Oscillations in self-aware networks

By Michael Gellman*

KBC Financial Products UK Limited, 111 Old Broad Street, London EC2N 1FP, UK

Adaptive routing is once again of interest owing to the possibility to couple online probing in networks with real-time dynamic and distributed control of paths and flows. Wireless networks, with their rapidly changing network conditions also create a need to revisit this issue. This paper uses measurements in a wired adaptive network test bed, the cognitive packet network (CPN), to investigate the pros and cons of adaptive routing. CPN routes packet flows through a store and forward network according to their quality of service (QoS) needs through an online distributed reinforcement learning mechanism. This paper investigates routing oscillations that occur due to the interaction of multiple flows and studies their effect on QoS in the context of CPN. Our results indicate that routing oscillations can be easily controlled by randomizing the route switching, and that from an overall QoS viewpoint increased switching can also lead to improved performance.

Keywords: oscillations; networks; engineering

1. Introduction

Routing oscillations (Gelenbe & Gellman 2007) are a networking phenomenon that have been observed since some of the earliest packet-switched networks (such as the ARPANET) where they caused performance to suffer under medium to high load (Khanna & Zinky 1989). These oscillations were due to the use of a load-sensitive routing metric that has led to a commonly taken-for-granted assumption among researchers that routing protocols should ensure that oscillations do not occur and furthermore, load-sensitive metrics should not be used in such a way that they could occur. We set out to test this assumption using an adaptive routing protocol, with some surprising results that question whether it still holds. Recent theoretical research (Gelenbe 2003, 2007) supports the idea that probabilistic routing can actually be used to improve overall network performance.

This research is conducted in the framework of autonomic communications (Gelenbe 2005; Dobson et al. 2006) which uses online network measurements to improve the quality of service (QoS) of the users and has diverse applications in both the civilian and the military domain (Ghanea-Hercock et al. 2007). The routing protocol in question is the cognitive packet network (CPN; Gelenbe et al. 2001, 2002, 2004c), which is a packet routing protocol that uses online measurement and reinforcement learning (RL) to adapt its routing to changing

* michael.gellman@kbcfp.com

One contribution of 6 to a Special Feature ‘Stochastic networks: from theory to practice’.
traffic or network conditions. It constantly probes paths in the network using smart packets, and selects the ‘best’ one for a given flow in a distributed manner based on a user-specified QoS metric. It has been experimentally shown in previous work (Gelenbe et al. 2004a; Gellman & Su 2004) that CPN can effectively offer best-effort QoS routing according to metrics that are selected for specific application’s needs. Being an adaptive routing protocol, we set out to question how routing oscillations impact performance and what methods exist towards controlling the rate at which routes oscillate.

In this paper, we challenge the assumption that routing oscillations always result in poor network performance. Using experiments conducted with CPN on a large networking test bed configured with real-world topology data, we show that increased switching does not always result in poor performance and can actually result in improved QoS for network flows. While an increase in route switching results in a corresponding increase in packet desequencing, we demonstrate two factors that function to trade-off between network performance (i.e. delay and loss) and desequencing.

A recently published study uses simulations (Gao et al. 2006) to examine routing oscillations and indicates that frequent route switching can lead to reduced performance. As a starting point for our study, we have conducted experiments on our CPN test bed using a certain number of scenarios that are quite similar to the ones simulated in Gao et al. (2006). Our results indicate that in simple cases our results appear to confirm the previously published simulation results. However, in a more complicated scenario with a large number of flows and stable background traffic, we discover different relationships between switching frequency and performance.

We begin our paper with an overview of the CPN protocol, focusing on its RL-based approach to routing (§2). We then demonstrate in §3 that CPN will suffer from routing oscillations due to the two causes outlined in Gao et al. (2006), namely the self-load effect where the impact of a flow on a path is not taken into account when probing, which can lead to permanent oscillations, and secondly when the measurement periods of multiple flows overlap causing them to switch in groups chasing perceived areas of better performance. While we do observe routing oscillations in CPN, in our larger scale experiments the results seem to contradict some of the conclusions of Gao et al. (2006) when the background traffic is stable. In §4, we show that CPN in effect can benefit from frequent route switching, yielding its best performance when it is allowed to switch whenever it finds a better performing route. This observation seems to indicate that myopic and selfish behaviour of users in a network, leading to frequent ‘changes of mind’ can under certain circumstances result in better performance for all parties concerned. This affirmation needs to be tempered when oscillations can have other side effects, such as packet desequencing for flows that are highly sensitive to sequence, such as real-time video or voice, or even Transmission Control Protocol (TCP) traffic (§4c).

