[15] L.P. Kaebling, M. L. Littman, and A.W Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[16] I. Matta and A. Udaya Shankar. Type-of-service routing in datagram delivery systems. *IEEE Journal On Selected Areas In Communications*, 13(8):1411–1425, october 1995.

[17] R Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunication networks. *Adaptive Behavior*, 5(2):169–207, 1996.

[18] D. Subramanian, P. Druschel, and J. Chen. Ants and reinforcement learning: A case study in routing in dynamic networks. In *Proceeding of IJCAI-97, International Joint Conference on Artificial Intelligence*, pages 832–838, Menlo Park, CA, 1997. Morgan Kaufmann.

[19] A. S. Tannenbaum. *Computer Networks*. Prentice-Hall, Inc, Upper Saddle River, NJ, 1996.

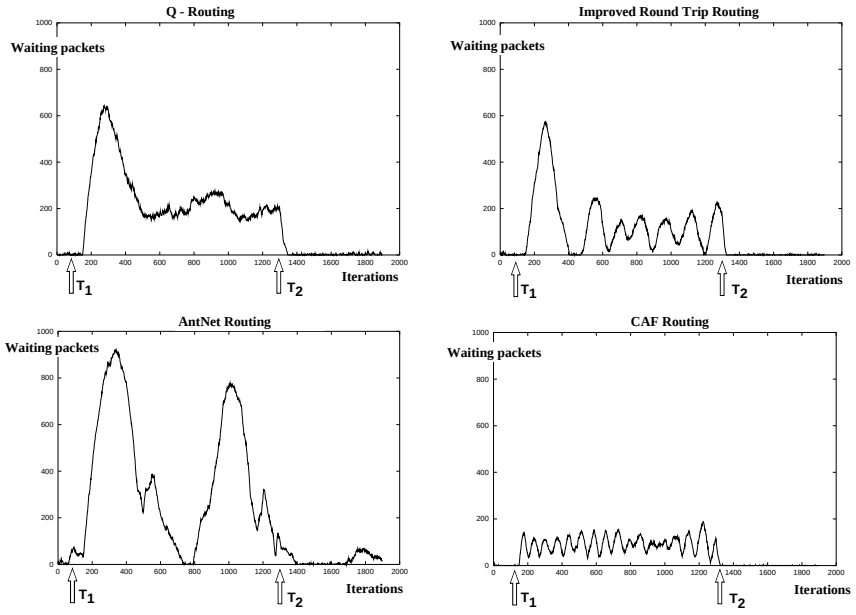

Figure 7: The evolution in the number of waiting packets during a representative simulation. During this simulation, a hot spot is created on a particular node between iteration T_1 and T_2, (i.e. the emission probability associated with the hot spot node is instantly increased). Oscillating transient responses are observed with a larger amplitude and a lower frequency for the AntNet system.

[8] S.P.M Choi and D.-Y. Yeung. Predictive Q-routing: A memory-based reinforcement learning approach to adaptive traffic control. In *Proceedings of NIPS-8, Neural Information Processing Systems*, pages 945–910, Cambridge, MA, 1996. The MIT Press.

[9] G. Di Caro and M Dorigo. Antnet: A mobile agents approach to adaptative routing. Technical Report IRIDIA/97-12, Université Libre de Bruxelles, Bruxelles, Belgium, 1997.

[10] G. Di Caro and M. Dorigo. Ant colonies for adaptive routing in packet-switched communication networks. In Springer-Verlag, editor, *PPSN V - Fifth International Conference on Parallel Problem Solving From Nature*, pages 673–682, 1998.

[11] G. Di Caro and M Dorigo. Antnet: Distributed stigmergetic control for communication networks. Technical Report IRIDIA/98-01, Université Libre de Bruxelles, Bruxelles, Belgium, January 1998. Accepted for publication in Journal of Artificial Intelligence Research (JAIR).

[12] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematcs*, 1:269–271, 1959.

[13] D. Glazer and C. Tropper. A new metric for dynamic routing algorithms. *IEEE Trans. on Communications*, 38(3):360–366, march 1990.

[14] S. Guérin. Multi-agent optimization in dynamic environment: Application to telecommunication network routing (in french). Dea dissertation, Ecole Nationale Supérieure des Télécommunications de Bretagne, 1997.

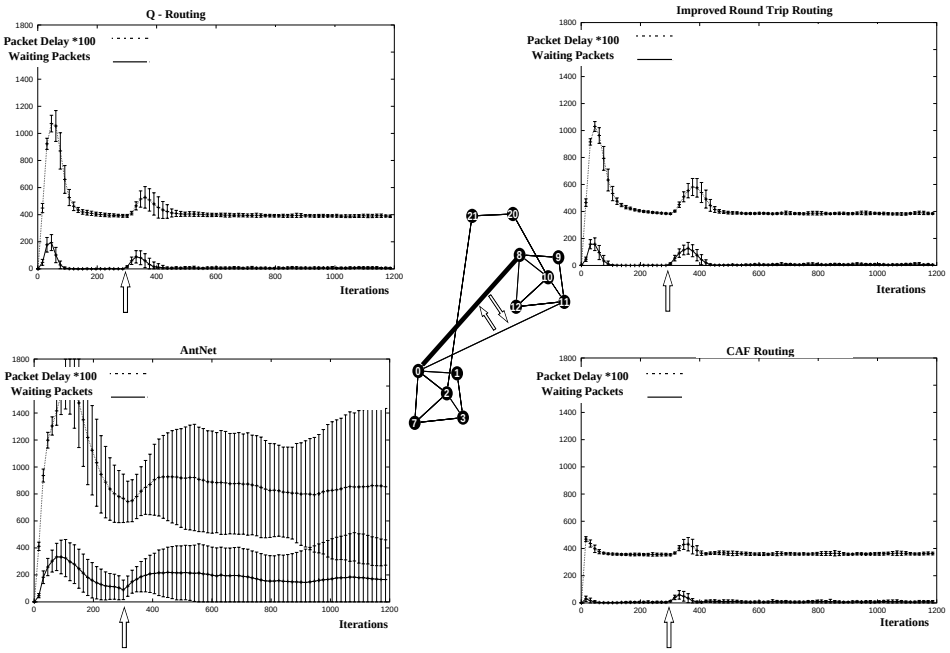

Figure 6: Average number of waiting packets and packet delay for Q-routing, Round Trip, AntNet and CAF routing on 25 simulations on the represented network (standard deviation as vertical lines). At time 300 (see the arrows), the originally large bandwidth associated with link (0, 8) is exchanged with the originally small bandwidth associated with link (0, 11).

