SAPIENZA
UNIVERSITÀ DI ROMA

Dottorato di Ricerca in Ingegneria dei Sistemi
XXI Ciclo

Tesi di Dottorato

Reinforcement Learning Algorithms for Technology Independent Resource Management in Next Generation Networks: Connection Admission Control Procedures

Relatore
Prof. Francesco Delli Priscoli

Candidato
Ing. Silvano Mignanti

Coordinatore del Dottorato
Prof. Carlo Bruni

ANNO ACCADEMICO 2007-2008

Ai ragazzi del Labreti,

ricordate sempre:

siete nel mio cuore!

Index

Index of figures

Index of tables

Chapter 1. Introduction

In the last years the telecommunication field is one of which had continuous technological advancements: in particular both new services and new products are continuously introduced to the users. This trend brings to several major problems: first of all, new technologies require new network infrastructures which have to be connected to existing ones; subsequently, interoperability of different technologies becomes a problem. Furthermore, often new technologies are introduced following independent standardization ways, augmenting incompatibilities (not only new technologies with old ones, but also among new technologies themselves): one of the last examples is the case of Wi-Fi and HiperLAN.

These problems affect both users and network service providers; in particular, for these last ones to invest in a new technology implies a wise strategy planning for the short, medium and long term, mainly because new infrastructures, software and solutions valid for a certain technology usually cannot be reused for a different one.

In order to overcome most of these problems, currently new technologies follow a long standardization period made in conjunction with tests: a lot of standardization fora (e.g.: 3GPP[1][2][3][4], ETSI[5][6][7], ITU-T [8], TISPAN [9]) are currently involved in the standardization of the last and the next network technologies. Most of the network solutions are emerging in adherence with the “Converged Network” concept which could be defined as the possibility to unify the classical telephone network, mobile networks and the internet in a single infrastructure, which offers Quality of Service (QoS) and high reliability. This new kind of network is also known as Next Generation Network (NGN [10]). NGN model is based on the decoupling of services and networks allowing them to evolve independently in a seamless fashion. It takes into account the formal separation of the network architecture into different planes to offer a platform for creating and managing different services. The majority of the European research projects ([11] [12], [13], [14], [15], [16]) focus on the definition of a converged architectural model that is transport-independent and service-independent. This goal is achieved decomposing the network functionalities in three main planes: Service Control Plane (SCP), Network Resource Control Plane (NRCP) and Transport Plane (TP). SCP is composed of a set of functional entities offering services under the control of a service provider which share a set of policies and common technologies. NRCP is constituted by a set of functional entities in charge of managing QoS policies received from SCP and forcing QoS mechanisms to Transport Plane. TP is composed of a set of transport resources sharing a common set of policies, QoS mechanisms and transport technologies under the control of a transport network operator.

Among other problems, one of the most challenging ones, from a control-theory point of view, is the problem of the resource management: new networks, even if having more bandwidth capabilities, lower bit error rates and so on, in any case have finite resources and an optimal use of them is advantageous both for the network operators and for the users. Network resource management comprises a lot of functionalities, e.g. Routing, Connection Admission Control (CAC), Traffic control each of which having challenging problems to face with. In the ambit of this thesis, the problem of Connection Admission Control was selected to propose a possible solution.

In the next chapter, a brief introduction to diverse approaches to the CAC problem solution is presented. The CAC functionalities are provided by the NRCP to the upper layer of the NGN architecture. The target audience of this solution are NGN providers interested to maximize their profit in resource-limited networks (i.e. wireless networks) guaranteeing Quality of Service by means of a new contractual probabilistic QoS assurance constraints.

In the context of my work some objectives have driven the design of the CAC solution. It must be decoupled from the other Resource Management procedures and must be technology independent, allowing network operators to update their transport technologies without modifying the resource control functions. The CAC should guarantee to the network operator the maximum income, preserving the QoS constrains. It means that on one hand the network resources should be exploited in order to optimally use the available resources and on the other hand the contractual QoS should be assured to all the already accepted end-users connections. The CAC must adapt to different network scenarios, thus two of its objectives are the robustness and the independence from the specific statistical behaviour of the traffic: the CAC algorithm should detect and react to the changes of the environmental conditions.

The thesis is organized as follows:

- chapter two about state of the art regarding Resource Management and, in particular to the CAC problem and solutions, and to Semi Markov Decision Processes;
- chapter three containing an overview about Reinforcement Learning;
- chapter four introducing adopted RL algorithms;
- chapter five depicting RL algorithm implementation and simulation tools;
- chapter six containing simulation results and explanation;
- a chapter with conclusions;
- an annex containing some information regarding the OMEGA project.

Chapter 2. State of the Art in Resource Management in Next Generation Networks

2.1. *Introduction to the Connection Admission Control (CAC) problem*

The CAC problem occurs whenever a new flow asks to be accepted requiring Quality of Service (QoS) parameter to be guaranteed. The easiest way to introduce the problem of CAC is a simple example. Imagine to have a certain link (wireless or wired is indifferent), which, in a real scenario (i.e., not an ideal one), even considering all other problems connected to a link (e.g. packet loss, delay, jitter) minor in a first instance, will have, in any case, a finite bandwidth. Each new connection wanting to pass through that link will require a certain amount of bandwidth: it is clear, after a certain number of new call arrivals, the available bandwidth will end and further new calls could not be accepted. The easiest solution, which is also the most intuitive one, is to accept new calls until there is enough available bandwidth, then start to reject new calls until new bandwidth is made available after an ongoing call ends. This solution, called the Peak-based solution (or also called Greedy Algorithm), incredibly, is the most used one in actual networks. If every new call requires the same amount of bandwidth and produces, to the network provider, the same income, the Peak-based solution is the optimal one: this was the scenario of the dawn of telecommunications. Today, instead, usually they exist different types of users trying to get access to a network link, each of which having different QoS requirements and guaranteeing to the network operator a different income. This brings to a much more complicated scenario where the Peak-based solution is no more the optimal one, as demonstrated by the following example.

In order to introduce a common understanding, the literature defines Class of Service (CoS) a certain type of service requiring certain QoS values and providing to the network operator a certain revenue. In this extremely simplified example, we define two different classes of service, characterized by the required bandwidth and the network operator's revenue.

CoS_1:

Requested Bandwidth = 100 Kbps

Network's operator revenue = 2 €

CoS_2:

Requested Bandwidth = 300 Kbps

Network's operator revenue = 7 €

Suppose the total available bandwidth is 2Mb and that there are currently 4 active connections of CoS_1 and one active connection of CoS_2, so that the total occupied bandwidth is $4 \cdot 100 + 1 \cdot 300 = 700$ Kbps. Suppose next four calls follows this order: CoS_1, CoS_2, Cos_1, CoS_1. The behaviour of the Greedy algorithm will be the following: accept the first call of CoS_1 (occupied bandwidth = 800Kbps, so that free bandwidth = 200Kbps), reject the call of CoS_2 (not enough remaining bandwidth), accept the second and the third calls of CoS_1. Summarizing, the total revenue of the network operator is $3 \cdot 2 = 6$ €. A better algorithm will have left space for the call of CoS_2: in fact, rejecting the first call of CoS_1, it had be possible to accept the call of CoS_2, which had been resulted in a combined revenue of 7€. A possible critic is that an algorithm could not know a priori the list of forthcoming calls: so that it could seem impossible for an algorithm to produce better results of the Greedy one. A lot of works demonstrated this is not true: summarizing their results, even if it is not possible to know a priori next calls, in usual scenarios the distribution of new calls is not completely unknown, rather new calls usual have well known distribution. Knowing these distributions it is possible to forecast the progress of new calls and so decide based on these forecast, “betting” if it will be convenient, or not, to accept each new call. The idea is simple: if a better call (in the sense of a call of a “better” CoS) is expected than the one just arrived, if accepting the current one will prevent the possibility to accept the better one, it will be better to reject it. Obviously next call could arrive much later than expected one or could be not of the desired CoS, so that having rejected results to be not the best solution, in this case: we have to highlight that such a kind of algorithm’s behaviour “bets” on the future. Being so that an algorithm based on “bets”, it could be, in the short term, it could work worst than the Greedy one, but, if the knowledge of new calls distribution is sufficiently right, on the long run it definitely will work better than the Greedy one.

2.2. *Approaches to the solution of the CAC problem*

The above mentioned example was extremely simple and passed over a lot of possible issues, details and so on: its only aim was to introduce the problem in the most comprehensible way as possible. More precision is necessary to introduce the different approaches to the solution of the problem of CAC. The most common way to differentiate different approaches to the CAC problem is to distinguish these based on the information an algorithm can have access to in order to function.

If it is available a measurement system, which is able to provide diverse measurements on the network link (e.g.: bit error rate, actually occupied and available bandwidth, jitter, delay, ...) it is possible to use these measures to adopt so called “measurement based” CAC algorithms.

If such a measurement system is not available, but it is well known the topology of the network (the graph of the underlying network is known to the algorithm), it is possible to use this knowledge to adopt so called “topology based” CAC algorithms.

Finally, if neither measurements are available nor it is known the topology of the network it is anyhow possible to perform CAC using the so called “model based” algorithm. In this last group of algorithms, the network is treated as a black box: just few information are known (e.g., total nominal available bandwidth, CoS of the incoming call and QoS parameters and rewards of each CoS), so that these algorithms well adapt to challenging scenarios were it is not possible to have sufficiently precise measures or network topology varies during the time (e.g. wireless networks). Furthermore, not requiring complex measurement systems, such algorithm are less expensive to be adopted by network operators than the first ones.

The class of model based CAC algorithms is quite large, comprehending a lot of diverse approaches: it is important to highlight that the Greedy CAC algorithm is part of the model based ones. The approach proposed in this thesis formulated the CAC problem as a Semi-Markov Decision Process and used an on-line Reinforcement Learning algorithm to learn the model of the environment from experience, in order to be able to perform the CAC. Next paragraph briefly introduces Markov Decision Processes, while to the Reinforcement Learning is dedicated the next Chapter.

2.3. *Introduction to Markov Property and Markov Decision Processes*

In order to introduce Markov processes we previously need some definitions. In a typical telecommunication scenario we can distinguish several actors:

- some callers and receivers, which are the end points of a communications;
- an *environment*, which is the entity where it is possible to transmit the signal (from

callers to receivers and vice versa), and which has a *state*, indicating the main properties the environment has which are useful to perform the CAC decisions;

- an *agent*, which is the entity that performs the decisions (*actions*) to determine where to accept or not a certain call; its decisions are based on the information contained in the state and furthermore, each action brings the environment to a new state.

Being a little bit more precise, at each time *t* the environment is in a certain state *s*: because the environment passes for a number of states during the time, in order to avoid confusion, we define it s_t, which means the state at time *t*. When it happens a new event (either a new call or a call closure), which we call e_t (i.e. the event when the state is s_t), the agent should decide whether to accept or not the new call, so that it takes a decision and performs an action *a*. In particular, if the environment is in the state s_t we call the action a_t, which means the action took at state s_t. So, starting from state s_t and taking the action a_t the environment arrives to the new state s_{t+1}, which in general is different from s_t (but it could also happen they are equal).

Focusing on the state, it could contain a number of information, but not all of them are useful the *agent* to perform the CAC algorithm. A common assumption is to assume that the state is given by some pre-processing system that is nominally part of the environment: the main concern in this thesis is not with designing or detecting the state signal, but with deciding what action to take as a function of whatever state signal is available. Entering in more details, the state signal should not be expected to inform the agent of everything about the environment, or even everything that would be useful to it in making decisions. If the agent is playing blackjack, we should not expect it to know what the next card in the deck is. If the agent is answering the phone, we should not expect it to know in advance who the caller is. In all of these cases there is hidden state information in the environment, and that information would be useful if the agent knew it, but the agent cannot know it because it has not the possibility to measure that information. An important property the state should have is to have memory of the past: in particular it could be very useful to have a state signal that summarizes past information compactly, yet in such a way that all relevant information is retained. This normally requires more than the immediate measurements, but never more than the complete history of all past sensations. A state signal that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property [17] (a formal definition follows below). For example, a checkers position - the current configuration of all the pieces on the board - would serve as a Markov state because it summarizes everything important about the complete sequence of positions that led to it. Much of the information about the sequence is lost, but all that really matters for the future of the game are retained. Similarly, the current position and velocity of a cannonball is all that matters for its future flight. It doesn't matter how that position and velocity came about. This is sometimes also referred to as an "independence of path" property because all that matters is in the current state signal; its meaning is independent of the "path", or history, of signals that have led up to it. Consider now a sequence of states $(s_{0,}s_1,...,s_t)$ and a sequence of actions $(a_{0,}a_1,...,a_{t-1})$ having brought the state from the state s_0 at time t_0 to the state s_t at time t. Now consider the probability the next state, s_{t+1}, to be equal to a certain state s'; this usually depends from the list of previous states and actions and the current action a_t i.e.:

$$\Pr\{s_{t+1} = s'\} = \Pr\{s_{t+1} = s' \mid s_{0,}a_0, s_{1,}a_1, ..., s_{t-1,}a_{t-1}, s_{t,}a_t\} \tag{1}$$

If the state signal has the Markov property, on the other hand, then the agent decision at s_t depends only on the state and action representations at time t, in which case the previously defined probability can be defined by specifying only

$$\Pr\{s_{t+1} = s'\} = \Pr\{s_{t+1} = s' \mid s_{t,} a_t\} \qquad (2)$$

In other words, a state signal has the Markov property, and is a Markov state, if and only if (2) is equal to (1) for all s', and histories $(s_{0,} a_0, s_{1,} a_1, ..., s_{t,} a_t)$. In this case, the environment and task as a whole are also said to have the Markov property.

A Markov Decision Process [18] is a discrete time stochastic control process characterized by a set of states; in each state there are several actions from which the decision maker must choose. For a state s and an action a, a state transition function $Pa(s)$ determines the transition probabilities to the next state. The decision maker earns a reward for each state transition.

The Markov property is important in the approach used in this thesis, the reinforcement learning based CAC, because decisions and values are assumed to be a function only of the current state. In order for these to be effective and informative, the state representation must be informative.

2.4. *The Peak-based CAC algorithm*

The easiest CAC algorithm is the one called peak-based or greedy. The idea beyond the algorithm is very simple: accept new calls whenever there is enough space to do this, else reject. This algorithm seems not to be a very good one, but, instead, it has a lot of good properties having made it the most diffuse algorithm to solve the CAC problem.

2.4.1. <u>Optimal solution on the short term</u>

First of all, it is possible to demonstrate it is the optimum solution to the CAC algorithm while the time window where it acts reduces to zero (which means, in practice, that peak-based is the optimal solution on the short term). Suppose to have an environment in a certain state s_0

at a certain time t_0 (as above mentioned the CAC problem has the Markov property), to have a set $(t_1, t_2, ..., t_n)$ of time instants at which happens the set of events $(e_0, e_1, ..., e_{n-1})$ each being or a new call or a call closure; suppose also the instant t_1 when e_0 happens is infinitely close to t_0 , and that e_0 is a new call. Departing from s_0 each CAC algorithm will have, in general, a different behaviour, so that each algorithm will take a set of different actions $(a_{0,k}, a_{1,k}, ..., a_{n-1,k})$ where $a_{i,k}$ is the *i*-th action took by the *k*-th algorithm: determine which is the best one is not easy. If the time window goes to zero, the only time instant entering in the time window will be just t_1. Here CAC algorithms could have different behaviours: if there is not enough space, departing from s_0 to accept the new calls, all of them, including the peak-based, will not accept the new call, so that the peak-based is the optimum algorithm. If there is enough space, some algorithms will not accept the call, some others (among which the peak-based) will accept. Obviously in that time window the choice guaranteeing the maximum reward is to accept the call, so that also in this case (which completes the possibilities) the peak-based algorithm acted as optimal.

2.4.2. Optimal solution in case of only one class of service

In case the system to be controlled has just one class of service, it is easy to demonstrate that the peak-based algorithm is the optimum one. Suppose to have an environment in a certain state s_0 at a certain time t_0 (as above mentioned the CAC problem has the Markov property), to have a set $(t_1, t_2, ..., t_n)$ of time instants at which happens the set of events $(e_0, e_1, ..., e_{n-1})$ each being or a new call or a call closure; suppose also all e_i behave to the same class of service. In this case, it is quite obvious to reject a certain call to wait for the next one hoping it will be of a more convenient CoS is useless. In any case, one could object that different calls could have different length so that conducing to different incomes. In the case the bandwidth is all occupied but a certain part just enough to accept a new call, it could seem to be convenient to wait in order to accept the longest new call. Suppose to have an algorithm able to determine the distribution of the calls (either the distribution of call arrivals and also of their duration) or that this distribution is known a-priori. Even with this knowledge (which the peak-based algorithm does not use) an algorithm could not state that, if the last call was a "short" ("long") one with respect to the known distribution, the next one will be a "long" ("short") one, due to the fact that, at least in a real environment, new calls arrivals and their duration is statistically independent from the previous ones. It is the same situation as playing roulette: even if last n extracts were all "rouge", it is not more probable the next one will be "noir". So that, the best solution in such a scenario is to accept new calls whenever there is enough space, which is exactly the behaviour of the peak-based algorithm.

2.4.3. <u>Optimal solution in case of low traffic</u>

The peak-based algorithm could be demonstrated to be the optimal solution to the CAC problem in case of "low traffic". The concept of "low traffic" could be defined as follows: $B_t^{req} < B_t^{av} \ \forall t$ where B_t^{req} is the requested bandwidth of the new call event e_t and B_t^{av} is the available bandwidth at time t: summarizing, there should always be enough bandwidth to accept the new call. This situation is not as unusual as one could suppose: currently telecommunication networks (in particular cellular ones) are over-provisioned, which means exactly that in most of cases there is enough bandwidth to accept a new call. Unfortunately this situation leads to a lot of waste: a better bandwidth usage could lead to very good results even without over-provisioning (and, in general, not all the scenarios could be resolved with over-provisioning). So, in a "low traffic" scenario it is counterproductive to reject a call, so that the optimal solution is to always accept new calls, which is exactly what the peak-based algorithm does. Even in the scenarios where the low traffic situation could not be assured $\forall t$ but just for some time intervals $[t_s, t_e]$ (where t_s is the time at which the interval starts and t_e is the time at which the interval ends - obviously we are interested to the cases where $t_s > t_e$), in such intervals the peak-based algorithm is the optimal one.

Chapter 3. Reinforcement Learning: an overview

3.1. *Introduction*

In order to introduce the Reinforcement Learning concepts and algorithm, it is useful to perform a brief introduction of the existing CAC algorithms, with particular attention to the model-based ones. Apart the Peak-based, there are a lot of solution to the CAC problem in the literature, e.g. [19], [20], [21], [22], [23]. Most of these approaches need a number of calculations to be performed or suffer of the so-called "curse of dimensionality" (e.g., the Dynamic Programming approaches): when the state space becomes huge and the number of possible classes of services increase, the number of information to be stored in order to perform the algorithm becomes so big that it is not possible to store it or, at least, manage it in a "reasonable" time (a user is not willing to wait minutes to have the response his call is accepted or not!).

A class of algorithms which well adapts to this particular problem is the class containing algorithms based on a neural-network like approach. Neural networks (NN [24]) are particular entities which are able to *learn* how to respond to certain inputs, i.e. to learn what outputs to produce having something in input. In order to make NN able to learn, there are two main approaches, a supervised and an unsupervised one. In the first approach, it is possible to create a training set, which has a number of different inputs and the respective correct outputs. This training set is put in input to the NN with its respective outputs and, thanks some backtracking algorithms, the NN learn how to behave in response to those inputs. Then the

NN is used on the real input and outputs are the result of the behaviour learned by the NN. Main problems with NN are:

- necessity to have a sufficiently varied training set, in order to “teach” to the NN how to behave in response to the more possible diverse inputs as possible (supervising);
- training sets not sufficiently varied could lead the NN to do not learn the expected functioning so that outputs will be wrong;
- training sets too wide will lead the NN to work perfectly on it but to not have a sufficient generalization

All these problems could be overcame using an unsupervised approach. In non supervised approaches there is not the necessity to have a training set: the idea is to reward the unsupervised agent when it takes the right decision, to penalize him when it takes wrong decisions. In this way, an agent could easily learn to try to get rewards and avoid penalizations. The real problem with these algorithms is to correctly determine positive and negative rewards: in case of an error in the reward functions, the algorithm will learn a wrong behaviour.

They exist a number of approaches of unsupervised learning [25]: among them, the one I determined to be the most promising to solve the CAC problem is the Reinforcement Learning [26] approach.

3.2. *The Reinforcement Learning Problem*

The objective of this chapter is to describe the reinforcement learning problem in a broad sense. The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker are called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a task, one instance of the reinforcement learning problem.

More specifically, the agent and environment interact at each of a sequence of discrete time steps, . At each time step , the agent receives some representation of the environment's state, , where is the set of possible states, and on that basis selects an action, , where is the set of actions available in state . One time step later, in part as a consequence of its action, the agent receives a numerical reward, , and finds itself in a new state, . Next picture diagrams the agent-environment interaction.

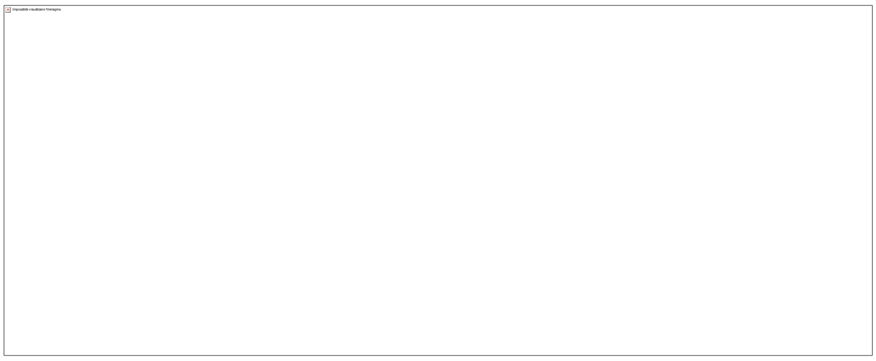


Figure 1 Agent-environment interaction in reinforcement learning

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted , where is the probability that if . Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run.

This framework is abstract and flexible and can be applied to many different problems in many different ways. For example, the time steps need not refer to fixed intervals of real time; they can refer to arbitrary successive stages of decision-making and acting. Similarly, the states can take a wide variety of forms: they can be completely determined by low-level sensations, such as direct sensor readings, or they can be more high-level and abstract, such as symbolic descriptions of objects in a room. The general rule I followed is that anything cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment; furthermore I consider the reward computation to be external to the agent because it defines the task facing the agent and thus must be beyond its ability to change arbitrarily.

The reinforcement learning framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behaviour can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards). This framework may not be sufficient to represent all decision-learning problems usefully, but it has proved to be widely useful and applicable.

3.3. *Goals and Rewards*

In reinforcement learning, the purpose or goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent. At each time step, the reward is a simple number, . Informally, the agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but cumulative reward in the long run. The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning. Although this way of formulating goals might at first appear limiting, in practice it has proved to be flexible and widely applicable. The best way to see this is to consider examples of how it has been, or could be, used. For example, to make a robot learn to walk, researchers have provided reward on each time step proportional to the robot's forward motion. In making a robot learn how to escape from a maze, the reward is often zero until it escapes, when it becomes Impossibile visualizzare l'immagine.. Another common approach in maze learning is to give a reward of Impossibile visualizzare l'immagine.for every time step that passes prior to escape; this encourages the agent to escape as quickly as possible. You can see what is happening in all of these examples. The agent always learns to maximize its reward. If we want it to do something for us, we must provide rewards to it in such a way that in maximizing them the agent will also achieve our goals. It is thus critical that the rewards we set up truly indicate what we want accomplished. In particular, the reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do. For example, a chess-playing agent should be rewarded only for actually winning, not for achieving sub-goals such taking its opponent's pieces or gaining control of the centre of the board. If achieving these sorts of sub-goals were rewarded, then the agent might find a way to achieve them without achieving the real goal. For example, it might find a way to take the opponent's pieces even at the cost of losing the game. The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it achieved.

3.4. *Returns*

A more precise definition of what is meant with "maximize the total amount of reward received" is needed. If the sequence of rewards received after time step [illegible] is denoted Impossibile visualizzare l'immagine., then what precise aspect of this sequence has to be maximize? In general, we seek to maximize the expected return, where the return, [illegible], is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the rewards:

Impossibile visualizzare l'immagine. (3)

where [illegible] is a final time step. This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent-environment interaction breaks naturally into subsequences, which we call episodes, such as plays of a game, trips through a maze, or any sort of repeated interactions. Each episode ends in a special state called the terminal state, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states. Tasks with episodes of this kind are called episodic tasks. In episodic tasks it is sometimes needed to distinguish the set of all non-terminal states, denoted [illegible], from the set of all states plus the terminal state, denoted [illegible].