(a) Related work

Routing oscillations are hardly a new phenomenon, and there have been many contributions to the subject in the field of networks. The ARPANET, one of the predecessors of the modern Internet was susceptible to routing
oscillations under heavy load (Khanna & Zinky 1989) that led to poor performance. The effects of Internet routing instability were discussed in Labovitz et al. (1998, 1999), which summarized their impact by saying that ‘Overall, instability has three primary effects: increased packet loss to unstable destination, delays in the time for network convergence, and additional overhead (memory, CPU, etc.) within the Internet infrastructure.’ A discussion of the negative impact of synchronization of periodic routing messages is found in Floyd & Jacobson (1994) who show that even if traffic sources are initially not synchronized, they can abruptly become synchronized. In Shaikh et al. (2000) route instability is discussed in the presence of network congestion, causing increased packet loss and latency.

The negative impact upon TCP performance of disturbances at the routing layer is examined in Ranadive & Medhi (2001) which considers both the case of routing oscillations due to the self-load effect and link failures. TCP is affected by these events owing to how it responds to asymmetric paths (i.e. the path that its data packets take is different from those of its acknowledgement, ACKs) and out-of-order packet delivery.

Routing oscillations in overlay networks are analysed in Keralapura et al. (2005). They deal with the negative consequences of synchronization of multiple overlay networks, and analytically determine an upper bound for the length of a period of synchronization.

Multiprotocol Label Switching Protocol (MPLS) is an industry standard routing protocol and should normally have been evaluated with respect to oscillations and similar phenomena by industry. Such information does not seem to be available in the open literature, and we do not have access to platforms where an external user can make direct measurements. In any case, studying MPLS is beyond the scope of this work.

2. The CPN

In this section, we briefly present an overview of the CPN routing protocol as described in Gelenbe et al. (2004c), focusing on the online path searching algorithm and on the manner with which it allows edge routers to source route the traffic. A more detailed description can be found in Gelenbe et al. (2001, 2004c).

The CPN has been shown to have applicability to a variety of networking problems beyond the scope of pure QoS routing. An admission control algorithm using the information that is gathered by the CPN was proposed in D’Arienzo et al. (2006). Another body of work (Gelenbe et al. 2004b, 2005; Gelenbe & Loukas 2007) has shown that CPN can be used as a way for constructing an online dynamic defence for denial of service attacks. Other ongoing research efforts include hardware implementation (Hey et al. 2005) and applications in the wireless ad hoc networks domain (Gelenbe & Lent 2004).

As opposed to routing protocols that rely on pre-computing routes to all possible destinations based on a fixed criterion, such as the choice of paths that are of minimum length, CPN works in an on-demand fashion, dividing the routing task among the following three distinct classes of packets: smart (SP); ACK; and dumb (DP) packets. The SP’s role is simply to discover paths, while
DPs carry payload and are source routed. ACKs bring back information that SPs have discovered so that both intermediate routers and the source router can update the information that they have.

In CPN, communication sources (which are supported by edge routers) create SPs at regular intervals, the final destination of which is the address of the edge router which supports the communication user’s destination. SPs contain the destination node identifier and the QoS criterion that they seek, and are routed in a distributed manner without any form of centralized control. Each intermediate node decides on the next node for a SP based on the final destination address and the QoS criterion. The intermediate router’s decision is taken by consulting a neural network, the weights of which have been updated with previous measurement data on routes to that destination for the same QoS criterion, using a RL algorithm described in Gelenbe et al. (2001). A fraction of SPs are routed at random each intermediate router so that a wider range of paths may be discovered. SPs collect both the router’s identity and data relevant to QoS at each router visited.

For instance, if the relevant QoS metric is ‘delay’ the SP will collect the local (not global) time at which it entered the node, so that if and when it reaches its final destination, it will contain a list of all nodes visited together with the times of the corresponding visits. The destination will then generate an ACK packet, to be sent back to the source along the reverse of the path that the SP followed so that all intermediate nodes may be visited. The ACK packet would contain all the information stored in the SP so that when the ACK visits a node, the current local time minus the time when the SP visited that same node in that order can be used as the ‘penalty’ of the RL algorithm (see Gelenbe et al. (2004c) for details of the RL algorithm).