References

[1] R. K. Ahuja, Magnanti T. L., and J. B. Orlin. *Network Flows: Theory, Algorithms and Applicaitons*. Prentice-Hall, Inc, Upper Saddle River, New Jersey, 1993.

[2] R. Beckers, Deneubourg J.-L., S. Goss, and J.M. Pasteels. Collective decision making through food recruitment. *Ins. Soc.*, 37:258–267, 1990.

[3] R. Bellman. On a routing problem. *Quartely of Applied Mathematics*, 16(1):87–90, 1958.

[4] D Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 1995.

[5] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, Inc, Upper Saddle River, New Jersey, 1992.

[6] E Bonabeau, F. Henaux, S. Guérin, D. Snyers, P. Kuntz, and G. Théraulaz. Routing in telecommunication networks with "smart" ant-like agents. In *Proceedings of the 2sd Int. Workshop on Intelligent Agents for Telecommunications Applications*, Paris, France, July 1998.

[7] J. A. Boyan and M. L. Littman. Packet routing in dynamically changing networks: A reinforcement approach. In *Advances in Neural Information Processing Systems*, volume 6, pages 671–678, San Mateo, CA, 1994. Morgan Kauffmann.

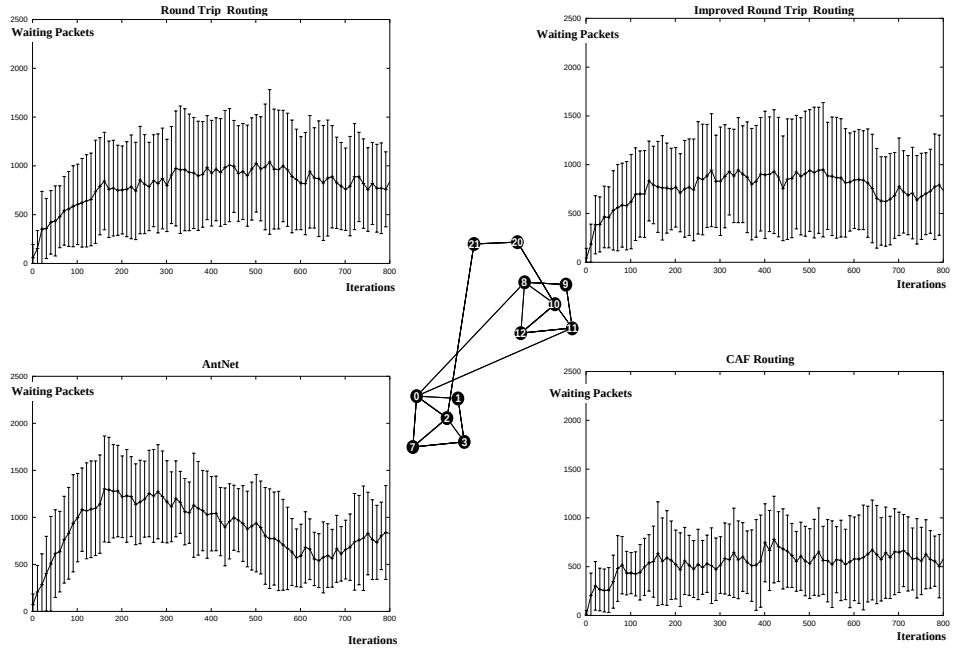

Figure 5: Average number of waiting packets for Round Trip, Improved Round Trip, AntNet and CAF routing on 25 simulations with identical Poissonian traffic ($\lambda = 0.25$) on the represented network.

In order to validate our algorithms and to compare them to already existing approaches, we have designed a network simulator. For the moment, this simulator has reduced functionalities but fits well to our first needs. In particular, it makes it possible to define and play dynamic scenarii and to validate the behaviour of our algorithms in numerous contexts. In this respect, it helped us to confirm the potentialities and the benefits of our model when dealing with network changes.

One of the main characteristic of the algorithms presented here is that they rely on two or three parameters only. Moreover, it is our experience that the range of efficient values for these parameters is rather large. This contrasts with many other optimisation heuristics.

The work in progress aims at two activities. The first one is to implement these algorithms on a realistic network simulator (we are currently working with results from the simulator. Another phase would be to implement the algorithms on a real set of routers and to gather results. The second in progress activity aims at investigating the approach in the context of Quality of Service. We believe that our proposed algorithms are of the highest interest for offering differentiated services, fault tolerant networks, and to deal with communication bursts in a timely manner.

Acknowledgements

This work was partly supported by the GIS (*Groupe d'Intérêt Scientifique*) of the CNRS and the *Région Bretagne* (project *Rodyref*). Martin Heusse also acknowledges support from Nortel, Harlow, UK.

