

Steps Toward Self-Aware Networks*

Erol Gelenbe
Dept. Electrical & Electronic Eng'g. Imperial College
London SW7 2BT, UK
e.gelenbe@imperial.ac.uk

ABSTRACT

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 at which the routing information can be practically updated throughout the network. In such systems it becomes necessary to allow individual nodes to pro actively discover the presence of other nodes, links and paths, as needed and on demand, leading to the design of networks which we call self-aware. This survey paper focuses on experimental and theoretical research concerning the technical steps that can lead to self-aware networks.

1. INTRODUCTION

As computer networks become extremely large, the information available concerning the state of a network at any given time becomes increasingly uncertain. Internet protocol offer an orderly update of the status of the network so that routing algorithms may operate seamlessly despite constant changes in the network's topology based on the shortest path algorithm [2, 19], with Distance Vector [6] and Link State [11, 7] techniques. However, the need to convey time-sensitive information such as voice and media has generated much interest in "quality of service" (QoS) routing [8]. Link state changes also become more frequent in larger networks, creating more overhead and delay due to updates throughout the network. Consequently, nformation about the network state, including connectivity, the condition of nodes, traffic conditions and QoS, may propagate more slowly than the rate at which changes occur. Thus it becomes necessary to allow nodes to discover the network state autonomously, without relying on an overall scheme that updates routing tables systematically *throughout* the network. Such updates can be initiated *by the nodes that need this information* and *at the time when the information is needed*, rather than throughout the network and whenever changes in network state occur. We use the term "self-aware network" (SAN) [23] for a system composed of

nodes which have the ability to join and leave the network autonomously, and to discover paths *when the need to communicate arises*. The nodes in SAN should be able to *sense the status* of other nodes, links and paths, including traffic levels and congestion, so as to update their own relevant information *about the paths they need to use*, based on criteria that are *specific to their own needs*, so that each connection may use the paths that optimise *their own* QoS criteria, rather than using a common criterion (such as the shortest path) for each connection. These needs may include user QoS requirements [22], or performance, reliability, security, defense against attacks [32, 33], power utilisation [24], etc. A SAN can be a wired, wireless, or a peer-to-peer system. A wireless ad-hoc network [20] is a practical example of a SAN which responds to the time-varying conditions related to the mobility of nodes, and changes in the conditions of wireless links (noise, physical obstructions, etc.). Networks which have to operate autonomously and remotely (such as sensor networks), can also benefit from self-aware capabilities for self-monitoring [13]. Research on effective SAN architectures has also motivated work on Autonomic Communications [28] and bio-inspired techniques for networking [30]. Ideally, self-awareness is a desirable property of most networked systems. However for SANs to become widely accepted, many fundamental questions need to be answered in a positive manner, including:

- (A) Assuming that in the worst case a node only knows its immediate neighbors, although the network is fully connected, can a node forward a packet successfully to any other node in the SAN in finite time without the existence of routing tables at each node?
- (B) What are practical means of gathering information about the 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 that is being gathered, in the presence of time varying network conditions, in a way which focuses on gathering information that is actually needed rather than trying to gather information about all possible paths? And how about any additional overhead?
- (C) Can self-awareness be exploited for timely decision making without risking the consequences of constant "changes of mind". These will happen, for instance, if distinct nodes select the same path in an uncoordinated manner due to the fact that it is momentarily underloaded, and then have to renege this decision

*Research sponsored by UK Eng. Phys. Sci. Res. Council (EPSRC), and the EU FP6 CASCADAS Project.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2008 ACM 0001-0782/08/0X00 ...\$5.00.

when they all use it and hence overload it. What are the risks, costs and mitigating factors associated with frequent “oscillations” regarding such decisions?

Other relevant questions include scaling, security, reliability and mobility. Our work shows that security [32, 33] and reliability (see Section 3.2) can be enhanced in a SAN. The effect of malicious nodes and users, and node mobility (studied for Mobile Ad Hoc Networks [20, 24]) both need further work. We have studied scalability of SANs via recursive routing [34], and it can also benefit from hierarchical routing methods of the Internet using Autonomous Systems.

2. RELIABLE COMMUNICATION IN UNRELIABLE NETWORKS

Travel time in unknown environments is of interest in networks and robotics, and rules for minimizing worst case travel times in finite graphs are considered in [3]. Here we address the average travel time in the *worst case* condition of an *infinite* graph, where packets may be *dropped*, using [35] answering question (A). Say that node U , wishing to forward a packet to a destination V , disposes of *randomised* routing information about how to reach V , allowing for approximate or erroneous routing. Since the network is infinite and nodes do not know the exact direction in which a packet needs to be forwarded, a packet can get lost and meander indefinitely from one node to another without ever reaching its destination. We also allow packet loss. Thus, a time-out is used so that 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 is $D < \infty$ in number of hops. The corresponding mathematical model [35] is a random walk, in which 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, arriving to destination at the first instant T when $X_T = 0$. The key question, whether there exists a *finite* T such that $X_T = 0$, is addressed in [35] where it is shown that:

