18

Inductive Approaches Based on Trial/Error Paradigm for Communications Network

Abdelhamid Mellouk
LISSI/SCTIC/QoSiDiN laboratory, University Paris XII-Val de Marne
France

1. Introduction

Today, providing a good quality of service (QoS) in irregular traffic networks is an important challenge. Besides, the impressive emergence and the important demand of the rising generation of real-time Multi-service (such as Data, Voice VoD, Video-Conference, etc.) over communication heterogeneous networks, require scalability while considering a continuous QoS. This emergence of rising generation Internet required intensive studies these last years which were based on QoS routing for heterogeneous networks on the one hand and on the backbone architecture level of communication networks characterized by a high and irregular traffic on the other hand (Mellouk et al., 2007b).

The basic function of QoS routing is to find a network path which satisfies the given constraints and optimize the resource utilization. The integration of QoS parameters increases the complexity of the used routing algorithms. Thus, the problem of determining a QoS route that satisfies two or more path constraints (for example, delay and cost) is known to be NP-complete (Gravey & Jhonson, 1979). A difficulty is that the time required to solve the Multi-Constrained Optimal path problem exactly cannot be upper-bounded by a polynomial function. Hence the focus has been on the development of pseudo-polynomial time algorithms, heuristics and approximation algorithms for multi-constrained QoS paths (Kuipers & Mieghem, 2005).

At present, several studies have been conducted on QoS routing algorithms which integrate the QoS requirements problematic for the routing algorithm. (Song & Sahni, 2006) introduce heuristics to find a source-to-destination path that satisfies two or more additive constraints on edge weights. (Jaffe, 1984) has proposed a polynomial time approximation algorithm for k multi-constrained path which uses a shortest path algorithm such as Dijkstra's (Sahni, 2005). (Korkmaz & Krunz, 2001) propose a randomized heuristic that employs two phases. In the first one, a shortest path is computed for each of the k QoS constraints as well as for a linear combination of all k constraints. The second phase performs a randomized breadth-first search for a solution of k multi-constrained problem. In (Kuipers & Mieghem, 2005), authors suggest that QoS routing in realistic networks could not be NP-complete regarding to a particular class of networks (topology and link weight structure).

Due this complexity, QoS routing problems are divided on several classes according to some aspects. For example, we distinguish the single path routing problem and the multipath routing problem, where routers maintain multiple distinct paths of arbitrary costs between a

Source: Reinforcement Learning: Theory and Applications, Book edited by Cornelius Weber, Mark Elshaw and Norbert Michael Mayer
ISBN 978-3-902613-14-1, pp.424, January 2008, I-Tech Education and Publishing, Vienna, Austria

source and a destination. The Multipath routing offers several advantages like good bandwidth, bounding delay variation, minimizing delay, and improved fault tolerance. So, it makes an effective use of the graph structure on a network, as opposed to single path routing which superimposes a logical routing tree upon the network topology. We find in literature many and various approaches that have been proposed to take into account the QoS requirement. The reader can refer to (Masip-Bruin et al., 2006) for an overview. Constraints imposed by QoS requirements, such as bandwidth, delay, or loss, are referred to as QoS constraints, and the associated routing is referred to as QoS routing which is a part of Constrained-Based Routing (CBR). Interest in constrained-based routing has been steadily growing in the Networks. Based on heuristics used in all of these approaches to reduce their complexity, we can classified it in three main categories:

Label Switching/Reservation Approaches- spurred by approaches like ATM PNNI, MPLS or GMPLS. With MPLS, fixed length labels are attached to packets at an ingress router, and forwarding decisions are based on these labels in the interior routers of the label-switched path. MPLS Traffic Engineering allows overriding the default routing protocol, thus forwarding over paths not normally considered. A resource reservation protocol such as RSVP must be employed to reserve the required resources. Another Architecture proposed for providing Internet QoS is the Differentiated Services architecture. Diffserv scales well by pushing complexity to network domain boundaries.

Multi-Constrained Path Approaches (MCP)- The goal of all of these approaches is to retrieve the shortest path among the set of feasible paths between two nodes. Considerable work in the literature has focused on a special case of the MCP problem known as the Restricted Shortest Path (RSP) problem. The goal is to find the least-cost path among those that satisfy only one constraint. An overview of these approaches can be found in (Kuipers et al., 2004).

Inductive approaches- To be able to make an optimal routing decision, according to relevant performance criteria, a network node requires to have a complete knowledge of the entire network state and an accurate prediction of the evolution of the networks and its dynamics. This, however, is impossible unless the routing algorithm is capable of adapting to the network state changes in almost real time. Thus, it is necessary to design intelligent and adaptive optimizing routing algorithms which take into account the network state and its evolution. We need to talk about QoS based state dependent routing algorithm.

In this chapter, we present an accurate description of the current state-of-the-art and give an overview of our work in the use of reinforcement learning concepts focused on communication networks. We focus our attention by developing a system based on this paradigm called KOCRA for K Optimal Constrained path Routing Algorithm. Basically, these inductive approaches selects routes based on flow QoS requirements and network resource availability. After developing in section 2 the concept of routing in high speed networks, we present in section 3 the family of inductive approaches. After, we present our works based on reinforcement learning approaches in three different communication networking domains: wired networks, mobile ad hoc networks, and packet router's scheduling networks. Last section concludes and gives some perspectives of this work.

2. Routing problem

As Internet is a large collection of more than 25,000 independent domains called autonomous systems (Ases), the cooperation between ASes is not optimized at the network

level, but rather it is based on the business relationships between organizations. The fully-independent management actions in each AS are expressed in terms of a policy-based routing strategy which primarily controls the outbound traffic of an AS and can include conflicting policies. A global solution for QoS routing over all the ASes must be able to handle both the differing QoS provisioning mechanisms and service specifications. This latter solution of building models of large ISP's is so complex to obtain (Quoitin & Uhlig, 2005). For this, Routing is divided onto two classes: IGP and EGP. IGP, such as OSPF or IS-IS, compute the interior paths in one AS, while EGP, such as BGP, is responsible for the selection of the interdomain paths. To fulfill application QoS requirements, many ISPs have deployed mechanisms to provide differentiated services in their networks. In fact, in the last decade, the development of none of QoS routing proposals has turned out to be sufficiently appealing to become deployed in practice. This is because ISPs have preferred to overprovision their networks rather than deliver and manage QoS (Yanuzzi et al., 2005).
In the IGP or EGP cases, a routing algorithm is based on the hop-by-hop shortest-path paradigm. The source of a packet specifies the address of the destination, and each router along the route forwards the packet to a neighbour located "closest" to the destination. The best optimal path is choosed according to given criteria. When the network is heavily loaded, some of the routers introduce an excessive delay while others are under-utilized. In some cases, this non-optimized usage of the network resources may introduce not only excessive delays but also high packet loss rate. Among routing algorithms extensively employed in the same AS routers, one can note: distance vector algorithm such as RIP and the link state algorithm such as OSPF or IS-IS (Grover, 2003).

2.1 Distance vector approach

Also known as Bellman-Ford or Ford-Fulkerson, the heart of this type of algorithm is the routing table maintained by each host. With the distance-vector (DV) routing scheme (e.g. RIP, IGRP), each node exchanges with its neighbouring nodes its distance (e.g. hop count) to other networks. The neighbouring nodes use this information to determine their distance to theses networks. Subsequently these nodes share this information with their neighbours, etc. In this way the reachability information is disseminated through the networks. Eventually each node learns, which neighbour (i.e. next hop router) to use, to reach a particular destination with a minimum number of hops. A node does not learn about the intermediate to the destination. These approaches suffers from a classic convergence problem called "count to infinity". It also does not have an explicit information collection phase (it builds its routing table incrementally). DV routing protocols are designed to run on small networks.

2.2 Link state approach

With link-state (LS) routing (e.g. OSPF or IS-IS), each node builds a complete topology database of the network. This topology database is used to calculate the shortest path with Dijkstra's algorithm. Each node in the network transmits its connectivity information to each other node in the network. This type of exchange is referred to as flooding. This way each node is able to build a complete topological map of the network. The computational complexity cost used here is lower than the DV protocol. However, LS algorithms trade off communication bandwidth against computational time.

3. Inductive approaches

Modern communication networks is becoming a large complex distributed system composed by higher interoperating complex sub-systems based on several dynamic parameters. The drivers of this growth have included changes in technology and changes in regulation. In this context, the famous methodology approach that allows us to formulate this problem is dynamic programming which, however, is very complex to be solved exactly. The most popular formulation of the optimal distributed routing problem in a data network is based on a multicommodity flow optimization whereby a separable objective function is minimized with respect to the types of flow subject to multicommodity flow constraints (Gallager, 1977; Ozdaglar & Bertsekas, 2003). In order to design adaptive algorithms for dynamic networks routing problems, many of works are largely oriented and based on the Reinforcement Learning (RL) notion (Sutton & Barto, 1997). The salient feature of RL algorithms is the nature of their routing table entries which are probabilistic. In such algorithms, to improve the routing decision quality, a router tries out different links to see if they produce good routes. This mode of operation is called exploration. Information learnt during this exploration phase is used to take future decisions. This mode of operation is called exploitation. Both exploration and exploitation phases are necessary for effective routing and the choice of the outgoing interface is the action taken by the router. In RL algorithms, those learning and evaluation modes are assumed to happen continually. Note that, the RL algorithms assigns credit to actions based on reinforcement from the environment. In the case where such credit assignment is conducted systematically over large number of routing decisions, so that all actions have been sufficiently explored, RL algorithms converge to solve stochastic shortest path routing problems. Finally, algorithms for RL are distributed algorithms that take into account the dynamics of the network where initially no model of the network dynamics is assumed to be given. Then, the RL algorithm has to sample, estimate and build the model of pertinent aspects of the environment.

Many of works has done to investigate the use of inductive approaches based on artificial neuronal intelligence together with biologically inspired techniques such as reinforcement learning and genetic algorithms, to control network behavior in real-time so as to provide users with the QoS that they request, and to improve network provide robustness and resilience. For example, we can note the following approaches:

Q-Routing approach- In this technique (Boyan & Littman, 1994), each node makes its routing decision based on the local routing information, represented as a table of Q values which estimate the quality of the alternative routes. These values are updated each time the node sends a packet to one of its neighbors. However, when a Q value is not updated for a long time, it does not necessarily reflect the current state of the network and hence a routing decision based on such an unreliable Q value will not be accurate. The update rule in Q-Routing does not take into account the reliability of the estimated or updated Q value because it depends on the traffic pattern, and load levels. In fact, most of the Q values in the network are unreliable. For this purpose, other algorithms have been proposed like Confidence based Q-Routing (CQ-Routing) or Confidence based Dual Reinforcement Q-Routing (DRQ-Routing).