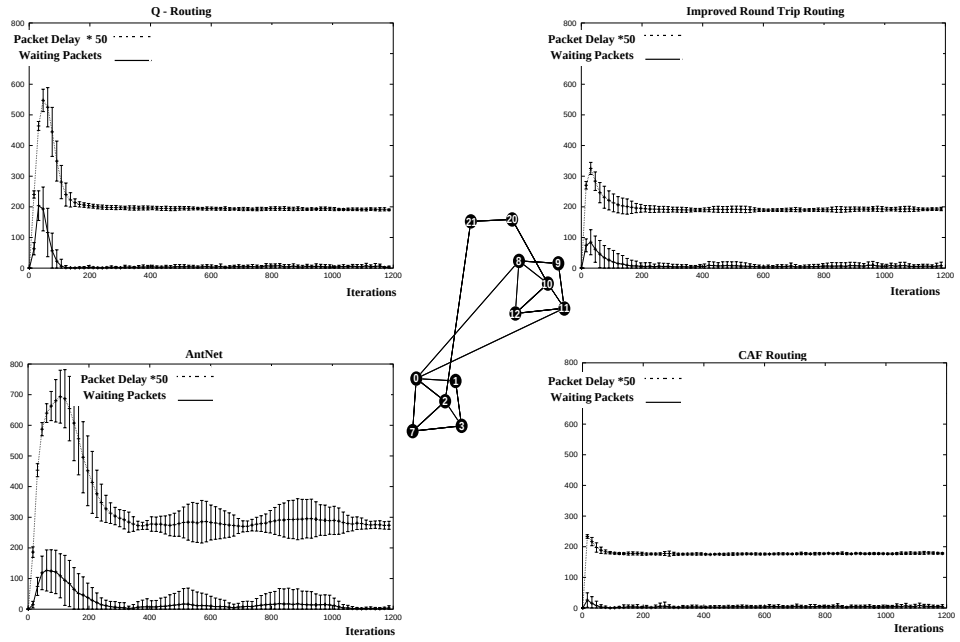

Figure 4: The average packet delay and the average number of waiting packets for Q-Routing, Improved Round Trip, AntNet and CAF routing on 25 simulations with identical random traffic loads on the network represented at the center. Vertical lines represent standard deviations.

react in a similar way to link failures. The transient after the network modification at iteration 300 is larger than the initial one. This is due to the presence of a large number of packets saturating the queue buffers after the modification.

Figure 7 shows the algorithm responses to the creation of hot spots. Between iteration T_1 and T_2, the emission probability of a node s is instantly increased. This results into an important traffic addition to the random one described before. This additional traffic originates from s toward random destinations.

All the algorithms show oscillating responses to this hot spot with a larger amplitude for the AntNet however which, on the other hand, has also a lower frequency due to its intrinsic low pass filter. CAF routing algorithm dynamically adapts its response to the traffic change with only a small transitory period. At the hot spot completion, the algorithm responses return to their normal behaviours.

6 Conclusion

In this paper, we have presented a new scheme based on agents for routing in communication networks. Our approach combines the benefits of the asynchronous distance vector routing with the adaptive link state routing approach. Moreover, we have addressed asymmetry in communication links with the CAF algorithm (Co-operative Asymmetric Forward) which is also derived from the previous modules. Our proposed algorithms are able to react and to deal with numerous changes in the communication bandwidth and the network topology: e.g., link removals, bandwidth reduction on a set of communication links. We believe that these properties are of the highest interest and can contribute to Quality of Service for network communications.

packets anymore and packets must sometimes wait on the transmission link buffers. To keep the average packet delay "reasonable" while crossing the network, adaptive load balancing must be introduced. In the simulations presented here, the delay associated with a transmission link – which is mainly due to the time spent in buffers before transmission – is used as the metric. Other metrics should be tested to further enhance the efficiency of the presented algorithms (e.g. [13], [16]).

In the sequel, the total traffic load (data plus routing traffic) will be more or less kept constant between simulations, close to the congestion level associated with the given bandwidths. We try to minimise the routing traffic and a routing overhead of 4%[5] , 8%, 8% and 20 % has been respectively used for CAF, Round Trip, Q-routing and AntNet routing. Each method keeps the same parameter set throughout the simulations. For Q-routing, we use $\beta = 4$ and $\eta = 0.5$; for Round Trip Routing, we use $\beta = 4$ and $\eta = 0.1$; for AntNet we set the ($c,\ a,\ a',\ \epsilon,\ h,\ t$) parameters as defined in Di Caro and Dorigo's paper [11]) to (2, 10, 9, 0.7, 0.04, 0.5); and for CAF ($\beta,\ \eta,\ \eta_r$) parameters we chose (4, 0.55, 0.6).

5.2.1 Load Balancing

Figure 4 studies the initial convergence in the case of a random traffic load. It compares the average packet delay and number of waiting packets for Q-routing, AntNet system, Improved Round Trip and CAF routing algorithms. It clearly exhibits a faster convergence for CAF routing and this with a lower routing overhead. The initial peak is due to the delay and local saturation resulting from the random values assigned to the routing table at the beginning.

Figure 5 similarly shows the waiting packets in the case of non uniform traffic load in which the packets are emitted from each node to a random destination following a Poissonian distribution. At each Poissonian event ($\lambda = 0.25$), a number of packets (around 136 here) are emitted toward a single destination. In this case also, CAF routing perform better than the other algorithms, but the advantage is smaller than previously. We believe that this smaller advantage is probably more due to the way the non uniform traffic is generated in the current version of the simulator: when the packets are created indeed, they enter the buffers before being really emitted; the time spent in these initial buffers is also taken into account in the statistics and depends more on the limited bandwidths than on the efficiency of the routing algorithm itself. Early results on implementing other traffic types seem to confirm this hypothesis.

5.2.2 Dynamic Load Balancing

So far, only the intrinsic environment variations due to the limitations on transmission link bandwidths has been studied. We will now study the algorithm responses to sudden modifications in the network by exchanging bandwidth on "close" links and by creating hot spots on some nodes.