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

Here b is the “drift” parameter, so that if $b < 0$ then the packet is on the average making progress toward the destination, while if $b > 0$ 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 that $c > 0$. $1/r$ is the average value of the time out, λ is the probability per unit time that the packet is lost. The expression (1) tells us, as expected, that if there is no time-out, i.e. $r = 0$, and losses are possible, i.e. $\lambda > 0$, then $E[T] = +\infty$, i.e. a packet will never makes it to its destination. If there are no packet losses, i.e. $\lambda = 0$, and there is also no time-out, i.e. $r = 0$, then $E[T] < \infty$ if $b < 0$, while if $b > 0$ then as expected $E[T] = +\infty$, the time-out is a protection against packets that “lose their way” by traveling on and on through the infinite network without ever reaching destination. Most interestingly, when $c > 0$, i.e. there is randomness in the path, we will have $E[T] < \infty$ as long as there are losses or a finite time-out. However if the path is deterministic, i.e. $c = 0$ then the travel time will be infinite unless $b < 0$. Thus we provide an answer to question (A) by saying that, even in the

worst case of an infinitely large network in which individual nodes may lose packets, and also 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. These [35] also covers the case of “wrong” routing information with $b > 0$, where packets are probabilistically sent away from the destination, and the case $b < 0$ of “partially correct” routing information where on the average packets get closer to the destination at each step. $b = -1$ is the ideal case where the packet is making the maximum possible progress to the destination.

3. APPROACHES TO SAN ROUTING

The question raised in (B) concerns routing, and 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 will be minimized, as will 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 some *a priori* choice of criterion such as the shortest path. Much of the published work on SAN routing follows two approaches using reinforcement learning (RL), which was apparently first proposed for packet routing in [4]. The Cognitive Packet Network (CPN) approach [21, 23] uses “smart packets” (SP) for path discovery, together with RL and neural networks installed in each network node. SPs are sent out by nodes which are actively involved in forwarding packets, to discover and assess paths which lead to destination nodes. The “Ant Colony” [9, 26, 16, 29] paradigm also searches for paths to specific destination nodes which are of interest to some source node. This paradigm emulates the pheromone based technique used by biological ants to mark their paths and communicate with other insects of the same colony. Both of these techniques include random search when no information about suitable paths is available, and a technique for reinforcing the importance attributed to paths which have been perceived to be the best, and also a way to use alternate paths when the previously used paths prove to be less desirable. The CPN routing algorithm adaptively selects paths so as to offer “best effort” QoS to the end users [23]. CPN includes *three* types of packets: *Smart Packets (SP)* are used to discover routes for connections to specific destinations. They are routed using reinforcement learning (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 find routes and collect measurements, they do not carry payload. It is based on three complementary elements:

- Each node which is engaged in forwarding packets to some destination sends out SPs which search for paths to the destination(s), and also gather measurement data about these paths. This data is not limited to things such as delay and packet loss, but may also include measurable information about power utilisation by nodes on those paths, the volume of traffic on the

paths, or the security of the nodes and links on the paths. SPs do not carry the actual traffic payload but are just used for measurement and exploration.

- Each node also maintains a neural network which is used to compute the next node that a smart packet from this node actually needs to go to. The weights of the neural network are updated using a reinforcement learning (RL) algorithm which uses data collected by the smart packets. In CPN, the role of the neural network is just to route SPs, and the “dumb packets” which carry the payload are actually routed differently.
- Each source node maintains an ordered list of paths to the destination(s) they are concerned with. This list contains paths that are discovered by the SPs, and is updated using the QoS information that is collected by the smart packets. The list is ordered with the best paths being at the top, so that the payload or “dumb packets” are forwarded along the complete path (i.e. they are source routed) and intermediate nodes do not normally interfere with them other than providing a store and forward capability.

When (and if) a SP arrives at its destination, an *acknowledgment(ACK) packet* is generated by the destination, and the ACK stores the “reverse route” as well as the measurement data collected by the SP. A SP which 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 a SP will travel along the “reverse route”, which is obtained from the SP’s route, examining it from right (destination) to left (source), and removing any sequences of nodes which begin and end in the same node. For instance, the path $\langle a, b, c, d, a, f, g, h, c, l, m \rangle$ will result in the reverse route $\langle m, l, c, b, a \rangle$. 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 towards the source node of the SP. On the other hand, *Dumb Packets (DP)* 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 having the same destination, until a new route is brought back by another ACK. A *Mailbox(MB)* in each node is used to store QoS information. Each MB is organized as a Least-Recently-Used (LRU) stack. The entries in 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 worthwhile paths for a given destination. Each node contains one or more Random Neural Networks (RNN) [5], 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 [22]. A “goal function” G is used to characterise the objective that one wishes to optimize for a given source to destination connection; this objective may be hop count (if one wants to minimize path length), or delay, or packet loss rate, or energy utilisation, etc., or a combination of these factors. The reward R which is defined as $R = \frac{1}{G}$, and successive values of R obtained from measurements carried

back by the ACK packets are denoted by $R_l, l = 1, 2, \dots$. These are used to compute a “historical value” of R :

$$T_l = \alpha T_{l-1} + (1 - \alpha) R_l \quad (2)$$

where α is some constant ($0 < \alpha < 1$) which determines the algorithm’s memory and R_l is the most recently measured value of reward. Suppose we have made the l th decision which corresponds to output link(neuron) j and that the l th 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 this is the case, then to strengthen or reward this success, we increase very significantly the excitatory weights going into neuron j and make a small increase of 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 an useful prediction. Finally, the excitation probabilities of each neuron in the RNN are computed, and the SP is forwarded to the output link which corresponds to the neuron which is the most “excited”. The arrival of an ACK to a node triggers the update of the wights of the RNN, while the arrival of a SP to the node triggers the execution of the RNN algorithm to make the routing decision. Thus several weight updates can occur between tow successive updates of a routing decision. Similarly, if no ACKs arrive to the node between two successive arrivals of a SP, the successive SPs will use the currently available routing decision.