Each ACK that arrives at the source contains a complete route from the source to the destination, along with its QoS; for instance, in the above example when QoS is delay, the ACK will have the time at which the SP was generated at the source, and the current time when the ACK is received by the source provides an estimate of the round-trip delay from source to destination when that path is used. Paths are stored at the source, and allow it to select among all recently tested paths the one that is best and which it will use to forward its DPs. The source can revise its decision more or less frequently based on the ACKs that it receives, and based on the difference between the best QoS path, and the one it may be currently using. Thus, the source node is free to reduce path switching when the perceived improvement in QoS does not exceed a given threshold.

(a) The CPN RL algorithm

In this section, we will detail the RL algorithm that is implemented in CPN. Each CPN router contains a (fully recurrent) random neural network (RNN; Gelenbe 1990; Gelenbe & Hussain 2002) whose weights are updated using RL. The RNN functions as a decision maker where each neuron corresponds to the choice of one of the possible output links of the router, and at any time the most excited neuron is selected to indicate the output port that the current SP should take.¹

¹ As indicated earlier, with a small probability (of 5% in the measurements that we report), instead of using the RNN, the SP will simply select an output port at random to allow it to increase its chance of discovering new routes or to probe old routes whose QoS may have improved since they were last tested.
The RNN weights are updated whenever an ACK visits the router. From the QoS information (loss, delay, etc.) brought back by an ACK, the relevant information is stored in the router that it is currently visiting and a new value of the ‘reward’ $R$ is calculated. Successive values of $R$ are denoted as $R_l, l=1, 2, \ldots$. These are used to compute a threshold or historical value of the reward

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

where $\alpha$ is some constant close to (but less than) 1.

Thus, $T_{l-1}$ represents our expectation of reward so that if $R_l \geq T_{l-1}$, we reward the previous decision by increasing the excitatory weights leading to its corresponding neuron, and slightly increase the inhibitory weights of the other neurons. On the other hand, if the reward is below the threshold $R_l \leq T_{l-1}$, then we punish the decision by increasing its inhibitory weights, and also slightly increasing the excitatory weights of the other neurons so that the other choices for output ports can compete more effectively in the next round of decision making.

As an example, in the case of the QoS metric delay, $R$ would be calculated as

$$R = \frac{1}{D},$$

where $D$ is the round-trip delay to the destination calculated from the arrival of the most recent ACK.

(b) Routes at the source

Each ACK packet that arrives back to the source as a result of an SP reaching its destination contains a functional loop-free route, along with its QoS data. Although an SP may in fact loop (i.e. visit the same node more than once), it is easy to construct a corresponding loop-free route by (for instance) deleting all appearances of a given node except for the last one from the list.

In the original CPN algorithm, a route brought back by the ACK would be immediately used, as it contains the most up-to-date snapshot of the network state. However, Gellman & Liu (2006) showed that this could lead to poor data packet performance due to the random component of SP exploration. Thus, we now switch routes only when the reward of the new route is greater than that of the current route. ACK packets are also generated at the destination when a DP reaches it (in which case it is called a DACK), and these can also serve to refresh the QoS information related to the paths that are stored at the source.

Complementing this strategy, we incorporate two additional parameters that influence when the source switches to a new route. The first of these enhancements (also described in Gao et al. (2006) though not explicitly used to control oscillations) we call a reward threshold, where we specify that a newly discovered route must be better than the active route by a fixed percentage in order for a switch to occur. This is a similar approach to the one used in Gelenbe et al. (2006a,b) where a group of routes within a given quality margin were all considered as viable routing options, and a round-robin policy was used for selecting a route. The second control we use is a fixed switching probability (FSP) which was also used in Gao et al. (2006). When a new route is discovered which

Oscillations in self-aware networks

is better by the reward threshold than the active one then with some probability it will cause a switch. As an example, in traditional CPN the FSP is equal to 1, whereas if no switching were allowed, the FSP would be 0.