At the beginning of the simulation represented on Figure 6 the bandwidth associated with link (0, 8) is much higher than the one associated with link (0, 11). At iteration 300, the network exchanges the two bandwidths and the traffic that originally went through link (0, 8) immediately saturates. Delays increase and new load balance has to be found. AntNet responds with an important transient before returning to normal. This transient decreases with round trip routing, Q-routing and CAF routing. The peak of waiting packets after the bandwidth exchange around iteration 300 is followed by a burst in the mean delay. The presented algorithms

[5] If the reverse distance vector cannot be directly updated by the data traffic for efficiency reasons, an additional reverse routing traffic can be used instead. In this case, the routing overhead will double.

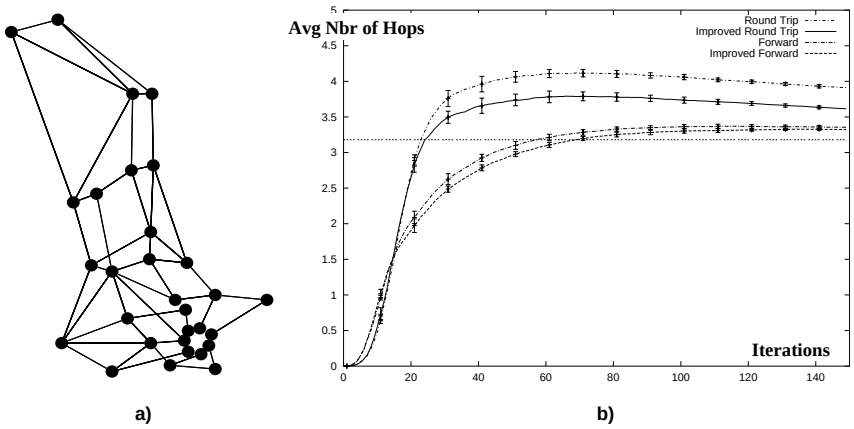

Figure 3: The curves in **b)** show the sum of the average cost on 25 runs, for all packets transiting in the symmetric network represented in **a)** over a sliding time window of 10 iterations. The packet cost is here the number of hops it takes for a packet to reach its destination. (Vertical lines represent standard deviations.) The horizontal line is the optimal mean number of hops given by the Dijkstra algorithm. The routing tables are initialised at random and the hop counter is only incremented when the packets reach their destination. (This explains the slow values observed at the beginning). The traffic load is uniform: 35 data packets are fed into the network per iteration, with random source and destination nodes, while 20 routing packets per node are maintained in the network.

5 Experimental Results

This network simulator is used to compare the different routing algorithms presented in the previous sections. We first consider some static environments associated with a random traffic load pattern (each node sends packets to a random destination at an uniform rate) and a Poissonian traffic load. The dynamic of these algorithms is then studied by restricting the bandwidths under the same random traffic load. This generates buffering delays on transmission links that sometimes lead to oscillating responses. Finally, we study the reaction of the algorithms to simulated events like hot spots (a bursting node emits many packets during a given period of time), bandwidth changes and link failures. In all the results presented here, the simulation were initialised with a uniform probability and the data packets started to be emitted from the very beginning.

5.1 Routing in Static Environments

We simulate static environments by considering symmetric networks with the number of hops as metric and with no restriction on the bandwidth. Simulations have been performed on a simplified network (Figure 3) which is sufficient to highlight the faster convergence of forward routing against the round trip one. Furthermore, they show the advantage of the improvement based on Bellman's principle presented in Section 3.

5.2 Routing in Dynamic Environments

As soon as we add some bandwidth constraints, the network becomes asymmetric due to the delay metric: the transmission links cannot handle all the incoming

with the packet destination d on all the visited nodes n are updated as follows:

$$D^n_{j,d}(t) = (1 - \eta)D^n_{j,d}(t - 1).$$

4 A Network Simulator

In order to compare these routing algorithms, a simplified network simulator has been designed. Dedicated to routing algorithm testing however, it does not implement all the network OSI layers. For example, connections are supposed to be perfect and packets are produced by simple random processes instead of simulated sessions or clients. Practically, it is a synchronous discrete event simulator running on a directed weighted graph representing the network. Every node is a router identical to the others. At each iteration, incoming packets are routed depending on their type (routing or data packets), then pushed to the buffer corresponding to the chosen outgoing link. These buffers are supposed infinite: packets are queued but never discarded and there is no congestion control other than the routing algorithm itself.

At each iteration, every link processes a given amount of waiting packets according to its associated parameters: *bandwidth* and *delay*. The bandwidth is the number of packets a link can take by unit of time, and the delay is the minimum time that it takes for a packet to pass through the link. All the packets appear similar to the link: this does not allow us to compare algorithms by considering the amount of data carried by each routing packet, only their number is critical. Actually, it should be noticed that for all the algorithms presented here, routing packets share basically the same structure and carry approximately the same amount of information; they differ only in their behaviours.

The network load is provided by Poissonian random processes. The basic traffic load is uniform: every node generates packet at a uniform rate with a random destination. Other non-uniform traffic patterns have been tested including the existence of hot spots: a node emitting a large amount of packets over a given period of time. And, in order to test the algorithm reactions in a dynamic context, the simulator also includes events such as link breakdowns or sudden parameter changes.

All the algorithms presented here use *probabilistic routing*, i.e. the routes are selected, following a random scheme, proportionally to the routing table values that give the probability of selecting a link for a given destination node. To improve the exploration of new routes however, there is also a tiny probability (between 0.05 and 0.01) of choosing the next hop with a uniform probability amongst the neighbouring nodes. Compared to deterministic routing, this method makes it possible to achieve load balancing, as flows in the network are naturally split among the different available routes.