On the other hand, in many cases the agent-environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. For example, this would be the natural way to formulate a continual process-control task, or an application to a robot with a long life span: these ones are called continuing tasks. The return formulation (3) is problematic for continuing tasks because the final time step would be Impossibile visualizzare l'immagine., and the return, which is what we are trying to maximize, could itself easily be infinite. (For example, suppose the agent receives a reward of [illegible] at each time step.) Thus, usually it is used a definition of return that is slightly more complex conceptually but much simpler mathematically. The additional concept that is needed is discounting. According to this approach, the agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses to maximize the expected discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{4}$$

where is a parameter, , called the discount rate. The discount rate determines the present value of future rewards: a reward received time steps in the future is worth only times what it would be worth if it were received immediately. If , the infinite sum has a finite value as long as the reward sequence is bounded. If , the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose so as to maximize only . If each of the agent's actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize (4) by separately maximizing each immediate reward. But in general, acting to maximize immediate reward can reduce access to future rewards so that the return may actually be reduced. As approaches 1, the objective takes future rewards into account more strongly: the agent becomes more farsighted.

3.5. *Unified Notation for Episodic and Continuing Tasks*

It is opportune to have a single notation that covers both episodic and continuing tasks. Having defined the return as a sum over a finite number of terms in one case (3) and as a sum over an infinite number of terms in the other (4), it is possible to unify them by considering episode termination to be the entering of a special absorbing state that transitions only to itself and that generates only rewards of zero. For example, consider the following state transition diagram

Here the solid square represents the special absorbing state corresponding to the end of an episode. Starting from , we get the reward sequence . Summing these, we get the same return whether we sum over the first rewards (here) or over the full infinite sequence. This remains true even if we introduce discounting. Thus, we can define the return, in general, according to (4).

3.6. *Value Functions*

Almost all reinforcement learning algorithms are based on estimating value functions, i.e. functions of states (or of state-action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state). The notion of "how good" is defined in terms of future rewards that can be expected, or, to be precise, in terms of expected return. Of course the rewards the agent can expect to receive in the future depend on what actions it will take. Accordingly, value functions are defined with respect to particular policies. Recall that a policy, , is a mapping from each state, , and action, , to the probability of taking action when in state . Informally, the value of a state under a policy , denoted , is the expected return when starting in and following thereafter. For MDPs, it is possible to define formally as:

(5)

where denotes the expected value given that the agent follows policy , and is any time step. Note that the value of the terminal state, if any, is always zero. The function is called the state-value function for policy . Similarly, it is possible to define the value of taking action in state under a policy , denoted , as the expected return starting from , taking the action , and thereafter following policy :

(6)

is called the action-value function for policy .

The value functions and can be estimated from experience. For example, if an agent follows policy and maintains an average, for each state encountered, of the actual returns that have followed that state, then the average will converge to the state's value, , as the number of times that state is encountered approaches infinity. If separate averages are kept for each action taken in a state, then these averages will similarly converge to the action values, . Estimation methods of this kind are called Monte Carlo methods because they involve averaging over many random samples of actual returns. Of course, if there are very many states, then it may not be practical to keep separate averages for each state individually. Instead, the agent would have to maintain and as parameterized functions and adjust the parameters to better match the observed returns. This can also produce accurate estimates, although much depends on the nature of the parameterized function approximator.

A fundamental property of value functions used throughout reinforcement learning and dynamic programming is that they satisfy particular recursive relationships. For any policy and any state , the following consistency condition holds between the value of and the value of its possible successor states:

$$= E_\pi\{R_t|s_t=s\}$$

$$=$$

$$=$$

$$=$$

$$= \qquad (7)$$

where it is implicit that the actions, , are taken from the set , and the next states, , are taken from the set , or from in the case of an episodic problem. Equation (7) is the Bellman equation for . It expresses a relationship between the value of a state and the values of its successor states. Think of looking ahead from one state to its possible successor states, as suggested by Figure 2. Each open circle represents a state and each solid circle represents a state-action pair. Starting from state , the root node at the top, the agent could take any of some set of actions (three are shown in Figure 2). From each of these, the environment could respond with one of several next states, , along with a reward, . The Bellman equation (7) averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way.

The value function is the unique solution to its Bellman equation. Diagrams like those shown in Figure 2 are called backup diagrams because they diagram relationships that form the basis of the update or backup operations that are at the heart of reinforcement learning methods. These operations transfer value information back to a state (or a state-action pair) from its successor states (or state-action pairs).

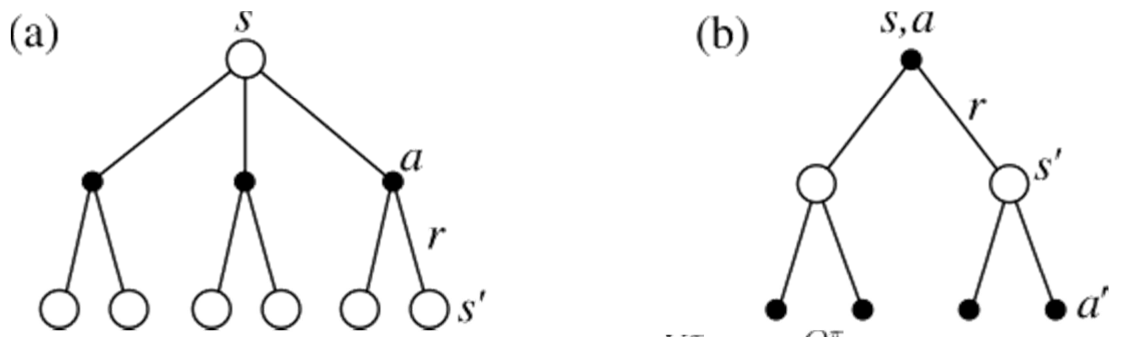


Figure 2 Backup diagrams for (a) V^π and (b) Q^π

3.7. *Optimal Value Functions*

Solving a reinforcement learning task means, roughly, finding a policy that achieves a lot of reward over the long run. For finite MDPs, it is possible precisely define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words, $\pi \geq \pi'$ if and only if $V^\pi(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$. There is always at least one policy that is better than or equal to all other policies. This is an optimal policy. Although there may be more than one, it is better to denote all the optimal policies by π^*. They share the same state-value function, called the *optimal state-value function*, denoted V^*, and defined as

$$V^*(s) = \max V^\pi(s) \quad (8)$$

for all $s \in \mathcal{S}$.

Optimal policies also share the same *optimal action-value function*, denoted Q^*, and defined as

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) \tag{9}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$. For the state-action pair (s,a), this function gives the expected return for taking action a in state s and thereafter following an optimal policy. Thus, it is possible to write Q^* in terms of V^* as follows:

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right\} \tag{10}$$

Because V^* is the value function for a policy, it must satisfy the self-consistency condition given by the Bellman equation for state values (7). Because it is the optimal value function, however, V^*'s consistency condition can be written in a special form without reference to any specific policy. This is the Bellman equation for V^*, or the Bellman optimality equation. Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\ &= \max_{a} E_{\pi^*}\left\{ R_t \,\middle|\, s_t = s, a_t = a \right\} \\ &= \max_{a} E_{\pi^*}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a \right\} \\ &= \max_{a} E_{\pi^*}\left\{ r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \,\middle|\, s_t = s, a_t = a \right\} \\ &= \text{Impossibile visualizzare l'immagine.} && (11) \\ &\text{Impossibi...} \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V^*(s') \right]. && (12) \end{aligned}$$

The last two equations are two forms of the Bellman optimality equation for V^*. The Bellman optimality equation for Q^* is

$$
\begin{aligned}
Q^*(s,a) &= E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \;\middle|\; s_t = s, a_t = a \right\} \\
&= \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right].
\end{aligned}
$$

The backup diagrams in Figure 3 show graphically the spans of future states and actions considered in the Bellman optimality equations for V^* and Q^*. These are the same as the backup diagrams for V^π and Q^π except that arcs have been added at the agent's choice points to represent that the maximum over that choice is taken rather than the expected value given some policy. Figure 3 a graphically represents the Bellman optimality equation (12).

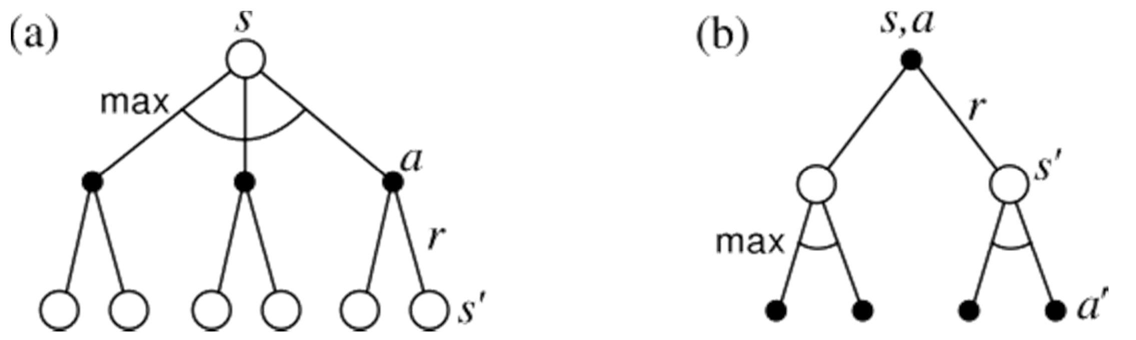


Figure 3 Backup diagrams for (a) V^* and (b) Q^*

For finite MDPs, the Bellman optimality equation (12) has a unique solution independent of the policy. The Bellman optimality equation is actually a system of equations, one for each state, so if there are N states, then there are N equations in N unknowns. If the dynamics of the environment are known ($\mathcal{R}^a_{ss'}$ and $\mathcal{P}^a_{ss'}$), then in principle one can solve this system of equations for V^* using any one of a variety of methods for solving systems of nonlinear equations. One can solve a related set of equations for Q^*.

Once one has V^*, it is relatively easy to determine an optimal policy. For each state s, there will be one or more actions at which the maximum is obtained in the Bellman optimality equation. Any policy that assigns nonzero probability only to these actions is an optimal policy: this could be called one-step search. Having the optimal value function, V^*, then the actions that appear best after a one-step search will be optimal actions. Another way of saying this is that any policy that is *greedy* with respect to the optimal evaluation function V^* is an optimal policy. The beauty of V^* is that if one uses it to evaluate the short-term consequences of actions (specifically, the one-step consequences) then a greedy policy is actually optimal in the long-term sense in which we are interested because V^*already takes into account the reward consequences of all possible future behaviour. By means of , the optimal expected long-term return is turned into a quantity that is locally and immediately available for each state. Hence, a one-step-ahead search yields the long-term optimal actions.

Having makes choosing optimal actions still easier. With , the agent does not even have to do a one-step-ahead search: for any state , it can simply find any action that maximizes . The action-value function effectively caches the results of all one-step-ahead searches. It provides the optimal expected long-term return as a value that is locally and immediately available for each state-action pair. Hence, at the cost of representing a function of state-action pairs, instead of just of states, the optimal action-value function allows optimal actions to be selected without having to know anything about possible successor states and their values, that is, without having to know anything about the environment's dynamics.

Explicitly solving the Bellman optimality equation provides one route to finding an optimal policy, and thus to solving the reinforcement learning problem. However, this solution is rarely directly useful. It is akin to an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirability in terms of expected rewards. This solution relies on at least three assumptions that are rarely true in practice:

1. to accurately know the dynamics of the environment;
2. to have enough computational resources to complete the computation of the solution;
3. the Markov property.

For most tasks it is generally not possible to implement this solution exactly because various combinations of these assumptions are violated: in reinforcement learning one typically has to settle for approximate solutions.

3.8. *Optimality and Approximation*

Even if it is possible to have a complete and accurate model of the environment's dynamics, it is usually not possible to simply compute an optimal policy by solving the Bellman optimality equation. For example, board games such as chess are a tiny fraction of human experience, yet large, custom-designed computers still cannot compute the optimal moves. A critical aspect of the problem facing the agent is always the computational power available to it, in particular, the amount of computation it can perform in a single time step.

The memory available is also an important constraint. A large amount of memory is often required to build up approximations of value functions, policies, and models. In tasks with small, finite state sets, it is possible to form these approximations using arrays or tables with one entry for each state (or state-action pair). This is called the tabular case, and the corresponding methods are called tabular methods. In many cases of practical interest, however, there are far more states than could possibly be entries in a table. In these cases the functions must be approximated, using some sort of more compact parameterized function representation.

Approximating is not always a great problem, in fact there may be many states that the agent faces with such a low probability that selecting suboptimal actions for them has little impact on the amount of reward the agent receives. The on-line nature of reinforcement learning makes it possible to approximate optimal policies in ways that put more effort into learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

Chapter 4. Introduction to Q(λ): the chosen RL algorithm

4.1. *Introduction*

In order to introduce the chosen RL algorithm, Q(λ), it is necessary to introduce two different concepts, in particular the Q-Learning and the Eligibility-Trace ones, of which Q(λ) is a fusion.

4.2. *Introduction to Q-Learning*

In order to introduce Q-Learning algorithm, it is important to briefly define the two concepts of *on-policy* and *off-policy* learning. On-policy methods attempt to evaluate or improve the policy that is used to make decisions, i.e. that they estimate the value of a policy while using it for control. In off-policy methods these two functions are separated. The policy used to generate behaviour, called the behaviour policy, may in fact be unrelated to the policy that is evaluated and improved, called the estimation policy. An advantage of this separation is that the estimation policy may be deterministic (e.g., greedy), while the behaviour policy can continue to sample all possible actions.

One of the most important breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989). Its simplest form, one-step Q-learning, is defined by

(13)

In this case, the learned action-value function, , directly approximates , the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs. The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated: this is a minimal requirement in the sense that any method guaranteed to find optimal behaviour in the general case must require it. Under this assumption (and some other regarding stochastic approximation conditions on the sequence of step-size parameters) has been shown to converge with probability 1 to . The Q-learning algorithm is shown in procedural form in Figure 4.

Figure 4 Q-learning: an off-policy TD control algorithm

It could be useful to determine the backup diagram for Q-learning The rule (13) updates a state-action pair, so the top node, the root of the backup, must be a small, filled action node. The backup is also from action nodes, maximizing over all those actions possible in the next state. Thus the bottom nodes of the backup diagram should be all these action nodes. Finally, remembering that taking the maximum of these “next action” nodes is indicated with an arc across them, the resulting diagram is depicted in Figure 5.

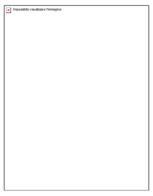


Figure 5 Backup diagram for Q-learning

4.3. *Introduction to Q-Learning*

Eligibility traces are one of the basic mechanisms of reinforcement learning. There are two ways to view eligibility traces. The more theoretical view is that they are a bridge from TD to Monte Carlo methods (remember that TD methods are BLABLABLA and Monte Carlo Methods are BLABLABLA). When TD methods are augmented with eligibility traces, they produce a family of methods spanning a spectrum that has Monte Carlo methods at one end and one-step TD methods at the other. In between are intermediate methods that are often better than either extreme method. In this sense eligibility traces unify TD and Monte Carlo methods in a valuable and revealing way. The other way to view eligibility traces is more mechanistic. From this perspective, an eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error. Thus, eligibility traces help bridge the gap between events and training information. Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment.

The more theoretical view of eligibility traces is called the forward view, and the more mechanistic view is called the backward view. The forward view is most useful for understanding what is computed by methods using eligibility traces, whereas the backward view is more appropriate for developing intuition about the algorithms themselves.

4.4. *n-Step TD Prediction*

Consider estimating V^π from sample episodes generated using . Monte Carlo methods perform a backup for each state based on the entire sequence of observed rewards from that state until the end of the episode. The backup of simple TD methods, on the other hand, is based on just the one next reward, using the value of the state one step later as a proxy for the remaining rewards. One kind of intermediate method, then, would perform a backup based on an intermediate number of rewards: more than one, but less than all of them until termination. For example, a two-step backup would be based on the first two rewards and the estimated value of the state two steps later. Figure 6 diagrams the spectrum of -step backups for , with one-step, simple TD backups on the left and up-until-termination Monte Carlo backups on the right.

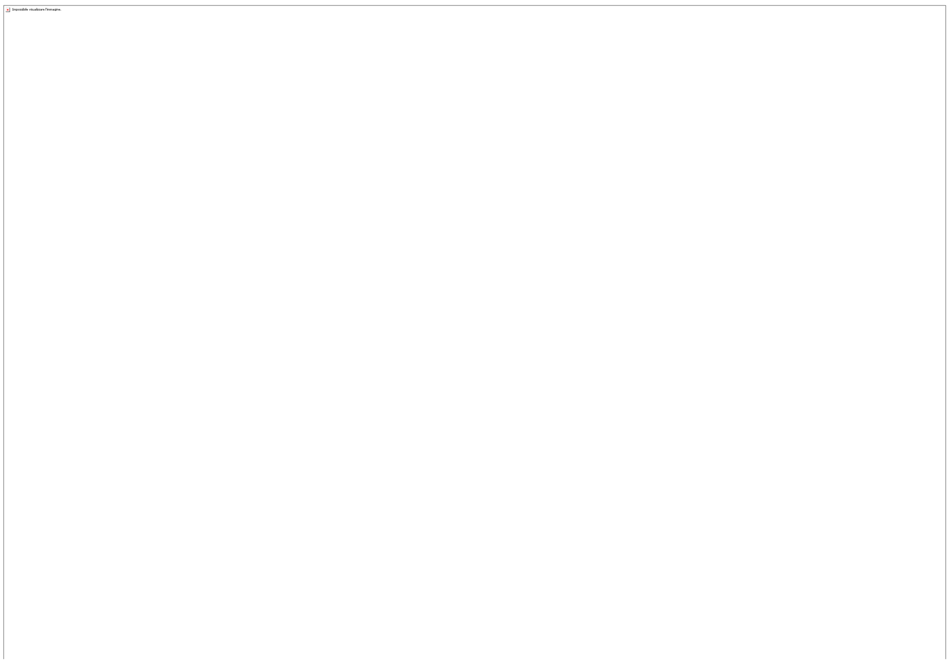


Figure 6 Spectrum ranging from the one-step backup of simple TD methods to the up-until-termination backups of Monte Carlo methods. In between are the *n*-step backups, based on *n* steps of real rewards and the estimated value of the *n*-th next state, all appropriately discounted

The methods that use n-step backups are still TD methods because they still change an earlier estimate based on how it differs from a later estimate. Now the later estimate is not one step later, but n steps later. Methods in which the temporal difference extends over n steps are called n-step TD methods. More formally, consider the backup applied to state as a result of the state-reward sequence, $s_t, r_{t+1}, s_{t+1}, r_{t+2}, \ldots, r_T, s_T$ (omitting the actions for simplicity). It is known that in Monte Carlo backups the estimate $V_t(s_t)$ of $V^\pi(s_t)$ is updated in the direction of the complete return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T,$$

where *T* is the last time step of the episode. Let us call this quantity the target of the backup. Whereas in Monte Carlo backups the target is the expected return, in one-step backups the target is the first reward plus the discounted estimated value of the next state:

$$R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1}).$$

This makes sense because $\gamma V_t(s_{t+1})$ takes the place of the remaining terms $\gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$. The point now is that this idea makes just as much sense after two steps as it does after one. The two-step target is

$$R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2}),$$

where now $\gamma^2 V_t(s_{t+2})$ takes the place of the terms $\gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots + \gamma^{T-t-1} r_T$. In general, the n-step target is

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}). \quad (14)$$

This quantity is sometimes called the "corrected n-step truncated return" because it is a return truncated after n steps and then approximately corrected for the truncation by adding the estimated value of the n-th next state. That terminology is descriptive but a bit long: usually it is also called as the n-step return at time t.

Of course, if the episode ends in less than n steps, then the truncation in an n-step return occurs at the episode's end, resulting in the conventional complete return. In other words, if $T - t < n$, then $R_t^{(n)} = R_t^{(T-t)} = R_t$.

An n-step backup is defined to be a backup toward the n-step return. In the tabular, state-value case, the increment to $V_t(s_t)$ (the estimated value of $V^\pi(s_t)$ at time t), due to an n-step backup of s_t, is defined by

$$\Delta V_t(s_t) = \alpha \Big[R_t^{(n)} - V_t(s_t) \Big],$$

where α is a positive step-size parameter, as usual. Of course, the increments to the estimated values of the other states are $\Delta V_t(s) = 0$, for all $s \neq s_t$. Here the n-step backup is defined in terms of an increment, rather than as a direct update rule, in order to distinguish two different ways of making the updates. In on-line updating, the updates are done during the episode, as soon as the increment is computed. In this case we have $V_{t+1}(s) = V_t(s) + \Delta V_t(s)$ for all $s \in \mathcal{S}$. In off-line updating, on the other hand, the increments are accumulated "on the side" and are not used to change value estimates until the end of the episode. In this case, $V_t(s)$ is constant within an episode, for all s. If its value in this episode is $V(s)$, then its new value in the next episode will be $V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$.

The expected value of all n-step returns is guaranteed to improve in a certain way over the current value function as an approximation to the true value function. For any , the expected value of the n-step return using is guaranteed to be a better estimate of than is, in a worst-state sense. That is, the worst error under the new estimate is guaranteed to be less than or equal to times the worst error under :

(15)

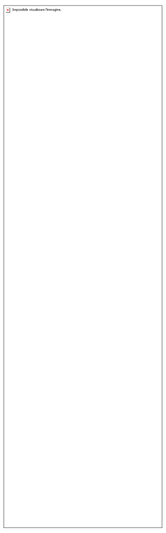
This is called the error reduction property of n-step returns. Because of the error reduction property, one can show formally that on-line and off-line TD prediction methods using n-step backups converge to the correct predictions under appropriate technical conditions. The n-step TD methods thus form a family of valid methods, with one-step TD methods and Monte Carlo methods as extreme members.

Nevertheless, n-step TD methods are rarely used because they are inconvenient to implement. Computing n-step returns requires waiting steps to observe the resultant rewards and states. For large , this can become problematic, particularly in control applications.

4.5. *The Forward View of TD(λ)*

Backups can be done not just toward any n-step return, but toward any average of n-step returns. For example, a backup can be done toward a return that is half of a two-step return and half of a four-step return: . Any set of returns can be averaged in this way, even an infinite set, as long as the weights on the component returns are positive and sum to 1. The overall return possesses an error reduction property similar to that of individual n-step returns (15) and thus can be used to construct backups with guaranteed convergence properties. Averaging produces a substantial new range of algorithms.

A backup that averages simpler component backups in this way is called a complex backup. The backup diagram for a complex backup consists of the backup diagrams for each of the component backups with a horizontal line above them and the weighting fractions below. For example, the complex backup mentioned above, mixing half of a two-step backup and half of a four-step backup, has the diagram:



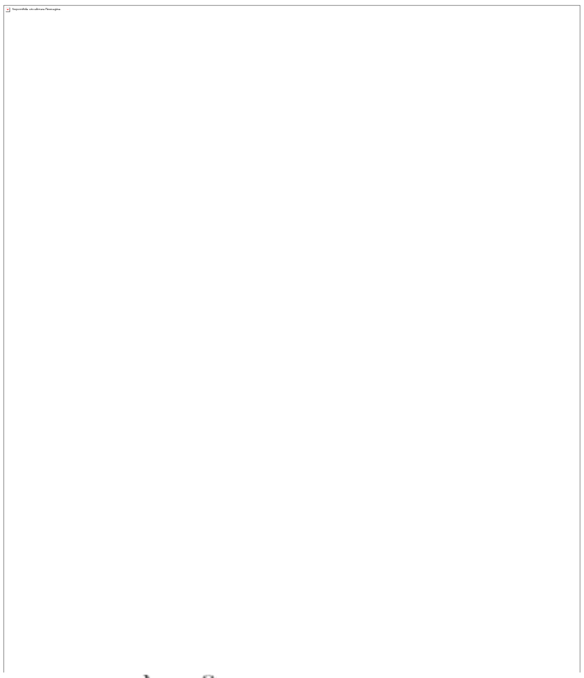


Figure 7 Backup diagram for TD(λ). If $\lambda = 0$, then the overall backup reduces to its first component, the one-step TD backup, whereas if $\lambda = 1$, then the overall backup reduces to its last component, the Monte Carlo backup

The TD(λ) algorithm can be understood as one particular way of averaging n-step backups. This average contains all the n-step backups, each weighted proportional to λ^{n-1}, where $0 \leq \lambda \leq 1$ (Figure 7) A normalization factor of $1-\lambda$ ensures that the weights sum to 1. The resulting backup is toward a return, called the λ-return, defined by

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)}.$$

Figure 8 illustrates this weighting sequence. The one-step return is given the largest weight, $1-\lambda$; the two-step return is given the next largest weight, $(1-\lambda)\lambda$; the three-step return is given the weight $(1-\lambda)\lambda^2$; and so on. The weight fades by λ with each additional step. After a terminal state has been reached, all subsequent n-step returns are equal to R_t. If we want, we can separate these terms from the main sum, yielding

$$R_t^\lambda \quad = \quad (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}R_t^{(n)} \quad + \quad \lambda^{T-t-1}R_t. \tag{16}$$

This equation makes it clearer what happens when $\lambda = 1$. In this case the main sum goes to zero, and the remaining term reduces to the conventional return, R_t. Thus, for $\lambda = 1$, backing up according to the λ-return is the same as the Monte Carlo algorithm. On the other hand, if $\lambda = 0$, then the λ-return reduces to $R_t^{(1)}$, the one-step return. Thus, for $\lambda = 0$, backing up according to the λ-return is the same as the one-step TD method, TD(0).