Cognitive Packet Networks (CPN)- CPNs (Gelenbe et al., 2002) are based on random neural networks. These are store-and-forward packet networks in which intelligence is constructed into the packets, rather than at the routers or in the high-level protocols. CPN is then a reliable packet network infrastructure, which incorporates packet loss and delays directly

into user QoS criteria and use these criteria to conduct routing. Cognitive packet networks carry three major types of packets: smart packets, dumb packets and acknowledgments (ACK). Smart or cognitive packets route themselves, they learn to avoid link and node failures and congestion and to avoid being lost. They learn from their own observations about the network and/or from the experience of other packets. They rely minimally on routers. The major drawback of algorithms based on cognitive packet networks is the convergence time, which is very important when the network is heavily loaded.

Swarm Ant Colony Optimization (AntNet)- Ants routing algorithms (Dorigo & Stüzle, 2004) are inspired by dynamics of how ant colonies learn the shortest route to food source using very little state and computation. Instead of having fixed next-hop value, the routing table will have multiple next-hop choices for a destination, with each candidate associated with a possibility, which indicates the goodness of choosing this hop as the next hop in favor to form the shortest path. Given a specified source node and destination node, the source node will send out some kind of ant packets based on the possibility entries on its own routing table. Those ants will explore the routes in the network. They can memory the hops they have passed. When an ant packet reaches the destination node, the ant packet will return to the source node along the same route. Along the way back to the destination node, the ant packet will change the routing table for every node it passes by. The rules of updating the routing tables are: increase the possibility of the hop it comes from while decrease the possibilities of other candidates. Ants approach is immune to the sub-optimal route problem since it explores, at all times, all paths of the network. Although, the traffic generated by ant algorithms is more important than the traffic of the concurrent approaches.

In the following, we give an overview of our work in the use of reinforcement learning concepts focused on communication networks. We focus our attention by developing a system based on this paradigm called KOCRA for K Optimal Constrained path Routing Algorithm and present our works based on reinforcement learning approaches in three different communication networking domains: wired networks, mobile ad hoc networks, and packet router's scheduling networks.

4. KOCRA system based reinforcement learning in routing wired networks.

KOCRA is the successor of KONRS, a K Optimal Neural Routing System (Mellouk et al., 2006a).

4.1 Brief summary of KONRS

In (Mellouk et al., 2006a), we have presented an adaptive routing algorithm based on Q learning approach, the Q function is approximated by a reinforcement learning based neural network (NN). As shown in figure 1, In this approach, NN ensure the prediction of parameters depending on traffic variations. Compared to the approaches based on a Q table, the Q value is approximated by a reinforcement learning based neural network of a fixed size, allowing the learner to incorporate various parameters such as local queue size and time of day, into its distance estimation. Indeed, a neural network allows the modelling of complex functions with a good precision along with a discriminating training and a taking into account of the context of the network. Moreover, it can be used to predict non-stationary or irregular traffics. In this approach, the objective is to minimize the average packet delivery time. Consequently, the reinforcement signal which is chosen corresponds

to the estimated time to transfer a packet to its destination. Typically, the packet delivery time includes three variables: The packet transmission time, the packet treatment time in the router and the latency in the waiting queue.

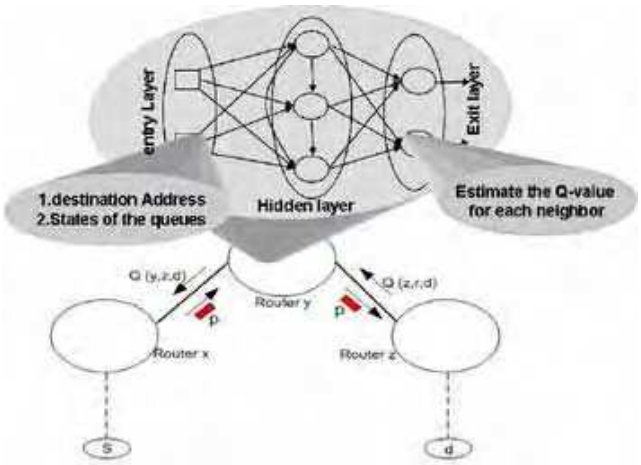

Fig. 1. Neural Net Architecture

The input cells in NN used correspond to the destination and the waiting queue states. The outputs are the estimated packet transfer times passing through the neighbours of the considered router. The algorithm derived from this architecture can be described according to the following steps:

When receiving a packet of information:
1. Extract a destination IP address,
2. Calculate Neural Network outputs,
3. Select the smallest output value and get an IP address of the associated router,
4. Send the packet to this router,
5. Get an IP address of the precedent router,
6. Create and send the packet as a reinforcement signal.

At the reception of a reinforcement signal packet:
1. Extract a Q estimated value computed by the neighbor,
2. Extract a destination IP address,
3. Neural Network updating using a retro-propagation algorithm based on gradient method,
4. Destroy the reinforcement packet.

This approach offers advantages compared to standard Distance Vector (DV) routing policy and earlier Q-routing algorithm, like the reduction of the memory space for the storage of secondary paths, and a reasonable computing time for alternative paths research. The Q value is approximated by a reinforcement learning based neural network of a fixed size. Results given in (Mellouk et al., 2006a) show better performances of the proposed algorithm

comparatively to standard distance vector and Q-routing algorithms. In fact, at a high load level, the traffic is better distributed along the possible paths, avoiding the congestion of the network.

4.2 The concepts behind KOCRA

This first version of our KONRS routing system explore all the network environment and do not take into account loop problem in a way leading to large time of convergence algorithm. To address this drawback and reducing computational time, we have worked on the evolution of our earlier Q-neural routing algorithm and present the enhanced version of KONRS called "K Optimal Constrained path Routing Algorithm (KOCRA)" (Mellouk et al., 2007a). KOCRA contains three stages. The objective of the first stage is to select the K Best candidate paths according to the cost cumulative path from the source and the destination nodes (for simplicity, we consider here all link costs equal to 1). The second stage is used to integrate the dynamics of traffic. For this, a continuous end-to-end delay among the K Best selected Paths is computed using a reinforcement Q-learning function. In order to force the router to take the alternative routes regarding to the second stage, we used a third one which compute automatically a probability affected to each path based on packet delivery time obtained by the second stage and the time latency in queuing file associated for each path.

4.2.1 First stage: constructing K-best paths

First of all, in spite of exploring the entire network environment which needs large computational time and space memory (Mellouk et al., 2006a), our approach reduces this environment to K Best no loop paths in terms of cost cumulative links. Thus, each router maintains a link state database as map of the network topology. We used a label setting algorithm based on the optimality principle and being a generalization of Dijkstra's algorithm (Sahni, 2005). In order to find these K best paths, a variant of Dijkstra's algorithm proposed in (Eppstein, 1999) was used. The space complexity is $O(Kmn)$, where K is the number of paths, m (resp. n) is the number of edges (resp. the number of links). By using a pertinent data structure, the time complexity can be kept at $O(m+nlogn+K)$ (Mellouk et al., 2007a). When a network link changes its state (i.e., goes up or down, or its utilization is increased or decreased), the network is flooded with a link state advertisement (LSA) message. This message can be issued periodically or when the actual link state change exceeds a certain relative or absolute threshold. Obviously, there is tradeoff between the frequency of state updates (the accuracy of the link state database) and the cost of performing those updates. In our approach, the link state information is updated when the actual link state change. Once the link state database at each router is updated, the router computes the K optimal paths.

Let a DAG (N; A) denote a network with n nodes and m edges, where N = {1.. n}, and A ={aij/i,j€N}. N}. The problem is to find the top K paths from source s to all the other nodes. Let's define a label set X and a one-to-many projection h: N→ X, meaning that each node i€ N corresponds to a set of labels h(i), each element of which represents a path from s to i.

---

* *S the source node*
* *N –set of nodes in network*

```
* X – the label set
* Count_i – Number of paths determined from S to I
* elm – Affected number to assigned label
* P – Paths list from S to destination (D)
* K – paths number to compute
      * h – corresponding between node and affected label number
/* Initialisation */
count_i = 0 /* for all i □ K */
elem = 1
h(elem) = s
h^-1(s) = {elem}
distance_elem = 0
X = {elem}
P^K = 0
While (count_t < K and X != { })
    begin
        /* find a label lb from X, such that
        distance_lb <= distance_lb1, □ lb1 □ X*/
        X = X – {lb}
        i = h(lb)
        count_i = count_i + 1
        if (i == D) then /* if the node I is the destination node D */
                begin
                        p = path for 1 to lb
                        P^K = P^K U {h(p)}
                end
        if (count_i <= K) then
              begin
                for each arc(i,j) □ A
                      begin
                       /* Verify if new label does not result in loop */
                       v=lb
                       While (h(v) != s)
                        begin
                          if (h(v) == j) then
                             begin
                                goto do_not_add
                             end
                           v = previous_v
                        end
                    /* Save information from new label */
                    elem = elem + 1
                    distance_elem = distance_n + c_ij
                    previous_elem = lb
                    h(elem) = j
                    h^-1(j) = h^-1(j) U {elem}
```

```
            X = X U {elem}
            do_not_add:
                        end
end
end
```

4.2.2 Second stage: Q-learning algorithm for optimizing the end-to-end delay

After finding our K best Optimal Paths based on link costs, the second step is to distribute the traffic on these K candidate paths. For this, we use another criteria based on the end-to-end delay. The reinforcement signal which is chosen corresponds to the estimated time to transfer a packet to its destination. This value is computed by a variant of Q-Routing algorithm which is considered as an asynchronous relaxation of the Bellman-Ford algorithm used in distance vector protocols. Typically, the packet delivery time includes three variables: the packet transmission time, the packet treatment time in the router and the latency in the waiting queue. In our case, the packet transmission time is not taken into account. In fact, this parameter can be neglected in comparison to the other ones and has no effect on the routing process.

In this approach, each router x maintains in a Q-table a collection of values of $Q(x, y, d)$ for every destination d and for every interface y. This value reflects a delay of delivering a packet for destination d via interface s. Then, the router x forwards the packet to the best next router y determined from the Q-table. Just after receiving this packet, the router y provides x an estimate of its best Q value to reach the destination. This new information is then added in the Q-values of the router x.

The reinforcement signal T employed in the Q-learning algorithm can be defined as the minimum of the sum of the estimated $Q(x, y, d)$ sent by the router y neighbour of router x and the latency in waiting queue q_x corresponding to router x.

$$T = \min_{y \in \text{neighboof x}} \{ \; q_x + Q(x,y,d) \; \} \tag{1}$$

Where $Q(x, y, d)$, denote the estimated time by the router x so that the packet p reaches its destination d through the router y. This parameter does not include the latency in the waiting queue of the router x. The packet is sent to the router y which determines the optimal path to send this packet.