To evaluate the result of each simulation we use several measurements: throughput, average delay on the network, number of waiting packets. The throughput is the number of packets (or the amount of information) that went through the network during one unit of time. The average delay experienced by all the packets arrived during a given interval of time gives another view of the general status of the network. The number of waiting packets, which is just the sum of the sizes of all the queues buffering the links, shows congestions very efficiently. In the case of infinite buffer lengths, the average packet delay is highly correlated with the overall throughput. Therefore, only the results associated with the delay metrics will be presented here.

puts into practice Bellman's principle of optimal path decomposition. The routing table probabilities $p^i_{j,s}$ of selecting link (i,j) are given by Equation (4).

3.2.2 CAF Routing

The *Co-operative Asymmetric Forward routing* (CAF routing) extends symmetric forward routing steps to asymmetric networks for which the estimate of the distance between two nodes in one direction cannot be used in the other direction. The basic idea of this algorithm is to switch routing packets differently as if there were tracing back their way from their destination to their source and to use the cost estimated by traffic moving in the opposite direction to update the routing tables.

Let F_k be a routing agent emitted by a node s with a destination node d. Since the network is supposed to be asymmetric, the cost $d^k_{s,n}$ encountered by F_k arriving on a node n through a link (j,n) cannot be considered as previously (Section 3.2.1) as a valuable estimation of the distance toward s for other agents arriving on n from another direction. It consequently cannot be used directly for updating the distance vector element $D^n_{j,s}$ anymore. The cost $d^k_{n,j}$used in the update of $D^n_{j,s}$ is now estimated by the traffic moving in the opposite direction (i.e. from n to j). Hence, the cost T^j_n associated with the last packet arriving on j from n is stored on j and, when a routing agent F_k leaves j to go to n it pushes the pair (j, T^j_n) on a stack S_k. The arrival of F_k on n entails the following update of $D^n_{j,S}$:

$$D^n_{j,s}(t) = (1-\eta)D^n_{j,s}(t) + \eta d^k_{n,s}$$

where $d^k_{n,s}$ is here the sum of each arc distance stored on S_k and measured by routing agents moving in the opposite direction. We also add, as in the previous section, the improvement based on Bellman's principle.

To ensure a correct distance vector update in the asymmetric case however, the data packets and the routing agents have to be routed differently. Data packets are routed on a node n as previously following Equation (4) with new D^n definition. For routing agents, the basic idea is to switch them differently as if they were tracing back their way from d to s. As a consequence, in addition to maintaining the routing table from distance vectors in one direction we must now also compute the routing tables for reverse routing traffic. Let R^n be a "reverse routing table" whose elements $R^n_{j,s}$ keep track on n of the link (j,n) usage by traffic originating from s; at the arrival of a data packet on n from s through (j,s), all the elements of R^n associated with destination s are multiplied by η_r (this implements the time window negative exponential) and $R^n_{j,s}$ is incremented by one. The routing agents are then routed on n according to the following probabilities

$$rp^n_{j,d}(t) = \frac{\left(R^n_{j,d}(t)\right)^\beta}{\sum_l \left(R^n_{l,d}(t)\right)^\beta}. \tag{5}$$

Maintaining these reverse routing table probabilities allows the CAF agents originating from s to be routed along the path used by information packets coming in the opposite direction, i.e. with a destination s.

Forward routing also requires a special treatment of detected loops to efficiently deal with dynamic changes in the network. In the case of a connection failure for example, the information about the open connection cannot be forwarded to the nodes on the other side, but it can at least be backpropagated on the previously visited nodes. Once a loop is detected[4] in our approach, the distance vector associated

[4] A loop is detected by agent k on n when every outgoing link $(n,\ j)$ leads to an already visited node j, i.e. j is already on the agent internal stack S_k.

inherent to distance vector algorithms they use a low pass filter in maintaining a list of estimates over a given moving time period of arithmetic mean values of the distances and their associated variance for all node n in the network. These values are then periodically used to directly update the routing tables. Their AntNet system has been shown to outperform OSPF and Bellman-Ford algorithms in their simulations[3] (see [11], [10] for more details and developments).

3.2 Forward Agent Routing

Backward routing is intrinsically slow since it requires the agent to reach its destination before any update to begin. This slow round trip reaction to changes in the network might induce oscillations. Forward routing offers an alternative by removing the need of round trips. It was first introduced by Schoonderwoerd et al. [17] in the case of virtual circuit based symmetric networks (e.g. identical costs associated with both link directions). We extended it to packet-switching symmetric networks and we added the optimisation based on Bellman's principle previously mentioned [6]. Subramanian at al. made the connection between Schoonderwoerd et al. work and reinforcement learning [18]. This paper extends this approach to asymmetric networks (e.g. packet-switching networks in which asymmetric delays occur due to different queue lengths). If the network is symmetric, the cost measured by forward agents on their path from the source to the current node can directly be used to update the estimation of the distance toward the source from the current node. For asymmetric networks, this cannot be done anymore. Subramanian et al. [18] avoid the problem by defining agents that choose the next node to visit randomly (i.e. with identical probability of choosing a neighbour node). But this random routing scheme annihilates any desirable auto-catalytic effect previously mentioned. This paper introduces a more efficient asymmetric forward routing: *CAF routing*.

3.2.1 Symmetric Forward Routing

Schoonderwoerd et al. [17] introduced symmetric forward routing in the context of virtual circuit-based networks in which bandwidth is allocated during transmission on the links between the origin and the destination of the call. Bandwidth is a limited resource and traffic should be adequately balanced. The performance of the network in this case is often measured by the proportion of calls which could be placed on the network in a given period of time.

Routing agents travel in the network and count the number of hops from the beginning of their journey. This routing metric is symmetric and the distance estimation can be used by the agents coming in the opposite direction for deciding of the route to be taken. Each time a routing agent arrives from the source s on a node n, the distance measured from s to n can directly be used to update the estimation of the distance toward s for other agents arriving on n from an other direction.