Numerous experiments have been run with CPN, both with simulation and in actual network test-beds [18, 21, 23, 22]. Here we will report on experiments carried out on three real networks with 17, 25 and 46 nodes and different topologies. We first report measurements made in the wired test-bed consisting of seventeen nodes shown in Figure 1, which was 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 is 1024KB. For each experiment, 10,000 packets are sent out from source to the destination and each measurement point provides averages or statistics for this 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 are reported. We use *Algorithm-H*, *Algorithm-D* and *Algorithm-HD* to denote the RNN routing algorithms using hop, delay and the combination of hop count and delay as QoS goal, respectively. The length of the shortest path between the source (#201) to the destination (#219) is 7 and there are five different shortest paths (see Figure 1).

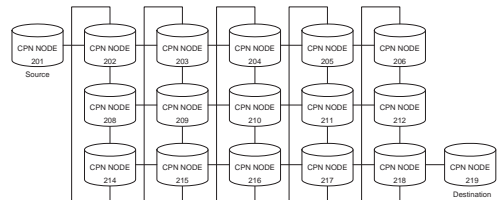


Figure 1: A test-bed

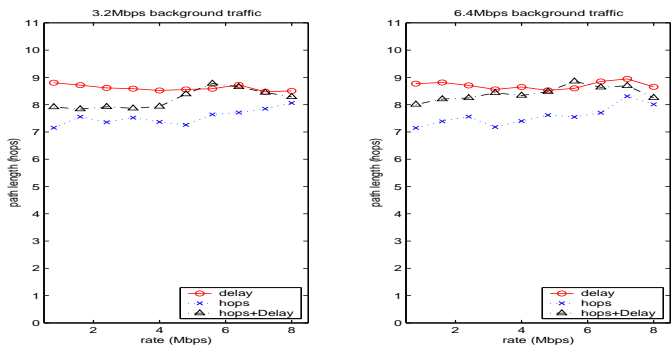


Figure 2: Path length comparison

Figure 2 shows that CPN is indeed providing the SAN capability that is being sought, by adapting its behavior to the QoS goal. The curves show the average number of hops of the routes 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 Figure 2), the average number of hops under different background traffic rates are close to the minimum value 7. When delay is used as the QoS goal (circle or “delay” in Figure 2), we see that the average path length is longer so that the SAN nature of CPN responds to the guidelines that it has received and chooses shortest delay paths rather than shorter paths. Note that when the source to destination traffic is high (right hand side of the figures) there seems to be little difference in path lengths for the different QoS goals; this is due to the fact that performance is equally bad for all possible criteria in this case. We have also measure the average packet forwarding delay for each of the goal functions as show in Figure 3 as a function of the amount of traffic from source to destination, for different levels of background traffic. The results confirm those shown in Figure 2. The blue curve in Figure 3 corresponds to using the number of hops as the QoS goal to be minimized and as expected it leads to the largest delay. Interestingly enough, the criterion that combines delay with the number of hops leads to the best results although they are quite comparable to the results based on using just the delay as the QoS goal. Again, these results confirm those of Figure 2 showing that the CPN algorithm is indeed self-aware in that it is able to translate its overall objectives into effective adaptive decisions which are taken in real time.

We examined the queuing plus forwarding delay experienced by SPs and DPs, and the effect of increasing the proportion of SPs in the network as reported in Figure 4: as the fraction of SPs is 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. Furthermore DPs experience better QoS, while SPs themselves “pay the price” of the additional search activity by experiencing a less favorable QoS. We can also estimate the overhead introduced by the SPs and ACKs. Since SPs and DPs are each approximately 10% of the length of full Ethernet packets, 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 *only* carried out for SPs, i.e. for a small fraction of the traffic, resulting in reduced router

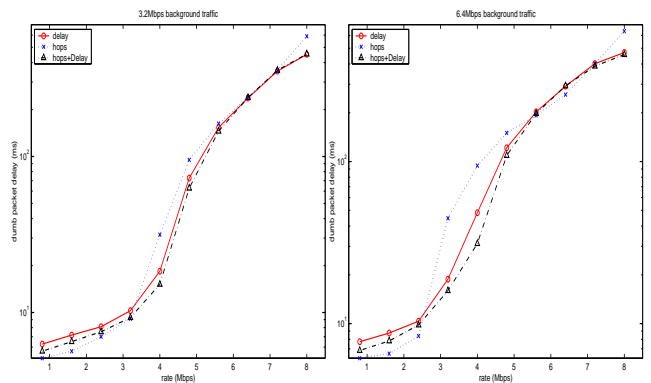


Figure 3: Total packet delay with 3.2Mbps and 6.4Mbps background traffic

computation overhead.