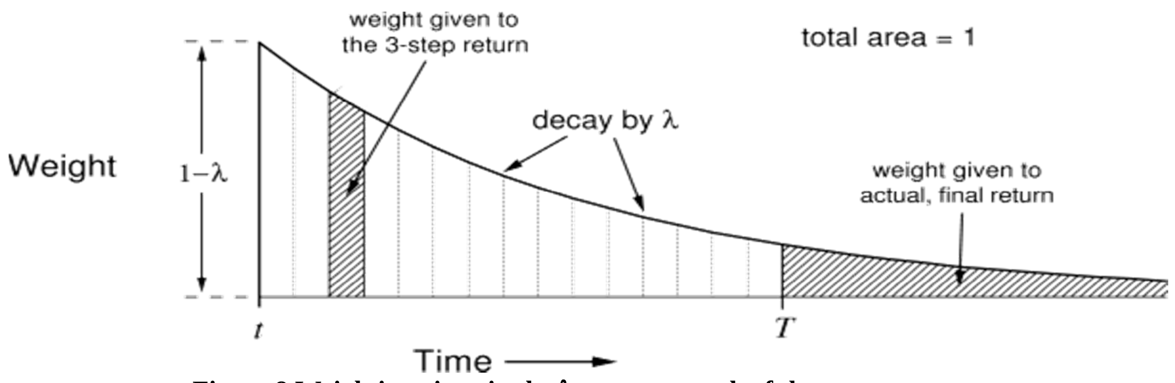

weight given to the 3-step return
total area = 1
decay by λ
Weight
1−λ
weight given to actual, final return
t
T
Time

Figure 8 Weighting given in the λ-return to each of the *n*-step returns

The algorithm that performs backups using the λ-return is defined the λ-return algorithm. On each step, t, it computes an increment, $\Delta V_t(s_t)$, to the value of the state occurring on that step:

$$\Delta V_t(s_t) = \alpha \Big[R_t^\lambda - V_t(s_t) \Big]. \tag{17}$$

(The increments for other states are of course $\Delta V_t(s) = 0$, for all $s \neq s_t$.) As with the n-step TD methods, the updating can be either on-line or off-line.

This approach is called the theoretical, or forward, view of a learning algorithm. For each state visited, the algorithm looks forward in time to all the future rewards and decide how best to combine them. It is possible to imagine a person riding the stream of states, looking forward from each state to determine its update, as depicted in Figure 9. After looking forward from and updating one state, the person moves on to the next and never have to work with the preceding state again. Future states, on the other hand, are viewed and processed repeatedly, once from each vantage point preceding them.

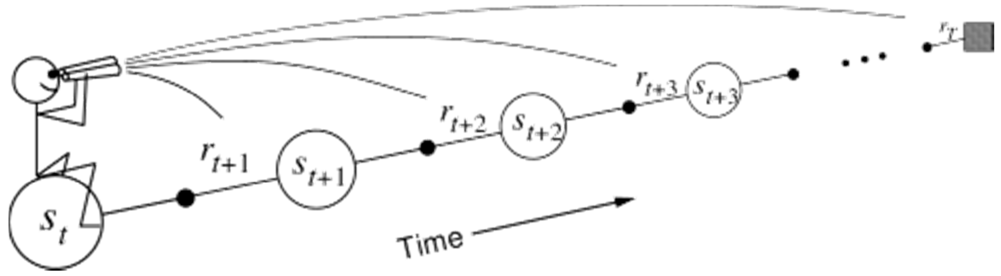


Figure 9 Forward or theoretical view. The algorithm decides how to update each state by looking forward to future rewards and states

The λ-return algorithm is the basis for the forward view of eligibility traces as used in the TD(λ) method. In fact, in the off-line case, the λ-return algorithm is the TD(λ) algorithm. The λ-return and TD(λ) methods use the λ parameter to shift from one-step TD methods to Monte Carlo methods. The specific way this shift is done is interesting, but not obviously better or worse than the way it is done with simple n-step methods by varying n. Ultimately, the most compelling motivation for the λ way of mixing n-step backups is that there is a simple algorithm -TD(λ) -for achieving it. This is a mechanism issue rather than a theoretical one.

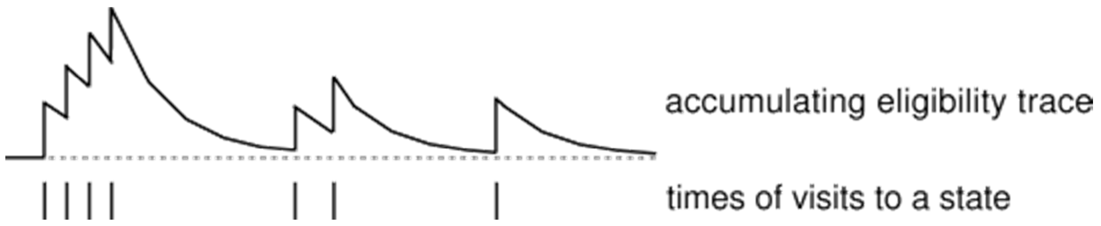
4.6. *The Backward View of TD(λ)*

The previous section presented the forward or theoretical view of the tabular TD(λ) algorithm as a way of mixing backups that parametrically shifts from a TD method to a Monte Carlo method. This section instead defines TD(λ) mechanistically, and in the next section it is shown that this mechanism correctly implements the forward view. The mechanistic, or backward, view of TD(λ) is useful because it is simple conceptually and computationally. In particular, the forward view itself is not directly implementable because it is non-causal, using at each step knowledge of what will happen many steps later. The backward view provides a causal, incremental mechanism for approximating the forward view and, in the off-line case, for achieving it exactly.

In the backward view of TD(λ), there is an additional memory variable associated with each state, its eligibility trace. The eligibility trace for state s at time $t$is denoted $e_t(s) \in \Re^+$. On each step, the eligibility traces for all states decay by $\gamma\lambda$, and the eligibility trace for the one state visited on the step is incremented by 1:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases} \tag{18}$$

for all non-terminal states s, where γ is the discount rate and λ is the parameter introduced in the previous section. Henceforth we refer to λ as the trace-decay parameter. This kind of eligibility trace is called an accumulating trace because it accumulates each time the state is visited, then fades away gradually when the state is not visited, as illustrated below:



At any time, the traces record which states have recently been visited, where “recently” is defined in terms of $\gamma\lambda$. The traces are said to indicate the degree to which each state is eligible for undergoing learning changes should a reinforcing event occur. The reinforcing events we are concerned with are the moment-by-moment one-step TD errors. For example, the TD error for state-value prediction is

$$\delta_t \ = \ r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t). \tag{19}$$

In the backward view of TD(λ), the global TD error signal triggers proportional updates to all recently visited states, as signalled by their nonzero traces:

$$\Delta V_t(s) = \alpha \delta_t e_t(s), \qquad \text{for all } s \in \mathcal{S}. \tag{20}$$

As always, these increments could be done on each step to form an on-line algorithm, or saved until the end of the episode to produce an off-line algorithm. In either case, equations ((18)-(20)) provide the mechanistic definition of the TD(λ) algorithm. A complete algorithm for on-line TD(λ) is given in Figure 10.

```
Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
    Initialize $s$
    Repeat (for each step of episode):
        $a \leftarrow$ action given by $\pi$ for $s$
        Take action $a$, observe reward, $r$, and next state, $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $e(s) \leftarrow e(s) + 1$
        For all $s$:
            $V(s) \leftarrow V(s) + \alpha\delta e(s)$
            $e(s) \leftarrow \gamma\lambda e(s)$
        $s \leftarrow s'$
    until $s$ is terminal
```

Figure 10 On-line tabular TD(λ)

The backward view of TD(λ) is oriented backward in time. At each moment we look at the current TD error and assign it backward to each prior state according to the state's eligibility trace at that time. It is possible to imagine a person riding along the stream of states, computing TD errors, and shouting them back to the previously visited states, as suggested by Figure 11. Where the TD error and traces come together, we get the update given by (20).

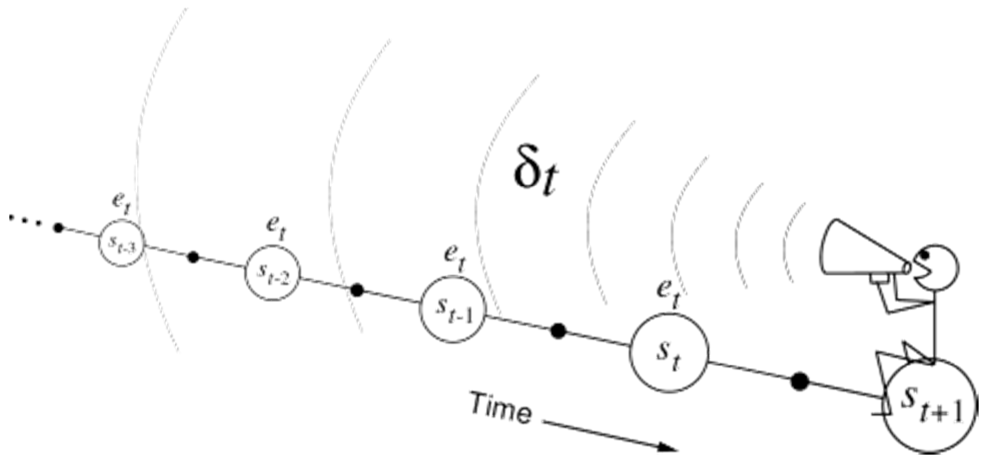


Figure 11 Backward or mechanistic view. Each update depends on the current TD error combined with traces of past events

To better understand the backward view, consider what happens at various values of λ. If $\lambda = 0$, then by (18) all traces are zero at t except for the trace corresponding to s_t. In terms of Figure 11, TD(0) is the case in which only the one state preceding the current one is changed by the TD error. For larger values of λ, but still $\lambda < 1$, more of the preceding states are changed, but each more temporally distant state is changed less because its eligibility trace is smaller, as suggested in the figure: it is possible to say that the earlier states are given less credit for the TD error.

If $\lambda = 1$, then the credit given to earlier states falls only by γ per step. This turns out to be just the right thing to do to achieve Monte Carlo behaviour. For example, remember that the TD error, δ_t, includes an undiscounted term of r_{t+1}. In passing this back k steps it needs to be discounted, like any reward in a return, by γ^k, which is just what the falling eligibility trace achieves. If $\lambda = 1$ and $\gamma = 1$, then the eligibility traces do not decay at all with time. In this case the method behaves like a Monte Carlo method for an undiscounted, episodic task. If $\lambda = 1$, the algorithm is also known as TD(1). TD(1) is a way of implementing Monte Carlo algorithms.

Concluding, it is possible to demonstrate that off-line TD(λ), as defined mechanistically above, achieves the same weight updates as the off-line λ-return algorithm: in this sense it is possible align the forward (theoretical) and backward (mechanistic) views of TD(λ) [27].

4.7. *Q(λ)*

Two different methods have been proposed that combine eligibility traces and Q-learning; sometimes these are referred as Watkins's Q(λ) and Peng's Q(λ), after the researchers who first proposed them.

Recall that Q-learning is an off-policy method, meaning that the policy learned about need not be the same as the one used to select actions. In particular, Q-learning learns about the greedy policy while it typically follows a policy involving exploratory actions -occasional selections of actions that are suboptimal according to Q_t. Because of this, special care is required when introducing eligibility traces.

Suppose to back up the state-action pair s_t, a_t at time t. Suppose that on the next two time steps the agent selects the greedy action, but on the third, at time $t + 3$, the agent selects an exploratory, non-greedy action. In learning about the value of the greedy policy at s_t, a_t it is possible to use subsequent experience only as long as the greedy policy is being followed. Thus, it is possible to use the one-step and two-step returns, but not, in this case, the three-step return. The n-step returns for all $n \geq 3$ no longer have any necessary relationship to the greedy policy.

Thus, unlike TD(λ), Watkins's Q(λ) does not look ahead all the way to the end of the episode in its backup. It only looks ahead as far as the next exploratory action. Aside from this difference, however, Watkins's Q(λ) is much like TD(λ). Its look ahead stops at episode's end, whereas Q(λ)'s look ahead stops at the first exploratory action, or at episode's end if there are no exploratory actions before that. Actually, to be more precise, one-step Q-learning and Watkins's Q(λ) both look one action past the first exploration, using their knowledge of the action values. For example, suppose the first action, a_{t+1}, is exploratory. Watkins's Q(λ) would still do the one-step update of $Q_t(s_t, a_t)$ toward $r_{t+1} + \gamma \max_a Q_t(s_{t+1}, a)$. In general, if a_{t+n} is the first exploratory action, then the longest backup is toward

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_a Q_t(s_{t+n}, a),$$

where we assume off-line updating. The backup diagram in Figure 12 illustrates the forward view of Watkins's Q(λ), showing all the component backups.

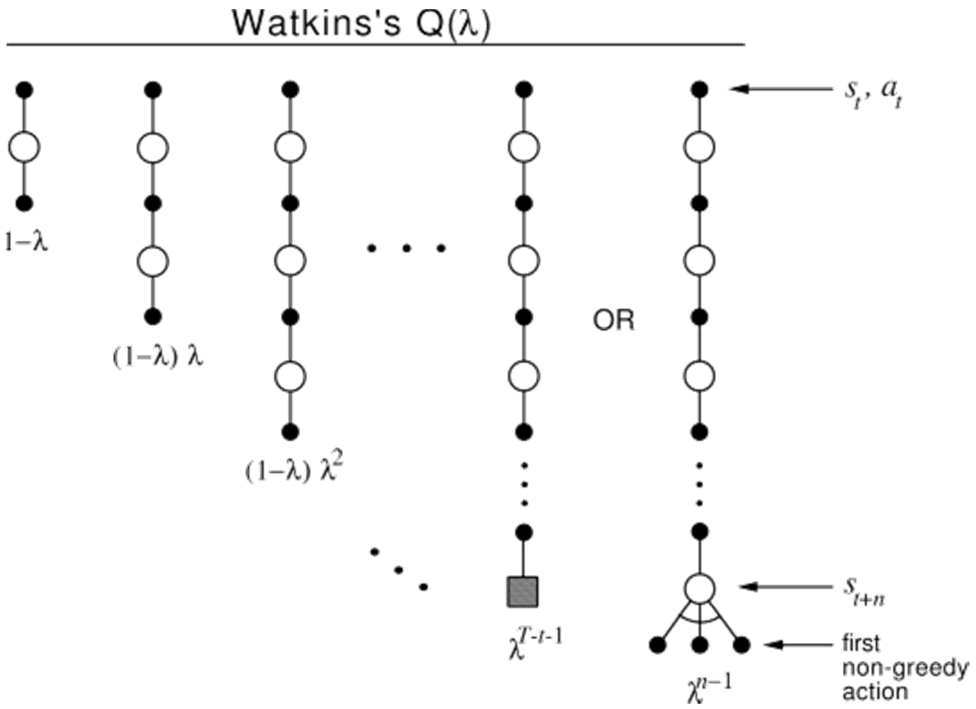


Figure 12 Backup diagram for Watkins's Q(λ). The series of component backups ends either with the end of the episode or with the first non-greedy action, whichever comes first

The mechanistic or backward view of Watkins's Q(λ) is also very simple. Eligibility traces are just set to zero whenever an exploratory (non-greedy) action is taken. The trace update is best thought of as occurring in two steps. First, the traces for all state-action pairs are either decayed by $\gamma\lambda$ or, if an exploratory action was taken, set to 0. Second, the trace corresponding to the current state and action is incremented by 1. The overall result is

$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a); \\ 0 & \text{otherwise,} \end{cases}$$

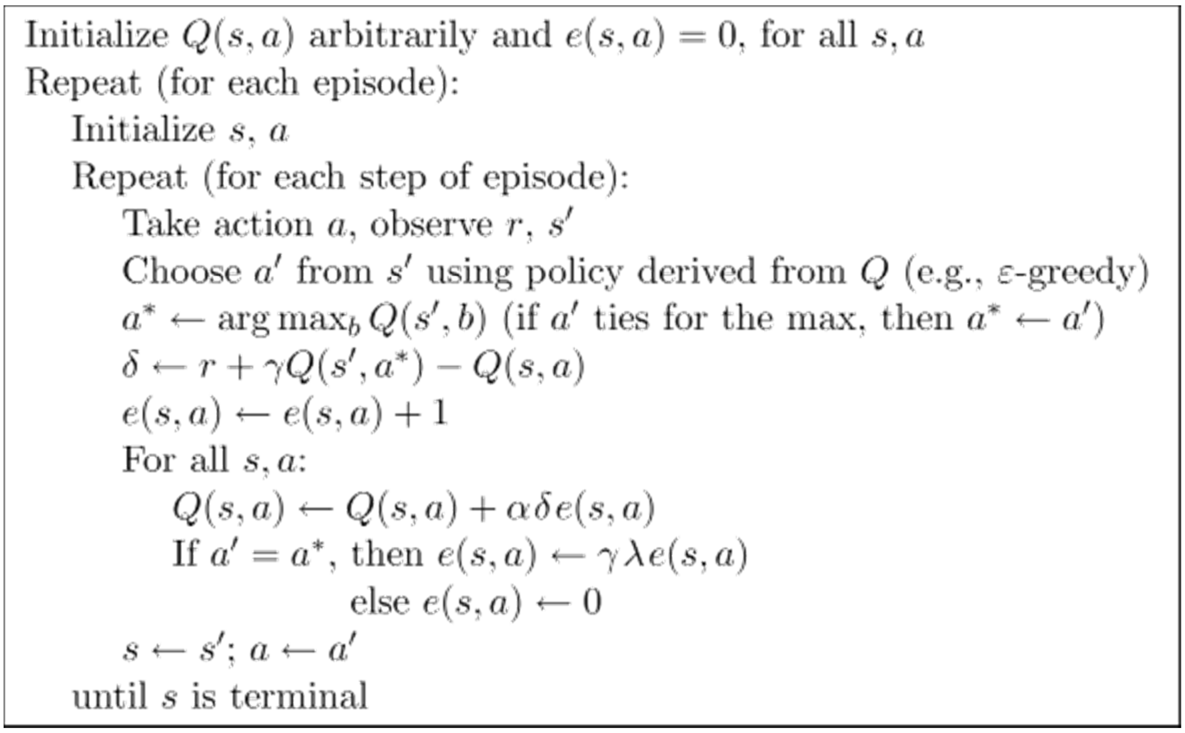
where, as before, $\mathcal{I}_{xy}$ is an identity indicator function, equal to 1 if $x = y$ and 0 otherwise. The rest of the algorithm is defined by

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\,\delta_t e_t(s,a),$$

Where

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1},a') - Q_t(s_t,a_t).$$

Next picture shows the complete algorithm in pseudocode:



```
Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
    Initialize $s$, $a$
    Repeat (for each step of episode):
        Take action $a$, observe $r$, $s'$
        Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $a^* \leftarrow \arg\max_b Q(s',b)$ (if $a'$ ties for the max, then $a^* \leftarrow a'$)
        $\delta \leftarrow r + \gamma Q(s',a^*) - Q(s,a)$
        $e(s,a) \leftarrow e(s,a) + 1$
        For all $s, a$:
            $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
            If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                            else $e(s,a) \leftarrow 0$
        $s \leftarrow s'$; $a \leftarrow a'$
    until $s$ is terminal
```

Figure 13 Tabular version of Watkins's Q( ) algorithm

Unfortunately, cutting off traces every time an exploratory action is taken loses much of the advantage of using eligibility traces. If exploratory actions are frequent, as they often are early in learning, then only rarely will backups of more than one or two steps be done, and learning may be little faster than one-step Q-learning. Peng's Q(λ) is an alternate version of Q(λ) meant to remedy this.

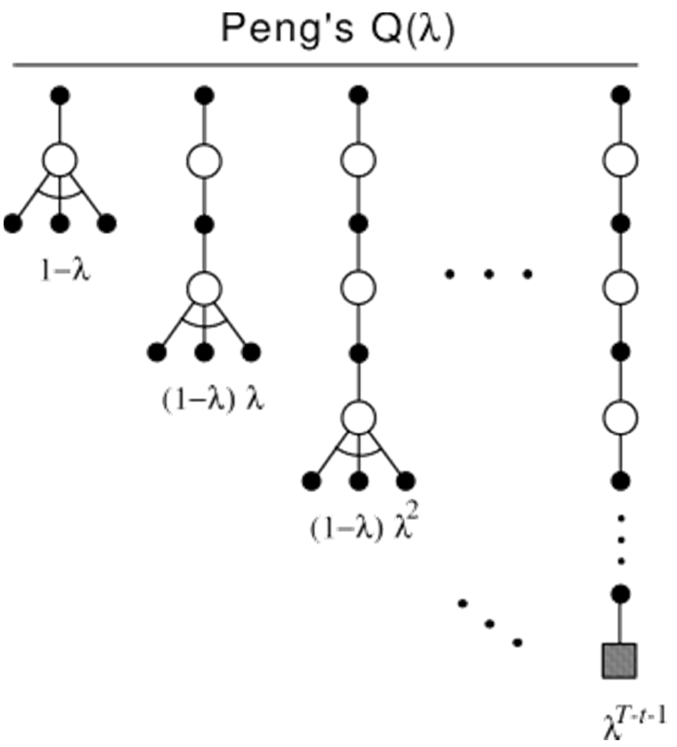


Figure 14 Backup diagram for Peng's Q(λ)

Conceptually, Peng's Q(λ) uses the mixture of backups shown in Figure 14. Unlike Q-learning, there is no distinction between exploratory and greedy actions. Each component backup is over many steps of actual experiences, and all but the last are capped by a final maximization over actions. The component backups, then, are neither on-policy nor off-policy. The earlier transitions of each are on-policy, whereas the last (fictitious) transition uses the greedy policy. As a consequence, for a fixed non-greedy policy, converges to neither nor under Peng's Q(λ), but to some hybrid of the two. However, if the policy is gradually made more greedy, then the method may still converge to , even if this has not yet been proved. Nevertheless, the method performs well empirically. Most studies have shown it performing significantly better than Watkins's Q(λ).

Chapter 5. Objectives and adopted approaches

5.1. *Objectives*

In the contest of this work, the most important target is to assure that the algorithms based on Reinforcement Learning perform better at least of the most common CAC algorithm, the peak based's one. That is, the RL based CAC algorithms (in the future CAC-RL) should guarantee to the network operator the maximum income, preserving the clients' quality of service. It means that on one hand the network resources should be exploited in order to optimally use the available resources and on the other hand the contractual quality of service should be assured to all the already accepted end-users connections.

In addition to that, the RL algorithm must be decoupled from other Resource Management procedures and must be technology independent, allowing network operators to update their transport technologies without modifying the resource control functions. This is especially true for the wireless networks market where new standards impose high investment in new infrastructures. The CAC-RL must adapt to different network scenarios, thus another objective is the robustness and independence from the specific statistical behaviour of the traffic; CAC-RL should detect and react to the changes of the environmental conditions.

Among the objectives depicted above, I've decided to subdivide part of them in order to introduce more complexity in RL algorithms in a step-by-step way:

- first of all an RL algorithm was developed able to control only bandwidth usage, while maximizing the income in the long run;
- then, when a good algorithm was found, another constrain was introduced, i.e. the control of the blocking probabilities;

- finally the final constrain was introduced, i.e. the control of the dropping probability.

5.2. *First RL approach*

In order to utilize the RL algorithm to solve the CAC problem, it is needed to identify the fundamental elements for the implementation of the admission control algorithm i.e. the system states, actions and rewards.

5.2.1. **Definition of state, actions and reward**

At any given step t, the system is in a particular configuration s_t which we identify with the number of active connection of each traffic class, i.e. $s_t = (n_1, n_2, ..., n_N)$, where n_i is the number of active connections of class *i*. At random times an event *e* can occur (we assume only one event can occur at any step), where *e* is either a new call arrival or a call termination. This assumption is not restrictive: in a time-continuous scenario the probability two events occur exactly at the same time is zero. The agent has to find an optimal policy which maximizes the cumulative measure of the rewards received over time choosing between two possible actions for each request: accepting (a = 1) and rejecting (a = 0) the arriving connection request of class *i*; connection terminations are not decision points so that no decision needs to be taken.

The crucial point in the definition of an environment behaviour is the analytical form of the reward for the acceptance or rejection of a connection request in a state *s*. It should take into account all the objectives established previously. A good guideline to achieve the goal is to build the reward function as a superposition of additive contributes, each of which regarding a different objective: in this way future refinements of the model such as the introduction of new objectives will be easily made. I have chosen the following general reward functions for the acceptance ($r_{Ai}(B,\vec{\lambda},\vec{\mu},\vec{\rho})$) and rejection ($r_{Ri}(B,\vec{\lambda},\vec{\mu},\vec{\rho})$) couple of a generic class of service:

$$\begin{aligned} r_{Ai}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) &= f(B) - \Delta_i^{Bw}(\vec{\lambda},\vec{\mu},\vec{\rho}) \\ r_{Ri}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) &= f(0) - r_{Ai}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) \end{aligned} \tag{21}$$

Where:

- $f(\cdot)$ is the bandwidth contribute;
- $\Delta_i^{Bw}(\cdot)$ is the inversion contribute (connected to the bandwidth only) for the *i*-th class of service;
- B is the used bandwidth in the state *s* – note that B is variable during the time; for notation simplicity we will omit to explicitly depict time dependences;
- $\vec{\lambda} = (\lambda_1,...,\lambda_N)$ is the new call frequency vector, being λ_i the frequency of new call for an *i*-th class of service connection;
- $\vec{\mu} = (\mu_1,...,\mu_N)$ is the holding time vector, being μ_i the average duration of an *i*-th class of service connection;
- $\vec{\rho} = (\rho_1,...,\rho_N)$ is the price vector, being ρ_i the price for band and time unit of an *i*-th class of service connection. In particular $\rho_i = \frac{r_i}{B_i}$ where r_i is the reward per time unit and B_i is the nominal bandwidth (occupied by a connection) of the i-th class of service.