Once the choice of the next router is made, the router y puts the packet in the waiting queue, and sends back the value T as a reinforcement signal to the router x. It can therefore update its reinforcement function as:

$$\Delta Q(x,y,d) = \eta(\alpha + T - Q(x,y,d) \;) \tag{2}$$

α and η are the packet transmission time between x and y and the learning rate respectively. So, the new estimation $Q'(x,y,d)$ can be written as follows:

$$Q'(x,y,d) = Q(x,y,d)(1-\eta) + \eta(T+\alpha) \tag{3}$$

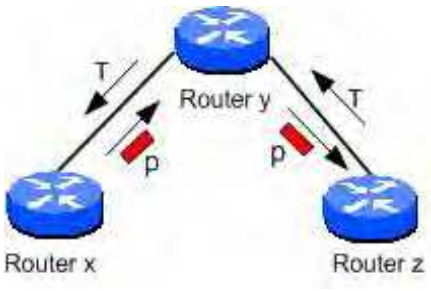

Fig. 2. Updating of the reinforcement signal.

4.2.3 Third stage: adaptive probabilistic path selection.

The goal of this stage is to distribute the traffic on K best paths in probabilistic manner. To force the router to take alternative routes find in K best paths and not only the best one, we compute a probability affected to each path automatically. We associated a maximal value P_{max} for the best path and divided the rest of probability (1- P_{max}) for the remaining (K-1) paths. The value of P_{max} is fixed by a counting process. To force the router to take the alternative routes find in the second stage and not only the best path, a uniform distributed random process is implemented in each router. This process chooses randomly a number between [0, 1]. Next, a router choose the path verifying the condition that it's probability is less than this random number. For example, in the situation characterized by K=2 (two paths), P1=0.8, P2=0.2, if the random number <=0.8, the router chooses the first path, otherwise the router takes the second one. In this manner, the flow packets reach their destination with a time close to optimal, while ensuring a good exploration of the remaining paths. Unfortunately, this kind of fixed hand probability don't take automatically into account the dynamic of the irregular traffic. We have proposed a second version of computing automatically the load balancing distribution. The process is based on the packet delivery time computed by our Q reinforcement learning and the latency in queuing file associated for each path.

Let $D_i(t)$ be the packet delivery time for path i at time t. Let $T_i^n(t)$ be the latency in queuing file associated to closest router n' in the direction of path i at time t (that is, the neighbour of router n). The following formula allows us to count the probability $P_i^n(t)$ for the i^{th} path in router n at time t:

$$P_i^n = \left[\left(\frac{1}{D_i}\right)^{\alpha} + \left(\frac{1}{T_i^n}\right)^{\beta}\right] \Big/ \left[\sum_{i=1}^{z}\left(\frac{1}{D_i}\right)^{\alpha} + \left(\frac{1}{T_i^n}\right)^{\beta}\right] \quad (4)$$

Where α and β are two tuneable parameters that determine respectively the influence of delay time and waited queue time. They have an equivalent influence in the case of $a=\beta$. This formula associates a very small probability for paths with high delay time and/or high queue time. This is due to the fact that when delay time (respectively waited time) increase the value of $[1/D_i(t)]^{\alpha}$ (respectively $[1/T_i(t)]^{\beta}$) decreases.

4.3 Performance evaluation

To validate our results in the case of irregular traffic in wired networks, we take the results given by a well-known Djikstra's algorithm (which offers to use an existing polynomial-time path computation) used in protocols such OSPF, IS-IS or CISCO EIGRP as a reference for our study. This choice of this classical approach is argued by the fact that the majority of ISP's used actually this kind of protocols to exchange routing information in their networks. In order to do comparison with KOCRA, parameters of standard approach used here are fixed in order to optimize the delay and cost criteria simultaneously (on the rest of paper, we used the notation "Standard Optimal Multi-Path Routing Algorithm (SOMRA)" for this kind of algorithm). All algorithms have been implemented with OPNET and used the same data structure. OPNET software constitutes for telecommunications networks an appropriate modelling, scheduling and simulation tool. It allows the visualization of a physical topology of a local, metropolitan, distant or on board network. The protocol specification language is based on a formal description of a finite state automaton.
The simulations presented in this article consisted of creating a traffic merged in irregular network topology, through which the two family of algorithms (KOCRA and SOMRA) computed the best paths between two nodes. QoS measures of each of tested algorithms concerns two additive constraints: cost and delay criteria. Results given in all the figures are evaluated in terms of average packet end-to-end delivery time on both topologies. Time simulation is represented on the other axis of the figures.

4.3.1 Simulation parameters on the irregular topology.

The topology of the network is specified by a collection of routers and a set of links that bind these routers elements. The network traffic is specified in the source router by setting several parameters like: the start time, the stop time, the statistical distribution for packet inter-arrival times, the statistical distribution for packet size and the destination node.
To ensure a meaningful validation of our algorithm performance, we devised a realistic simulation environment in terms of network characteristics, communications protocols and traffic patterns. We focus on IP datagram networks with irregular topology. The topology of the network employed for simulations includes 36 interconnected nodes with essentially two parts of the network, as shown in Fig 3. This topology is the same used in (Boyan & Littman, 1994) for their Q learning approach.
We model traffic in terms of requests characterized by its source and destination. While we concern ourselves with arrival and departure of flows, we do not model the data traffic of the flows. For simplicity, we also chose not to implement a proper management of error, flow and congestion control. In act, each additional control component has a considerable impact on the network performance, making very difficult to evaluate and to study properties of each control algorithm without taking in consideration the complex way it interacts with all the other control components (Dorigo & Stüzle, 2004). Therefore, we chose to test the behavior of our algorithm such that the routing component can be evaluated in isolation.
The traffic is sent/received by four end nodes (marked in the figure noeud100, noeud101, noeud102 and noeud103).
For our simulation results, we studied the performance of the algorithms for increasing traffic load, examining the evolution of the network status toward a saturation condition,

and for temporary saturation conditions. For this topology, we study the performance of our routing strategies according a Poisson Law inter-arrival times statistical distribution.

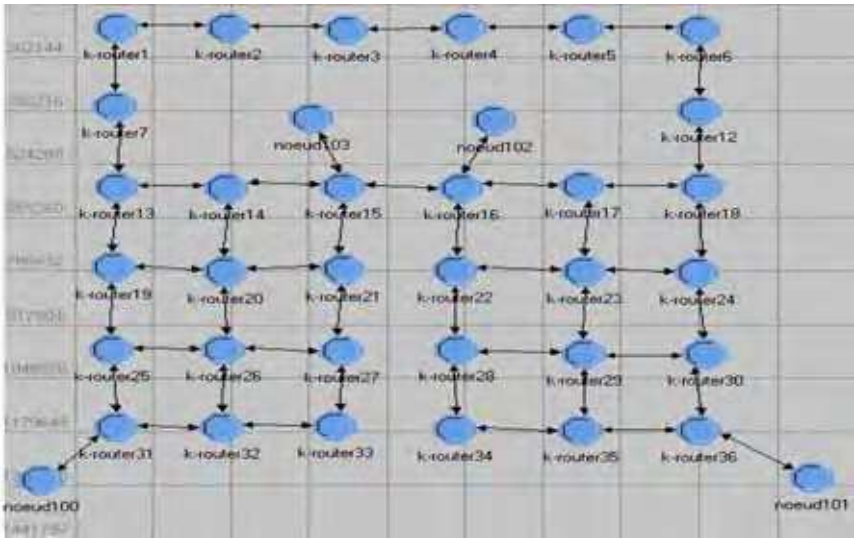

Fig. 3. Network topology.

4.3.2 Poisson distribution of model traffic.

In probability theory, the Poisson distribution is a discrete probability distribution which expresses the probability of a number of events occurring in a fixed period of time if these events occur with a known average rate, and are independent of the time since the last event. The It is represented by random variables N that count a number of discrete occurrences (called "arrivals") that take place during a time-interval of given length. The probability that there are exactly k occurrences (with k a non-negative integer, k = 0, 1, 2, ...) is:

$$P(k,\lambda) = \frac{e^{-\lambda}\lambda^{k}}{k!} \tag{5}$$

Where λ is a positive real number and is the mean number of occurrences k. The Poisson law is then defined by its mean λ parameter.

In our simulations, we suppose the mean λ of the inter-arrival times is 3 s and fix the time start to 1 min and the stop time to the end of simulation is fixed to 6 h.

4.3.3 Simulation results.

As shown in Fig. 4 which represent time simulation versus the average packet delivery time, our probabilistic K Optimal Constrained path Routing Algorithm (KOCRA) give better results than the well-known N best optimal path routing Algorithm SOMRA. This is due to the fact that in our new approach, routers are able to take into account not only the average of delivery delay but also the waiting queue time. Thus, they are able to adapt their

decisions very fast and in close concordance with the network dynamics. In spite of the many packages taking secondary ways, N-optimal routing does not present better performances because it rests on a probabilistic method to distribute the load of the network over the closest cost paths, and not on the degradation of the times of routing. So, in classical approach, the routers take their decisions only according to the average of delivery delay and the exploration of potentials good paths, none trivially best and that can give us betters results, is not realized. Our approach, with the introduction of a probabilistic module, responds to this inconvenience and shows better results for Poisson law distribution of traffic. Thus, mean of average packet delivery time obtained by KOCRA is reduced by 37% compared to traditional N best optimal routing Algorithm.

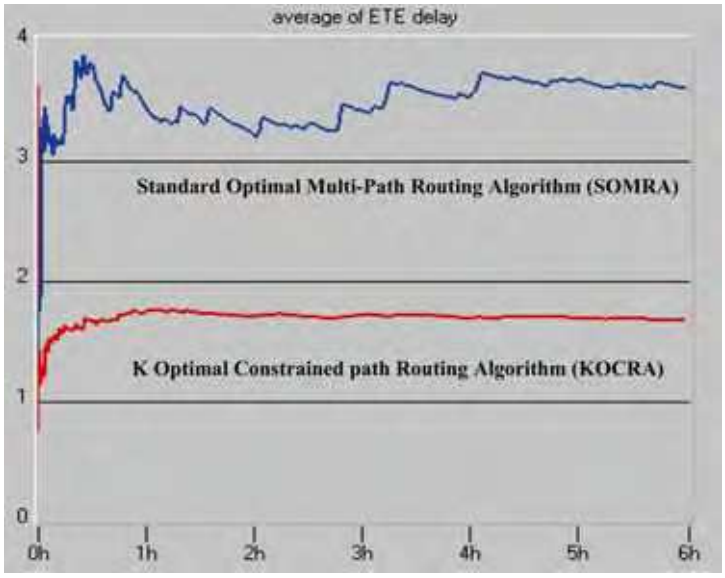

Fig. 4. Poisson law distribution simulations results

5. AMDR based reinforcement learning in mobile ad hoc networks.

AMDR (Adaptive Mean Delay Routing) is a new adaptive routing protocol based on probabilities and built around two exploration RL agents. Exploration agents gather mean delay information available at each node in their route and calculate total delay between source and destination. According to the delay value gathered, probabilistic routing tables are updated at each intermediate node. In order to deal with mobile nodes synchronisation we consider, in our protocol, delay estimation model proposed in [Naimi, 2005; Naimi & Jacquet, 2004], instead of instantaneous delay considered in the most oriented delay routing protocols.