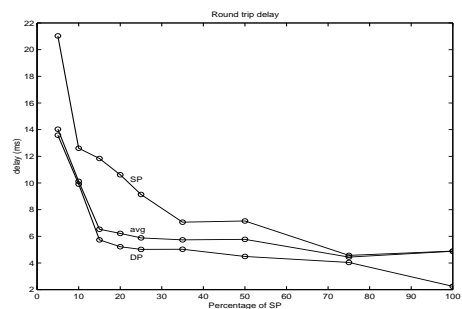


Figure 4: Average total delay for SPs (top) and DPs (bottom) packets, and average delay for all packets (center) as a function of the percentage of SPs

We also examine whether CPN spreads traffic among many paths. We see that as traffic rate goes from 100 packets/sec in Figure 5 to 1000 packets/sec in Figure 6, there is a more even distribution of traffic over a smaller number of better QoS paths.

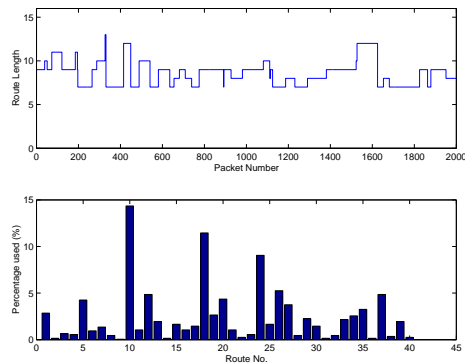


Figure 5: Usage of routes with low traffic rate and delay as the QoS goal

3.1 The 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 whether

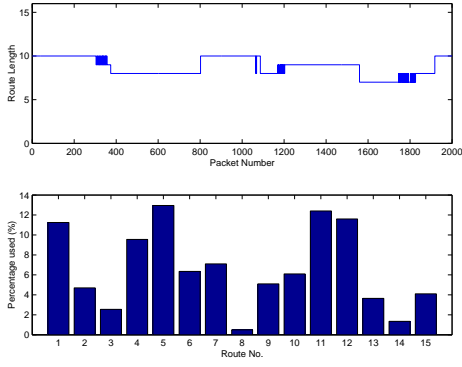


Figure 6: Usage of routes with high traffic rate and delay as the QoS goal

one should provide decisions as soon as they can be formulated and hope for the best. Thus the experiments described in the previous section refer to a situation where decisions are taken 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. In this section we will evaluate the effect of a more sophisticated mode of decision making, which uses the underlying RNN and RL, but also uses a further optimization step, 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, for each of the destinations that the source communicates with. Now suppose that two *distinct* paths $SxIyD$ and $SuIvD$ connect source S to destination D through the same intermediate node I , and that these paths have been discovered by SPs and that they provide QoS metrics $G(SxIyD)$ and $G(SuIvD)$. Obviously then $SuIyD$ and $SxIvD$ are also valid paths. If they have not been discovered by SPs, we can infer their estimated QoS (not the measured QoS) using the data brought back from the SPs, and we denote this 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 (e.g. smallest delay, or smallest loss, or best value of some other metric of interest). Then we propose to use the hitherto untested path $SuIyD$ to forward DPs, rather than the best of the two paths that were actually tested. Note that this resembles the “crossover operation” in a genetic algorithm, where in this case the path is the genotype and we create new paths by splicing existing paths which have a common intermediate node like construct for network routing [31]. Experiments were run with 1024 byte DPs, and the DP rate was varied between 100 packets/sec to 800 packets/sec. Without any background traffic we observed that this additional optimization provided a small improvement both in the packet loss rate and in the average packet delay. However when a significant amount of background traffic was added to each link, we see in Figure 7 that even with relatively low DP rate exceeding a certain value (300 packets/sec), the original CPN algorithm performs significantly better. Thus the slower optimization process using older data introduced on top of CPN by the genetic algorithm-like approach cannot respond fast enough to changing network conditions and is actually detrimental to QoS.

3.2 Adaptation to failures

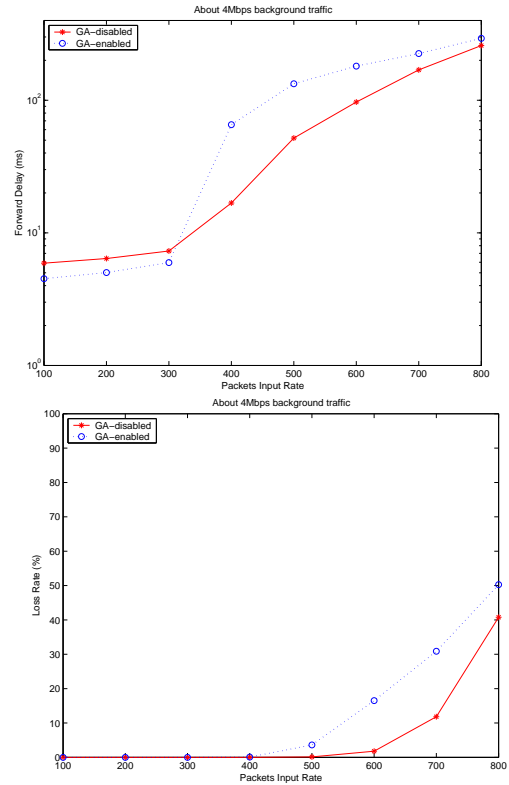


Figure 7: Average delay and loss rate with 4Mbps background traffic

Here we show that CPN can also provide some level of stable operation in the presence of worm-like failures. In experiments conducted in the 46-node test-bed of Figure 8 running CPN, failures begin at a given node which then randomly causes other nodes to fail. A node that fails is unable to forward payload traffic, but causes its neighbours to fail at a rate that we vary from 0.02 to 0.1 nodes/sec. Node failure is followed by recovery, representing cleaning and patching, at a constant rate of 1 node/second. Figure 9 reports the measured average delay, jitter and packet loss over ten successive experiments, for the User 1 that is sending 7MB/sec CBR traffic from Node 6 to Node 24. We see that CPN’s route adaptation allows QoS to remain stable over a wide range of failure rates, but degrades significantly at high failure rates.