The definition of reward for the call rejection comes from the choice to satisfy the following constraint, and so to assure a good control of the algorithm behaviour:

$$r_{Ai}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) + r_{Ri}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) = f(0).$$

The first term of the first equation in (21), the bandwidth term, should assure that the agent learns to accept a connection only if at least a portion of the bandwidth is available, and to reject it if the bandwidth of the link is filled. I have chosen for $f(B)$ the following sigmoid function,

$$f(B) = \frac{1}{1+e^{(B-B_L)/B_0}},$$

where:

- B is the actual bandwidth;
- B_L is the nominal bandwidth of the link;
- B_0 is a free parameter to be chosen conveniently for an optimal tuning.

The behaviour of the couple of reward functions for a generic class of service is shown in Figure 15 (where we temporarily considered $\Delta_i^{Bw}(\cdot) = 0$). As it is possible to see, changing the value of the tuning parameter B_0 the graph results in curves more "crushed" but the 0.5 value is always reached for $B/B_L = 1$. We call this point the "*inversion point*", i.e. the point where $r_{Ai} = r_{Ri}$: before there is a greater reward accepting calls, after rejecting them. This choice is intuitive: supposing requested bandwidth for a certain call is its maximum occupied bandwidth and that there is no bandwidth occupied for signalling (or, as preferred, consider this bandwidth not available to be assigned to call requests), the best solution to augment operator incoming is to accept new calls as long as there is bandwidth, then the operator is forced to reject in order to continue to guarantee QoS constrains.

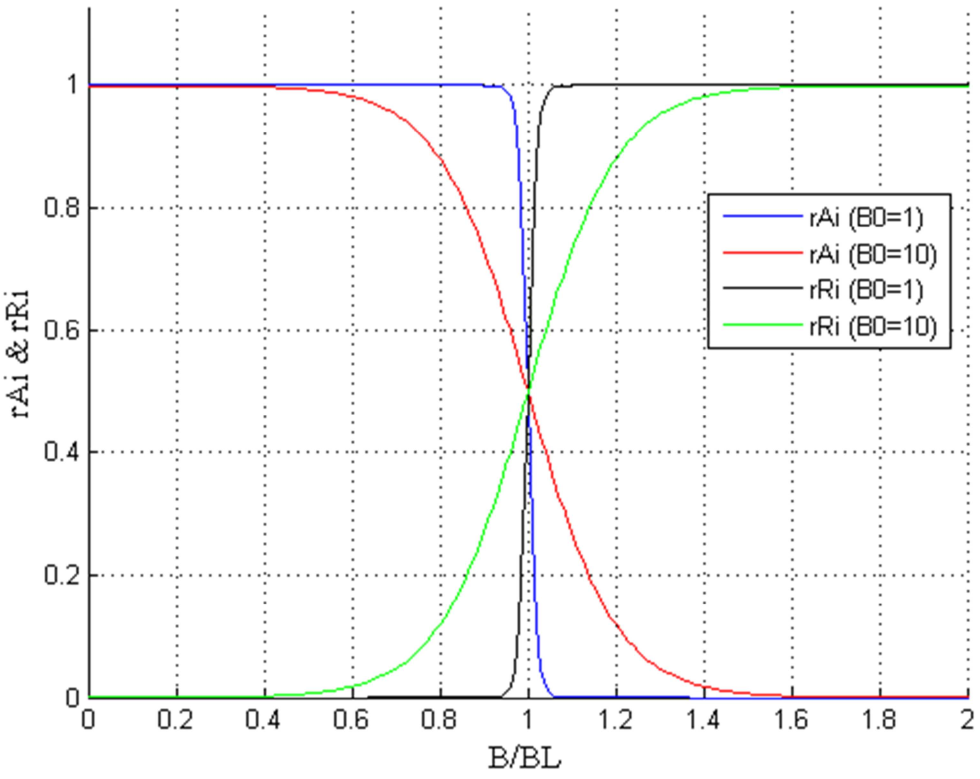


Figure 15 Acceptance and rejection reward functions for B_0=1 and B_0=10

The second term in (21), the inversion term, comes from the network operator need to maximize its profit over the long run. More in detail I have introduced it to assure significant improvement in the management of connection requests in congestion conditions.

The value of $\Delta_i^{Bw}(\cdot)=0$ should be as higher as the class of services i is considered not convenient by the network operator, as smaller as considered convenient. Figure 15 shows the couples of reward functions for four classes of service, in which four decreasingly values of Δ has been considered.

The main effect of Δ is the left shift of the inversion point. The more Δ increases the more the shift increases. The rationale is the following: without Δ, the policy is exactly the greedy one, i.e., accept if there is bandwidth. In this way all calls are considered equivalently convenient; vice versa, usually certain classes of service are more convenient than others and so it could be intuitive to try to leave some bandwidth free for them: this is exactly what Δ

permits. The best classes have Δ near zero, which implies accept them whenever remains bandwidth (i.e. unless occupied bandwidth is around 100%); the worst classes have high Δ, which implies to accept them whenever the B/B_L ratio is sufficiently low (i.e., accept till a certain percentage, considerably less than 100%, of the bandwidth is occupied). In this way the agent could follow a policy which accepts only the most convenience class of service requests in situation next to the bandwidth saturation.

First of all I have considered the problem of bandwidth control only, so that I set the inversion contribute as follows:

$$\Delta_i^{Bw} = c \cdot g_i(\cdot) \tag{22}$$

where c is a free parameter that it is possible to use to obtain an optimal tuning of the algorithm ($0 \le c \le 0.5$), and $g_i(\cdot)$ is the so called "*inversion function*" ($0 \le g_i(\cdot) \le 1$). It takes into account the convenience in accepting the other class of service requests respect to a *i*th one; an analytical expression assuring our scopes is the following:

$$g_i(\vec{\lambda}, \vec{\mu}, \vec{\rho}) = \frac{\sum\limits_{k \neq i}^{N} \lambda_k \cdot \rho_k \cdot \mu_k}{\sum\limits_{k=1}^{N} \lambda_k \cdot \rho_k \cdot \mu_k} \tag{23}$$

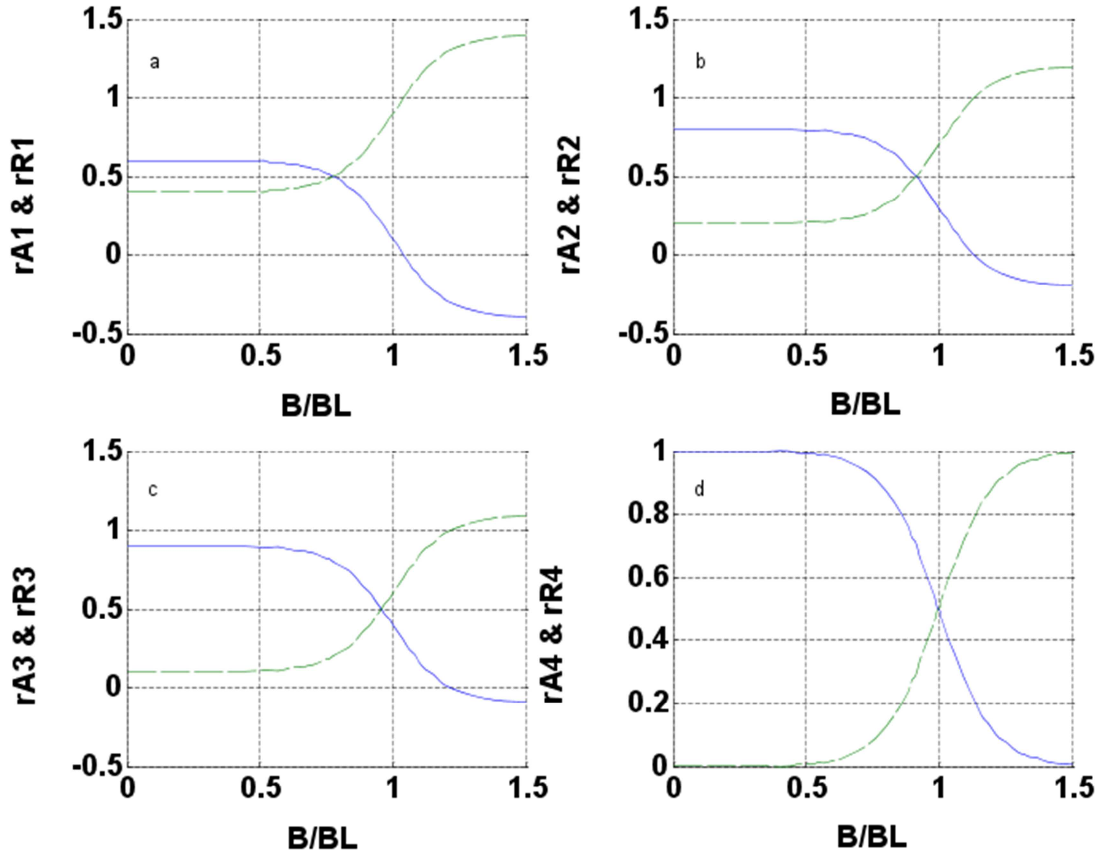


Figure 16 Reward functions for a) $\Delta = 0.4$**, b)** $\Delta = 0.2$**, c)** $\Delta = 0.1$**, d)** $\Delta = 0$ **respectively**

The rationale is the following: a certain class *k* of service is more convenient:

- if its price (for bandwidth and time units, ρ_k) is higher
- if its frequency (λ_k) is higher
- its duration (μ_k) is higher.

While the first point is intuitive, the second and the third could be not. Regarding the second, suppose to have two different classes of services, *i* and *j*, which have all parameters equals but $\lambda_i > \lambda_j$. This implies the probability to have a new call of class *i* is greater than having one of class *j*: so that, it will be counter-productive to leave space for calls of class j rejecting calls of class *i*, because *j*th ones are more rare. In the worst case (i.e. λ_j □ 0) leaving space for calls of class *j* will be useless, because calls of class *j* are so rare that probably these will not arrive at all in the near future. Regarding the third, supposing again to have two classes of services, *i* and *j*, which have all parameters equals but $\mu_i > \mu_j$: in this case it will be convenient to accept calls of class *i* because these will go on further than *j*th ones, guaranteeing so higher revenues. One could counter that accepting *i*th calls will occupy bandwidth, but, with respect to leave space for class *j*, it is convenient because a call of class *j* will occupy the same bandwidth (per euro) and usually to last less than calls of class *i*, so that guaranteeing less reward.

5.2.2. <u>Control of blocking an drop probabilities</u>

The previously depicted algorithm is able to maximize the network operator income while guaranteeing the bandwidth constrain, i.e. while not accepting new connections if there is no sufficient bandwidth for them. In a real telecommunication scenario, the bandwidth constrain is not the only one. As already stated, in model based algorithms, in particular in these which does not use any measurement system, it is not possible to take care of constrains like bit-error-rate (BER), delays, jitter and similar issues (which require measurements), but it is still possible to try to maintain sufficiently low two other important parameters, namely the blocking probability and the dropping probability. These last two indicates, roughly speaking, the probability of a connection to be blocked or to be dropped by the system.

It is possible to use several definition of the last two probabilities: I've chosen the following ones, which better adapt to the scenario:

$$P_i^b = \frac{blocked\_call\_amount_i}{total\_call\_amount_i}$$

$$P_i^d = \frac{dropped\_call\_amount_i}{total\_call\_amount_i} \qquad (24)$$

where *b* means blocked, *d* dropped and *i* indicates the *i*-th class of service. Note that defined in this way, these should be better named blocking ratio and dropping ratio instead of probabilities, but in the literature the preferred naming is "probability". Furthermore, it becomes crucial to decide if these ratio should be calculated in the (theoretically infinite) window of time during which the algorithm works or if these ones are to be defined in a fixed time window of length *T*. The last one is the most commonly adopted, in particular because it is usually the more stringent: so that, we will use $P_i^b(t)$ and $P_i^d(t)$ defined as in (24) but where we consider only calls (arrived, blocked and dropped) in the "sliding" time window (*t*, *t*-*T*].

In order to introduce blocking probability and dropping probability control, we need to slightly modify the (21), as depicted below:

$$r_{Ai}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) = f(B) - \Delta_i^{Bw}(\vec{\lambda},\vec{\mu},\vec{\rho}) - \Delta_i^{Pb}(P_i^b(t),P_i^{bMax}) - \Delta_i^{Pd}(P_i^d(t),P_i^{dMax})$$
$$r_{Ri}(B,\vec{\lambda},\vec{\mu},\vec{\rho}) = f(0) - r_{Ai}(B,\vec{\lambda},\vec{\mu},\vec{\rho})$$

Where:

- $\Delta_i^{Pb}(P_i^b(t),P_i^{bMax})$ is the inversion contribute (connected to the blocking probability only) for the *i*-th class of service;
- $\Delta_i^{Pd}(P_i^d(t),P_i^{dMax})$ is the inversion contribute (connected to the dropping probability only) for the *i*-th class of service.

In particular

$$\Delta_i^{Pb}(P_i^b(t),P_i^{bMax}) = -d\,\frac{P_i^b(t)}{P_i^{bMax}};$$

$$\Delta_i^{Pd}(P_i^d(t),P_i^{dMax}) = -e\,\frac{P_i^d(t)}{P_i^{dMax}}.$$

In both formulas the minus is due to the fact that $r_{Ai}(\cdot)$ should augment as much the $\frac{P_i^x(t)}{P_i^{xMax}}$ ratio augment; in other words, if the system reaches the limit of blocking and dropping probabilities, it tries to compensate augmenting the number of accepted calls of that class of service (or, equivalently, to avoid to reject or to drop calls of that class of service); finally, *d* and *e* are two tuning parameters.

Furthermore, while introducing dropping probability control, it is obvious there should be the possibility to drop calls, so that there should be a policy to determine which ongoing connections should be dropped. The proposed algorithm will drop only one call at a time selecting the ongoing call which has the minimum expected income, selecting it using the following rule:

$$\min_i((T_i^{mean} - \max_k(T_i^k))\rho_i) \tag{25}$$

In particular, tanks to the previous one, it is selected the k-th ongoing call of the i-th class of service which expected income is the lowest, among all the ongoing calls. The rationale of (25) is the following: $T_i^{mean} - \max_k(T_i^k)$ indicates the difference between the mean duration time of the calls of i-th class of service and the actual duration of the ongoing longest (thanks to the max) call of that class. Shorter calls are expected to last more than longer ones, so that the first ones will guarantee more incomes (having other parameters all equal); ρ_i is the usual price for band and time unit of the *i*-th class of service connection.

5.3. *Second RL approach*

The second RL approach uses an extremely simple assumption: RL is able to learn from experience, stating that an opportune reward function is provided to the algorithm. In this second approach the reward function chosen is extremely simple:

$$r_{Ai} = \begin{Bmatrix} +mht_i \cdot r_i, & in\ case\ of\ acceptance \\ -K, if\ no\ sufficient\ available\ bandwidth\ to\ accept\ the\ call \end{Bmatrix}$$

$$r_{Ri} = -mht_i \cdot r_i \tag{26}$$

where

- $mht_i \cdot r_i$ is the mean duration (at time *t*, at which the calculation of r_{A_i} is done) of the calls of class of service *i*;
- r_i is the reward for time unit of the class of service *i*;
- *K* is a constant (*K*>0): the more higher is *K*, the more the agent is penalized when accepting a call if there is not sufficient bandwidth for it.

Summarizing, apart the case of no sufficient bandwidth, the agent gets the same absolute value for both actions, but positive in case of acceptance, negative otherwise.

This simple couple of reward functions demonstrated to be really good to solve the CAC problem (as depicted in the results chapter) and their rationale is extremely simple to explain.

Remember that the Q(λ) algorithm chooses the best action to be performed starting from the Q(s,a) and following an epsilon-greedy policy, i.e., selecting the action corresponding to the maximum Q(s,a), apart a certain percentage epsilon of the times, in which it explores, selecting the action to be followed randomly.

Suppose to have just two classes of services, CoS_1 and CoS_2, occupying respectively the bandwidth B_1 and B_2; suppose also the respective rewards are r_1 and r_2. Note that, for every state *s,* there are four possible actions to be chosen: A_1, A_2, R_1, and R_2, which are respectively accept a call of class 1 and 2, and reject a call of class 1 and 2.

Consider initially to be in a certain state *s* in which neither a call of CoS_1 nor of CoS_2 can be accepted, because the remained bandwidth B_r is less than B_1 and B_2 ($B_r < B_1$ and $B_r < B_2$);

consider also this state was never visited before. In this case, the value of the Q function is depicted in the following table:

Q function	Q(s,A_1)	Q(s,A_2)	Q(s,R_1)	Q(s,R_2)
Value	0	0	0	0

Table 1 Initial values of Q(s,a) As it is possible to see, all Q(s,a) are equals to zero; in this case, the algorithm will select randomly one of these actions. Suppose the incoming call is of CoS$_1$, then it will select or A_1 or R_1. Consider the selected action is R_1, the agent gets the reward "-*K*"; it could be useful to remember here the updating rule:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a),$$

where

$$\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t).$$

and consider γ, α and e_t all equals to 1 just for facilitate calculations (obviously their values should be opportunely set, as previously highlighted, otherwise the algorithm will not converge; putting them to 1 in this example is just to give an idea about how the process goes), so that they become:

$$Q_{t+1}(s,a) = Q_t(s,a) + \delta_t$$

$$\delta_t = r_{t+1} + \max_{a'} Q_{t+1}(s_{t+1}, a') - Q_t(s,a)$$

Considering $s_{t+1} = s_t$, $\max_{a'} Q_{t+1}(s_{t+1}, a') = 0$, and $Q_t(s_t, R_1) = 0$, the table will update as follows:

Q function	Q(s,A_1)	Q(s,A_2)	Q(s,R_1)	Q(s,R_2)
Value	0	0	$-mht_1 \cdot r_1$	0

Table 2 Values of Q(s,a) after rejection of a call of CoS$_1$

Suppose now another call of CoS_1 arrives, now the best action to be performed is A_1, so that the table will update as follows:

Q function	$Q(s,A_1)$	$Q(s,A_2)$	$Q(s,R_1)$	$Q(s,R_2)$
Value	$-K$	0	$-mht_1 \cdot r_1$	0

Table 3 Values of Q(s,a) after acceptance of a call of CoS_1

If one guarantees

$$|K| > |mht_i \cdot r_i| \quad \forall i \tag{27}$$

then the action R_1 is more convenient than A_1, so that the algorithm learns that, in this state, it is better to reject calls of CoS_1 than accept them. Similar behaviours there are for the CoS_2; note also the algorithm arrives to the same results also if it decides to accept the first call. The "final" value of Q(s,a) is depicted in the following table:

Q function	$Q(s,A_1)$	$Q(s,A_2)$	$Q(s,R_1)$	$Q(s,R_2)$
Value	$-K$	$-K$	$-mht_1 \cdot r_1$	$-mht_2 \cdot r_2$

Table 4 "Final" values of Q(s,a)

In the reality, if one continues with the updating rule, there will be some influences between the two classes of services, which enormously complicates calculations: in any case, if (27) is valid, then the algorithm will always converge to the fact that it is better to reject any incoming call in the state *s* instead accepting them, so that we have depicted how the algorithm is able to respect the bandwidth limit constrain.

A similar argumentation is possible to be done regarding the maximization of network provider incoming. In the previous algorithm, the sigmoid function guaranteed there should be an inversion point starting from which it is more convenient to reject calls instead of accepting them. Opportunely tuning some parameters, one could guarantee that, when mostly of the bandwidth is used, only calls of more convenient classes of services are accepted, while the calls of other classes are rejected to leave some space free for the first ones. It is possible to show the Q(λ) algorithm, with the rewards as in (formula), reaches the same results.

With the same assumptions of the previous example, but supposing in the state *s* there is remaining bandwidth just to accept or a call of CoS$_1$ or a call of CoS$_2$, we will start, as usual, with an "empty" table of Q(s,a) (Table 1); suppose also calls of CoS$_2$ are more convenient of the ones of CoS$_1$ (this is true if $mht_2 \cdot r_2 > mht_1 \cdot r_1$); finally remember that $mht_i \cdot r_i > 0 \ \ \forall i$.

Suppose, as usual, a call of CoS$_1$ arrives, suppose the algorithm accepts it and suppose that the state s_{t+1} where the system goes to has a $Q(s_{t+1}, a)$ as in Table 1 (just for sake of simplicity: the algorithm will converge to similar results also for $Q(s_{t+1}, a)$ much more complex of the ones in Table 4, just guaranteeing (27); in any case all states were never visited have a Q(s,a) equals to the one in Table 0, so that, in this example, it is correct to consider s_{t+1} as a state never visited) . So that, because $\max_a Q(s_{t+1}, a)$ the resulting Q(s,a) table is

Q function	Q(s,A$_1$)	Q(s,A$_2$)	Q(s,R$_1$)	Q(s,R$_2$)
Value	$mht_1 \cdot r_1$	0	0	0

Table 5 Values of Q(s,a) after acceptance of a call of CoS$_1$

If now, in s_{t+1}, a call of CoS$_2$ arrives, the agent is forced to reject it, so that the network owner has earned, in total, $mht_1 \cdot r_1$. Suppose, while being in s_{t+1}, a call of CoS$_1$ ends, the system returns to the "initial" state *s*. Now, the same happens with a call of CoS$_2$: a new arrival and a termination, in order to return to the state *s*. The final table will be:

Q function	Q(s,A$_1$)	Q(s,A$_2$)	Q(s,R$_1$)	Q(s,R$_2$)
Value	$mht_1 \cdot r_1$	$mht_2 \cdot r_2$		0

Table 6 Values of Q(s,a) after acceptance of a call of CoS$_2$

Suppose now the agent has sufficiently explored the state s_{t+1} in order to have for it the complete Table 4: at this stage the Q(s,a) is arrived to a crucial point, in which it is already able to determine that it is better to reject calls of CoS$_1$ instead of accept them, in order to leave some space for calls of CoS$_2$.

In fact, suppose now it arrives a call of CoS$_1$. We have two different values of δ_t; the fist value corresponds to *a*=A$_1$, and:

$R_{t+1} = mht_1 \cdot r_1$;

$Q(s_t, A_1) = mht_1 \cdot r_1$;

$\max_a Q(s_{t+1}, a) = -mht_1 \cdot r_1$;

for a total $\delta_t^{A_1} = -mht_1 \cdot r_1$.

The second value corresponds to *a*=R$_1$, and

$R_{t+1} = -mht_1 \cdot r_1$;

$Q(s_t, R_1) = 0$;

$\max_a Q(s_{t+1}, a) = mht_2 \cdot r_2$;

for a total $\delta_t^{R_1} = -mht_1 \cdot r_1 + mht_2 \cdot r_2 - 0 = mht_2 \cdot r_2 - mht_1 \cdot r_1$.

With these values of deltas, it is easy to see that Q(s,A$_1$) goes to zero, while Q(s,R$_1$) goes to $mht_2 \cdot r_2 - mht_1 \cdot r_1$ (which is higher than 0 for hypothesis), so that the "final" table of Q(s,a) is:

Q function	Q(s,A$_1$)	Q(s,A$_2$)	Q(s,R$_1$)	Q(s,R$_2$)
Value	0	$mht_2 \cdot r_2 - mht_1 \cdot r_1$	$mht_2 \cdot r_2 - mht_1 \cdot r_1$	X

Table 7 Final values of Q(s,a)

where $0 > X > -mht_1 \cdot r_1$ could be calculated only fixing and considering all parameters, but, in any case, Q(s,R$_2$) < Q(s,A$_2$).

Summarizing, the algorithm learnt that in this case accepting calls of CoS_2 is the best action, while, with respect to CoS_1, the best action is to reject new calls.

Chapter 6. Implementation

6.1. *Introduction*

In this chapter implementation of previously depicted approaches will be presented, indicating used tools and API, in terms of functioning and capabilities.

6.2. *Implementation details*

In order to validate and test the presented algorithms, I used a library of classes for implementing reinforcement learning in Java called Piqle [28]. The core component of this library is the standard Java interface for programming RL problems. This Java interface is an adaptation of the C++ standard of Sutton and Santamaria's RL interface [29].

The Java language has been chosen for several reasons: it is powerful enough for simulation needs and it is relatively platform independent. It also allowed me to interface the platform with existing strong machine learning implementations.

Like the C++ version, the Java platform is comprised of three core classes, which correspond to the three basic entities of the RL problem. They are the Agent, the Environment, and the Simulation. This code includes interfaces and classes for implementing the communication between agents and environments. The library contains also Action and State interfaces and classes for programming TD learning algorithm. Specifically, I had to add some classes to Piqle package in order to adapt them to the CAC problem.