Unlike data packets, control packets, used in adaptive routing, are sent in broadcast manner and so treated at IEEE 802.11, MAC layer differently than unicast packets. For this, we consider that trip delay of a control packet is not the same of a data packet.

In AMDR, routing function is determined by means of very complex interactions of forward and backward network exploration agents. Forward agents report network delay conditions to the backward ones. So, no node routing updates are performed by the forward agents.
AMDR uses two kinds of agents: Forward Exploration Packets (FEP) and Backward Exploration Packets (BEP). Forward agents explore the paths of the network, for the first time in reactive manner, but it continues the exploration proactively.
FEP packets create a probability distribution at each node for its neighbours. Backward agents are used to propagate the information gathered by forward agents through the network, and to adjust the routing table entries.

5.1 Forward exploration packet

When a new traffic arrives at a node *n*, periodically the node *n* generates a forward exploration packet called FEP. The FEP packet is then sent to the destination of traffic. Each FEP packet contains the following fields: *source_node_address, destination_node_address, ext_hop_address, stack_of_visiting_nodes_addresses, total_delay*.
If the entry of the current destination does not exist then a routing table entry is immediately created. The algorithm of FEP sending is the following:

```
Algorithm (Send FEP)
    At Each T_interval_secondes Do
    Begin
        Generate a FEP
        If any entry for this destination Then
                Create an entry with uniform probabilities.
      End If
        Broadcast the FEP
    End
End (Send FEP)
```

When a FEP arrives to a node *i*, it checks if the address of the node *i* is not equal to the *destination address* contained in the FEP agent then the FEP packet will be forwarded. FEP packets are forwarded according to the following algorithm:

```
Algorithm (Receive FEP)
        If any entry for this destination Then
                Create an entry with uniform probabilities.
        ELSE If my_adress ≠ dest_adress Then
            IF FEP not already received then
                    Stock address of the current node,
                    Recover the available mean delay,
                    Broadcast FEP
            ELSE
                    Send BEP
          End IF
      End IF
End (forward FEP)
```

5.2 Backward exploration packet

As soon as a forward agent FEP reaches its destination, a backward agent called BEP is generated and the forward agent FEP is destroyed. BEP inherits the stack and the total delay information contained in the forward agent. We define five options for our algorithm in order to reply to a FEP agent. The algorithm of sending a BEP packet depends on the chosen option. The five options considered in our protocol are:

Reply to All: for each FEP reception, the destination a BEP packet is generated. In this case, the delay information is not used and the overhead generated is very important.

Reply to First: Only one BEP agent is generated for a FEP packet. The mean delay module is not exploited. It's the same approach used in the AntNet. The overhead is reduced but any guarantee to have the best delay paths.

Reply to N: destination node can generate until N BEP packet for the same FEP. The overhead is reduced compared to reply to N but it is more important than the Reply to first option.

Reply to the Best: We save at each node the information of the best delay called "Node.Total_delay". When the first FEP arrives to the destination, Node.Total_delay takes the value of total delay of the FEP packet. When another FEP arrives, its total delay is compared to the node total delay, and we reply only if the FEP has a delay better or equal to the node total delay.

Reply to delay Constraint: This option focuses on Delay-Constrained-Path (DCP) unicast routing. The DCP issue is to select the path with given delay requirement. This is the case of real time applications having serious delay constraints.

Applications needs in term of delay are determined in a max delay supported "D" value. Arriving to destination, this one compares FEP *total_delay* to the application *delay_constraint* "D". If the FEP *total_delay* is equal or less than "D" then a BEP is generated and sent to the source of the FEP. We give in the following a part of BEP sending algorithm. We focus on "Reply to best" option which will be used in simulation part:

Algorithm (Send BEP)

```
Select Case Option
   Case: Reply to All
         ....
   Case: Reply to first
   If (First (FEP) ) Then
         …..
     endIf
   Case: Reply to N
   If (N>0) Then
         .......
     endIf
   Case: Reply to Best
   If (FEP.Total_Delay <= Node.Total_Delay) Then
         Genarate BEP
         BEP.Total_Delay=0,
         BEP.dest = FEP.src,
```

```
        Send BEP,
        Node.Total_Delay= FEP.Total_Delay,
    endIf
  Case: Reply to Delay Constraint (D)
  If (FEP.Total_Delay <= D) Then
        .....
    End If
End (Send BEP)
```

Backward Exploration Packet (BEP) retraces the inverse path traversed by the FEP packet. In other words, unlike FEP packets, a BEP packet is sent in a unicast manner because it must take the same path of its FEP generator. During its trip, the BEP agent calculates the total mean delay of its route and uses this new delay to adjust the probabilistic routing table of each intermediate node. The algorithm of forwarding BEP agent is the following:

```
Algorithm (Receive BEP)
    If (my_adress = BEP.dest) Then
        Update probabilistic routing table
    Else
        Update probabilistic routing table
        Forward BEP
    End If
End (Receive BEP)
```

5.3 Updating routing tables

Routing tables are updated when a BEP agent is received. The probabilities updating can take many forms, and we have chosen updating rules (6), (7), (8) and (9) described in (Baras & Mehta, 2003). As soon as, routing table is calculated, data packets are then routed according to the highest probabilities in the probabilistic routing tables.

Unlike on demand routing protocols, there is no guarantee to route all packets on the same route because of the proactive exploration. The BEP agent make changes to the probability values at the intermediate and final node according to the following update rules:

$$p_{fd} \leftarrow (p_{fd} + r)\,(1+r) \tag{6}$$

$$p_{nd} \leftarrow p_{nd}/(1+r) \tag{7}$$

$$p_{nd} \leftarrow p_{nd} - rp_{nd} \tag{8}$$

$$p_{fd} \leftarrow p_{fd} + r\,(1-p_{fd}) \tag{9}$$

In both the above cases, the reinforcement parameter *r* can be defined as a function of delay. Here, r=k /f(c), where k > 0 and f(c) is the cost function (Baras & Mehta, 2003).

5.4 Flooding optimization

In order to improve the performance of our routing protocol, we introduce the MPR (Nguyen & Minet, 2006) concept in the broadcast process. However, the MPR selection

according to native OLSR is unable to build path satisfying a given QoS request. To avoid this problem, we propose a new algorithm for MPR selection. We keep at each node a table called MPR table containing a partial view of MPR neighbours. Our algorithm takes into account the mean delay available at each node. The MPR selection algorithm based on mean delay is the same proposed for bandwidth in (Nguyen & Minet, 2006), unlike their approach for bandwidth MPR; we define only one kind of MPR which are delay MPR. Mean delay MPR selection algorithm is composed of the following steps:

1. A node N_i selects, first, all its neighbours that are the only neighbours of a two hop node from N_i.
2. Sort the remaining one-hop delay neighbours in increasing order of mean delay.
3. Consider each one-hop neighbour in that order: this neighbour is selected as MPR if it covers at least one two-hop neighbour that has not yet been covered by the previous MPR.
4. Mark all the selected node neighbours as covered and repeat step 3 until all two-hop neighbours are covered.

With the present MPR selection algorithm, we guarantee that paths having best delays will be discovered but there are any guarantees about the overhead generated (Ziane & Mellouk, 2006).

5.5 Performance evaluation

We use NS-2 simulator to implement and test AMDR protocol. We present in this section two scenarios of simulation. In the first one, we define a static topology of 8 nodes. To compare AODV, OLSR and AMDR, we have chosen the Reply to Best option of AMDR.

5.5.1 Static scenario

The following table summarizes the simulation environment:

Routing	AODV, AMDR, OLSR
MAC Layer	802.11
Bandwidth	11Mb/s
TERRAIN	1000m,1000m
Nodes	8
Simulation time	1000 sec
Data traffic	exponential

Table 1. Simulation settings scenario 1

We injected three types of traffic in the network and compared for each simulation the file trace for each routing protocol. Figure 5 shows the comparison of end to end delay realized by AODV, AMDR and OLSR protocols. We can see that, at first OLSR realizes the best

delays when AMDR and AODV show a large initial delay, which is required for routes to be set up.
A few times after initialisation stage, AMDR shows more adaptation to changes in the network load and realizes the best end to end delay followed by AODV and at last OLSR.
On the other hand, comparing loss rate performances of the three protocols shows in figure 6, that OLSR realizes the best performances followed by AMDR and then AODV. AMDR performance is justified by keeping alternative paths used when the actual path is broken. Any additional delay is need to route waiting traffics and deliverance ratio is well improved than AODV.

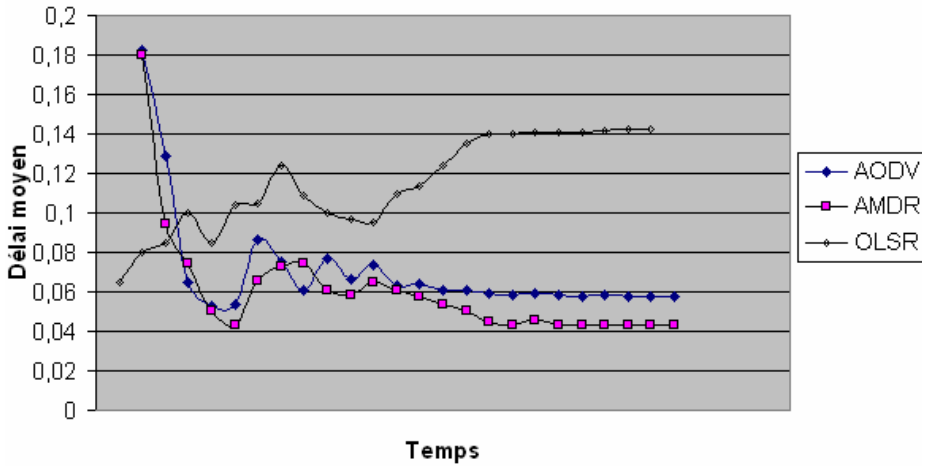

Fig. 5. Packets delay comparison in static scenario

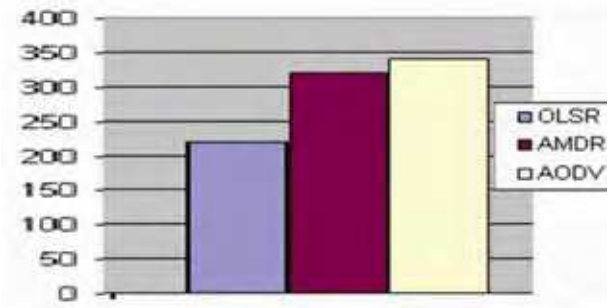

Fig. 6. Comparison of loss rate for static scenario

From the overhead comparison results, we saw that overhead generated by OLSR was the most important followed by AMDR and than AODV. This is justified by the proactive exploration process used by AMDR even a route is already established. The difference between overhead of AMDR and AODV is not very important because of optimization flooding mechanism used in AMDR.