3.3 Ant Colony algorithms

Ant Colony Routing Algorithms [9, 26, 16, 29] are in many ways similar to CPN, but differ in the manner that they use RL and in other details. They are inspired by the use of pheromones by “real” ants to mark their paths and communicate to each other the way to find the location of sources of food. In the network case, packets replace the ants, and nodes or links represent locations. Packets will move toward their destination based on preferring branches and paths with stronger markings. When a packet reaches its destination, a corresponding “marking” packet will head back to the source following the successful path in reverse or quasi-reverse order (or again following the “strongly marked trail”, and strengthen the marking at each link and node that it

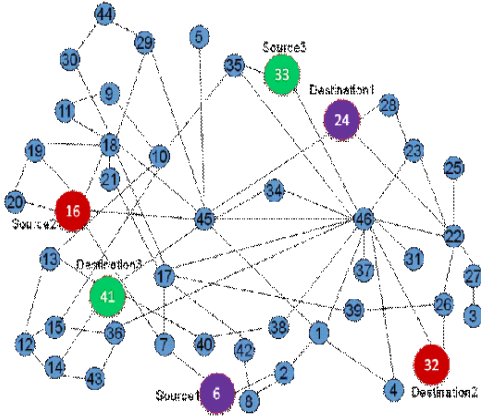


Figure 8: 46 node CPN test-bed with failures

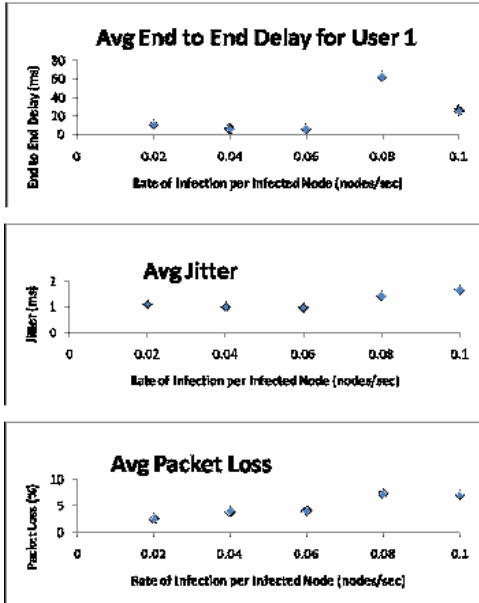


Figure 9: Stable QoS in the presence of failures

visits. These numerical or symbolic values will degrade with time (“forgetfulness”) if they are not reinforced by the passage of other packets. The routing algorithm will be initiated by a random search until the destination node is found, and then the discovered path(s) will be reinforced by the returning packets. Note that the theoretical results summarised in Section 2 also indicate that the destination can be found in finite time even when no initial path information is available. The returning packet from the destination is similar to the ACK packet of CPN. While CPN typically separates the search role via SPs from the payload role via DPs, Ant Colony routing algorithms will typically use payload packets for both purposes; in early versions of CPN, SPs also carried payload, but this approach was abandoned because it was found that the resulting QoS was not as good as when DPs specialised in payload conveyance while SPs were restricted to search. Thus CPN will use more packets, since SPs are constantly sent forward to accomplish the search function and represent a constant fraction (e.g. 10%) of to-

tal payload. Furthermore, Ant Colony algorithms do not use a neural network (as does CPN) to store the RL information inside a given node. Because DPs are source routed CPN does not require route computation to be carried out for payload packets, so that in CPN routers only carry out route computation for SPs which represent a small fraction of total traffic in the network, as well as for ACKs which are also source routed, while Ant Colony algorithms typically require route computation to be carried out for all packets. Clearly though, Ant Colony algorithms are better adapted to networks which may be frequently disrupted, since all packets are in a sense autonomous, while in CPN if a DP’s path is disrupted, the packet will have to be retransmitted at the source thanks to information brought back by a subsequent SP which finds another path. Thus CPN will be on the average better at forwarding packets, but slower in responding to changes in topology.