Schoonderwoerd et al. method is adapted here to packet-switching networks and their update scheme is slightly modified. Each time a routing agent F_k arrives on node n through link (j, n) from s, the estimated distance $D^n_{j,s}$ to reach s through link (n, j) – which is equivalent to (j, n) due to the symmetry property – is updated as in (3) and becomes

$$D^n_{j,s}(t) = (1 - \eta)D^n_{j,s}(t - 1) + \eta d^k_{n,s}$$

where $d^k_{n,s}$ is the distance measured by F_k between s and n. We augment this method with the efficient improvement described previously (Section 3.1.1) which

[3]The implementation presented here correspond to the one found in the first given reference.

time spent in queues plus transmission delay. The associated cost $d^k_{n,d}$ between n and d is the sum $d^k_{n,j_1} + d^k_{j_1,j_2} + ... + d^k_{j_m,d}$ of each cost between n and d on the journey J_k. As soon as F_k reaches the destination d, a backpropagating agent B_k is sent from d to s on the arcs of J^k. The arrival of B_k on a node n of J^k from a node j updates the estimation of $D^n_{j,d}(t)$ as follows:

$$D^n_{j,d}(t) = (1-\eta)D^n_{j,d}(t-1) + \eta d^k_{n,d} \tag{3}$$

where η is a learning parameter as defined previously in Section 2.4. On Figure 2, taken as an example, when a forward agent F_k arrives on node n from source s it pushes the pair $(s, d^k_{s,n})$ on its stack. Arriving on destination d, the stack contains $\left\{(s, d^k_{s,n}), (n, d^k_{n,s_1}), (s_1, d^k_{s_1,d})\right\}$. When the associated backpropagating agent B_k arrives on n from s_1 it updates the $D^n_{s_1,d}$ value as follows:

$$D^n_{s_1,d}(t) = (1-\eta)D^n_{s_1,d}(t-1) + \eta(d^k_{n,s_1} + d^k_{s_1,d}).$$

In [6] we introduce a very efficient improvement that updates all distances corresponding to intermediate nodes visited by the agent at once. We apply the well-known Bellman's principle of decomposing an optimal path into optimal sub-paths. Every intermediate node on J^k is treated like a destination and its associated distance is updated according to the relative distance between that visited node and the current updating ones. This is as if the set of routing agents associated with the source-destination pair s-d were duplicated on each node such that each new set is associated with an intermediate node on J^k taken as a destination - this is the reason why we called this approach "multiple round trip"-. On node n from Figure 2, for example, the backpropagating agent B_k updates – in addition to updating $D^n_{s_1,d}$ as previously described – the distance vector for the intermediate node s_1:

$$D^n_{s_1,s_1}(t) = (1-\eta)D^n_{s_1,s_1}(t-1) + \eta d^k_{n,s_1}.$$

The routing table on n, which gives the probability $p^n_{j,d}$ of selecting each arc (n,j) linked to n for a packet with destination d, is then periodically recalculated at every time step t as follows[1]:

$$p^n_{j,d}(t) = \frac{\left(\frac{1}{D^n_{j,d}(t)}\right)^\beta}{\sum_l \left(\frac{1}{D^n_{l,d}(t)}\right)^\beta} \tag{4}$$

with β is a non-linearity parameter taken superior to 1 to favour short paths.

In order to deal with the dynamics of the networks (i.e. link breakdowns etc.) that force forward agents to loop back to a previously visited node, a special treatment has been added. Instead of simply destroying the agent, the valid distance information gathered so far is backpropagated. Hence, when an agent with destination d arrives on n through arc (j,n) and detects a loop, it simply returns to the previously visited node j and starts a standard backpropagation from there, as if it would have reached its destinations j (and not d).

3.1.2 AntNet

Di Caro and Dorigo's AntNet system [9] uses similar forward agents with a different loop detection behaviour[2] but with more complex backpropagating agents and a different routing table update procedure. Indeed, in order to minimise the oscillations

[1] Computing the routing table and updating the distance estimation could be combined for efficiency but we keep them separate here for clarity reason.

[2] Di Caro and Dorigo simply erases the loop from the agent memory and let them continue with the risk of falling in the same loop again.

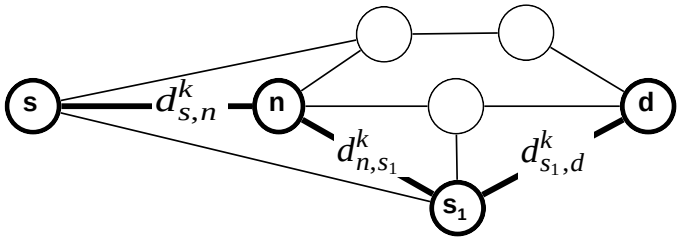

Figure 2: The path followed by a packet/agent P from s_0 to d with the associated measured distances $d^k_{s,n}, d^k_{n,s_1}$ and $d^k_{s_1,d}$.

For routing and load-balancing, we consider two types of agents depending on when the distance vector is updated. This update can either be performed by routing agents going from the source to the destination (*forward routing*) or by agents retracing their way back to their source (*round trip routing*). We first present two versions of round trip routing: an original approach derived from the online asynchronous distance vector routing algorithm presented in Section 2.4. [14] and the *AntNet* system that has been shown to outperform many aspects of the OSPF and Bellman-Ford routing algorithms on a packet-switching network simulator by Di Caro and Dorigo [11] and [10]. We propose an efficient improvement based on Bellman's principle that increases the transmission of information on the network between agents [6]. The advantages of forward routing are then presented for symmetric and asymmetric networks. We review the Schoonderwoerd et al.'s approach on forward routing [17] which has been developed in the context of virtual circuit based symmetric networks and extend it to packet-switching symmetric networks. Finally, we propose a very efficient new forward routing method in the general context of asymmetric packet-switching networks: *CAF routing* (Co-operative Asymmetric Forward routing).