5.5.2 Mobility scenario

In this scenario, we test the impact of mobility on AMDR and compare its performances with OLSR and AODV. We define a random topology of 50 nodes.

Traffic model	Exponential
Surface of simulation	1000m,1000m
Packets size	512 byte
Bandwidth	1Mbs
Rate of mobility	5m /s , 10m/s
Number of connections	5, 10, 15, 20, 25
Rate	5 paquets/s
Simulation duration	500 s

Table 2. Simulation settings scenario 2

Table 2 summarizes the simulation setting. We injected at first five traffics, ten, fifteen, twenty and at last twenty five traffics. After each simulation we calculate the end to end delay realized by each protocol. Figure 7 summarizes our comparison. We can observe that with low load, there is no difference in end to end delays. However, more the network is loaded more AMDR is better in term of delay. Such performance is justified by the adaptation of AMDR to changes in the network load. In the case of AODV and OLSR an additional delay is impossible to circumvent for adapting to changes.

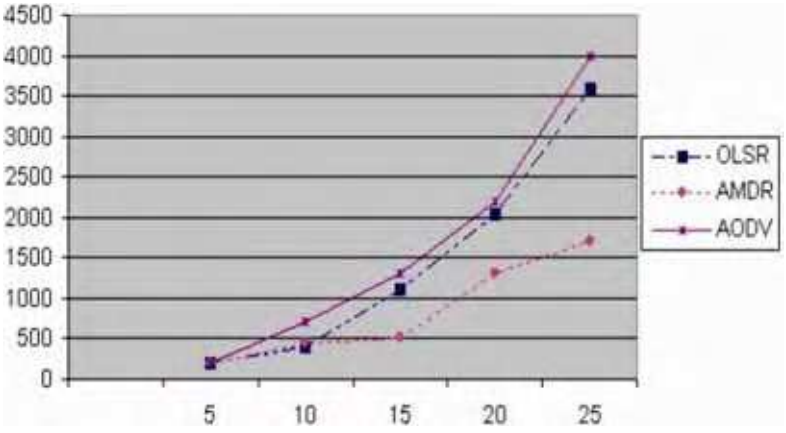

Fig. 7. Packets delay comparison for mobility scenario

Comparing loss rate performance between AODV, AMDR and OLSR, shows in figure 8 that both AMDR and OLSR have, in a low loaded network, the same performance when AODV realises the best performances. However, in a high loaded network (case of 20 or 25 connexions), AODV becomes less good than AMDR and OLSR. We justify such results by the adaptation of AMDR to load changes when AODV needs more route request function.

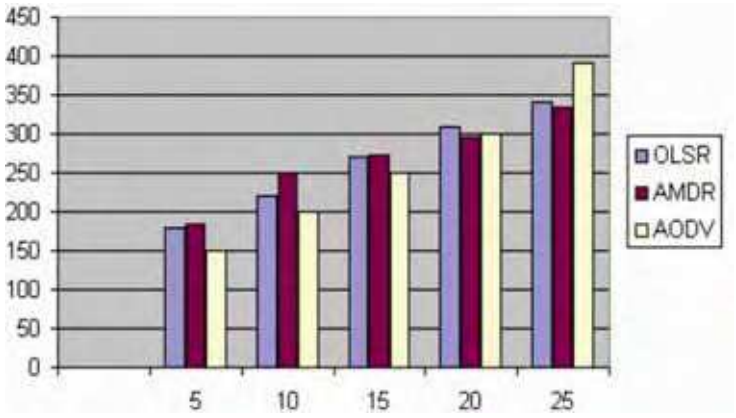

Fig. 8. Loss rate comparison for mobility scenario

6. A system based reinforcement learning in packet scheduling communications network routing.

6.1 Problem formulation

In the dynamic environment the scheduler take the actual evolution of the process into account. It is allowed to make the decisions as the scheduling process actually evolves and more information becomes available. For that, we consider at each router an agent that can make decision. This decision-maker collects information gathered by mobile agents and then decides which action to perform after learning the current situation. We will focus on dynamic technique and will formulate the packet scheduling problem through several routers as a multi-agent Markov Decision Problem (MDP). As Machine learning techniques, we use reinforcement learning to compute a good policy in a multi-agent system (Mellouk & Hoceini, 2005; Hoceini et al., 2005). Simultaneous decision making in a dynamic environment is modelled using multi-agent Markov Decision Processes (MMDPs) (Puterman, 2005). However, learning in multi-agent system suffers from several limitations such the exponential growing of number of states, actions and parameters with the number of agents. In addition, since agents carry out actions simultaneously so they have evolving behaviours, transitions are non-stationary. Since centralized MAS may be considered as a huge MDP, we work with decentralized system where each agent learns individually in environment improved with information gathered by mobile agents.

6.2 Markov decision processes

Markov decision problem (MDP) can be used for modelling the interaction of an agent with its environment. It is defined as a 4-tuple < S, A, P, r > where S is a finite set of states, A is a finite set of actions, P is the transition probability function and r is the reward function. Given a state s and an action a, P(s, a, s') denotes the transition probability of the system to state s' when action a is executed in state s. Hence, the dynamics of the environment can be characterized by the transition probabilities. The process is said to be Markovian when the transition function depends only on the current state and not on any previously traversed states or any previous actions. If the state and action spaces are finite, then it is called a finite

Markov decision process (FMDP). The reward function r is defined as a real function r: S x A x S → đ, where the scalar reinforcement signal r(s, a, s′) is the reward of taking action a in state s and observing state s′ as the next state. A policy π is denoted for a description of behaviours of an agent. It is a function that maps the current state s of the system into an action. The value of a state s under a policy π is the expected discounted sum of rewards obtained following this policy. The action value of a state according to the policy π noted Qπ(s, a) is the expected discounted sum of reward obtained by taking action a in state s and following policy π. We use the reward as feedback to find an optimal policy π* by iteratively refining an initial policy π0.

In a Markov decision process, the objective of the agent is to find a policy π so as to maximize the expected sum of discounted rewards, It has been proved that there exists an optimal policy π* such that for any s □ S, the following Bellman equation holds:

$$V(s, \pi^*) = \max_a \{r(s, a) + \gamma \Sigma_{s'} P(s, a, s') V(s', \pi^*)\} \quad (10)$$

where V(s, π*) is the optimal value for state s. When the transition function is unknown, the Q-learning (Watkins, 1989) is one of the algorithms most used to find an optimal policy. In Q-learning, Q*(s, a) is the total discounted reward obtained in state *s* after performing action *a* and following the optimal policy π*. Then, the above equation becomes:

$$V(s, \pi^*) = \max_a Q^*(s, a) \quad (11)$$

On basis of Q*(s, a) the optimal policy π* can be found by performing an action *a* in state *s* so as to maximise Q*(s, a).

The Q-Learning algorithm builds values of Q(s, a) for all *s* □ S and *a* □ A whose initial values may be arbitrarily chosen. If the agent, after executing an action a, moves from state *s* to *s′* and receives an immediate reward r(s, a), the current Q(s, a) values are updated using the following formula:

$$Q(s, a) \leftarrow (1-\alpha)Q(s, a) + \alpha [r(s, a) + \gamma \max_{a'} Q(s', a')] \quad (12)$$

where α, 0≤ α ≤1 is the learning rate. (Watkins & Dayan, 1992) proved that equation (12) converges to optimal Q*(s, a). At the end of the learning, an optimal policy π* can be derived such as π*(s) = arg $\max_a$ Q*(s, a).

6.3 Multi-agent MDPs

The single MDP and Q-learning are defined for the case where only one action is selected with each iteration. In this case the existence of optimal policy π* is guaranteed. Nevertheless, the formalism can be extended to problems where multiple actions can be carried out simultaneously by several agents (Boutilier, 1999). In this way, we consider n agents each one of them having learned the optimal solution from its own MDP. The effort of these n agents in individual learning is combined to learn the joint optimal policy of this multi-agent MDP.

We define a multi-agent MDP as 4-tuple (S, A, P, R) where the state space is a subset of the joint state space of n agents such that $S = S^1 \times \ldots \times S^n$ in which each S^i is a discrete state space of each agent, the action space is the joint action space of n agents $A = A^1 \times \ldots \times A^n$ in which

a joint action (a^1, a^2, …., a^n) corresponds to the concurrent execution of the actions a^i by each agent i. The transition probabilities and rewards are factorial and defined for all states s, s′ □ S and for all actions a □ A respectively as:

$$P^a_{ss'} = \prod_{i=1}^{n} P(s^i, a^i, s^{i'}) \tag{13}$$

and R is given by the function R: S × A → đ such that:

$$R = \sum_{i=1}^{n} r^i(s^i, a^i, s^{i'}) \tag{14}$$

where $r^i(s^i, a^i, s^{i'})$ is the reward obtained by agent i when it performs action a^i in state s^i and move to state $s^{i'}$.

However, the multi-agent MDP approach has two disadvantages. The learning is centralized, i.e. the task consisting in finding an optimal policy for the group cannot be distributed among the agents. Moreover, the need to consider all the joint actions increases considerably the complexity of learning, since the size of the space of joint actions grows in an exponential way according to the number of agents. We are interested in a decentralized MDP with communication where each agent takes into account only its actions but considers that all the other agents are part of the environment. The communication is governed by mobile agents.

6.4 Mobile agent

An important aspect of multi-agent systems is to construct intelligent agent able of achieving goals in complex environment. They address specifically the behaviour of the agent in its environment changes [Hadeli et al., 2004; Mellouk et al., 2006). Reinforcement learning provides a framework of adaptation of the agent's behaviour according to its environment. As mobile agents we consider an ants' colony. The structure of the model identifies two kinds of agents, their responsibilities, and the way they interact. The structure consists of a scheduler agent that deals with management of queues on the basis of available information (resource capacity) and a resource agent that measures the resource amount and gives this information to the scheduler.