4. OSCILLATIONS IN A SAN

Route oscillations [2] in networks have been observed since the early days of the ARPANet, causing performance to suffer under medium to high load conditions. They can occur if load-sensitive routing metrics are used, and [27] frequent route switching can lead to reduced performance. Internet routing instability can increase packet losses and delay in network convergence, and cause overhead (memory, CPU, etc.). In [14] route instability is discussed in the presence of network congestion, and seen to cause increased packet loss and latency. The negative impact on TCP performance of disturbances at the routing layer is examined in [17] who consider routing oscillations due to the self-load effect and link failures. TCP is affected by these events because of how it responds to asymmetric paths (i.e. the path that its data packets take is different from those of its ACKs), and out-of-order packet delivery. Routing oscillations in overlay networks are analyzed in [25]. *Self-load* occurs when the transfer of flows to a lightly loaded path creates path overload, leading to subsequent transfer of the same flows to other paths, which then in turn may become overloaded, leading to switching of flows between paths. Also different routers’ measurement windows can overlap, leading to oscillations when flows interfere and prevent the network from stabilizing. Detailed experiments regarding oscillations in self-aware networks have been reported in [37], allowing us to address question (C) of Section 1. We therefore examine whether (i) frequent oscillations can occur in a SAN, (ii) whether these can be easily mitigated or reduced, and (iii) why oscillations are detrimental to network performance. Concerning point (iii), we note that each time a path is selected, traffic is indeed forwarded along that path until the path is changed. Thus while the path is being used, useful work is done and packets are delivered to the destination. When a path switch occurs, packets that are already engaged in the path will continue flowing to the destination. This certainly occurs in CPN, since each DP stores the path it is following, and the change in path decided at the source only affects subsequent packets and not those which are already engaged in the current path. As we see in this section, oscillations can have an adverse effect on performance via packet “desequencing” [12]: when a path is switched, a subsequently sent packet from the same flow may arrive to destination before its predecessor(s). The output node will then have to delay incoming packets so as to reassemble

them before they are delivered. This has a negative effect on real-time applications such as voice and media, and can also result in packet loss due to the overflow from the output node's packet "resequencing" buffer. Concerning (ii), there are different ways to mitigate or reduce oscillations. For instance, the source node can decide to allow switching only if the QoS gain exceeds a significant threshold. Another simple approach is to require that each time a path is used, the usage must exceed a certain number of packets before switching can be considered. We now review experimental results [37] concerning the effect of oscillations on performance, and the techniques to mitigate these consequences that have been mentioned earlier.

The measurements reported were carried out on a testbed emulating the Swiss Education & Research Network (as of 2007) in Figure 10, but with off-the-shelf components and running the CPN self-aware network software in each router. The curves include 95% bars for the measurement values. The routers are Pentium IV-class machines with 4-port 10/100 Ethernet interfaces running Linux 2.6.15, where CPN is implemented as a loadable kernel module. Each link is full-duplex, and runs at 10Mb/s. 24 constant bit-rate flows, each of 1.66Mb/s were generated to offer just over 40Mb/s of DP traffic resulting (with the SP and ACK traffic) in a slightly overloaded system as in [27]. SP traffic is set at 10% of DPs, and each source sends traffic to the same destination, which has 4 ingress interfaces with 40Mb/s of available incoming bandwidth. Self-awareness was set to minimize delay as the QoS goal.

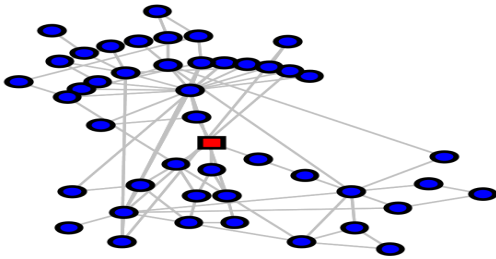


Figure 10: CPN testbed emulating the Swiss Ed. & Res. Net. The square node is the sink and 24 others are sources.

Figure 11 illustrates the effect of introducing a simple rule for limiting path switches: each time a source selects a new path which is identified as offering the smallest delay, the decision is only accepted with probability P (the switching probability). Thus if $P = 1$ all recommended path switches are used, while when $P = 0.001$ only once every one thousand times does the path actually switch paths. Thus the top curve shows that the switching probability affects *path oscillations*, which are patterns of switching starting from a given path and returning to it repeatedly: as the switching probability increases so does the rate at which the routes oscillate. The adverse effect of path switching on the packet drop rate at the output resequencing buffer is shown in the bottom figure. However, from Figure 12 we see that in order to achieve the desired effect of improving QoS (delay in this case) it actually suffices to accept switches with a low probability ($P = 0.01$ or a little higher). We therefore conclude that path switching does improve average delay (which is in fact why CPN attempts to switch paths), but that this

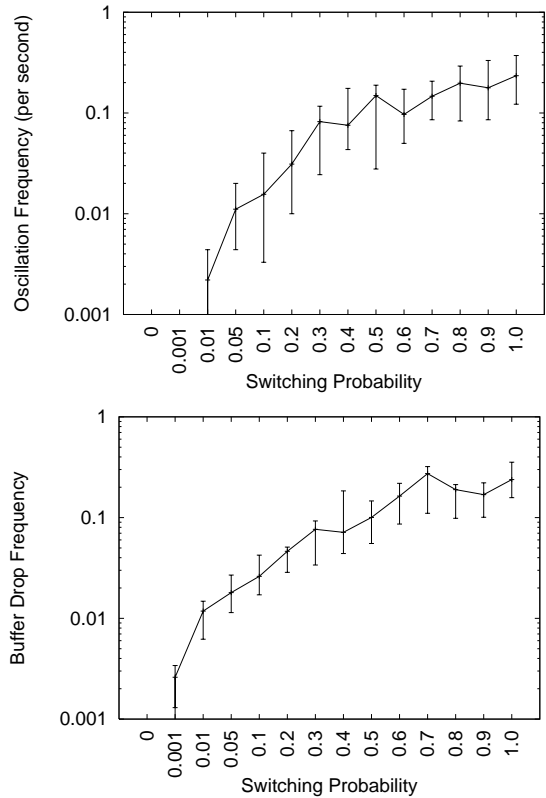


Figure 11: Oscillation Frequency (Top), Packet Drop Rate (Bottom) vs switching probability

comes at the cost of packet loss due to desequecing [1]. However, it does not appear difficult to mitigate for this by probabilistically limiting the switching, while retaining the benefits in terms of improving the QoS goal that the SAN is pursuing, which is packet delay in this case.