3.1 Round Trip Agent Routing

Round trip routing is based on two sets of homogeneous mobile agents, respectively called the *forward agents* and the *backpropagating agents*. The *forward agents* share the same queues as data packets and use the same routing tables. They keep track of their journey and the associated costs between hops in an internal stack. Forward agents also implement a mechanism to avoid loops in their routes. The *backpropagating agents* retrace their way back to the source and update the distance vector accordingly. These bactracking agents however have a higher priority over data for a faster propagation of the accumulated information.

3.1.1 Multiple Round Trip Routing

We first implemented round trip routing as a Monte Carlo extension to the online asynchronous distance algorithm presented in Section 2.4. Each node emits a set of forward agents and, in the case of multiple round trip routing a backpropagating agent is associated with each forward agent. For a greater intelligibility, we only describe the steps of the algorithm for a single forward agent F_k originating from any node s (source node) of N and going toward any node d (destination node) of N. We first detail the updates of the distance vectors D^n and then, we look at the probabilities p^n of arc selection in the routing tables.

The forward agent F_k keeps track of its journey J^k from s to d and of the associated cost $d^k_{n,d}$ between any node n and d on J^k defined as, for example, the

cost. Once again, let $D^i_{n,d}$ be what node i estimates it costs to deliver a packet toward destination d by way of its neighbour node n. This estimates could then be incrementally improved by direct feedback from the measured cost when the packet reaches its destination.

Following reinforcement learning [15], the estimates can be updated locally before the packet reaches its destination. As described in [7] indeed, on sending a packet to n, i immediately gets back n's estimate of the cost associated with the remaining part of the trip, namely $min_{j \in N(n)}\{D^n_{j,d}(t-1)\}$ where $N(n)$ is the neighbour node set of node n. At time t, i revises its estimate as follows:

$$D^i_{n,d}(t) = (1-\eta)D^i_{n,d}(t-1) + \eta(d_{i,n} + min_{j \in N(n)}\{D^n_{j,d}(t-1)\}) \qquad (2)$$

where η is the so called reinforcement "learning rate" of the gradient descent. Instead of synchronously updating all distance vectors as in (1) only $D^i_{n,d}$ is here asynchronously updated online every time a packet uses arc (i,n) on its way to destination d.

Choi and Yeung [8] further improve this algorithm by keeping track of congested paths (which usually correspond to optimal paths) and by periodically probing them for recovery.

3 Agent-Based Routing

The new family of algorithms presented here is an extension to these classical algorithms: it combines the ideas of online asynchronous distance vector routing with adaptive link state routing. As in online asynchronous distance vector routing, the associated costs are directly measured from the network traffic instead of being estimated by each node according to on-site data (e.g. from the waiting buffers length). Here, at regular interval, every network node emits a set of *routing agents* - usually implemented as routing packets that share the same transmission line and waiting queues with the data packets - which measure, for example, traffic delays and allow an online and asynchronous update of the routing tables. From link state routing, these new methods retain the idea of keeping topological information about the network. But instead of having an identical map of the network duplicated on the nodes, the topological information is here distributed on the routing agents themselves. Every routing agent memorises the sequence of switching nodes visited during its journey. The data packets select their next hop, following a probabilistic scheme function of the information stored in the routing table. Instead of broadcasting the topological information from a node to the whole network, as in OSPF for example, the routing agents incrementally update the distance vectors as they move along the network nodes. The important point is that each update on a node can immediately influence the routing on this node which in turn further influence the agents arriving afterwards.

This process of information transmission creates an autocatalytic effect similar to the so-called "stigmergy process" which explains the collective behaviour of some social insects and which has inspired the approaches described here. Stigmergy is a collective co-ordination process based on indirect communication between individuals only (usually through chemical substances called pheromones laid down in the environment). Dynamic structuring phenomena have been highlighted in some insect societies indeed; in particular, some entomologists have shown that the shortest path between an ant nest and a food source can emerge from pheromone trail laying and following behaviours between non directly communicating individuals [2]. Ants returning to the nest from a food source leave pheromone behind them, this pheromone trail attracts other individuals which in turn reinforces the pheromone trail. This auto-catalytic reaction makes the shortest path to rapidly emerge.

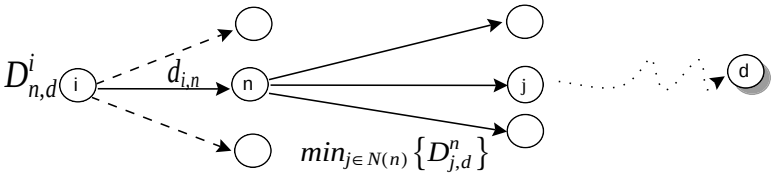

Figure 1: The Bellman-Ford algorithm updates the distance vectors on node i by combining the known distances $d_{i,n}$ and the best estimate from the neighbour nodes for the remaining part of the path.

by allowing non-deterministic routing along a small number of best routes. But all of these algorithms fail to work properly with very dynamic networks.

2.2 Adaptive Distance Vector Routing

Dynamic distance vector routing periodically updates the distance vectors by exchanging information between neighbours. The Bellman-Ford distance vector routing algorithm [3], for example, was the original ARPANET routing algorithm and was also used in INTERNET under the name RIP. It is based on the principle of dynamic programming [4]: an optimal path is made of sub-optimal paths. Each node i periodically updates its distance vector from the distance vector regularly sent by its neighbours as follows:

$$\begin{cases} D^i_{n,n}(t) &= d_{i,n}, & \forall i; \\ D^i_{n,d}(t) &= d_{i,n} + min_{j \in N(n)}\{D^n_{j,d}(t-1)\}, & \forall i,\ \forall n \neq d \end{cases} \quad (1)$$

where $D^i_{n,d}(t)$ is the cost estimated by i for delivering a packet from i to d by the way of the neighbour n at time t as shown on Figure 1, and where $d_{i,n}$ is the known distance between i and its neighbour n.