As simulation tool I used OMNET++ ([30], [31], [32], [33]) simulator. The simulations have been done to show the efficiency of the algorithm. OMNeT++ is a open-source, component-based, modular and open-architecture simulation environment with strong GUI support and an embeddable simulation kernel. Its primary application area is the simulation of communication networks and because of its flexible architecture, it has been successfully used in other areas like the simulation of IT systems, queuing networks, hardware architectures and business processes. I have developed a complete framework to simulate and analyze CAC algorithms based on reinforcement learning approach.

In the following are depicted in details the three fundamental parts of the framework: Piqle, OMNeT++ and CAC.

6.2.1. PIQLE

The PIQLE simulation tool was primarily designed for implementing and testing the algorithms and problems described in Sutton's and Barto's Reinforcement Learning, An Introduction. Reinforcement learning algorithms are relatively generic, as based on a quite abstract notion of state, action, reward. I tried to respect the generality and transcript it in Java, a language very well suited for this. Defining and designing separately the notions of agent, algorithms, environment, and how those objects communicate with each other made it possible to obtain a very general platform for reinforcement learning experiments, easy to understand, easy to use, and also easy to extend, by mean of adding new algorithms, new problems, new agents.

Piqle is organized around three main entities:

- Environments representing the universe of the problem, i.e. the physics (or the rules for a game) and the different methods to describe states and actions.
- Algorithms which are software elements able to choose the next action to be performed. Within the framework, algorithms are called Selectors. Interesting algorithms are those which are able to learn.
- Agents are the "interface" between Environments and Algorithms.

On top of those three main features, the referees are scheduling the succession of phases, as illustrating here after:

- The Environment tells the Agent in which state it is at this time, and provides the list of possible actions (Figure 17).
- The Agent communicates this information to its Algorithm (Figure 18).
- The Algorithm chooses the action to perform, and gives its answer back to the Agent (Figure 19).
- The Agent tells the Environment what action it wants to perform (Figure 19).
- The Environment computes the new state, the reward, and the new list of possible actions, and sends those information to the Agent (Figure 19).
- The Agent transmits everything to the Algorithm, which can now learn from experience, as it received a reward for its former choice, choose the next action to perform, and the cycle begins again (Figure 20).

Agents are just interfaces between Environments and Algorithms: they just receive and transmit information; they do not need to know exactly in which universe they are moving. Algorithms are just asked to choose an action into a given list of actions, and receive a reward for their last choice. The thing they have to remember (or learn) is, roughly speaking, what was (or what will be) the next reward if they choose a certain action. This reasoning scheme can be made independent of the Environment, as the Algorithm mainly has to store (and retrieve) its past experience as triples (state, action, reward). Finally, environments can be described as generic and abstract entities.

Hence the only place where one will have to really describe the problem one wants to solve is the instantiation of the generic classes of the environment package.

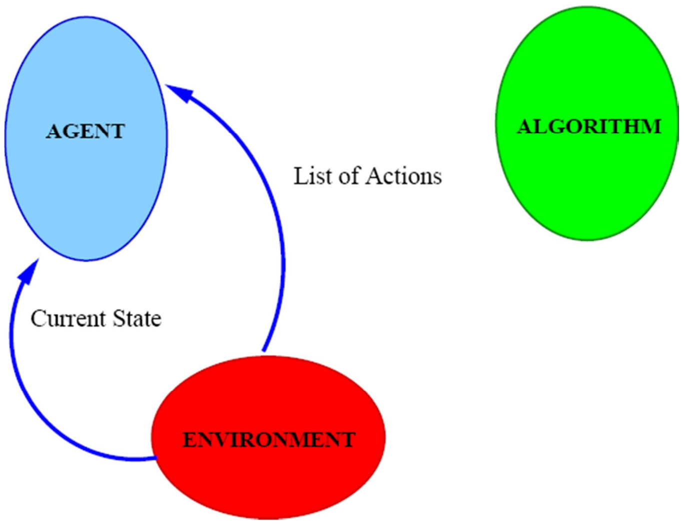


Figure 17 The Agent learns its State

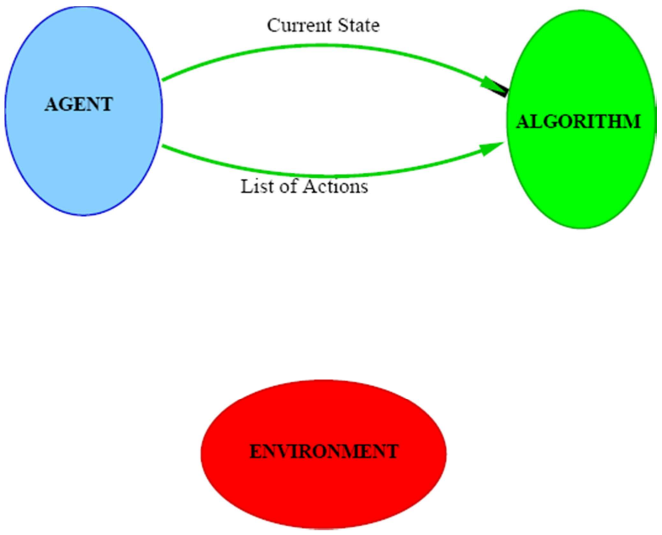


Figure 18 Algorithm knows its current State

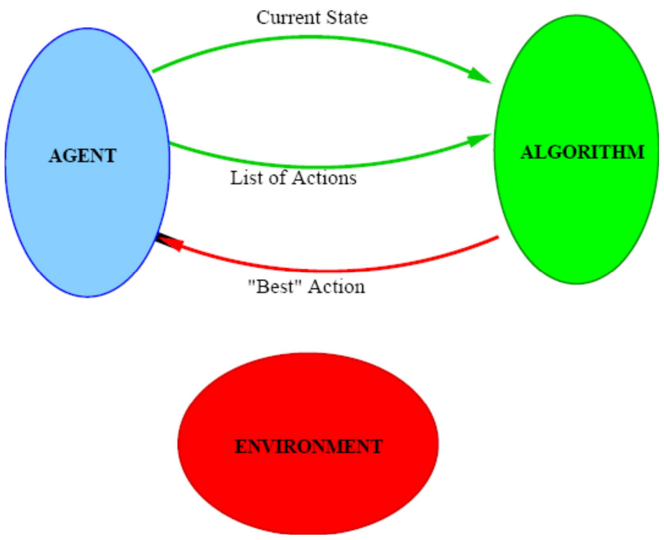


Figure 19 The Algorithm chooses the Action

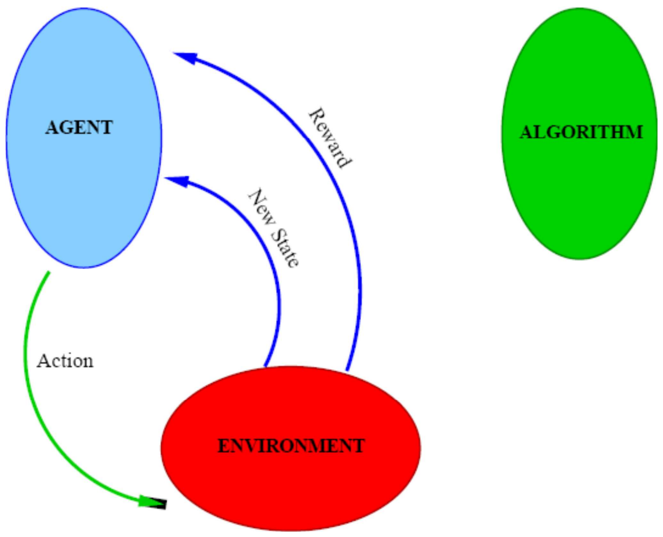


Figure 20 The Environment computes reward and New State

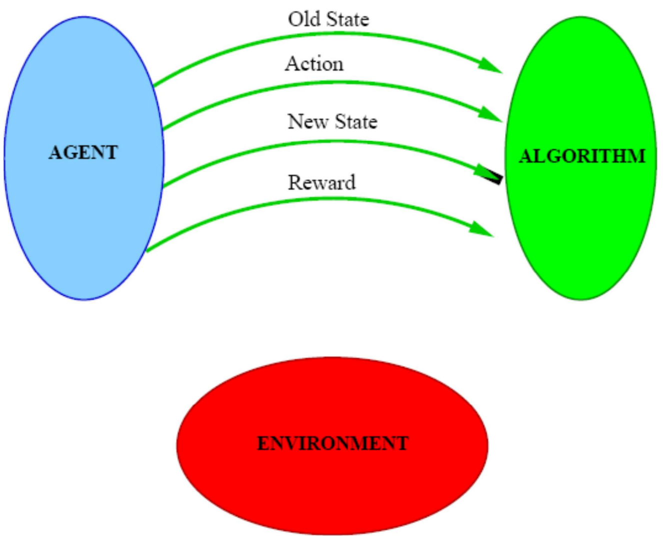


Figure 21 The Algorithm can learn

Those remarks directly lead to the Java organization of Piqle:

1. Three main packages independent from the problem to solve: agents, referees, algorithms.
2. One generic environment package for the common behaviour of any depictable environment.
3. For each problem, a new package to instantiate the environment package.

Now it is possible to enter a little bit more in details regarding each entity, starting with Agents.

Agents have two closely related roles:

- Allowing the environment and the algorithm to communicate with each other.
- Communicate itself with the referee, in order to schedule each episode.

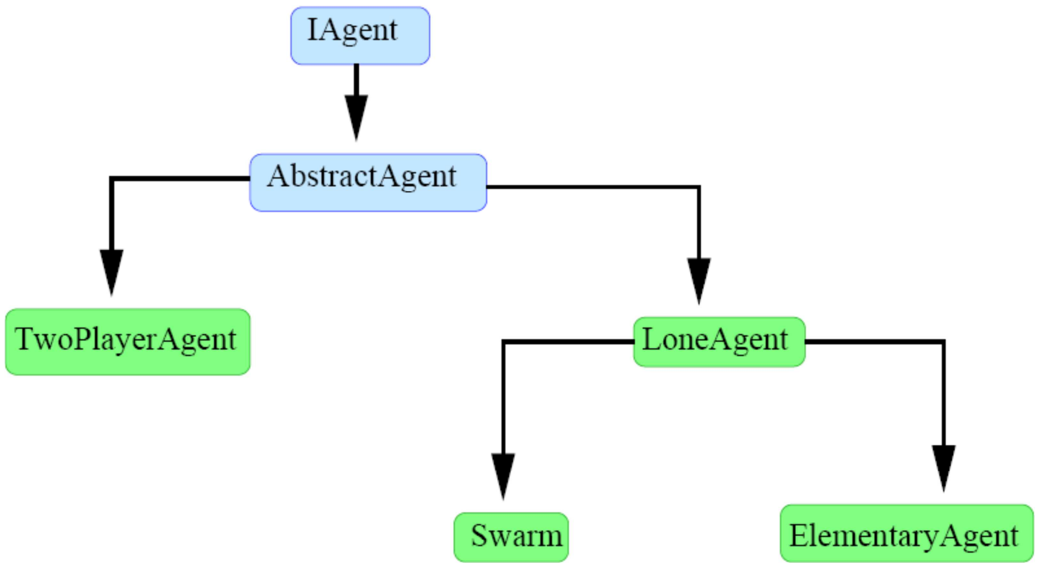


Figure 22 The Agent's hierarchy

Here following the blocks of the above picture are explained.

- As the agent provides an interface between an algorithm and an environment, it must contain two fields for those two entities. The first two methods: *getAlgorithm* and *getEnvironment* allow access to each of those fields.
- The next two methods, *enableLearning* and *freezeLearning*, are controlling the learning behaviour of the agent: one can ask the agent to stop learning at a certain stage, and see whether this agent behaves cleverly enough.
- Method *getLastReward* asks the algorithm (in cascade, through the current state) for the reward corresponding to the last action of the agent. This is used by referees to compute the total reward for an episode.

- Method *act* is the core of the agent behaviour: here the agent will ask its algorithm for the best action to perform, and return the chosen action to the referee (which will then communicate this choice to the environment).
- Both methods *saveAgent* allow saving an agent state into a file. Each non abstract agent classes (*LoneAgent* and *TwoPlayerAgent*) also implement *readAgent* methods. As those methods are static, they cannot appear into the definition of an interface.
- In case there are some settings to be done at the beginning of an episode (resetting reward, resetting the algorithm, …) the interface provides the method *newEpisode*.

Directly above the *IAgent* interface, the abstract class *AbstractAgent* defines the code of quite all the methods defined in the interface. Static methods *readAgent* can still not be written here, as we need to know whether we are speaking of a *LoneAgent* or of a TwoPlayerAgent.

The basic class for one player game agent is *LoneAgent*. It is simply an *AbstractAgent* with the only constraint of having an *IEnvironmentSingle* associated environment. It thus manipulates states of class *AbstractState*. Note that now, it is possible to write the code for the *readAgent* methods.

Two classes are derived from the *LoneAgent* class: both are related to the multi-agent extension of Piqle. Roughly speaking, a multi-agent system is a gathering of limited (in terms of perception and of behaviour) agents: the *ElementaryAgents*. Those agents are grouped into a *Swarm*, which is also an *IAgent*. This Swarm communicates with the environment in both directions, as any *IAgent* does:

- The Swarm receives the description of the current environment’s state.
- The Swarm dispatches this information to the ElementaryAgents it is composed of.

- Each ElementaryAgent sees this current state through its limited perception: a Filter extracts from the original state's information the only part of it visible for this ElementaryAgent.
- Each ElementaryAgent chooses the action it will perform.
- The Swarm collects all those individual actions into a ComposedAction, and sends it to the environment.
- The environment uses this ComposedAction to compute its next state, and sends back to the Swarm the reward for the current episode's step.
- Finally, the Swarm informs all its components of this reward.

Now it could be introduced the Environment package; inside of it there are three main entities:

1. Environments which are all that is needed to compute new state and reward, indicating also when the simulation should stop.
2. Actions which are able to generate any action for a given problem, tools for comparing, copying and coding actions.
3. States Code which compare, copy and manage states supplying also auxiliary methods to access the associated environment.

All environments must implement the *IEnvironment* interface. This interface defines the expected behavior of any instantiated environment, which means:

- Given a state of the environment, telling what actions are possible in it.
- Given a state and an action, compute the next state. This computation can also be non deterministic or probabilistic.
- Computing a reward from the previous state, the current state, the action taken (as usual in reinforcement learning).

- Indicating whether the simulation (or the game) is over or not (and, in case, who won).

Now it is possible to introduce Actions. *A*ctions are probably the simplest classes in Piqle. The only things one must be able to do while dealing with Actions is to create them, compare them for equality, code them for learning or storing algorithms. The basic definition of interface *IAction* is described by following methods:

- ***copy*** which creates a new Action which is the copy of the previous one and is used when storing (state, action) pairs.
- ***nnCoding, nnCodingSize*** used for defining a coding of an action as a real-valued vector, to be used in algorithms based on neural networks.
- ***hashCode, equals*** considering it is up to the programmer to define when two actions are declared equal. Equality is tested in Java HashMaps, using those two methods.

Moreover *ActionList* is an auxiliary class to store the list of possible actions from a given state. This list is given to the algorithm which makes it choice among the elements (Actions) of this list.

Finally it is possible to introduce States. A state must define accurately a given configuration of the environment: one must be able to decide whether two states of the environment are equals, or equivalent. The basic definition of interface *IState* is described below:

- ***setEnvironment, getEnvironment*** which connects the state with the environment it belongs to.
- ***getActionList, modify, getReward, isFinal*** which just call the corresponding methods in the Environment class.
- ***copy*** which is used to clone a state (mainly when storing the state into a HashMap).

- ***nnCodingSize, nnCoding*** which define the format of the state's representation for use in algorithms based on neural networks.
- ***hashCode*** which is equals to the same described above for *IAction*

The abstract class *AbstractState* only defines the code of the first four methods seen above, the ones connecting the state with its environment. It also sets the field corresponding to that environment.

It should be remarked that agents, and thus algorithms, may not be able to perceive all the details of a state. This is reflected in the fact that states may contain fields which are used by the environment to compute a new state, while those fields are used neither for the computation of hashCode and equal method, nor for the definition of nnCoding.

Now it is time to introduce the last entity of the Piqle framework, the Algorithms. **A**n algorithm is defined for each IAgent: its primary role is to choose an action to perform, given a list of possible actions.

Good algorithms will choose the best action. Learning algorithms will use their past experience to improve their choice, hopefully choosing best actions.

Thus algorithms in Piqle mainly implements:

- A method for choosing an action within a list of them.
- A method for learning, given the starting state, the chosen action, the new state after applying the action, and the reward given by the environment for this action's choice: this is the standard paradigm of reinforcement learning.

Some algorithms may not learn, some others may choose their action at random, depending of those two methods' implementations. Algorithms have several ways to store their past experiences, different ways to use this past experiences, and also different ways to use this experience. All that results in a package containing a lot of algorithms, organized in a multiple levels hierarchy.

Main activity of an algorithm is to choose an action This part of its behavior is contained in the basic interface IStrategy. Furthermore it learns, which is the main method of interface IStrategyLearner. This interface also contains the method newEpisode which performs all the necessary initialization at the beginning of an episode.

The API provide also methods to retrieve and analyze values calculated during learning.

After this introduction it is possible now to easily describe the top of the algorithms' hierarchy:

- A basic interface defining the getChoice method: IStrategy
- Another basic interface defining learn and newEpisode: IStrategyLearner
- A third interface gathering the two previous ones, and adding method extractDataset: ISelector

All algorithms defined so far in Piqle are implementations of the ISelector interface, some directly, others through a cascade of intermediate classes, forming the sub-hierarchy of reinforcement learning algorithms.

A complete view of Piqle algorithms hierarchy is shown in Figure 23.

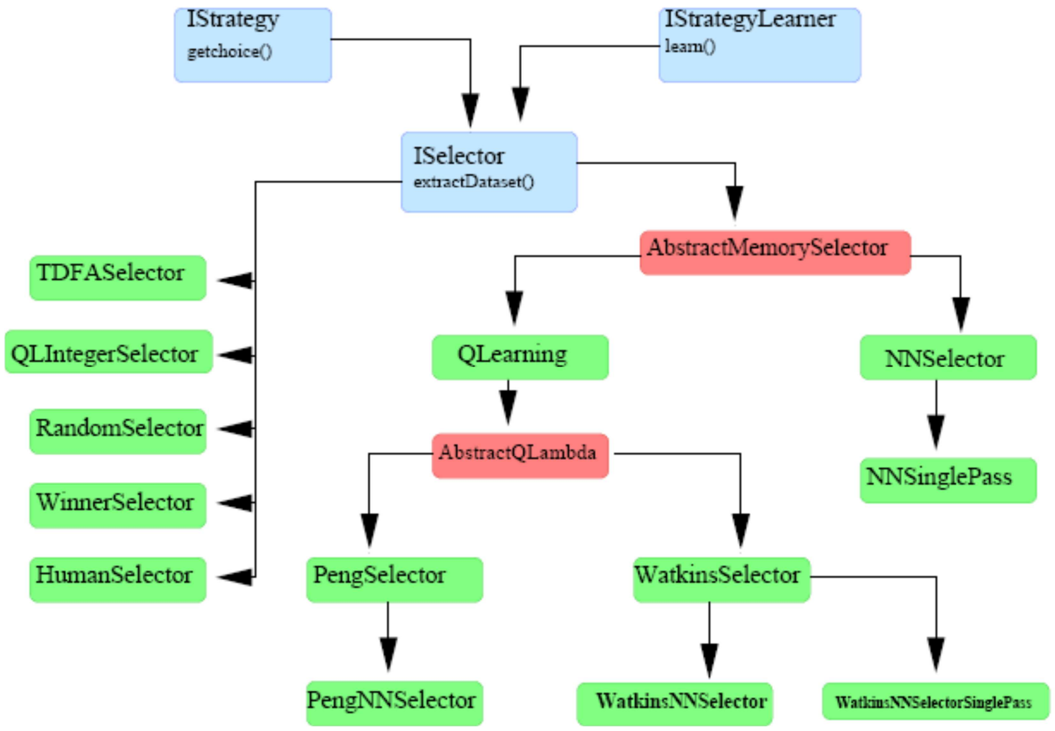


Figure 23 Java hierarchy of Algorithms

As far as Piqle is concerned, reinforcement learning can be described roughly as follows:

- The algorithm receives from its master (the agent) a state, and a list of possible actions.
- It estimates the value of each possible action, according to the value it has stored or computed.
- It returns the action that it believes will produce the greater reward (not only on the short, but also on the long run).
- After the action has been taken, and the state of the environment has been updated, the algorithm updates its Q(s, a) estimation.

This leads to the following preliminary remarks:

1. A class is needed which can store and retrieve values indexed with a pair (state, action).
2. It is needed to know when two states (and two actions) are equals.
3. It is not needed to estimate values for (state, action) pairs never encountered.

We are now ready to detail the reinforcement learning algorithms part of the class hierarchy, beginning with the abstract class that gathers the most important part of the structure and code.

AbstractMemorySelector

This abstract class defines:

- A Memory. the place where the Q(s, a) will be stored (whatever the meaning of "store" is).
- A method for learning, i.e. the standard method described in paragraph 4.7.
- A method for choosing the next action to perform, currently three different methods, depending on the strategy used to keep a little bit of randomness in the choice.
- Parameters for tuning the algorithm.

Each item is now described more precisely..

The memory

In this class, the exact kind of memory used is left abstract, and will be instantiated differently in each of its implementations.

This memory must implement the *IRewardStore* interface, from the qlearning package, which means it is asked to be able to:

- retrieve a value associated with a (state, action) pair: method *put*().
- store a value associated with a (space, action) value: method *get*().

The learning method

The algorithm used for learning is exactly the one describe in paragraph 4.7: it will be used without modification in both direct son classes, *QlearningSelector* and *NNSelector*.

The choice of next action

Roughly speaking, method *getchoice*() returns the best action, as far as Q(s, a) are concerned. But, in reinforcement learning, algorithms must preserve a balance between exploration (visiting new states) and exploitation (choosing the more rewarding action). This is done by introducing a little bit of randomness in the choice of the next action to perform. Several ways for implementing and controlling randomness have been proposed. Piqle defines three implementations:

- ***Greedy***: a random action is chosen with a certain probability, instead of the best action.
- ***Roulette***: turns each action that can be chosen with a probability proportional to its Q(s, a) associated value.
- ***Boltzmann***: selection is as in the roulette turn procedure, each action can be chosen with a probability related to its Q(s, a) associated value, but using a different function than the linear one.

The default behavior is the greedy one; hence the *getChoice*() method is just a branching to another choice function, depending of the chosen implementation.

Parameters

Some parameters are necessary to control and tune the behavior of reinforcement learning algorithms, some others to define precisely the randomness functions. Those values are given with their setters and getters. Some of them also need to change over time, and the class indicates how and how much they must change between two successive episodes.

- ***Alpha***: step-size parameter. It should theoretically decrease to ensure convergence. Two decrease method are proposed, exponential and geometrical. Only the first ensures convergence, but both give good practical results.

- ***Gamma***: discount rate parameter.

- ***Tau***: temperature for the Boltzmann selection implementation of randomness.
- ***Epsilon***: the probability of choosing an action at random in the-greedy case. Note that epsilon is supposed to decrease over time.

QLearningSelector

This class is a direct implementation of *AbstractMemorySelector* class, setting its memory to a Java implementation of HashMaps. The keys for this HashMap will be computed from the values of states and actions.

6.2.2. OMNeT++

For simulations performed in this thesis, the OMNeT++ simulator was used.

OMNeT++ is a discrete event simulator based on C++, highly modular, well structured and scalable. It provides a basic infrastructure wherein modules exchange messages. The name OMNeT++ stands for Objective Modular Network Testbed in C++. It has an open-source distribution policy and can be used free of charge by academic research institutions. It runs on Windows and Unix platforms, including Linux, and offers a command line interface as well as a powerful graphical user interface. The simulator can be used, for instance, to model communication and queuing networks, multiprocessors and other distributed hardware systems as well as to validate hardware architectures.

Modelling Concept

An OMNeT++ model consists of hierarchically nested modules, which communicate by passing messages each other. OMNeT++ models are often referred to as networks. The top level module is the system module. The system module contains sub-modules, which can also contain sub-modules themselves. The depth of module nesting is not limited; this allows the user to reflect the logical structure of the actual system in the model structure.

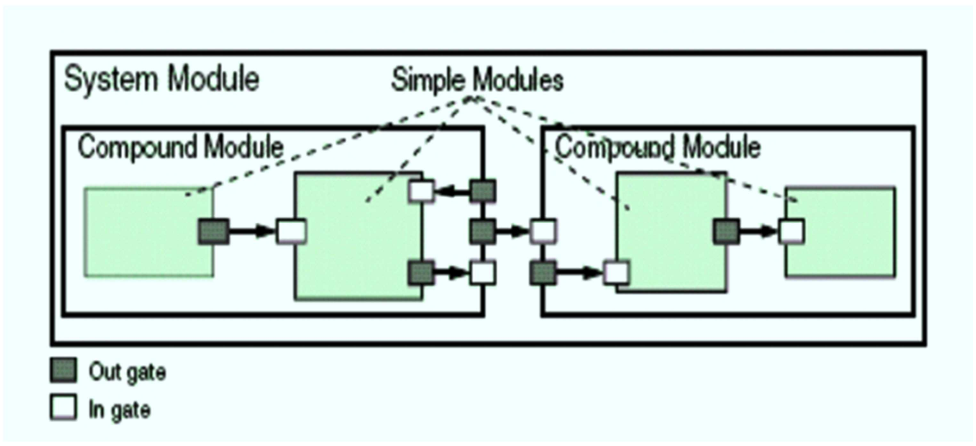


Figure 24 OMNeT++ Module Hierarchy

The model structure is described with OMNeT's NED language. Modules that contain sub-modules are termed compound modules, as opposed to simple modules which are at the lowest level of the module hierarchy. Simple modules contain the algorithms in the model. The user implements the simple modules in C++, using the OMNeT++ simulation class library.