3. Oscillations in intelligent route control systems

Routing oscillations in intelligent route control (IRC) systems have been explored in Gao et al. (2006) through simulation. They define IRC systems as aiming ‘to optimize the cost and performance of outgoing traffic, based on measurement-driven dynamic path switching techniques’. The CPN also carries out path switching as needed for QoS purposes based on dynamic measurement and control, and (as previously described) it constantly explores the network state using SPs.

In Gao et al. (2006), two causes of the oscillations are identified. The first is what they term the self-load effect, where the impact of a flow on the metric that is measured is not taken into account in the probing process so that once a path whose load appears to be light is actually used, the resulting load is significantly higher than the one that was previously observed. This first phenomenon has also been studied in D’Arienzo et al. (2006) with respect to the use of CPN in admission control algorithms. Of more difficulty is the second cause they outline: that routers’ measurement windows can overlap, leading to persistent switching as flows interfere with each other, preventing the network from stabilizing, and causing uneven use and poor QoS. In this section, we provide measurements to show that CPN can also suffer from both of these causes of path oscillations.

A solution proposed in Gao et al. (2006) is to use knowledge of a path’s available bandwidth to help reduce or eliminate oscillations. This is because, if an estimate of a flow’s bandwidth and the available bandwidth of a path are known, then the impact of switching a flow to a path can be estimated before switching to the path. However, very often the measuring of available bandwidth occurs at longer time scales than the routing process itself. A recent comparison of tools that may be used in this process (Shriram et al. 2005) found that the two most accurate tools had running times of 5.5 s (pathchirp) and between 7 and 22 s (pathload). Another more recent study (Shriram & Kaur 2006) concluded that a time of 5 s was needed for accurate bandwidth estimation. These times indicate that it would be difficult to satisfy the need to have routing intervals that are of similar or greater length than the measurement time (i.e. \(T_m = T_r\) from Gao et al. (2006) when \(T_r = 1\) s).

(a) CPN and the self-load effect

We have examined the extent to which CPN suffers from the self-load effect and have tried to match as closely as possible the experiment described in Gao et al. (2006, §III.A). We have set up a simple four-router square topology with one source \(S\) and one destination \(D\) on opposite corners (i.e. there are two paths to the destination, \(p_1\) and \(p_2\)). Four flows were started where three are statically routed using the Internet Protocol (IP), with two along path \(p_2\) and one on path \(p_1\). The fourth flow uses CPN with delay as its routing metric so that it generates SPs that are constantly searching for paths with lower end-to-end delay. The
traffic generated by each flow is constant bit rate (CBR) with rate equal to 25% of the capacity of a path (2.5 Mb s$^{-1}$). In order to compare the effect of routing oscillations on the performance of the CPN flow, we used two different configurations. In the first, CPN was configured to optimize delay while in the second, path switching was disabled and CPN was forced to stay on the optimal path $p_2$ (labelled good path). The results are shown in figure 1, where the round-trip packet delay is a sliding window average over 30 s of measurements per packet. While the time scales of the two datasets differ by an order of magnitude.

Figure 1. The self-load effect path switching (above graph) shows that oscillations are occurring, and the resulting QoS (loss in the above set of measurements and delay in the bottom set) shows the negative impact that they can have on performance (a) adapted from Gao et al. (2006; solid line, loss-delay; dashed line, avail-BW) and (b) CPN results (upper graph, delay; lower graph, good path).
e.g. CPN switches paths on the order of hundreds of milliseconds, while their simulations take seconds), we clearly see that CPN also suffers from the self-load effect, to the detriment of performance.

(b) Synchronization-caused oscillations

In addition to the self-load effect, routing oscillations can be caused as a result of multiple flows in the network making synchronized decisions or decisions are made quasi-concurrently.

To determine whether CPN can also suffer from oscillations due to concurrent decisions, we conducted an experiment using 10 CPN flows with the topology shown in figure 2. We artificially extended the measurement period within CPN to be 1 s and started each of the flows approximately 0.1 s apart so that their initial, and presumably subsequent, measurement periods would overlap. In addition to the CPN flows, there are also 10 IP flows (five per path) which function as background traffic. Oscillations caused by synchronization are shown in figure 3. Their period is close to 2 s, matching well the simulations in Gao et al.

Figure 2. Topology for the experiments in §3b. There are 10 constant bit rate (CBR) sources all with the same destination.