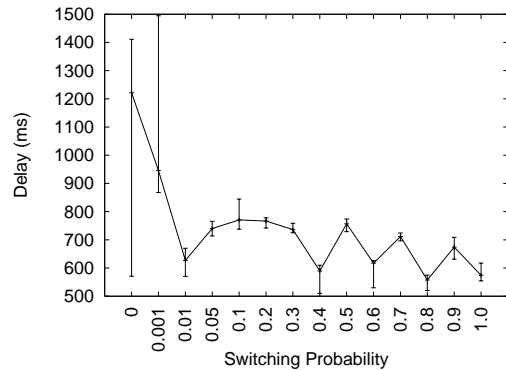


Figure 12: Avg. Packet Delay vs switching probability

Path switching and oscillations can also be limited by not allowing a switch unless the projected improvement in QoS exceeds some *threshold*. A small threshold will allow more frequent switches and hence potentially more oscillations, but a large threshold may worsen QoS. Figure 13 shows that if the threshold is small the observed delay is large, and as the threshold increases there is an improvement in

delay, but that packet delay increases again for even larger thresholds. The large delay for small threshold values indicates that switching is occurring based on “noise”, rather than on the gain that is expected. Figure 14 shows that an increasing threshold limits oscillations, but that this advantage levels out rapidly. Thus the QoS threshold can limit the negative effect of switching, while preserving the gains obtained from adaptivity and self-awareness.

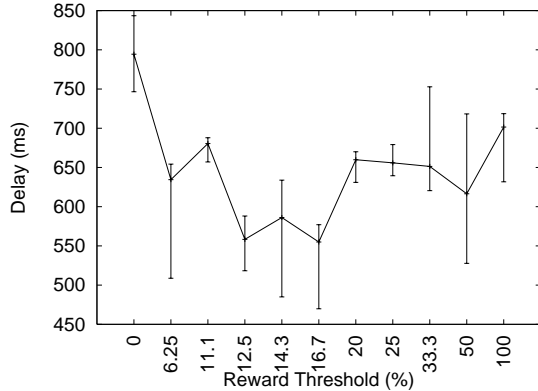


Figure 13: Avg. Packet Delay the threshold

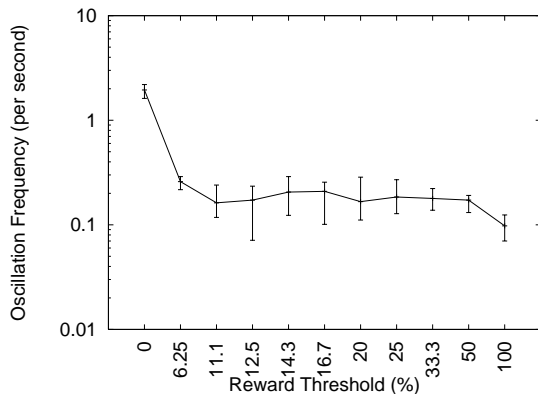


Figure 14: Rate of path oscillations vs threshold

5. CONCLUSIONS

We have presented an approach to self-aware networks where end users are allowed to explore the state of the network so as to find ways to best meet their individual communication needs. The paper has focused on the path finding and routing functions of a network. We have attempted to answer a few of the questions concerning the feasibility of such networks by focusing on whether reliable communications can be conducted in largely unknown networks, on 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 the users, and on whether the user’s ability to adapt to changing circumstances in the network can reduce the consequences of network failures. The laboratory experiments reported in this paper relate to small 17, 25 and 46 node networks. The Internet is composed of hierarchically organised Autonomous Systems of

relatively small size, and one can imagine that routing inside and between Autonomous Systems can benefit from the techniques discussed in this paper. We feel that future research will have to be devoted to the investigation how such techniques can be integrated into existing systems, on how they can withstand malicious behaviour of users and network nodes, and on whether such systems can adequately support mobile end users.

