switches and hence potentially more oscillations, but a large threshold may hurt QoS. Figure 13 shows that if the threshold is small, the observed packet delay is large, and as the threshold increases delay improves, but packet delay increases again for larger thresholds. For small threshold values, longer packet delays indicate that switching occurs based on "noise" rather than on real gain. Increasing the threshold in Figure 14 would reduce the oscillations, though the effect would level out quickly. The threshold thus limits the negative effect of switching but preserves the advantages of self-awareness and adaptation.

Conclusion

The approach to developing self-aware networks presented here gives end users the means to explore the state of the network so as to find the best ways to meet their communication needs. Focusing on the primary function of packet routing, I have tried to answer a number of questions concerning the feasibility of such networks and whether reliable communications is possible in largely unknown networks. I have also addressed whether there is a risk of unstable behavior in such systems due to constant "changes of mind" and oscillations as new information becomes available to users and whether a user's ability to adapt to changing circumstances in the network reduces the consequences of network failure. The experiments reported relate to small (up to 46-node) networks; more results are available at http://san.ee.ic.ac.uk.

The Internet consists of hierarchically organized autonomous systems of relatively small size, and one can imagine that routing inside and among them would benefit from the techniques discussed here. Future research is likely to investigate how these ideas can be integrated into existing networks, how they scale to large networks, how they might be able to withstand the malicious behavior of users and network nodes, and how they can support mobile users.

Acknowledgments

Research sponsored by the U.K. Engineering and Physical Sciences Research Council Grant GR/S52360/01 on self-aware networks and quality of service; and by E.U. FP6 Projects on self-aware networks, performance, and adaptivity and componentware for autonomic situation-aware communications and dynamically adaptable services.

References

1. Bennett, J.C.R., Partridge, C., and Shectman, N. Packet reordering is not pathological network behavior. *IEEE/ACM Transactions on Networks 7*, 6 (1999), 789–798.
2. Boyan, J.A. and Littman, M.L. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the Advances in Neural Information Processing Systems Conference* (Denver). Morgan Kaufmann, San Francisco, 1994.
3. Chen, G., Branch, J., and Szymanski, B.K. A self-selection technique for flooding and routing in wireless ad-hoc networks. *Journal of Network and Systems Management 14*, 3 (2006), 359–380.
4. Di Caro, G. and Dorigo, M. Antnet: Distributed stigmergetic control for communication networks. *Journal of AI Research 9* (1998), 317–365.
5. Dobson, S. Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., and Zambonelli, F. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems 1*, 2 (2006), 223–259.
6. Gao, R., Dovrolis, C. and Zegura, E. Avoiding oscillations due to intelligent route control systems. In *Proceedings of the 25th IEEE International Conference on Computer Communications* (Barcelona, Catalunya, Apr. 23–29). IEEE Press, New York, 2006.
7. Gelenbe, E., Sakellari, G., and D'Arienzo, M. Admission of QoS-aware users in a smart network. *ACM Transactions on Autonomous and Adaptive Systems 3*, 1 (Mar. 2008), 1–28.
8. Gelenbe, E. A diffusion model for packet travel time in a random multihop medium. *ACM Transactions on Sensor Networks 3*, 2 (June 2007).
9. Gelenbe, E. and Loukas, G. A self-aware approach to denial-of-service defence. *Computer Networks: The International Journal of Computer and Telecommunications Networking 51*, 5 (Apr. 2007), 1299–1314.
10. Gelenbe, E., Liu, P., and Lainé, J. Genetic algorithms for route discovery. *IEEE Transactions on Systems, Man, and Cybernetics B 36*, 6 (Dec. 2006), 1247–1254.
11. Gelenbe, E., Lent, R., and Nunez, A. Self-aware networks and QoS. *Proceedings of the IEEE 92*, 9 (Sept. 2004), 1478–1489.
12. Gelenbe, E. and Lent, R. Power-aware ad hoc cognitive packet networks. *Ad Hoc Networks 2*, 3 (July 2004), 205–216.
13. Gelenbe, E., Gellman, M., Lent, R., Liu, P., and Su, P. Autonomous smart routing for network QoS. In *Proceedings of the First International Conference on Autonomous Computing* (May 17–19). IEEE Computer Society Press, New York, 2004, 232-239.
14. Gelenbe, E., Lent, R., and Xu, Z. Measurement and performance of a cognitive packet network. *International Journal of Computer and Telecommunications Networking 37* (2001), 691–791.
15. Gelenbe, E. Learning in the recurrent random neural network. *Neural Computation 5*, 1 (1993), 154–164.
16. Gellman, M. Oscillations in self-aware networks. *Proceedings of the Royal Society A 464*, 2096 (2008), 2169–2186.
17. Keralapura, R., Chuah, C.N., Taft, N., and Iannaccone, G. Can coexisting overlays inadvertently step on each other? In *Proceedings of the 13th IEEE International Conference on Network Protocols* (Boston, Nov. 6–9). IEEE Computer Society, New York, 2005, 201–214.
18. Khanna, A. and Zinky, J. The revised Arpanet routing metric. *SIGCOMM Review 19*, 4 (1989), 45–56.
19. Koenig, S., Szymanski, B.K., and Liu, Y. Efficient and inefficient ant coverage methods. *Annals of Mathematics and Artificial Intelligence 31*, 1–4 (2001), 41–76.
20. Labovitz, C., Malan, G.R., and Jahanian, F. Internet routing instability. *IEEE/ACM Transactions on Networks 6*, 5 (1998), 515–528.
21. Liu, P. and Gelenbe, E. Recursive routing in the cognitive packet network. In *Proceedings of the Third International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities* (May 21–23, 2007, Trento, Italy), 21–23.
22. Malkin, G. RIP Version 2. RFC 2453, 1998; http://www.faqs.org/rfcs/rfc2453.html.
23. Moy, J. OSPF Version 2. RFC 2328, 1998; http://www.ietf.org/rfc/rfc2328.txt.
24. Öke, G.G., Loukas, G., and Gelenbe, E. Detecting denial-of-service attacks with Bayesian classifiers and the random neural network. In *Proceedings of the IEEE International Conference on Fuzzy Systems* (London, 2007), 964–1969.
25. Papadimitriou, C.H. and Yannakakis, M. Shortest paths without a map. *Theoretical Computer Science 84*, 1 (1991), 127–150.
26. Ranadive, U. and Medhi, D. Some observations on the effect of route fluctuation and network link failure on TCP. In *Proceedings of the 10th International Conference on Computer Communications and Networks* (Oct. 15–17, 2001), 460–467.
27. Sakellari, G. and Gelenbe, E. Adaptive resilience of the cognitive packet network in the presence of network worms. In *Proceedings of the NATO Symposium on C3I for Crisis, Emergency and Consequence Management* (May 11–12, 2009, Bucharest, Romania).
28. Shaikh, A., Varma, A., Kalampoukas, L., and Dube, R. Routing stability in congested networks: Experimentation and analysis. In *Proceedings of SIGCOMM 2000*, (Stockholm, Aug. 29–Sept. 2). ACM Press, New York, 2000, 163–174.
29. Thorup, M. and Zwick, U. Compact routing schemes. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures* (Heraklion, Crete Island, Greece, July 4–6). ACM Press, New York, 2001, 1–10.

Erol Gelenbe (e.gelenbe@imperial.ac.uk) is Head of the Intelligent Systems and Networks Research Group and Professor in the Dennis Gabor Chair, Electrical and Electronic Engineering Department, Imperial College London.