Scheduling in this system is done as follows: Before performing action selection and then scheduling the different queues based on their QoS, each scheduler agent sends ants moving downstream to control actual situation. They gather information about the availability of the resource and then return to the sender agent with the information. On the basis of this information the scheduler agent chooses a schedule and sends ants to reserve the needed resources by deposing pheromone. After that the scheduler agent regularly sends ants to reserve the previously found best resource capacity because if the reservation is not refreshed, the pheromone evaporates after a while. From time to time, the scheduler agent sends ants to survey the possible new (and better) amount of this resource. If they find a better measurement, the scheduler agent reserves the resource that is needed for the new schedule and the old reserving information evaporates. Ants are used also to distribute information and make their current state known throughout the system. The scheduler agent is not restricted to send the ants in one direction only. The ants are sent towards various directions which are directly connected with the router containing this scheduler

agent. The scheduler agent waits until all the ants arrive back with the gathered information and than decides to keep only the ant with the best information, the others terminate. Also, each agent sends ants to distribute information about its current state to the other agents. Every time an ant arrives at a scheduler agent, it gives a reward according to the information that was investigated. This reward is in form of a belief factor which will allow each scheduler agent in the multi-agent system to make update on their scheduling policies. The belief factor is a function of the synthetic pheromone concentration. It reflects the degree of confidence that an agent will consider on the information established by other agents from the same cooperating group. The belief factor might be useful in situations where the information is not reliable due to changes in the environment.

6.4.1 Combining pheromone and Q-learning

At level on each router, the scheduler agent performs a reinforcement learning to schedule the service of queues. This scheduling is done by taking account the current state of the environment provided by the ant agent. During the learning, the Q-function is updated based on the concentration of pheromone in the current state and the neighbour states. The used technique combines Q-learning (Sutton & Barto, 1998) with a synthetic pheromone introducing a belief factor into the update equation. The formula bellow describes the belief factor (Monekosso & Remagnino, 2004):

$$B(s, a) = \frac{\Sigma_{s \in Na}\ \Phi(s)}{\Sigma_{\sigma \in Na}\ \Phi_{max}(\sigma)} \tag{15}$$

where Φ(s) is a synthetic pheromone, a scalar value that integrates the basic dynamic nature of the pheromone, namely aggregation, evaporation and diffusion.

where Φ(s) is a synthetic pheromone, a scalar value that integrates the basic dynamic nature of the pheromone, namely aggregation, evaporation and diffusion.

6.5 Proposed model.

Definition: A decentralized multi-agent MDP is defined as 4-tuple (S, A^i, P^i, r^i), where S is a set of all states, A^i is the set of all actions of agent i, $P^i: S \times A^i \rightarrow \Delta(S)$ is the state transition function where Δ(S) is the set of probability distributions over the set S and $r^i: S \times A^i \times S \rightarrow đ$ is the individual reward such that $r^i(s, a^i, s')$ is the reward obtained by agent i when it performs action a^i in state s and move to state s'.

We define a local policy π^i for each agent i such that:

$$\pi^i : S \rightarrow A^i \tag{16}$$

The expected discounted value function of agent i is the following:

$$V^i(s, \pi^i) = E(\pi^i)[\ r^i \mid s_0 = s] = E(\pi^i)[\ \Sigma_{t=0}^{\infty} \gamma^t r^i_{t+1} \mid s_0 = s] \tag{17}$$

where r^i_t is the immediate reward at time step for agent i and γ is a discount factor.

We consider also Q^i as local Q-function, defined for each state-action pair as:

$$Q^{i}\pi^{i}(s, a^{i}, s') = E(\pi^{i})[\, r^{i} \mid s_0 = s, a^{i}, s'] = r^{i}(s, a^{i}, s') + \gamma \Sigma\, P(s' \mid s, a^{i})\, V^{i}(s', \pi^{i}) \tag{18}$$

The Q-learning update equation adapted to the local decision process according the global state space and modified with synthetic pheromone is given by the following formula:

$$Q^{i}(s, a^{i}) \leftarrow Q^{i}(s, a^{i}) + \alpha \{R + \gamma \max_{a^{i'}} [Q^{i}(s', a^{i'}) + \xi\, B(s', a^{i'})] - Q^{i}(s, a^{i})\} \tag{19}$$

where the parameter ξ is a sigmoid function of time periods such that $\xi \geq 0$. The value of the parameter ξ increases with the number of agents which achieve successfully the current task.

The optimal policy $\pi^{i,*}$ for each agent i can be obtained by using the modified formula (11):

$$V^{i}(s^{1}, \ldots, s^{n}, \pi^{i,*}) = \max_{a^{i}} Q^{i,*}(s^{1}, \ldots, s^{n}, a^{i}) \tag{20}$$

The effect of one agent's action depends on the action taken by others or choosing an action by an agent may restrict actions that can be selected by others. In this case each agent should change its local learned policy in order to achieve a multi-agent global optimal policy. For any global state s = (s1, ….., sn) and any joint action a = (a1, …., an), the optimal action value function Q*of the multi-agent MDP is the sum of the optimal action value functions Qi,*learned by a decentralized multi-agent MDP for each agent. The agents could have different estimations on the optimal state-action values according to the environment. These estimations wi(s, a) can be computed as:

$$w^{i}(s, a^{i})) = \frac{\exp(h^{i}(s, a^{i})/\eta)}{\Sigma_{j=1}^{n} \exp(h^{j}(s, a^{j})/\eta)} \tag{21}$$

where $h^{i}(s, a^{i})$ is the number of estimations' updates achieved by agent i for $(s^{1}, \ldots.., s^{n}, a^{i})$ and η is an adjustable parameter. So,

$$Q^{*}(s^{1}, .., s^{n}, a^{1}, \ldots., a^{n}) = \sum_{i=1}^{n} w^{i}(s, a^{i})\, Q^{i,*}(s^{1}, \ldots.., s^{n}, a^{i}) \tag{22}$$

When the learning is finished, an optimal policy can be directly derived from the optimal action value Q*(s, a) by:

$$\pi^{*}(s^{1}, \ldots.., s^{n}) = \arg\max_{(a^{1}, \ldots., a^{n})} Q^{*}(s^{1}, \ldots.., s^{n}, a^{1}, \ldots., a^{n}) \tag{23}$$

6.5.1 Learning algorithm

The model of the environment's dynamics, the transition probabilities and rewards is unknown in learning of a single agent MDP and consequently the subsequent multi-agent MDP. So, the learning of the optimal solution of a problem is done by agents through interaction with the environment.

We describe the global scheduling problem as a multi-agent MDPs in a decentralized approach. We derive a multi-agent learning algorithm from traditional reinforcement learning method based on Markov decision process to construct global solutions from

solutions to the individual MDPs. In this case, we assume that the agents work independently by making their trials in the simulated environment. The system state s is described by the space state of all agents; an action a^i describes which queue is serviced in the time slot. Therefore, the goal of scheduling is to find an optimal policy π^* such that the rewards accumulated are maximized.

The proposed algorithm converges to the optimal policy and optimal action value function for the multi-agent MDP since the difference between standard multi-agent and our decentralized multi-agent MDP model is the global states space for each action set A^i of an agent i.

The rewards may depend both on the current situation and on the selected action and express the desired optimization goal. In our approach, the global action a is a vector of single action made by distributed agents each associated with one of the n routers.

Learning here means iteratively improving the selection policy according to the maximization of the global reward. This is done by a Q-learning rule adapted to the local selection process (eq. 19). The learning rule relates the local scheduling process of agent i to the global optimization goal by considering the global reward R.

If Q^i converges the $Q^{i,*}$ predicts if the action a^i would be selected next. This action will be chosen by a policy greedy.

In a single-agent learning case, Q-learning converges to the optimal action independent of the action selection strategy. However, in a multi-agent situation, the action selection strategy becomes crucial for convergence to any joint action. A major challenge in defining a suitable strategy for the selection of actions is to make a trade-off between exploration of new policies and exploitation of existing policies.

In our research, we use a Boltzmann distribution (Katanakis & Kudenko, 2002) for the probability of choosing an action by each agent. In this strategy, each agent derive a scheduling policy from the current value of Q^i matrix and then update Q^i using the rewards from actions chosen by the current scheduling policy according to a probability distribution $\pi^i(s, a^i)$:

$$\pi^i(s, a^i) = \frac{\exp(Q^i(s, a^i)/T)}{\sum_{a^{i'} \in A^i} \exp(Q^i(s, a^{i'})/T)} \tag{24}$$

where exp is the exponential function and T is a parameter called temperature. The value of the temperature determines the possibility for an agent to balance between exploration and exploitation. For high temperature, even when an expected value of a given action is high, an agent may still choose an action that appears less desirable. In contrast, low temperature values support more exploitation, as the agent is more expected to have discovered the true estimates of different actions. The three important settings for the temperature are the initial value, the rate of decrease and the number of steps until it reaches its lowest limit. This lower limit must be set to a value close enough to 0 to allow the learners to converge by stopping their exploration.

In our work, we start with a very high value for the temperature to force the agents to make random moves until the temperature reaches a low enough value to play a part in the

learning. This is done when the agents are gathering information about the environment or the other agents. The temperature defined as a function of iterations is given by:

$$T(x) = (e^{-sx} * T_{max}) + 1 \tag{25}$$

where x is the iteration number, s is the rate of decay and T_{max} is the starting temperature.

In this section we present an algorithm called DEMAL (Decentralized Multi-Agent Learning) that uses Q-learning and decentralization on the level of the action.