6. REFERENCES

- [1] F. Baccelli, E. Gelenbe and B. Plateau. An end to end approach to the resequencing problem. *J. ACM* 31(3):474-485, 1984.
- [2] A. Khanna and J. Zinky. The revised arpanet routing metric. *SIGCOMM Rev.* 19(4):45-56, 1989.
- [3] C.H. Papadimitriou and M. Yannakakis “Shortest paths without a map”, *Th. Comp. Sci.*, 84(1): 127-150, 1991.
- [4] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Proc. NIPS*, 671-678, 1993.
- [5] E. Gelenbe. Learning in the Recurrent Random Neural Network. *Neural Computation*, 5(1):154-164, 1993.
- [6] G. Malkin. RIP Version 2. RFC 2453, 1998.
- [7] J. Moy. OSPF Version 2. RFC 2328, 1998.
- [8] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network* 12(6): 64-79, 1998.
- [9] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communication networks. *J. AI Res.* 9:317-365, 1998.
- [10] C. Labovitz, G.R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Trans. Net.*, 6(5):515-528, 1998.
- [11] D. Williams G. Apostolopoulos. QoS Routing Mechanisms and OSPF Extensions. RFC 2676, 1999.
- [12] Jon C. R. Bennett, Craig Partridge, and Nicholas Shectman. Packet reordering is not pathological network behavior. *IEEE/ACM Trans. Net.* 7(6):789-798, 1999.
- [13] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Comm. ACM* 43 (5): 551-558, 2000.
- [14] A. Shaikh, A. Varma, L. Kalampoukas and R. Dube. Routing stability in congested networks: experimentation and analysis. *Proc. SIGCOMM 2000* 163-174, ACM Press, New York.
- [15] M. Thorup and U. Zwick “Compact routing schemes”, *Proc. 13th Ann. ACM Symp. Par. Alg. & Arch.* 1-10, ACM Press, New York, 2001.
- [16] S. Koenig, B. K. Szymanski and Y. Liu. Efficient and Inefficient Ant Coverage Methods. *Ann. Math. & AI* 31(1-4):41-76, 2001.
- [17] U. Ranadive and D. Medhi. Some observations on the effect of route fluctuation and network link failure on TCP. In *Proc. 10th Intl. Conf. Comp. Comms. & Nets.* 460-467, 2001.
- [18] E. Gelenbe, R. Lent, and Z. Xu. Measurement and performance of a cognitive packet network. *J. Comp. Net.* 37: 691-791, 2001.
- [19] K. Kowalik and M. Collier. Should QoS routing algorithms prefer shortest paths? *Proc. IEEE Intl.*

- Conf. Comms.* 213–217, 2003.
- [20] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc on-demand distance vector (AODV) routing. RFC 3561, 2003.
- [21] E. Gelenbe, M. Gellman, R. Lent, P. Liu, and Pu Su. Autonomous smart routing for network QoS. In *Proc. First Intl. Conf. Auton. Comp.* 232–239, New York, 2004.
- [22] M. Gellman and P. Su. Using adaptive routing to achieve quality of service. *Performance Evaluation* 57(2): 105–119, 2004.
- [23] E. Gelenbe, R. Lent, and A. Nunez. Self-aware networks and QoS. *Proc. IEEE* 92(9): 1478–1489, 2004.
- [24] E. Gelenbe, and R. Lent. Power aware ad hoc cognitive packet networks *Ad Hoc Networks*, 2: 205–216, 2004.
- [25] R. Keralapura, C.-N. Chuah, N. Taft, and G. Iannaccone. Can coexisting overlays inadvertently step on each other? In *Proc. 13th IEEE Intl. Conf. Net. Protocols*, 2005.
- [26] F. Ducatelle, G. Di Caro and L. M. Gambardella. An analysis of the different components of the anthocnet routing algorithm. *ANTS Workshop*, 37–48, 2006.
- [27] R. Gao, C. Dovrolis and E. Zegura. Avoiding oscillations due to intelligent route control systems. *IEEE Infocom*, 2006.
- [28] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt and F. Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.* 1(2): 223–259, 2006.
- [29] G. Chen, J. Branch, and B.K Szymanski. A self-selection technique for flooding and routing in wireless ad-hoc networks. *J. Net. & Syst. Manag't.* 14(3):359–380, 2006.
- [30] Ö. Babaoglu, G. Canright, A. Deutsch, G. Di Caro, F. Ducatelle, L. M. Gambardella, N. Ganguly, M. Jelasity, R. Montemanni, A. Montresor and T. Urnes. Design patterns from biology for distributed computing. *ACM Trans. Auton. Adapt. Syst.* 1(1):26–66, 2006.
- [31] E. Gelenbe, P. Liu and J. Lainé. Genetic algorithms for route discovery. *IEEE Trans. SMC B* 36(6): 1247–1254, 2006.
- [32] E. Gelenbe and G. Loukas. A self-aware approach to denial of service defence. *J. Comp. Nets.*, 51(5):1299–1314, 2007.
- [33] G. G. Öke, G. Loukas, and E. Gelenbe. Detecting denial of service attacks with bayesian classifiers and the random neural network. *Proc. Fuzz-IEEE 2007*, pp. 1964–1969, July 2007.
- [34] P. Liu and E. Gelenbe “Recursive routing in the cognitive packet network”, *Proc. 3rd TridentCom Conf.*, 1–6, 2007.
- [35] E. Gelenbe. A diffusion model for packet travel time in a random multihop medium. *ACM Trans. Sensor Nets.*, 3(2):10, 2007.
- [36] E. Gelenbe, G. Sakellari and M. D’ Arienzo. Admission of QoS aware users in a smart network. *ACM Trans. Auton. Adapt. Syst.* 3(1), March 2008.
- [37] M. Gellman. Oscillations in self-aware networks. *Proc. Royal Soc. A* 464 (2096): 2169–2186, 2008.