Figure 3. Oscillations due to synchronization (a) adapted from Gao et al. (2006) and (b) CPN results. Solid line, path 1; dotted line, path 2.
(2006) where the decision time is also 1 s. Of course, these CPN oscillations are indeed somewhat artificial because the observation and decision times in CPN tend to be much shorter (of the order of 50–100 ms) and this kind of synchronous behaviour may be difficult to observe in practice.

4. Oscillations under stationary load

After showing that CPN can suffer from similar oscillations as have been seen in other adaptive networks, we wanted to scale up our analysis and conduct experiments using a realistic environment. Here we could observe routing oscillations and their impact on the performance of a large number of flows. In this section, we experimentally investigate the assumption that oscillations result in poor performance, with surprising results.

(a) Experimental set-up

Our networking test bed consists of 46 Pentium IV-class machines, each equipped with one (or more) 4-port 10/100 Ethernet interfaces. The operating system is Linux v. 2.6.15, where CPN is implemented as a loadable kernel module. Each link is full duplex and is configured to run at 10 Mb s⁻¹. The topology that we use is based on the information we received about the Swiss Education & Research Network depicted in figure 4.
Our experiments use 24 constant bit-rate flows, each of 1.66 Mb s\(^{-1}\), which makes for (just above) 40 Mb s\(^{-1}\) of application data (not including SP and ACK overhead). This yields similar parameters to the 105% of capacity case studied in Gao et al. (2006). Each client sends traffic to the same destination, which has four ingress interfaces that provide it with 40 Mb s\(^{-1}\) of available incoming bandwidth. For each dumb packet sent, the probability of generating a smart packet is 0.1. In order to estimate the impact of unresponsive traffic, we configured this probability to be zero for two of the flows (this matches the 90% case in Gao et al. 2006). Each CPN flow is configured to optimize delay and each individual experiment lasted 15 min (table 1).

Our experiments use 24 constant bit-rate flows, each of 1.66 Mb s\(^{-1}\), which makes for (just above) 40 Mb s\(^{-1}\) of application data (not including SP and ACK overhead). This yields similar parameters to the 105% of capacity case studied in Gao et al. (2006). Each client sends traffic to the same destination, which has four ingress interfaces that provide it with 40 Mb s\(^{-1}\) of available incoming bandwidth. For each dumb packet sent, the probability of generating a smart packet is 0.1. In order to estimate the impact of unresponsive traffic, we configured this probability to be zero for two of the flows (this matches the 90% case in Gao et al. 2006). Each CPN flow is configured to optimize delay and each individual experiment lasted 15 min (table 1).

### Table 1. Experimental parameters.

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>flows</td>
<td>24</td>
</tr>
<tr>
<td>rate per flow</td>
<td>1.66 Mb s(^{-1})</td>
</tr>
<tr>
<td>ingress links at destination</td>
<td>4</td>
</tr>
<tr>
<td>available ingress bandwidth</td>
<td>40 Mb s(^{-1})</td>
</tr>
<tr>
<td>smart packet</td>
<td>10%</td>
</tr>
<tr>
<td>experiment length</td>
<td>15 min</td>
</tr>
<tr>
<td>reward threshold</td>
<td>12.5%</td>
</tr>
</tbody>
</table>

Figure 5. Comparing (a) simulated and (b) experimental performance. While in the simulation increasing the switching probability results in poorer performance, the opposite trend is observed in our experiments. (a) Simulated performance while varying the switching probability (from Gao et al. 2006; solid line, 105% cap. 90% IRC; long-dashed line, 105% cap. 50% IRC; short-dashed line, 125% cap. 90% IRC; dotted line, 125% cap. 50% IRC) and (b) CPN’s performance while increasing switching. The average delay of each client is computed individually, and the median is plotted in the graph. Error bars indicate the first and third quartile.

(b) **Impact of switching on performance**

In order to study the extent to which CPN’s performance is affected by routing oscillations, we began by experimenting with the FSP parameter. The most striking difference between our experiments and the simulation results of Gao et al. (2006) is shown in figure 5. In the simulations, as the switching
probability increases, the performance of the system quickly decreases. The exact opposite trend is observed in CPN. When the switching probability is zero (i.e. the path of every flow is chosen at random at the beginning of the experiment, and then never switched), the delay is at its highest. As CPN is allowed more freedom to switch routes, it is able to better optimize its performance, yielding a median value half of that obtained without switching.