Algorithm DEMAL

Repeat

Initialize $s = (s^1, ..., s^n)$

  Repeat

    For each agent i

    Choose a^i using Boltzman formula

    Take action a^i, observe reward r^i and state s'

    $Q^i(s, a^i) \leftarrow Q^i(s, a^i) + \alpha\{R + \gamma \max_{a^{i'}} [Q^i(s', a^{i'}) + \xi B(s', a^{i'})] - Q^i(s, a^i)\}$

    $s \leftarrow s'$

  until s is terminal

until algorithm converges

6.5.2 Approximation

In this model, we define the optimal policy by using optimal action value function Q*of the multi-agent MDP as the sum of the optimal action value functions $Q^{i,}$*learned by a decentralized multi-agent MDP for each agent. We can apply this learning algorithm directly to solve multi-agent MDP but this method would not be very efficient because of the state and action spaces dimension that can be huge since they increase exponentially with the number of agents. This increase influences the complexity of the learning algorithm since this one depends on the number of states and actions. In tabular methods, Q^i values were assumed to be stored in lookup tables which can be large since each one depends on the number of states and actions. In order to approximate the tabular Q^i function, a feed-forward multilayer neural network like the MLP can be used. Its structure is a three-layered model containing an input layer, a hidden layer, and an output layer. The input variables of the NN are the states of the system and the set of actions A^i for each agent, which have n and m dimensions respectively. The output of the network corresponds to the Q^i value for the current states and actions. Each node in every layer subsequently calculates its activation as the weighted sum over its inputs according to:

$$\mu_j = \sum_i x_i w_{ij} \tag{26}$$

where x_i is the i^{th} input to node j and w_{ij} is the weight of the synapse connecting node i with node j from a higher level. A common activation function for MLPs is the sigmoidal function which is applied to the hidden layer and which will also be used for our specific implementation.

$$\sigma(y) = (1+e^{-y})^{-1} \tag{27}$$

The technique used to learn the Q-function is the back propagation algorithm [18, 19, 20]. The weight update can be expressed by gradient descent, where the weights are added, at each iteration by the value of:

$$\Delta w = -\eta \frac{\partial E}{\partial w} \tag{28}$$

where η is a small learning rate.
The network error is backpropagated through the network from output to input space, where at each node we aim to match the node's output o as closely to the target t as possible. The network error δ_k for the output nodes k, can be calculated as:

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k) \tag{29}$$

For the hidden nodes h δ_h can be calculated as:

$$\delta_h \leftarrow o_h(1-o_h) \sum_{k \in \text{outputs}} w_{kh}\delta_k \tag{30}$$

The error terms are directly derived from the mean-squared error (MSE) which is defined according to formula:

$$E(w) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2 \tag{31}$$

The approximate state-action value function Q^i is proven to converge to the optimal function $Q^{i,*}$ (and hence π^i to $\pi^{i,*}$) given certain technical restrictions on learning rates.

6.6 Performance evaluation.

We carried out our evaluation in two stages. The first stage consists to realizing the scheduling on level of one router. For that, we just consider in this stage a single agent MDP. In the second stage, we solve the whole problem which concerns the optimization of the end to end queuing delay through the global scheduling. Hence, we apply our algorithm based on the multi-agent MDP in its decentralized version. We start to describe the context of the first phase.
In each router, an agent deals with scheduling N classes of traffic, where each traffic class has its own queue q_i, for i = 1…N. Let q_N denote the queue for best-effort traffic, which has no predefined delay requirements and R_1, R_2,……,R_{N-1} denote the delay requirements of the remaining classes. Let M_1, M_2,…,M_{N-1} denote the measured delays of these classes observed over the last P packets. We assume that all packets have a fixed size. We consider also that a fixed length timeslot is required for transmitting a packet and at most one packet can be

serviced at each timeslot. The arrival of packets is described by a Bernoulli process, where the mean arrival rate μ_i for q_i is represented by the probability of a packet arriving for q_i in any timeslot. Our goal is to learn a scheduling policy that ensures $M_i \leq R_i$ for i=1,…,N-1. For the simulation, we used a three queue system that is Q_1, Q_2 and the best effort queue and the parameters of this simulation are given in table 3. We have considered two cases according to the availability of resource. For investigating the case where the output link capacity of the router is sufficient we assume that this capacity is 500 Kbps. In this case, a sufficient amount of capacity is provided for each queue so our algorithm satisfied the mean delay requirements for Q_1 and Q_2 (see fig.9). We have also observed that our approach requires 1.5 x 10^4 timeslots in terms of convergence time. In the second scenario (table 4) we consider the case where the output link capacity of the router is small and equal to 300 Kbps. The result of this case is shown in fig. 10. We observe that an allocation of a share of the available bandwidth is given to the delay-sensitive class Q_1 and then to Q_2 and the best effort queue. This is carried out on the basis of information gathered by a mobile agent. Also, $\varepsilon = 0.2$; $\gamma = 0.5$.

In the second part of our evaluation, we consider a network with several routers connected to each other like in (Bourenane et al., 2007). We introduce also the mobile agents to gather and distribute necessary and complete information in order to help the agents to update their knowledge of the environment. The figures 10 and 11 show that in both scenarios, the presence of mobile agents provides a better queuing delay for all routers.

Queue	Arrival Rate (packets/timeslot)	Mean Delay Requirement	eBi Kbps
Q_1	0.30	8	64
Q_2	0.20	2	128
BE	0.40	Best-effort	Best-effort

Table 3. Simulation Parameters: Scenario 1

Queue	Arrival Rate packets/timeslot	Mean Delay Requirement	eBi Kbps
Q_1	0.30	4	128
Q_2	0.20	6	256
BE	0.40	BE	BE

Table 4. Simulation Parameters: Scenario 2

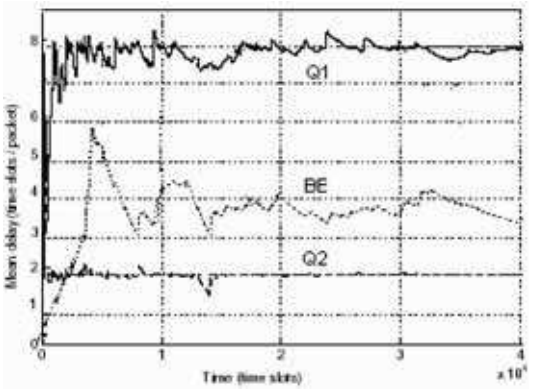

Fig. 9. Mean Delay for three classes

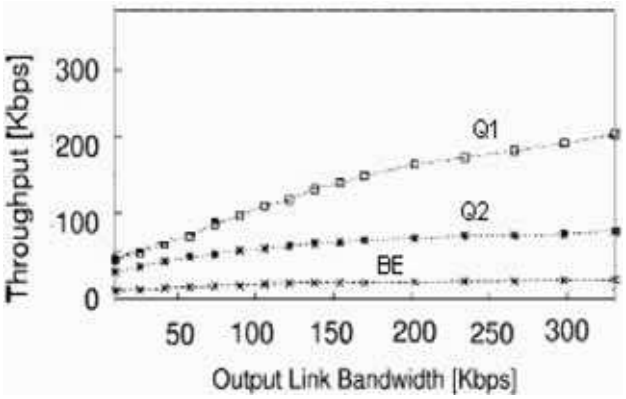

Fig. 10. Average throughput of three queues

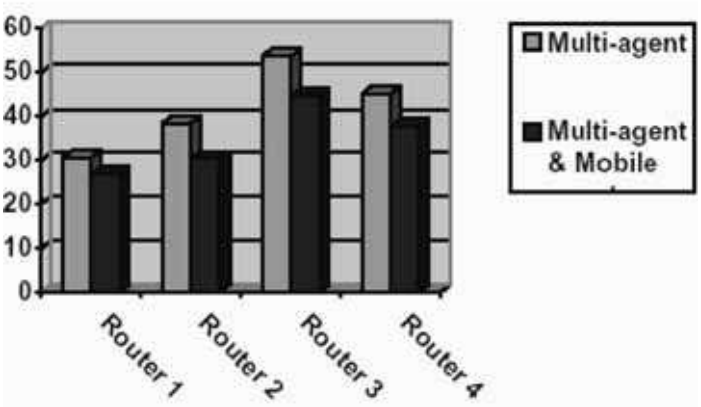

Fig. 11. Average queuing delay (scenario 1)

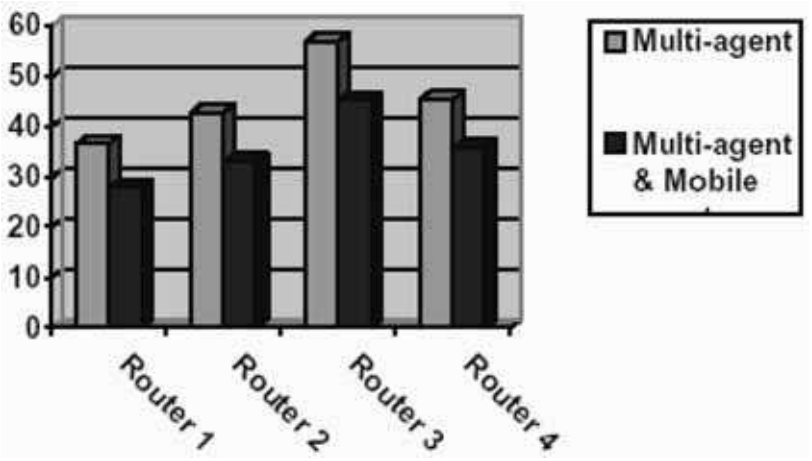

Fig. 12. Average queuing delay (scenario 2)

7. Conclusion

Due to the growing needs in telecommunications (VoD, Video-Conference, VoIP, etc.) and the diversity of transported flows, communication networks does not meet the requirements of the future integrated-service networks that carry multimedia data traffic with a high QoS. The main drivers of this evolution are the continuous growth of the bandwidth requests, the promise of cost improvements and finally the possibility of increasing profits by offering new services. First, it does not support resource reservation which is primordial to guarantee an end-to-end Qos (bounded delay, bounded delay jitter, and/or bounded loss ratio). Second, data packets may be subjected to unpredictable delays and thus may arrive at their destination after the expiration time, which is undesirable for continuous real-time media. In this context, for optimizing the financial investment on their networks, operators must use the same support for transporting all the flows. Therefore, it is necessary to develop a high quality control mechanism to check the network traffic load and ensure QoS requirements. It's clear that the integration of these QoS parameters increases the complexity of the used algorithms. Anyway, there will be QoS relevant technological challenges in the emerging hybrid networks which mixed several different types of networks (wireless, broadcast, mobile, fixed, etc.).

QoS management in networking has been a topic of extensive research in a last decade. As the Internet network is managed on a best effort packet routing, QoS assurance has always been an open issue. Because the majority of past Internet applications (email, web browsing, etc.) do not used strong QoS needs, this issue is somewhat made less urgent in the past. Today, with the development of internet real-time application and the convergence of voice and data networks, it is necessary to develop a high quality control mechanism to check the network traffic load and ensure QoS requirements. Constraints imposed by QoS requirements, such as bandwidth, delay, or loss, are referred to as QoS constraints, and the associated routing is referred to as QoS routing which is a part of Constrained-Based Routing (CBR).

Several methods have been proposed to integrate QoS constraints and to reduce their complexity. However, for a network node to be able to make an optimal routing decision, according to relevant performance criteria, it requires not only up-to-date and complete knowledge of the state of the entire network but also an accurate prediction of the network dynamics during propagation of the message through the network. This problem is naturally formulated as a dynamic programming problem, which, however, is too complex to be solved exactly. Reinforcement learning (RL) is used to approximate the value function of dynamic programming. In these algorithms, the environment is modeled as stochastic, so routing algorithms can take into account the dynamics of the network. However no model of dynamics is assumed to be given.