Modules communicate by exchanging messages. In an actual simulation, messages can represent frames or packets in a computer network, jobs or customers in a queuing network or other types of mobile entities. The local simulation time of a module advances when the module receives a message. The message can arrive from another module or from the same module (self messages are usually used to implement timers).

Gates are the input and output interfaces of modules; OMNeT++ supports only simplex (one-directional) connections, so there are input and output gates. Messages are sent out through output gates and arrive through input gates.

Due to the hierarchical structure of the model, messages typically travel through a series of connections, to start and arrive in simple modules. Such series of connections that go from simple module to simple module are called routes. Compound modules act as "cardboard boxes" in the model, transparently relaying messages between their inside and the outside world.

Connections can be assigned three parameters, which facilitate the modelling of communication networks, but can be useful in other models too: propagation delay, bit error rate and data rate, all three being optional. One can specify link parameters individually for each connection, or define link types and use them throughout the whole model.

The simple modules of a model contain algorithms as C++ functions. The full flexibility and power of the programming language can be used, supported by the OMNeT++ simulation class library. The simulation programmer can choose between event-driven and process-style description, and can freely use object-oriented concepts (inheritance, polymorphism etc.) and design patterns to extend the functionality of the simulator.

Basic Parts of an OMNeT++ Model

An OMNeT++ model physically consists of the following parts:

- NED language topology description(s)
- Message definitions
- Simple modules implementations and other C++ code

To build an executable simulation program, user firstly need to translate the NED files and the message files into C++, using the NED compiler (nedtool) and the message compiler (oppmsgc). NED files can also be loaded dynamically, in which case they don’t need to be compiled beforehand. After this step, the process is the same as building any C/C++ program from source.

Interesting Features

The cycle length of a random number generator (RNG) is fundamental, especially when RNGs are used for simulation purposes. OMNeT++ releases prior to 3.0 used a linear congruent generator (LCG) with a cycle length of 231 − 2. This RNG is still available but is only suitable for small-scale simulation studies. Newer OMNeT++ releases use by default the Mersenne Twister RNG (MT) by Matsumoto and Nishimura. MT has a period of $2^{19937-1}$, and 623-dimensional equidistribution property is assured. MT is also very fast: as fast or faster than ANSI C's rand(). In addition, OMNeT++ allows to plug in own RNGs as well.

In many simulations, only the steady state performance (i.e. the performance after the system has reached a stable state) is of interest. The initial part of the simulation is called the transient period. After the model has entered steady state, simulation must proceed until enough statistical data has been collected to compute results with the required accuracy.

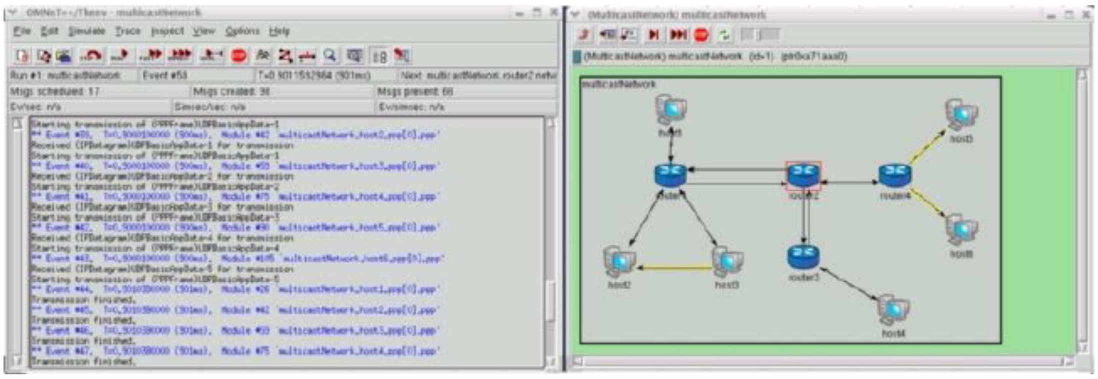
Detection of the end of the transient period and a certain result accuracy is supported by OMNeT++. The transient detection and result accuracy objects will do the specific algorithms on the data fed into the result object and tell if the transient period is over or the result accuracy has been reached. The transient detection algorithm uses a sliding window approach with two windows, and checks the difference of the averages of the two windows to see if the transient period is over. The accuracy detection algorithm divides the standard deviation by the square of the number of samples and checks if this is within the accuracy range specified by the user.

Comparison with other simulators

Available Models Non-commercial simulation tools cannot compete with some commercial ones (especially OPNET) which have a large selection of ready-made protocol models. OMNeT++ is no exception, it clearly lacks models, also compared with non-commercial tools such as Ns-2 (but it has to be considered that OMNeT++ is a rather new tool, it was originally released in 1999). On the other hand OMNeT++ provides a larger variety of models (that allows the user to simulate more than just communication networks) as compared to NS, which mainly provides TCP/IP centred models.

The modularity of OMNeT++ is a plus. For example NS-2 tends to be monolithic: to add new models to it, one needs to download the full source and modify it a bit, copy files to specific locations, add constants in other files etc.

Performance is a particularly interesting issue with OMNeT++ since the GUI debugging/tracing support involves some extra overhead in the simulation library. Simulating large networks (e.g. MQTT networks with hundreds of clients) results in unacceptable performance. But this is also a big problem with other popular simulators such as NS-2.


OMNeT++/Tkenv - multicastNetwork
File Edit Simulate Trace Inspect View Options Help
(MulticastNetwork) multicastNetwork
multicastNetwork
host1
host2
host3
host4
host5
host6
router1
router2
router3
router4

```
Run #1: multicastNetwork   Event #53   T=0.5011592984 (501ms)   Next: multicastNetwork.router2.netw
Msgs scheduled: 17         Msgs created: 98         Msgs present: 66
Ev/sec: n/a                Simsec/sec: n/a          Ev/simsec: n/a
Starting transmission of (PPPFrame)UDPBasicAppData-1
** Event #39,  T=0.5000130000 (500ms),  Module #42 `multicastNetwork.host2.ppp[0].ppp'
Received (IPDatagram)UDPBasicAppData-1 for transmission
Starting transmission of (PPPFrame)UDPBasicAppData-1
** Event #40,  T=0.5000130000 (500ms),  Module #55 `multicastNetwork.host3.ppp[0].ppp'
Received (IPDatagram)UDPBasicAppData-2 for transmission
Starting transmission of (PPPFrame)UDPBasicAppData-2
** Event #41,  T=0.5000130000 (500ms),  Module #75 `multicastNetwork.host4.ppp[0].ppp'
Received (IPDatagram)UDPBasicAppData-3 for transmission
Starting transmission of (PPPFrame)UDPBasicAppData-3
** Event #42,  T=0.5000130000 (500ms),  Module #90 `multicastNetwork.host5.ppp[0].ppp'
Received (IPDatagram)UDPBasicAppData-4 for transmission
Starting transmission of (PPPFrame)UDPBasicAppData-4
** Event #43,  T=0.5000130000 (500ms),  Module #105 `multicastNetwork.host6.ppp[0].ppp'
Received (IPDatagram)UDPBasicAppData-5 for transmission
Starting transmission of (PPPFrame)UDPBasicAppData-5
** Event #44,  T=0.5010380000 (501ms),  Module #28 `multicastNetwork.host1.ppp[0].ppp'
Transmission finished.
** Event #45,  T=0.5010380000 (501ms),  Module #42 `multicastNetwork.host2.ppp[0].ppp'
Transmission finished.
** Event #46,  T=0.5010380000 (501ms),  Module #55 `multicastNetwork.host3.ppp[0].ppp'
Transmission finished.
** Event #47,  T=0.5010380000 (501ms),  Module #75 `multicastNetwork.host4.ppp[0].ppp'
Transmission finished.
```

Figure 25 OMNeT++ screenshot

JSimpleModule

In order to permit OMNeT++ to interface with Java classes, (because, as already said, it was used a Java library of classes for implementing reinforcement learning) it was used a module for communication between Java and OMNeT++ (this module is obviously written in C++ Language). This module is referred to as *JSimpleModule*.

JSimpleModule is an extension that makes it possible to write OMNeT++ simple modules in Java. Java and C++-based simple modules can be freely mixed in simulations. Integration is not seamless though, there are limitations as to what OMNeT++ features are available, as well special coding rules to obey in the Java code.

Operation is based on the *JSimpleModule* class. This is an ordinary C++ simple module, which receives the name of a Java class in a module parameter. During initialization, it loads the Java Virtual Machine (if not already loaded), instantiates the given Java class, then simply delegates *initialize*(), *handleMessage*() and finish() calls to it. The Java class should be a subclass of *org.omnetpp.simkernel.JSimpleModule*, which provides the usual methods: *send*(), *scheduleAt*(), *cancelEvent*(), etc. Messages are instances of *org.omnetpp.simkernel.cMessage*, and the *org.omnetpp.simkerne* package contains Java versions of most simulation kernel classes. The *org.omnetpp.simkernel.** Java classes are just JNI-based wrappers around the corresponding C++ classes: every Java object holds a pointer to a corresponding C++ object, and delegates all method calls to it. This has a few consequences about the programming model. The wrapper classes have been generated using SWIG [34].

Chapter 7. Results

7.1. *Introduction*

This chapter describes simulation scenarios and illustrates how RL is applied in a real situation, showing system’s behavior during simulation runs. These results are obtained using simulator tool open source OMNeT++, described in the previous chapter. To valuate performance of the proposed Reinforcement Learning Connection Admission Control solutions, these were tested in different scenarios, with advanced multimedia services and various traffic intensity. Results are presented in the following order:

- Comparison between first RL approach (taking care only of bandwidth limit control) and peak based algorithm;
- Comparison between first RL approach (taking care of bandwidth limit and blocking probability control) and peak based algorithm;
- Comparison between first RL approach (taking care of bandwidth limit and blocking and drop probability control) and peak based algorithm;
- Comparison between second RL approach and peak based algorithm;
- Comparison between two RL approaches.

7.2. *Comparison between first RL approach (with bandwidth limit control) and peak based algorithm*

7.2.1. Introduction

To simplify simulation process it has been developed an appropriate package able to load scenario parameters (number of classes in the system and their statistical traffic information) and Reinforcement Learning Controller configuration parameters (like α, γ, ε, etc...).

In particular a GUI (Graphical User Interface) was created in order to ease to run and define simulations. When GUI is started, a new window (see Figure 27) is loaded. This window is formed of:

- A ListBox where there are all simulations (old and new) are listed: clicking on a simulation its parameters are loaded (Figure 28). Simulation information are stored on (and loaded from) a XML file (an example is depicted in Figure 26): to parse XML file, it is used the open source library “jdom4'“. In the first TabPanel, namely “Tree”, contents of file are shown, so it is possible to change on-the-fly simulation parameters, without having to manually edit XML file or to modifying the code.
- In the second TabPanel “Output”' (Figure 29) OMNeT++ simulator output is shown, comprehending simulation time and log or warning messages.
- In the third TabPanel (Figure 30), user can run a sequence of simulation (batch-simulation), also tuning some parameters; this is very useful to analyze performance of different scenarios.
- In the last TabPanel (Figure 31) it is possible to configure some useful parameters for OMNeT++ simulator management, such as running speed, to activate log of

simulation, decide whether to use OMNeT's GUI interface or CmdEnv environment (the alternative without any GUI) and auto-loading of agents' state.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<scenario>
  <num_CoS>3</num_CoS>
  <capacità>2500</capacità>
  <controllore>
    <alpha>0.5</alpha>
    <alphaDecay>0.999</alphaDecay>
    <gamma>0.1</gamma>
    <epsilon>0.0</epsilon>
    <epsilonDecay>0.998</epsilonDecay>
    <Qlambda>0.5</Qlambda>
    <cost>0.38</cost>
    <dropActive>false</dropActive>
    <FB>
      <limit>false</limit>
      <window>6000000</window>
      <alpha>1</alpha>
    </FB>
  </controllore>
  <classi>
    <classe id="CoS_1">
      <banda>50</banda>
      <lambda>0.099</lambda>
      <MHT>120</MHT>
      <reward>1</reward>
      <sogliaPB>0.1</sogliaPB>
      <sogliaPD>0.002</sogliaPD>
    </classe>
    <classe id="CoS_2">
      <banda>80</banda>
      <lambda>0.089</lambda>
      <MHT>100</MHT>
      <reward>5</reward>
      <sogliaPB>0.05</sogliaPB>
      <sogliaPD>0.001</sogliaPD>
    </classe>
    <classe id="CoS_3">
      <banda>110</banda>
      <lambda>0.04</lambda>
      <MHT>200</MHT>
      <reward>7</reward>
      <sogliaPB>0.15</sogliaPB>
      <sogliaPD>0.01</sogliaPD>
    </classe>
  </classi>
</scenario>
```

Figure 26 XML scenario example

This approach was followed in order to totally hidden all business logic.

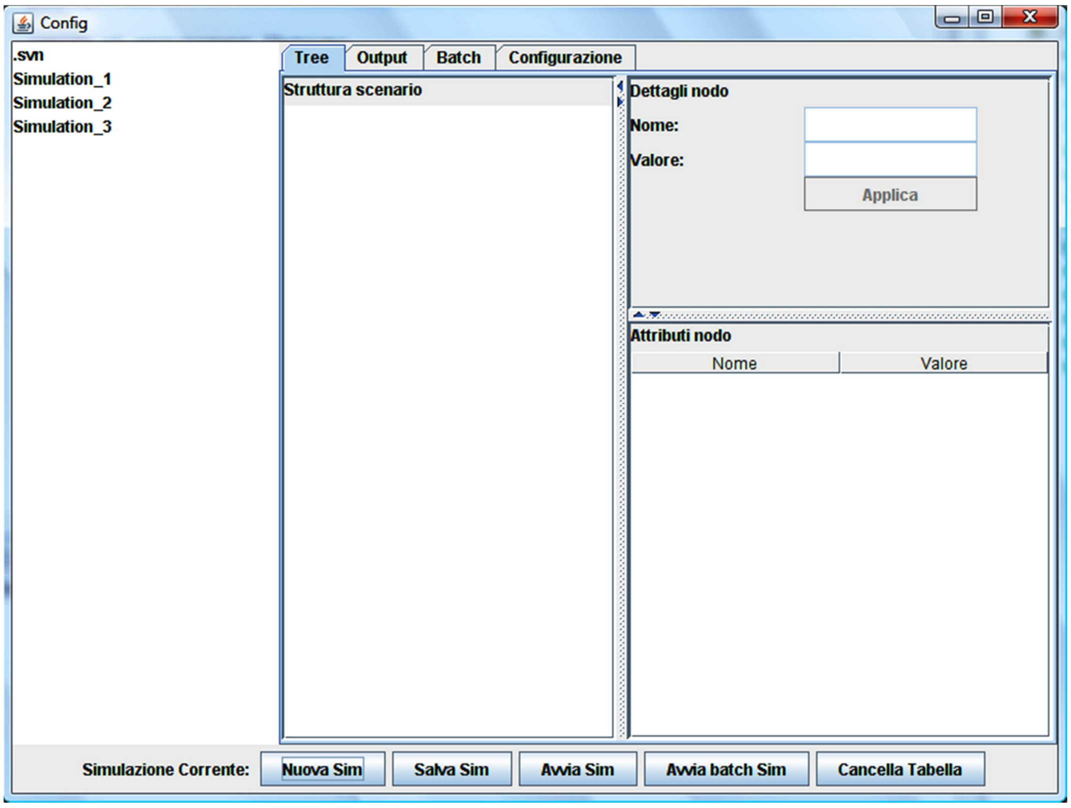


Figure 27 Graphical User Interface

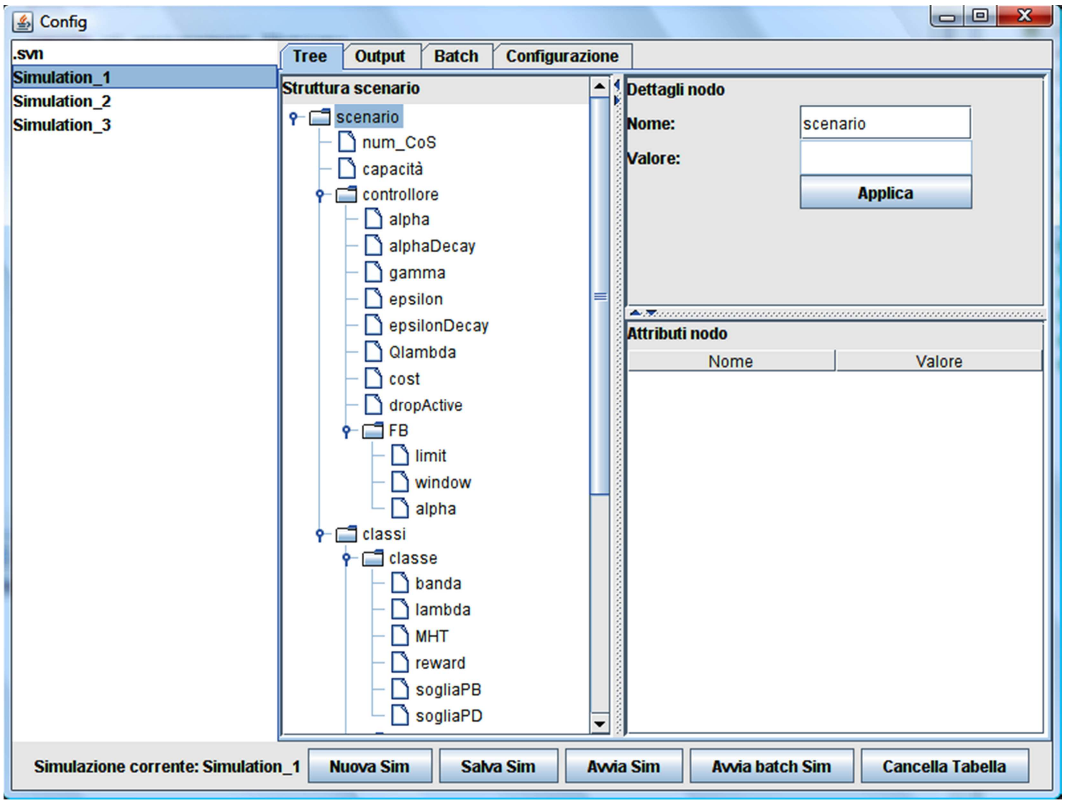
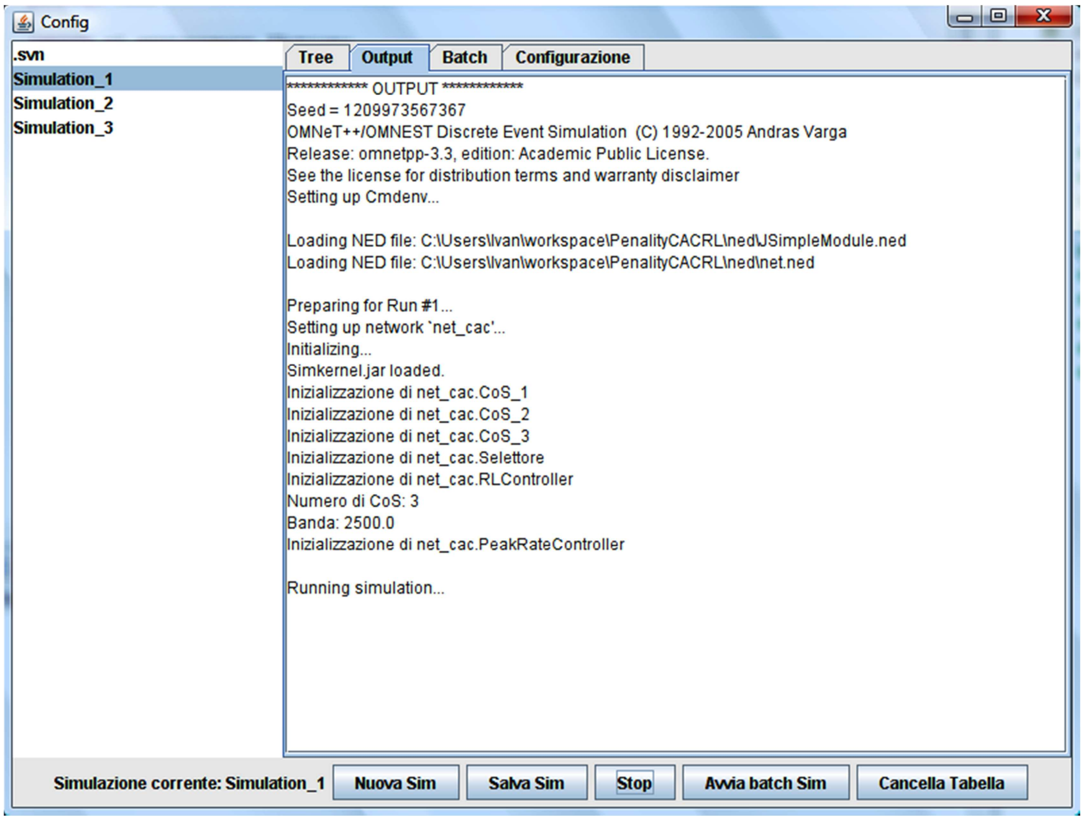
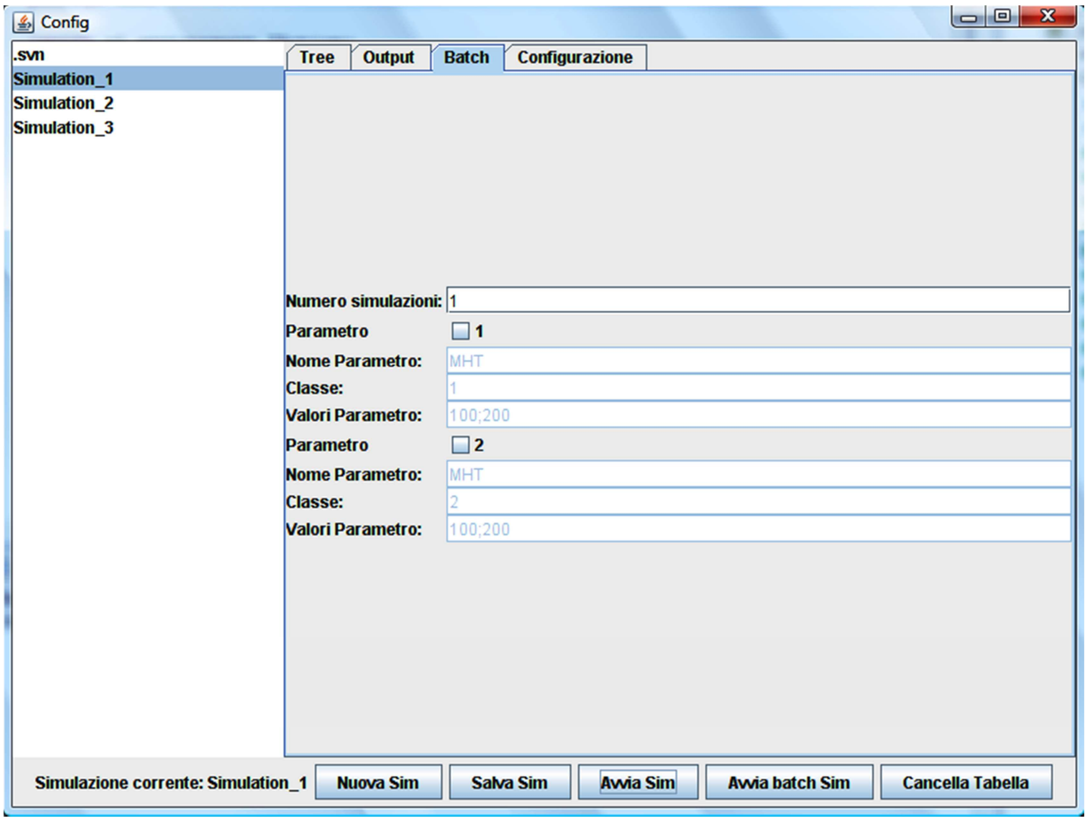


Figure 28 Simulation parameters



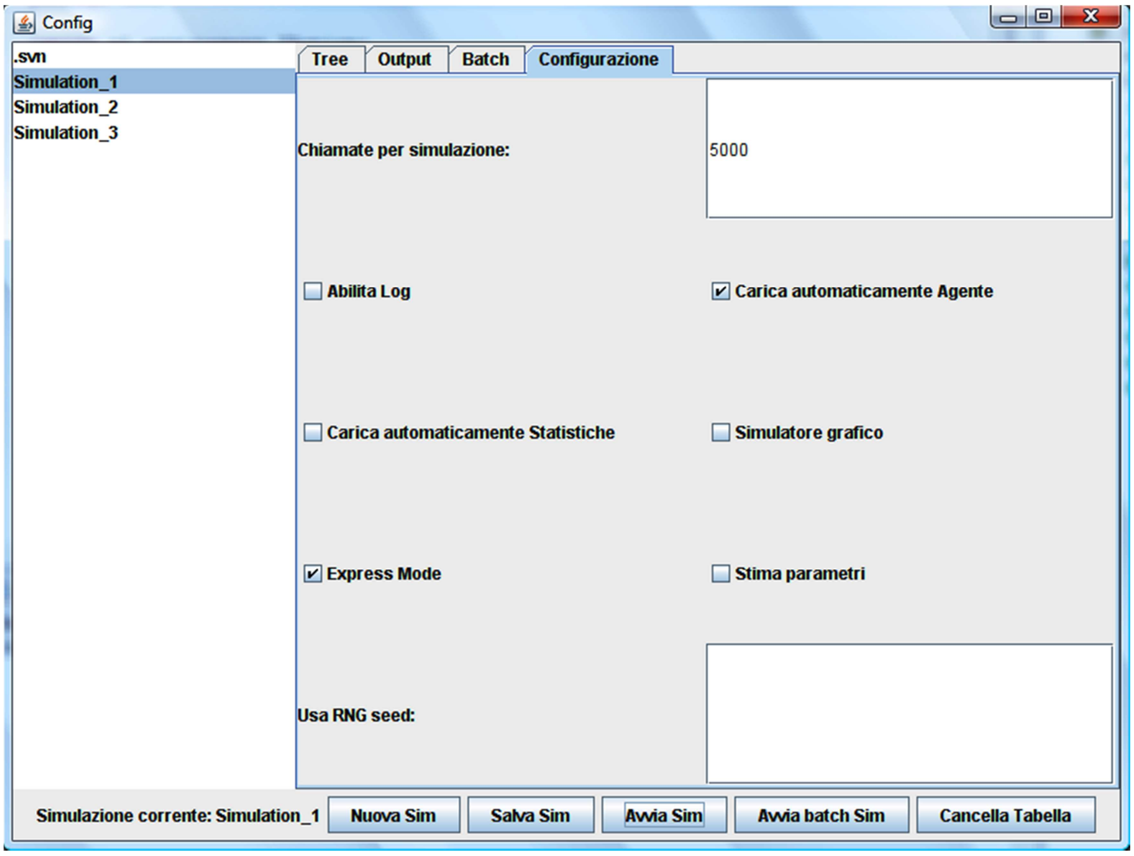
```
************ OUTPUT ************
Seed = 1209973567367
OMNeT++/OMNEST Discrete Event Simulation  (C) 1992-2005 Andras Varga
Release: omnetpp-3.3, edition: Academic Public License.
See the license for distribution terms and warranty disclaimer
Setting up Cmdenv...