(i) Differences between experiments and simulations

Despite our efforts to match the experimental conditions to that of the simulator described in Gao et al. (2006), we highlight the following items that could account for the differences we observe.

(i) CPN probing occurs much more rapidly than that reported in the simulations. Indeed, any information gathering scheme must sample the environment at a high rate in order to obtain an accurate picture of what is going on. In the simulations, one measurement period consists of 10 samples each of which lasts for 100 ms, and an average over the 10 samples is used as an estimate of the path’s performance. Our system uses exponential averaging over a relatively short memory span (eight samples on average) at each hop in the RL algorithm, and samples each node of the network roughly every 1 ms, which is a factor of 100 times faster.

(ii) The RL algorithm used by CPN differs from the straightforward rules used by the IRC flows in Gao et al. (2006)—the sophistication of the RL algorithm may allow CPN to benefit from increased switching.

(iii) CPN uses delay as its QoS metric in order to adapt towards better performance, whereas the simulation uses bandwidth.

(iv) Finally, the simulations in Gao et al. (2006) only include the effect of the ingress links at the destination node, while our measurements cover a more realistic and complex network topology that includes the effects of possible bottlenecks and capacity sharing throughout the network.

In addition to these more conceptual differences, there are also some other differences at the experimental level that may impact our results.

(i) It uses a fractional Gaussian noise (FGN) model for traffic. At each 100 ms, the rate of a flow changes according to this model which attempts to reproduce the self-similar traffic that may be seen in a backbone network. We do not have a FGN traffic generator and used constant bit rate traffic instead.

(ii) It simulates 100 sources each sending out one flow, while we conduct measurements on the test bed with 24 connections. This is due to resource limitations.

While these two factors could impact our results, it is our feeling that they are not enough to account for the different performance trends that we have observed.

We wanted to confirm that the increase in the switching probability corresponded to an increase in not only switching but also oscillations. We defined an oscillation as a special pattern of route switching where a route is used, then...
another and then back to the original route. For instance, if route $A$ is used, followed by route $B$ and then by route $A$ again, we define that as a single oscillation. The rate of these occurrences over the experiment length is plotted in figure 6. As the switching probability increases so does the rate at which the routes oscillate.

When taking figures 5 and 6 together, we can see some interesting relationships. The difference in performance between probabilities 0.01 and 1 is not very dramatic, yet their rate of oscillation differs by a factor of 100. Thus, our results are not indicating that oscillations are necessary for good performance—rather we are saying that they do not degrade performance in our network.

(c) Packet desequencing

One of the disadvantages to frequent path switching is that it can cause packets to arrive at the destination in a different order from which they were generated. This desequencing\(^2\) negatively impacts the performance of many different types of traffic. The negative impact of reordering on a TCP connection is discussed at length in Bennett et al. (1999), and it is an ongoing research activity to design a TCP implementation that can withstand desequencing without suffering a performance penalty (Bohacek et al. 2006). It is also well known that real-time traffic such as Voice over IP needs its packets to be received in order, or else they will be buffered, increasing the memory and processing requirements at the destination, and, in cases of excessive desequencing, it leads to dropped packets and poor performance.

\(^2\)This has also been referred to as packet reordering and out-of-sequence packets in the literature. We use the term desequencing to reflect the fact that we are measuring the degree to which the arriving packets are out of order.
Owing to the negative impact on application performance, we measured the amount of desequencing as a result of increasing the switching probability. We make the assumption that our client streams each represent a single flow of real-time traffic. While in our traces, the application has not used a monotonically increasing sequence number, we can deduce the order in which each packet was generated by looking at the $id$ field, which is the time stamp at the source when the packet was originally sent. By considering the sequence of received ids, we can reconstruct the order in which the packets were generated, and, in turn, the amount to which it became desequenced by the network.