Although this procedure converges to the correct answer, it may do so slowly. In particular, this procedure is known to react promptly to good news (e.g. a new transmission lines) but slowly to bad news (e.g. a link failure) [19] and it is also prone to oscillations [5]. For these reasons, it is nowadays often replaced by *link state routing*.

2.3 Adaptive Link State Routing

The link state algorithm essentially maintains a dynamic map of the complete network. This dynamic map is replicated on each router and is used to estimate the optimal distances between nodes (usually with Dijkstra's algorithm). Each node periodically broadcasts its routing information to all destinations with a distributed flooding mechanism [19, 5] trying to minimise the number of re-transmissions. The metric usually estimates the delays between a node and its neighbours based on the queue lengths on transmitting and receiving nodes.

The OSPF protocol, which is increasingly being used in the Internet, uses such a link state algorithm.

2.4 Online Asynchronous Distance Vector Routing

Recently, Boyan and Littman [7] propose an online and asynchronous version of the Bellman-Ford distance vector algorithm based on reinforcement learning [4]. The routing policy tries to find the optimal adjacent node the current node should send its packet to, in order to reach the destination with a minimal associated

from *routing packets* sent in the network by the routers. These routing packets mix with the regular information packets and keep track, for example, of the delays encountered during their journey.

In the first part, we recall the basic principles of the classical routing algorithms. The new class of distributed routing and load-balancing algorithms is detailed in Section 3. We present a unified overview of the literature related to this new family of algorithms and we propose two new approaches. The network simulator used to test these new protocols is described in Section 4 and comparative results are given in Section 5.

2 Classical Routing Algorithms

Historically, the routing algorithms used in communication networks have evolved from static routing in which "good routes" are computed off-line to more dynamic routing in which the routes are computed online to take the node congestion level into account. Classical routing protocols were successively based, for example, on *Static Routing*, *Adaptive Distance Vector Routing* in which the routing tables are regularly updated and *Adaptive Link State Routing* which also maintains a map of the network topology and load pattern on each of its nodes. This evolution is associated with an increase in the number of routing packets transiting on the network. However, these routing algorithms react rather slowly to changes in the network load or topology and they are prone to oscillations. In addition, these algorithms face a major increase in their required memory when several metrics are taken into account for guaranteeing different qualities of services.

Throughout this paper we consider that the topology of the telecommunication network is modelled by a non directed weighted graph $G = \{N, A\}$ with a node set N and an arc set A. At each node n of N, the following information is updated:

- D^n, the distance vector on n whose element $D^n_{(j,d)}$ is the estimated distance of a best route from a node n to a node d through the arc (n, j);
- $p^n_{(j,d)}$, the probability of selecting the arc (n, j) for a packet with destination d.

2.1 Static Routing

Basic static routing in a communication network is equivalent to finding the shortest paths between the nodes of an associated graph in which each node represents a router and in which each weighted arc corresponds to a communication line. The metric used here can be the number of hops between two routers, the physical distance, the transmission delay, etc. The classical Dijkstra algorithm [12] solves the shortest path problem in polynomial time. It can be used to build the routing tables required by the router to transmit entering packets toward their destination. The routing tables are built from the so called *distance vector*, which assigns the optimal distances to each destination for every outgoing lines on each nodes.

This method only takes the topology into account but the network load also need to be considered when the lines only have a limited transmission capacity or bandwidth. And while finding the shortest path can be solved in polynomial time, flow optimisation, i.e. maximising the number of packets transiting in the network per second (*throughput*), when lines have such transmission limitations is known to be a $\mathcal{NP}$-complete problem [1] for which existing heuristics are still quite complex.

Flow-based routing, for example, uses transmission delays as metric. It computes off-line the mean line delay for different flows under deterministic single path routing and statically optimises the average delay [5]. The load can also be further balanced

ENST de Bretagne Technical Report **RR-98001-IASC**

Adaptive Agent-Driven Routing and Load Balancing in Communication Networks

Martin Heusse, Dominique Snyers*, Sylvain Guérin and Pascale Kuntz
ENST de Bretagne, BP 832, Brest Cedex, France

Abstract

This paper presents an unified overview of a new family of distributed algorithms for routing and load balancing in dynamic communication networks. These new algorithms are described as an extension to the classical routing algorithms: they combine the ideas of online asynchronous distance vector routing with adaptive link state routing. Estimates of the current traffic condition and link costs are measured by sending routing agents in the network that mix with the regular information packets and keep track of the costs (e.g. delay) encountered during their journey. The routing tables are then regularly updated based on that information without any central control nor complete knowledge of the network topology. Two new algorithms are proposed here. The first one is based on round trip routing agents that update the routing tables by backtracking their way after having reached the destination. The second one relies on forward agents that update the routing tables directly as they move toward their destination. An efficient co-operative scheme is proposed to deal with asymmetric connections. All these methods are compared on a simulated network with various traffic loads; the robustness of the new algorithms to network changes is proved on various dynamic scenarii.

1 Introduction

This paper describes a new family of distributed algorithms for routing and load balancing in dynamic communication networks. These are indeed critical operations that directly influence the throughput and average delays of information messages and, hence have an impact on the overall performance of the network.

Nowadays, networks are characterised by a very fast evolution. Network topologies are not only continuously growing (see the Internet for instance), but the networks usage is also changing: many devices become mobile and some sort of guaranteed Quality of Service (QoS) is now often required. This results in an increasingly dynamic network to which classical - usually more or less centralised - routing methods are poorly adapted. This is the reason why new protocols should be defined to efficiently deal with the changing traffic loads and topologies.

This paper examines the potential of a new family of distributed methods for packet-switching networks in which the information is split into packets and transits in the network along potentially different routes. The routing tables are here regularly updated without central control nor complete knowledge of the network topology. An estimate of the current load is measured from statistics gathered

*Dominique.Snyers@enst-bretagne.fr