The second part of this chapter was devoted the study of our system based on reinforcement learning for different network communication domains.

First of all, we have focused our attention in some special kind of Constrained Based Routing in wired networks which we called QoS self-optimization Routing. Our algorithm is based on a multi-path routing technique combined with the Q-Routing algorithm and is tested for improving distribution of traffic on N-Best paths. The learning algorithm is based on founding N-Best paths in term of hops router and the minimization of the average packet delivery time on these paths. The performance of our algorithm is evaluated experimentally with OPNET simulator for different levels of traffic's load and compared to standard optimal path routing algorithms. Our approach prove superior to a classical algorithms and is able to route efficiently in networks even when critical aspects are allowed to vary dynamically. The fact that the reinforcement signal is continuously updated, parameter's adaptation of our system take into account variations of traffic.

Secondary, we study the use of reinforcement leaning in AMDR algorithm in the case of Mobile Ad Hoc Networks. It is shown from simulation results that combining proactive exploration agents with the on-demand route discovery mechanism, the AMDR routing algorithm would give reduced end-to-end delay and route discovery latency with high connectivity. This is ensured because of the availability of alternative routes in our algorithm. The alone case where our approach can provide more important delay is the first connection where any route is yet established. On the other hand, the use of delay-MPR mechanism, guarantees that the overhead generated will be reduced.

In the last part, we address the problem of optimizing the queuing delay in several routers of a network, through a global packet scheduling. We formulated this problem as a multi-agent MDP and used the decentralized version since multi-agent MDPs usually have huge state and action spaces (because they grow exponentially with the number of agents). This decentralized MDP is improved by ant-like mobile agent on the level of each router to guarantee a global view of the system's state. We presented a modified Q-learning algorithm in the decentralized approach. Our simulation shows that the proposed approach leads to better results than when the multi-agent system acts alone.

Finally, extensions of the framework for using these techniques across hybrid networks to achieve end-to-end QoS needs to be investigated, in particular on large scalable networks. Another challenging area concerns the composite metric used in routing packets (especially residual bandwidth) which is so complex and the conditioning of different models in order to take into account other parameters like the information type of each flow packet (real-time, VBR, …).

8. Acknowledgments

The work presented here is a part of QoSiDiN SCTIC LISSI research activities team, especially in the scope of the supervising Malika Bourenane, Saida Ziane and Said Hoceini PhD's thesis. I would like to thank them for their support and their suggestions.

9. References

Baras, J.S., Mehta, H. (2003). A Probabilistic Emergent Routing Algorithm (PERA) for Mobile Ad Hoc Networks, *Proceedings of WiOpt '03: Modeling and Optimization in Mobile, AdHoc and Wireless Networks*, Sophia-Antipolis, France.

Bourenane M, Mellouk A., Benhamamouche D., (2007). A QoS-based scheduling by Neurodynamic Learning. *System and Information Sciences* Journal, Vol. 2, n° 2, pp 138-144.

Boutilier C., (1999). Sequential Optimality and Coordination in Multiagent Systems, *Proceedings of IJCAI*, pp. 478-485.

Boyan, J. A., Littman, M. L., (1994). Packet routing in dynamically changing networks: A reinforcement learning approach, Advances in Neural Information Processing Systems 6, Morgan Kaufmann, San Francisco, CA, pp. 671-678.

Christopher, W. , Dayan, J.C.H., (1992). Q-learning. *Machine Learning*, vol. 3, pp. 279-292.

Dorigo, M., Stüzle, T., (2004). *Ant Colony Optimization*, MIT Press, Cambridge, MA.

Eppstein, D., (1999). Finding the K shortest paths, *SLAM J. Computing* 28:0, pp. 652-673.

Gallager, R.G. (1977). A minimum delay routing algorithm using distributed computations. *IEEE Transactions on Communications*, 25(1), 73-85.

Garey, M.R., Jhonson, D. S., (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

Gelenbe, E., Lent, L., Xu, Z, (2002). Networking with Cognitive Packets, *Proc. ICANN 2002*, Madrid, Spain, pp. 27-30.

Grover, W.D. (2003). *Mesh-based Survivable Transport Networks: Options and Strategies for Optical, MPLS, SONET and ATM Networking*. Ed. Prentice Hall PTR.

Hadeli, M., Valckenaers, P., Kollingbaum, M., Van Brussel, H., (2004). Multi-Agent Coordination and Control Using Stigmergy. *Computers in Industr,* vol. 53, pp. 75 – 96.

Hoceini S., Mellouk A., Amirat Y., (2005). A New QoS Routing Algorithm in Dynamic Traffic's Network : N-Best Routing Policy based on Reinforcement Learning, *International Transactions on Computer Science and Engineering (GESTS)*, vol. 8, pp 25-36.

Jaffe, J.M., (1984). Algorithms for Finding Paths with Multiple Constraints, *IEEE Networks*, Vol. 14, pp. 95-116.

Kapetanakis S., Kudenko, D., (2002). Reinforcement learning of coordination in cooperative multi-agent systems. *Proceedings of AAAI 2002*, pp 326–331.

Korkmaz, T., Krunz, M., (2001). A randomized Algorithm for Findind a Path Subject to Multiple QoS Requirements, *Computer Networks*, Vol. 36, pp. 251-268.

Kuipers, F. A., Korkmaz, T., Krunz, M., Van Mieghem, P., (2004). Performance Evaluation of Constraint-Based Path Selection Algorithms, *IEEE Network*, Vol. 18, No. 5, pp. 16-22.

Kuipers, F. A., P. Van Mieghem, (2005). Conditions that impact the Complexity of QoS Routing, *IEEE/ACM Transaction on Networking*, Vol. 13, No. 4, pp. 717-730.

Masip-Bruin X. et al., (2006). Research challenges in QoS Routing, *Computer Communications*, Vol. 29, pp. 563-581.

Mellouk, A., Hoceini S., (2005). A Reinforcement Learning Approach for QoS Based Routing Packets in Integrated Service Web Based Systems, *Lecture Notes in Artificial Intelligence*, LNAI 3528, Springer-Heidelberg GmbH, vol. 3528, pp 299-305.

Mellouk, A., Hoceini, S., Amirat, Y., (2006a). Adaptive Quality of Service Based Routing Approaches: Development of a Neuro-Dynamic State-Dependent Reinforcement Learning Algorithm, *International Journal of Communication Systems*, Ed. Wiley InterSciences, on-line september, to appear.

Mellouk A., Hoceini S., Larynouna S., (2006b). Self-Optimization Quality of Service by Adaptive Routing in Dynamic Communication Networks, *International Transactions on Systems Science and Applications*, Xiaglow UK Ed., vol. 2, n°3, pp 265-273.

Mellouk, A., Hoceini, S., Cheurfa, M., (2007a). Reinforcing Probabilistic Selective Quality of service Routes in Dynamic Heterogeneous Networks, *Journal of Computer Communication*, Elsevier Ed., to appear, on line.

Mellouk, A., Lorenz, P., Boukerche, A., Lee, M.H., (2007b). Impact of Adaptive Quality of Service Based Routing Algorithms in the next generation heterogeneous networks, *IEEE Communication Magazine*, IEEE Press, vol. 45, n°2, pp. 65-66.

Monekosso, N., Remagnino P., (2004). Analysis and performance evaluation of the pheromone-Q-learning algorithm, *Expert Systems* 21 (2), pp 80–91.

Naimi, A.M., Jacquet, P., (2004). One Hop Delay Estimation In 802.11 Ad Hoc Networks Using The OLSR Protocol. *Research Report INRIA*, N° 5327.

Naimi, A.M., (2005). *Délai et Routage dans les réseaux ad hoc 802.11*, PHD Thesis, INRIA Rocquencourt, France.

Nguyen, D.Q., Minet, P., (2006). Analysis of Multipoint Relays Selection in the OLSR Routing Protocol with and without QoS Support, *Research Report INRIA*, N° 6067.

Ozdaglar, A.E., Bertsekas, D.P. (2003). Optimal Solution of Integer Multicommodity Flow Problem with Application in Optical Networks. *Proc. Of Symposium on Global Optimisation*, June, 411-435.

Puterman, M., (2005). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley-Interscience.

Quoitin, B., Uhlig, S., (2005). Modeling the Routing of an Automous System with C-BGP, *IEEE Network*, Vol. 19, No.6, pp. 12-19.

Sahni, S., (2005). *Data Structures, Algorithms, and Applications in C++*, second Edition, Silicon Press.

Song, M., Sahni, S., (2006). Approximation Algorithms for Multiconstrained Quality-of-Service Routing, *IEEE Transaction on Computers*, Vol. 55, No. 8, pp. 1048-1056.

Sutton, R.S., Barto, A.G. (1997). *Reinforcement Learning*. Ed. MIT Press.

Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*, MIT Press/Bradford Books.

Yanuzzi, M., Masip-Bruin, X., Bonaventure, O., (2005). Open Issues in Interdomain Routing : A Survey, *IEEE Network*, Vol. 19, No.6, pp. 49-56.

Ziane, S., Mellouk, A. (2006). A Swarm Quality of Service Based Multi-Path Routing Algorithm (SAMRA) for Wireless Ad Hoc Networks, *International Review on Computers and Software Journal*, Vol.1, n°1., pp. 11-20.

Reinforcement Learning

Edited by Cornelius Weber, Mark Elshaw and Norbert Michael Mayer

ISBN 978-3-902613-14-1
Hard cover, 424 pages
Publisher I-Tech Education and Publishing
Published online 01, January, 2008
Published in print edition January, 2008

Brains rule the world, and brain-like computation is increasingly used in computers and electronic devices. Brain-like computation is about processing and interpreting data or directly putting forward and performing actions. Learning is a very important aspect. This book is on reinforcement learning which involves performing actions to achieve a goal. The first 11 chapters of this book describe and extend the scope of reinforcement learning. The remaining 11 chapters show that there is already wide usage in numerous fields. Reinforcement learning can tackle control tasks that are too complex for traditional, hand-designed, non-learning controllers. As learning computers can deal with technical complexities, the tasks of human operators remain to specify goals on increasingly higher levels. This book shows that reinforcement learning is a very dynamic area in terms of theory and applications and it shall stimulate and encourage new research in this field.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Abdelhamid Mellouk (2008). Inductive Approaches Based on Trial/Error Paradigm for Communications Network, Reinforcement Learning, Cornelius Weber, Mark Elshaw and Norbert Michael Mayer (Ed.), ISBN: 978-3-902613-14-1, InTech, Available from: http://www.intechopen.com/books/reinforcement_learning/inductive_approaches_based_on_trial_error_paradigm_for_communications_network

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821