Loading NED file: C:\Users\Ivan\workspace\PenalityCACRL\ned\JSimpleModule.ned
Loading NED file: C:\Users\Ivan\workspace\PenalityCACRL\ned\net.ned

Preparing for Run #1...
Setting up network `net_cac'...
Initializing...
Simkernel.jar loaded.
Inizializzazione di net_cac.CoS_1
Inizializzazione di net_cac.CoS_2
Inizializzazione di net_cac.CoS_3
Inizializzazione di net_cac.Selettore
Inizializzazione di net_cac.RLController
Numero di CoS: 3
Banda: 2500.0
Inizializzazione di net_cac.PeakRateController

Running simulation...
```

Figure 29 OMNeT++ simulator output



Numero simulazioni:	1
Parametro	☐ 1
Nome Parametro:	MHT
Classe:	1
Valori Parametro:	100;200
Parametro	☐ 2
Nome Parametro:	MHT
Classe:	2
Valori Parametro:	100;200

Figure 30 Batch simulation



Chiamate per simulazione: 5000

- [ ] Abilita Log
- [x] Carica automaticamente Agente
- [ ] Carica automaticamente Statistiche
- [ ] Simulatore grafico
- [x] Express Mode
- [ ] Stima parametri

Usa RNG seed:

Simulazione corrente: Simulation_1

Figure 31 Management page for OMNeT++ simulator

As stated before, OMNeT++ provides an intuitive graphical environmental simulator (called GNED); it contains a textual representation of the model topology and a graphical interface (Tkenv) for running simulations which allows to see and edit the modules of the model.

In Figure 31**Errore. L'origine riferimento non è stata trovata.**, selecting the field "Simulatore Grafico" it is possible to visualize GNED as depicted in Figure 32 and Figure 33.

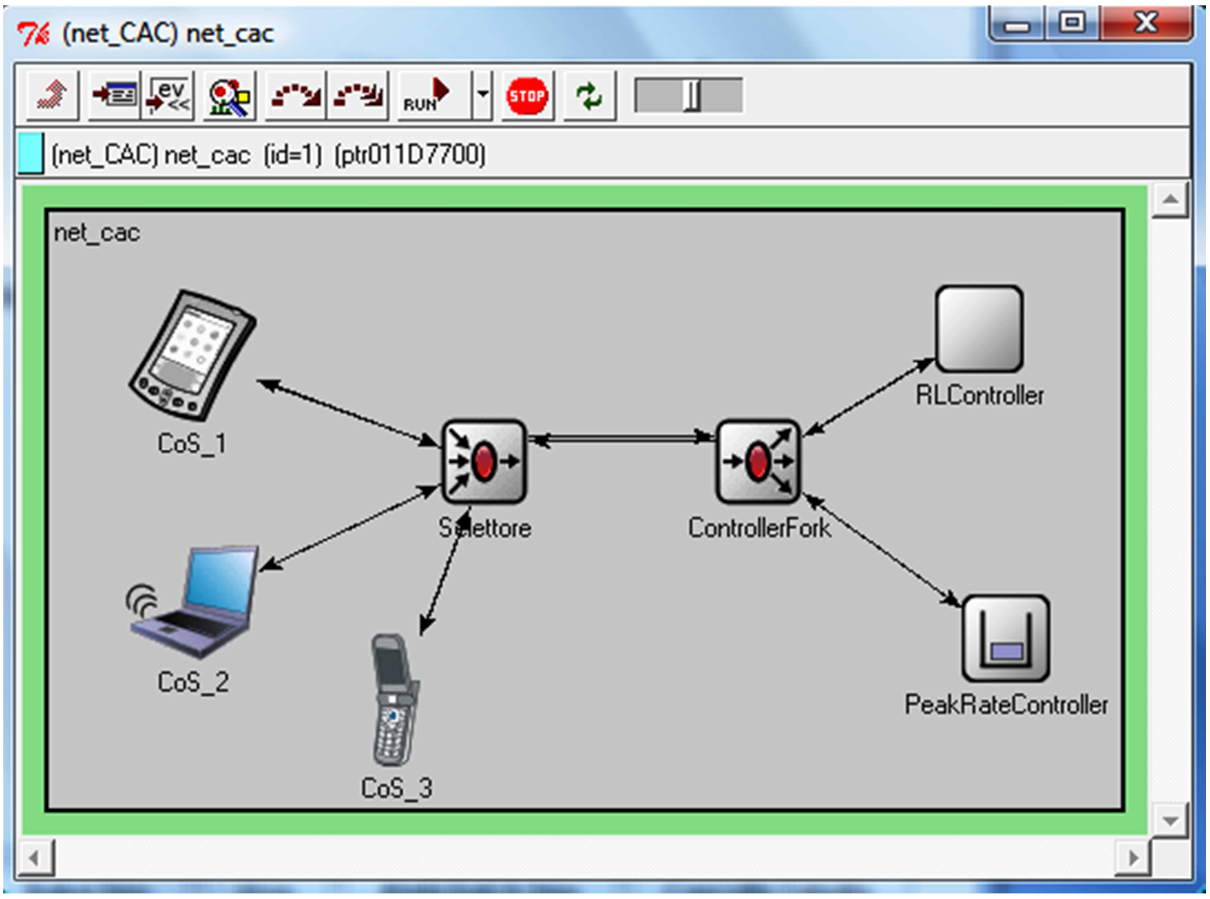
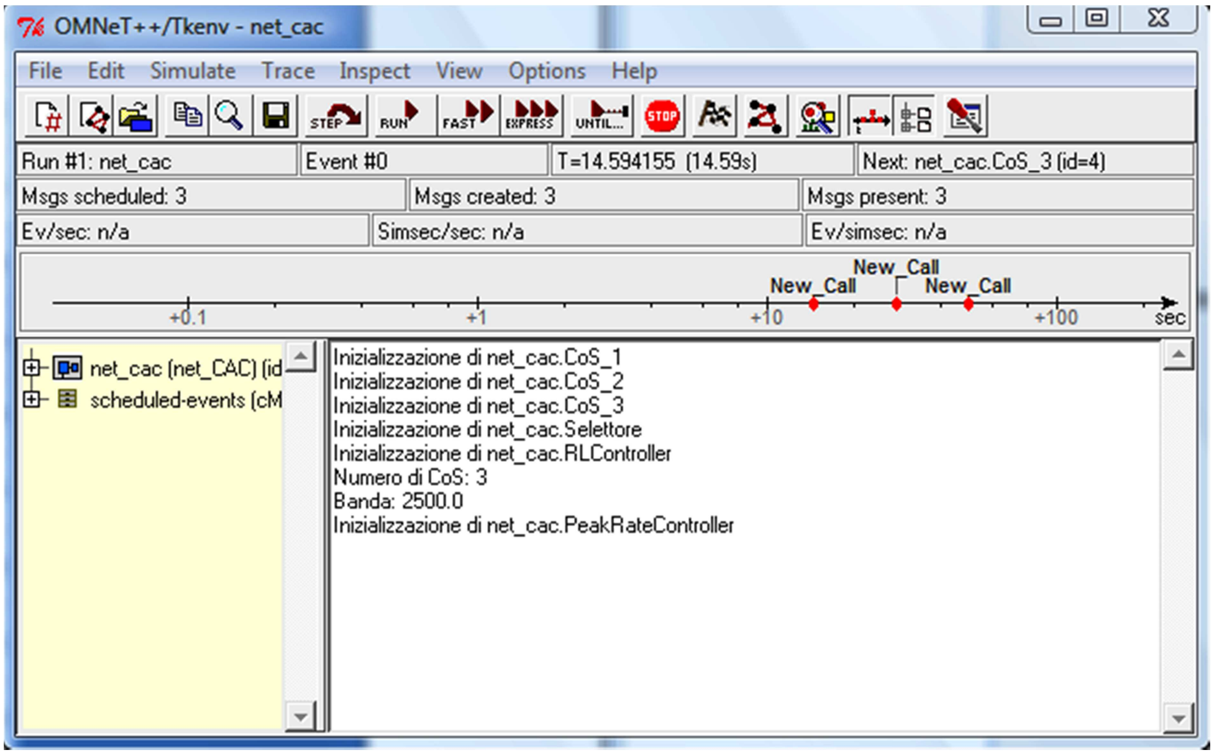


Figure 32 GNED window



```
Run #1: net_cac | Event #0 | T=14.594155 (14.59s) | Next: net_cac.CoS_3 (id=4)
Msgs scheduled: 3 | Msgs created: 3 | Msgs present: 3
Ev/sec: n/a | Simsec/sec: n/a | Ev/simsec: n/a
```