In order to quantify the level of desequencing, and in keeping with our focus on real-time traffic, our metric use a reordering buffer that acts to store out-of-order packets. When an out-of-order packet arrives at the destination, it is put into a buffer until it can be freed by the arrival of the expected sequence number. If an attempt is made to add an arriving packet to a full buffer, then it is assumed that the expected packet has been lost, and the packet with the lowest sequence number in the reorder buffer becomes the new expected packet. This acts to time-out packets that arrive excessively late. The packets arriving with sequence number less than the expected one are simply dropped. The reorder buffer has the ability to smooth out small amounts of desequencing, but under higher levels it can lead to packet drops. This approach is similar to the reorder buffer density metric proposed in Banka et al. (2002), except that we are concerned with the number of packets dropped by the buffer—not its size distribution.

For a reorder buffer of size 10, we obtained the drop frequencies in figure 7. As the switching frequency increases, the drop rate quickly exceeds the tolerable limits of many applications.

Figure 7. Impact of switching on desequencing. The number of packets that would be dropped by a reorder buffer increases with the switching rate.

---

3 Any arriving packet having a sequence number greater than the expected packet is out of order.

So far, we have demonstrated the impact that the switching probability has on performance, both in terms of delay and desequencing. Another factor that has a large impact on CPN’s ability to select the best-performing routes is the reward threshold that has been introduced in §2b (which we refer to as simply as threshold for the rest of the paper), which is how much better the estimated reward for a newly discovered route must be than the active route in order to result in a route switch. We express the threshold in terms of a percentage; thus, the scenario where a new route would have to be twice as good as the current route corresponds to a threshold value of 100%. Here, we discuss the impact that this threshold has on both performance and oscillations.

The threshold serves as a parameter which can be tuned to make CPN more or less sensitive to improvements in route quality. If the value is too low, any improvement in route quality, no matter how small, will result in a route change. This can lead to unnecessary route switches due to, for instance, the self-load effect where the smart packets of a flow do not impose the entire load of the flow on a path, thereby viewing improved levels of QoS that are not experienced by the flow after it switches, subsequently leading to further switches and high rates of oscillation. Similarly, if the threshold is too high, CPN will not be sensitive enough and will not take advantage of routes that offer improved performance. Thus reduced switching may also lead to reduced performance. Both of these phenomena are observed in figure 8. There appears to be a definite value of the relative threshold, in this case around 16.7% of the current value, which provides better performance than with greater or smaller values.

When figure 8 is considered along with the rate of oscillations in figure 9 we can confirm our analysis above, but also draw a stronger relationship between oscillations and performance. When the threshold is zero the oscillation rate is
nearly two times per second, a factor of 10 increase over all the other switching values. Similarly, at the highest threshold value, the oscillation rate is at its lowest. It is when we analyse these two graphs in tandem, however, that we see some interesting trends. First of all, for oscillation rates that are very nearly identical, we see significant performance differences. This tells us that the rate of oscillation is not by itself enough to impact performance, and that there are other factors at work, corroborating our analysis of the results in §2.

5. Summary

This paper has examined routing oscillations and their impact on performance, based on experiments on an adaptive network test bed that uses the CPN routing protocol. We first confirmed that CPN indeed suffers from routing oscillations as a result of different factors, similar to other real networks and results that have been observed by others in simulation experiments. We then turned our attention to studying the impact of routing oscillations on performance. The test-bed network was configured as realistically as possible given our resource (total number of nodes and links) constraints. Our measurements indicated that the CPN routing protocol provided somewhat different results from those that we would have expected based on long-held views (dating back to the ARPANET), that path adaptation will lead to oscillations and that this will result in poor performance under medium to high load. Our results indicate that routing oscillations do not severely degrade performance as would be expected; rather we show that even when they are present we can still obtain high performance.

We also examined how oscillations can be controlled in the network and tested two different parameters that largely impact the rate at which oscillations are observed. In particular, we studied the use of probabilistic path switching which
can be used both to make path switching more asynchronous, and to vary the rate at which switching decisions are made. We also examined the value of introducing a decision threshold which will only allow path switching if the gain expected from switching exceeds a certain minimal value. Both of these control schemes are easy to implement and provide an effective way to limit oscillations and their negative consequences.

More generally, we feel that this paper brings into question whether some long-held assumptions in the networking community regarding the viability of load-sensitive routing metrics are applicable for new classes of routing protocols which are based on self-monitoring and adaptation such as CPN.

This research was supported by the Engineering and Physical Sciences Research Council under grant no. GR/S52360/01 within the project entitled ‘Self-Aware Networks and Quality of Service’.

References


Oscillations in self-aware networks