```
Inizializzazione di net_cac.CoS_1
Inizializzazione di net_cac.CoS_2
Inizializzazione di net_cac.CoS_3
Inizializzazione di net_cac.Selettore
Inizializzazione di net_cac.RLController
Numero di CoS: 3
Banda: 2500.0
Inizializzazione di net_cac.PeakRateController
```

Figure 33 Tkenv window

7.2.2. Scenario: VOIP, video call and FTP data services

The adopted scenario consists of a link, supporting three different classes, selected to characterize three typical services that can be found in a real network having limited resources:

- VOIP (CoS_1)
- Video Call (CoS_2)
- FTP-Data (CoS_3)

Statistics on the traffic of class *k* generated by the users are usually given in terms of BHCA (Busy Hour Call Attempts), equals to the mean number of call attempts done by each user during the network traffic busiest hour, and MHT (Mean Holding Time), which is the average duration in minutes of a call. Arrival and termination rates (expressed in [min^{-1}]) are then easily computed as follows:

$$\begin{cases} \lambda_k = \dfrac{BHCA_k}{60} u_k(t) \\ \mu_k = \dfrac{1}{MHT_k} \end{cases} \tag{28}$$

where $u_k(t)$ is the number of users of class *k* present in the cell at time *t*. Moreover each class occupies a certain fraction of the available bandwidth, depending on the kind of service it belongs to (in terms of Bit Rate). Another parameter that has to be considered in the simulation scenario is the total available capacity of link, which represents the resource to be shared among the traffic classes' requirements, choosing the best policy. In the first simulation, it is assumed that the bandwidth capacity *C* = 2,5Mbit/s.

As initial state of link is chosen a state characterized by the absence of any active connection: $\overline{s_0} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ where each number in the vector $\overline{s_0}$ represent the number of ongoing calls for each classes; in other words a generic state $\bar{s} = [n_1 \quad n_2 \quad n_3]$ has n_1 active connections of class 1, n_2 active connections of class 2, and n_3 active connections of class 3.

Q(λ) parameters

It is useful to remember that as behavior policy the epsilon greedy one is used (greedy action selection with probability 1 – ε and random action selection with probability ε). The chosen value for epsilon is equal to 0.5 as starting point, and then it decreases rapidly, so that agent(s) can explore all possible actions early, and then , while learned, in order to better evaluate final performance of RL algorithm, the final value is set to ε = 0.1 The Q-table is initialized with zero values. The step-size α is not constant, but it decays following a geometrical law. Its starting value is α=0.5. The discount factor is fixed to γ=0.1.

Medium, Medium-High and High Load

In order to evaluate algorithm's results, three different traffic condition have been considered: medium load, medium-high load and high load.

CoS	BitRate [Kbit/s]	λ $[s^{-1}]$	MHT[s]	reward [€/s]
1	50	0,096	120	1
2	80	0,087	100	5
3	110	0,042	200	7

Table 8 Traffic parameters of medium load scenario

CoS	BitRate [Kbit/s]	λ $[s^{-1}]$	MHT[s]	reward [€/s]
1	50	0,69	120	1
2	80	0,46	100	5
3	110	0,11	200	7

Table 9 Traffic parameters of medium-high load scenario

CoS	BitRate [Kbit/s]	$\lambda\ [s^{-1}]$	MHT[s]	reward [€/s]
1	50	1,23	120	1
2	80	1,09	100	5
3	110	0,61	200	7

Table 10 Traffic parameters of high load scenario

In the picture below, the two algorithms are compared: the total incoming obtained using the peak-based algorithm is set to 1, and, with respect to it, it is calculated the relative gain of the RL algorithm:

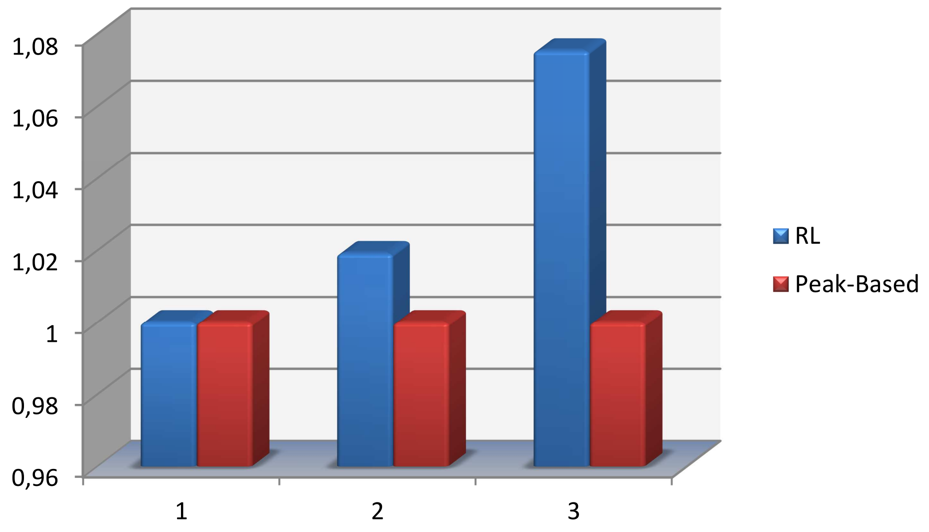


Figure 34 Perceptual Gain in three different network loads. 1 referred to Medium Load, 2 referred to Medium-High Load and 3 referred to High Load

Note that gains of peak-based algorithm are not always the same in all three conditions: the figure only remarks the relative convenience of using the RL algorithm instead of the peak-based one.

In the picture it is possible to see that, for medium load (as also for low load) the proposed algorithm has about the same behavior of the peak-based algorithm. For medium-high load the proposed algorithm's incoming is about 2% higher with respect to peak-based one and, for high load, the proposed algorithm's incoming is about 8% higher with respect to peak-based one. These results could be easily interpreted: for low and medium loads there is about no reason to leave space for calls of better classes of services: the best solution is to always accept new calls; when the load is higher, this is no more true, so that RL performs better than peak-based algorithms.

These results are obtained without take into account any control of the blocking or of the drop frequencies; so that in the next paragraphs these controls are introduced.

7.3. *Comparison between first RL approach (with bandwidth limit and blocking probability control) and peak based algorithm*

This paragraph depicts results of proposed RL approach while taking also into account blocking probability control, in particular in the medium-high load scenario; note that both the RL algorithm and the peak-based one guarantee the link does not arrive at the saturation.

Figure 35 illustrates the blocking frequency trend of this scenario without introducing control of it.

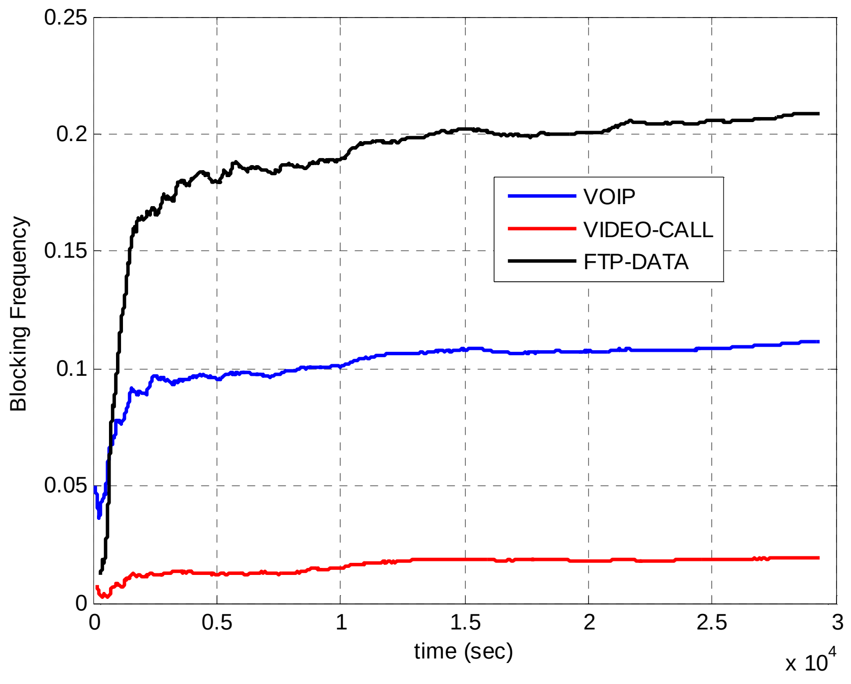


Figure 35 Blocking frequency in medium-high load scenario without control

This trend, i.e. blocking probabilities increasing indefinitely, is not acceptable in most of real cases; on the other side, it is not possible to block indefinitely the incoming connections in a network. Usually an upper limit to blocking probabilities is given; setting this limit to 5% for video-calls, to 10% for VOIP and to 15% for FTP-Data, the obtained trend is depicted in Figure 36.

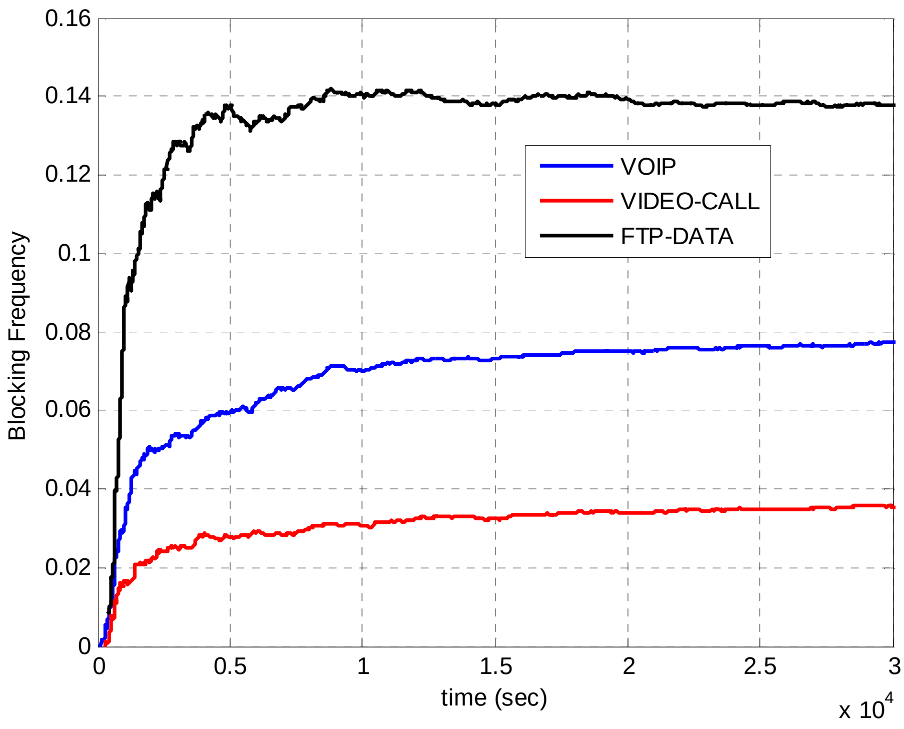


Figure 36 Blocking frequency in medium-high load scenario with limitation

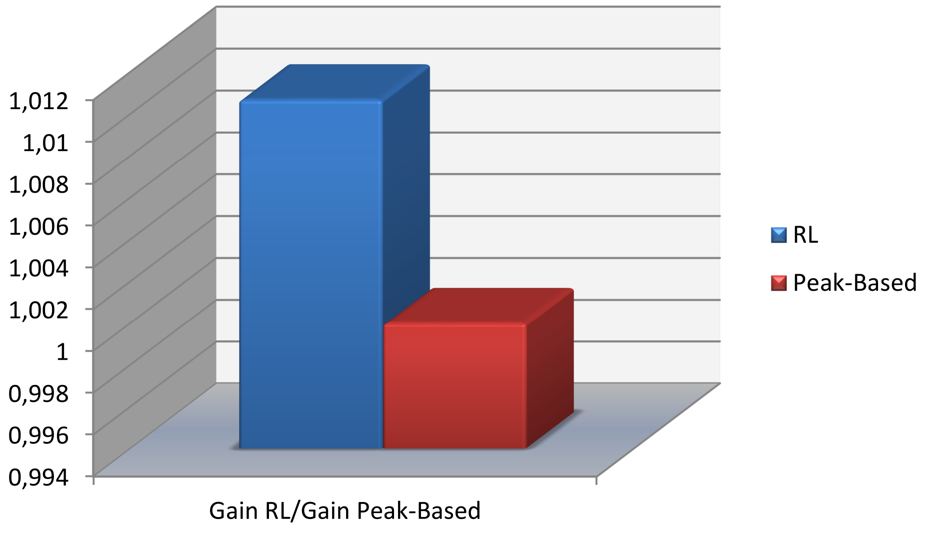


Figure 37 Gain in medium-high load scenario with limitation

In **Errore. L'origine riferimento non è stata trovata.**Figure 36 it is possible to see that the blocking frequencies are hold under thresholds. Because of introduction of these thresholds, the relative gain of RL on peak-based algorithms is decreased, but it is always present, and it is about 1%.

In the high load scenario, where the network arrives to the saturation, the blocking frequency can't be hold under threshold indefinitely, unless we introduce the possibility to drop calls, as illustrated in the following paragraph.

7.4. *Comparison between first RL approach (with bandwidth limit and blocking and drop probability control) and peak based algorithm*

Setting lambdas as in the high traffic scenario, even introducing the possibility to drop ongoing calls, having the above mentioned bandwidth capacity, it is not possible to control blocking probabilities indefinitely. This is due to the fact one, in a real scenario, could not drop ongoing calls without any limit, but should guarantee to respect some higher boundaries: in other words, if the traffic is too high for the link capability, it is not possible to control blocking probabilities at all and also the introduction of the possibility to drop calls (considering a real case where it is not possible to drop calls without any limit) could be not a solution. So that a not extreme scenario was considered, having parameters as depicted in the table below (we will refer to it as semi-high load scenario):

CoS	BitRate [Kbit/s]	λ $[s^{-1}]$	MHT[s]	reward [€/s]
1	50	1,03	120	1
2	80	0,89	100	5
3	110	0,51	200	7

Table 11 Traffic parameters of semi-high load scenario

In this scenario the considered thresholds for the dropping probability are 1% for VOIP, 2% for Video-Call, and 5% for FTP-Data: in the following picture trends of blocking probabilities are depicted, without (Figure 38) and with (Figure 39) possibility to drop calls:

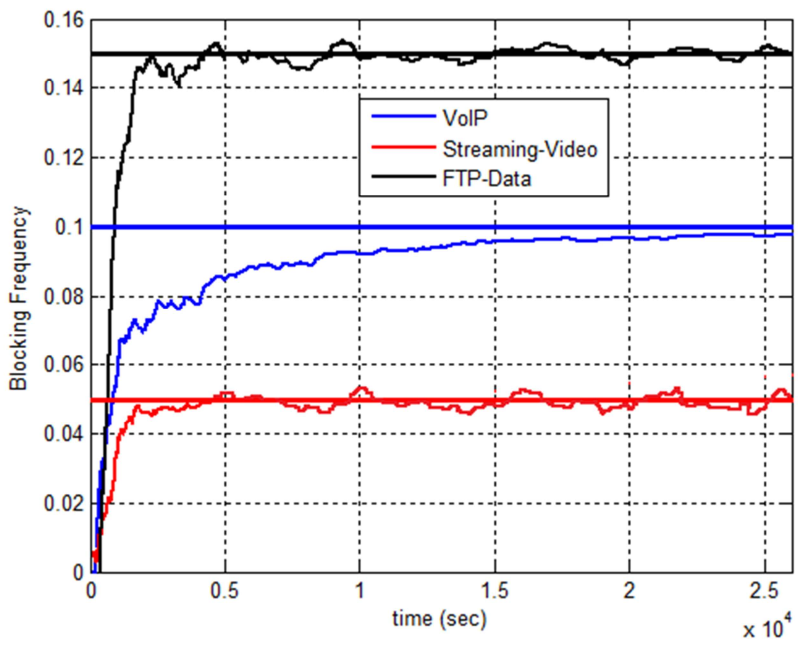


Figure 38 Blocking frequency in semi-high load scenario

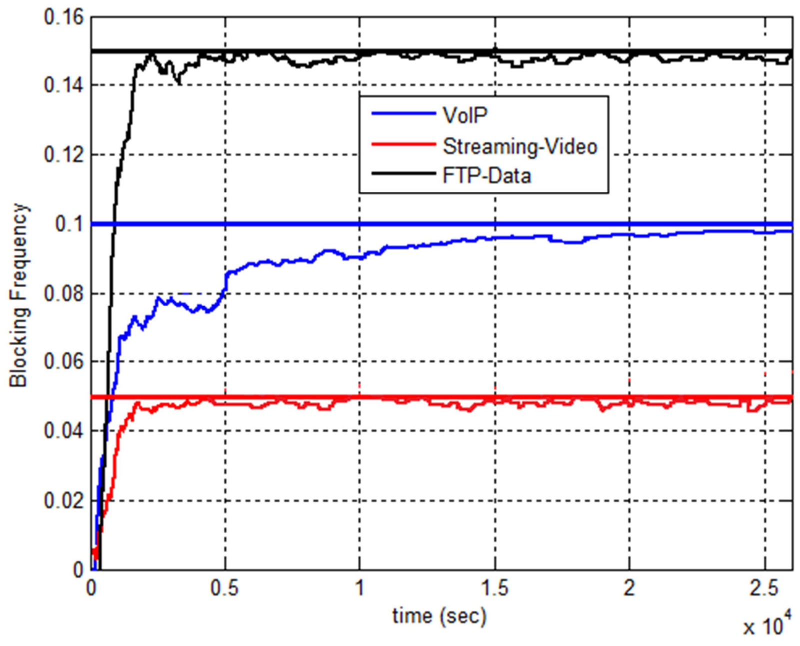


Figure 39 Blocking frequency in semi-high load scenario with drop

Following picture indicates the comparison of RL gain with respect to peak-based one, using either blocking and dropping frequencies control.

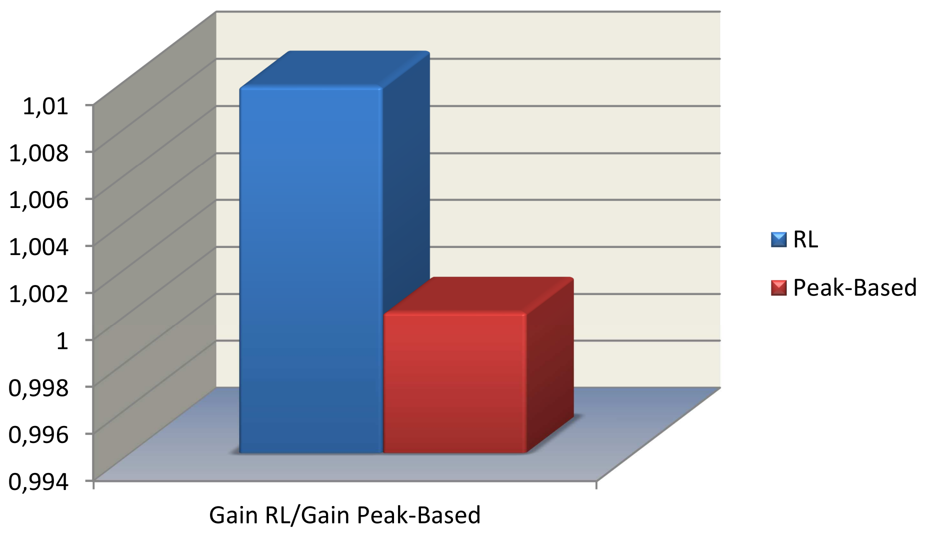


Figure 40 Gain in semi-high load scenario with drop

As it is possible to see, while, thanks to the introduction of dropping possibility one could expect RL to gain more than RL with respect to the previous scenario, the introduction of a limit on it act counter-productively, so that, at the end, in this case RL gain is about 1% higher than the one of peak-based, as in the previous scenarios. In any case this is a good result: the presented RL algorithm is able to guarantee QoS (in terms of bandwidth, blocking and drop probability control) while resulting in higher incomes than peak-based ones.

The following picture indicates dropping probabilities in the above depicted scenario. Is it possible to note that all probabilities are below their threshold:

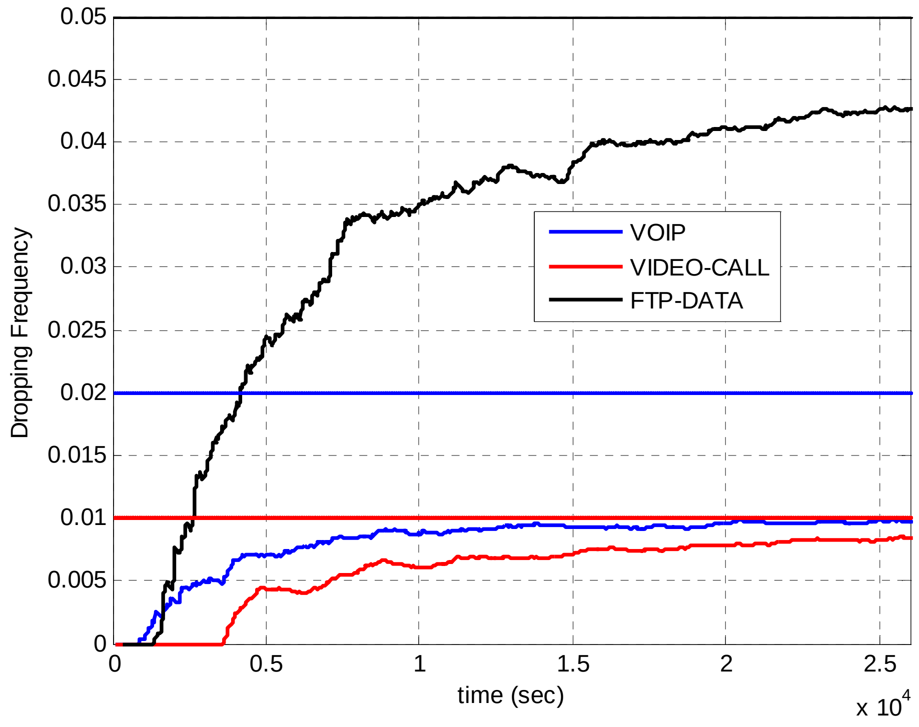


Figure 41 Dropping frequency in semi-load scenario

A final test could be done: to introduce in the algorithm a penalty function. One could suppose that, when the peak-based or the RL algorithms doe not respect the imposed limits on the blocking or drop frequencies, the network operator has to pay a “penalty” (in term of money). Here the adopted approach is to suppose that, when it arrives a connection request of a certain class of service and for it is not possible to respect the limits on the blocking and/or drop frequencies, the network operator has to pay a “fee” equal to the value of the economic gain that the connection averagely gives him (price/sec * MHT).

In the next figures the resulting values are depicted, either in medium-high load and semi-high load scenarios.

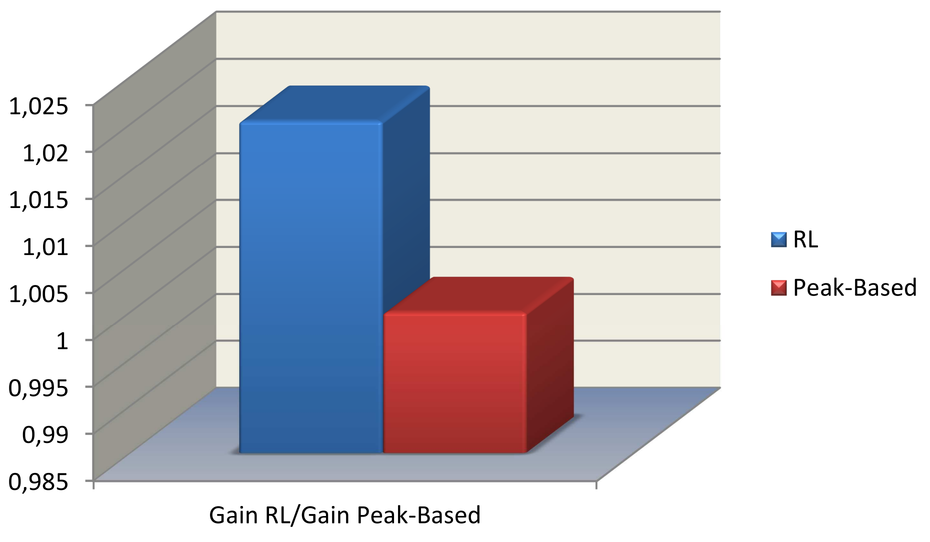


Figure 42Gain in medium-high load scenario with penalty

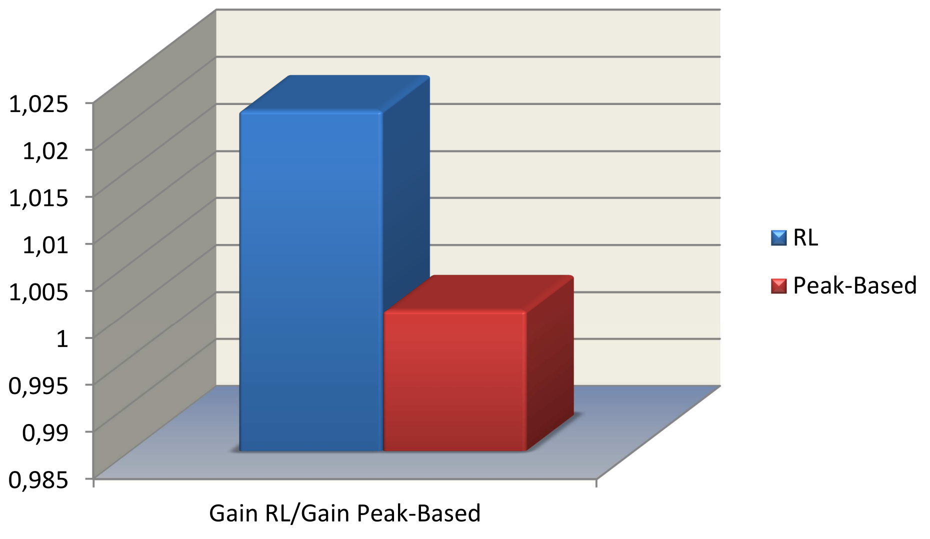


Figure 43 Gain in medium-high load scenario with penalty and drop probability control

In both scenarios the gain of RL is increased (with respect to previous cases): this is due to the fact that the peak-based algorithm accepts calls of the FTP-Data class of service, while RL algorithm rejects them, so that peak-based remains more often at saturation level, so that increasing the blocking and dropping probabilities over thresholds and so forcing the network operator to pay the penalties much more often than RL algorithm.

7.5. *Comparison between second RL approach and peak based algorithm*

This paragraph introduces simulation results of the second RL approach, compared with the peak-based algorithm. For these simulations, the same tools and network topology have adopted of the ones described above, but with different parameters, as indicated in the following scenarios.

7.5.1. Scenario 1

CoS	BitRate [Kbit/s]	$\lambda\ [s^{-1}]$	MHT[s]	reward [€/s]
1	50	0,099	120	1
2	80	0,089	100	5
3	110	0,04	200	7

Table 12 Traffic parameters of medium load scenario

This scenario is characterized by two classes of service that are similar in terms of arrival rate, but the second is more convenient of the first one, so, the agent, in order to have an income greater than the one guaranteed by peak-based algorithm, has to learn that, sometimes, it has to block calls belonging to CoS_1, in order to have free bandwidth for the more convenient calls belonging to CoS_2. The third CoS is the most convenient but its arrival rate is very small, so, the agent has to try to balance preserving free bandwidth for an arriving call belonging to this class of service (thus earning more than Peak-Based: in fact in most of cases peak-based will not have available bandwidth due to the fact that it has accepted all other calls of other CoS that were arriving with a higher frequency) and accepting calls of more frequent CoS. In order to analyze network operator's revenue of these two strategies, two kinds of tests were performed. The first test was very detailed: it has involved 10 simulations (each having 5000 calls), and, for every simulation, it has been compared CAC-RL's revenue with Peak-Based one (the comparison has been made between the same sequence of calls, obtained using the same seed). This kind of test demonstrated that, in every simulation, CAC-RL earned more than Peak-Based (a range of about 2000-100000 €).

Results are depicted in the following picture:

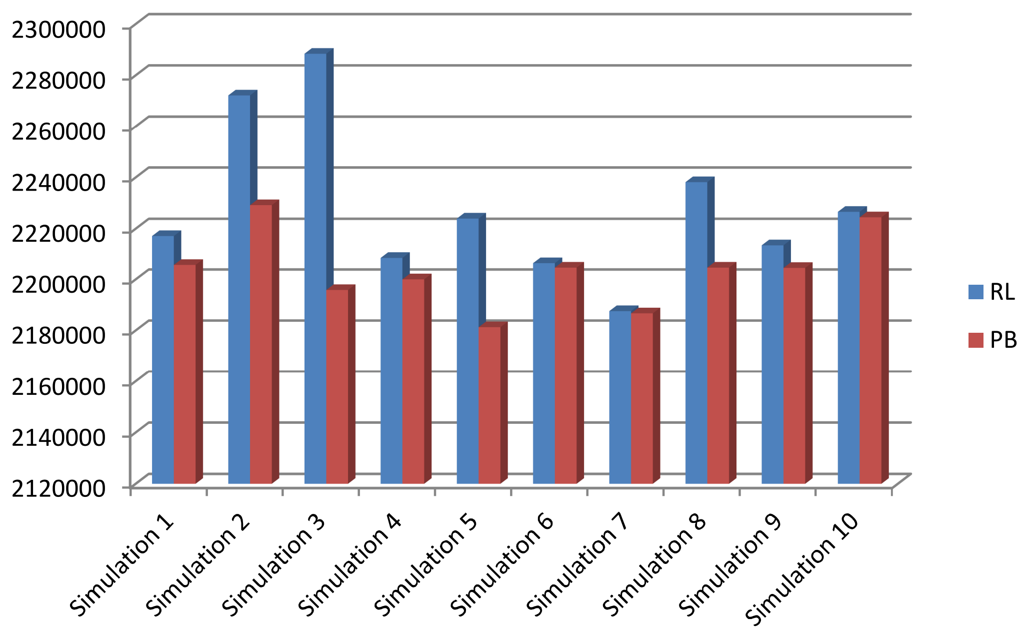


Figure 44 Simulation gains for medium load scenario

Analyzing these results, several considerations can be made. First of all, it is clear that RL earns more than Peak-Based. Then, the fact that the gap between the two incomes is not always the same can be explained considering that the agent learns its best policy making, sometimes, a random exploration, so that there are times in which this randomness allows to obtain a very great income, other times instead, this income is smaller; obviously, at the end, RL income is anyhow higher then Peak-Based one, due to the fact the agent is able to favourite calls more convenient against those less convenient.

Starting from these good results, the second analysis is a more general test, in order to understand what the mean behaviour of the two algorithms is. The second test has involved 30 simulations (each of about 5000 calls), and the results of these simulations have been averaged and normalized with respect to Peak-Based revenue. The bar-chart demonstrates that the proposed RL solution earns a more (about 1%) than the traditional Peak-Based, how it is visible in the following figure:

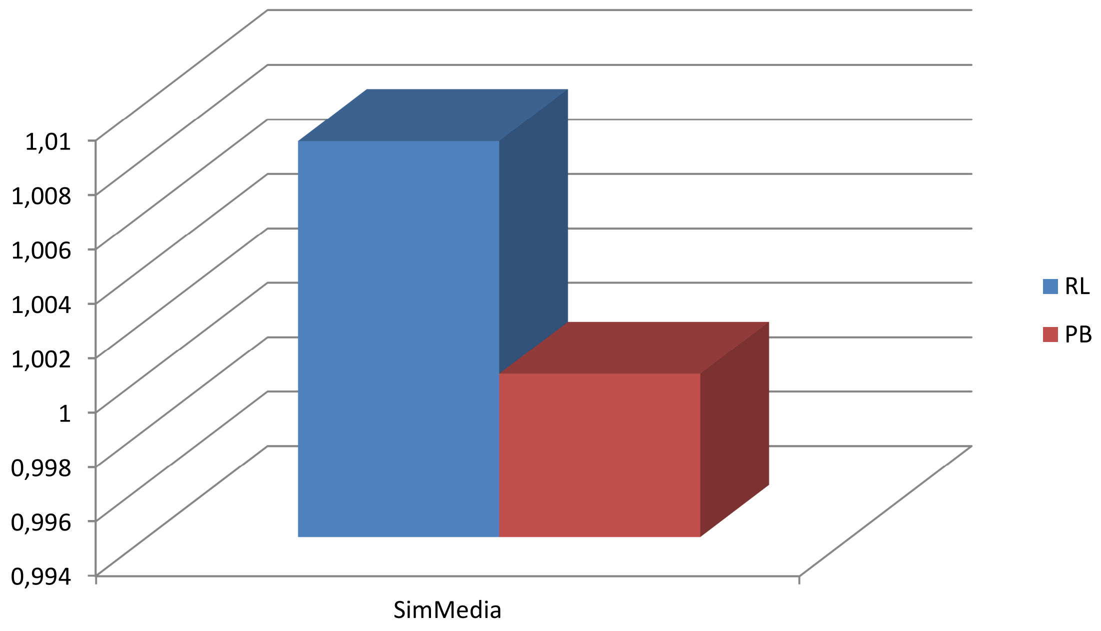


Figure 45 Mean gain comparison in medium load scenario

7.5.2. Scenario 2

This second scenario is similar to the first one: they only change the values of the arrival rates, which are grater then the previous ones. In this way it is possible to test this second RL algorithm with a network with more traffic. Values for this scenario are illustrated in the following table:

CoS	BitRate [Kbit/s]	$\lambda\ [s^{-1}]$	MHT[s]	reward [€/s]
1	50	0,7	120	1
2	80	0,4	100	5
3	110	0,1	200	7

Table 13 Traffic parameters of medium-high load scenario

Also in this case, the first test has involved only 10 simulations, in which both the algorithm have been tested using, every time, the same sequence of arriving connections. Results are shown in the following figure.

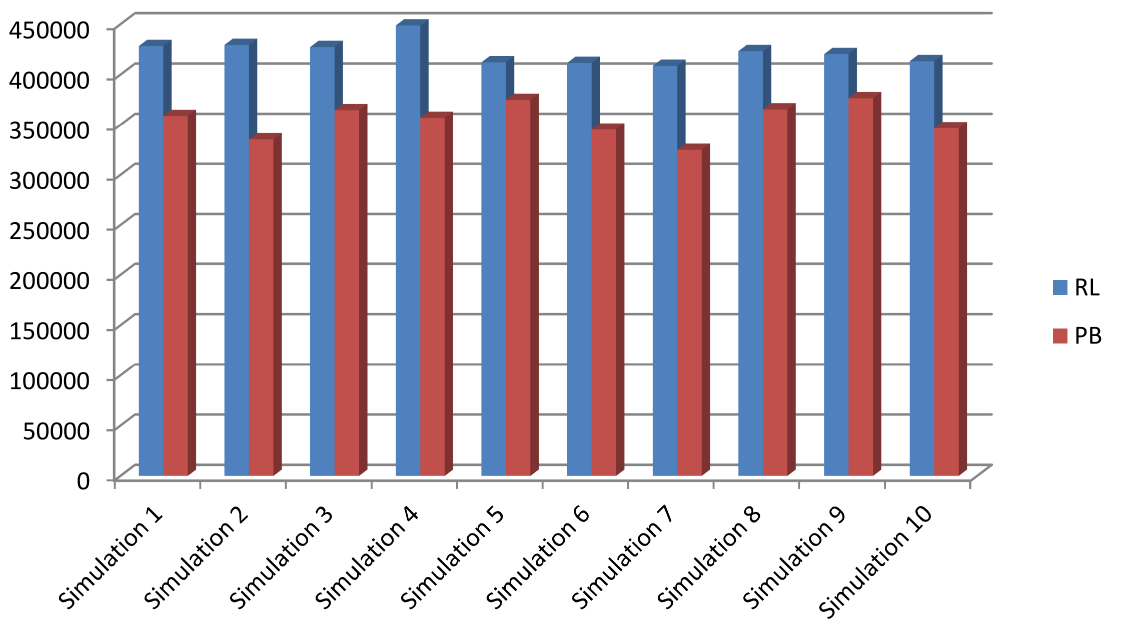


Figure 46 Simulation gains for medium-high load scenario

Subsequently the algorithm has tested in this same scenario, but considering others 30 simulations. Results of these simulations have been averaged, normalized respect Peak-Based, and reported in the following picture:

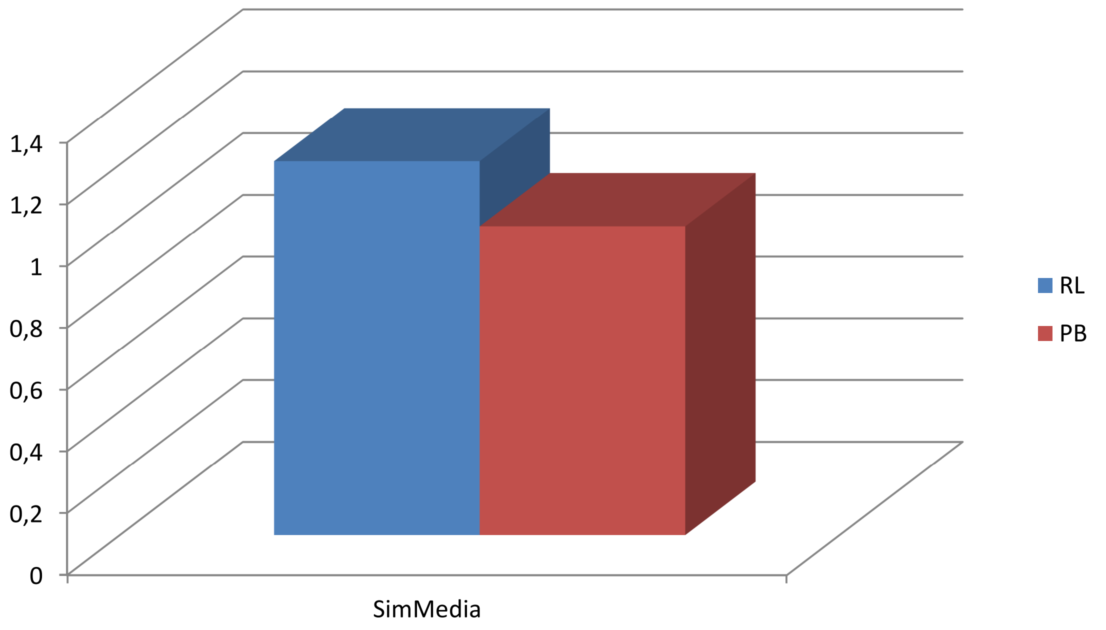


Figure 47 Mean gain comparison in medium-high load scenario

7.5.3. Scenario 3

This third scenario has higher arrival rates than first two ones; scenario values and simulation results are depicted below:

CoS	BitRate [Kbit/s]	λ $[s^{-1}]$	MHT[s]	reward [€/s]
1	50	1,2	120	1
2	80	1,0	100	5
3	110	0,6	200	7

Table 14 Traffic parameters of high load scenario

Also in this case, the first test compared both strategies with 10 simulations:

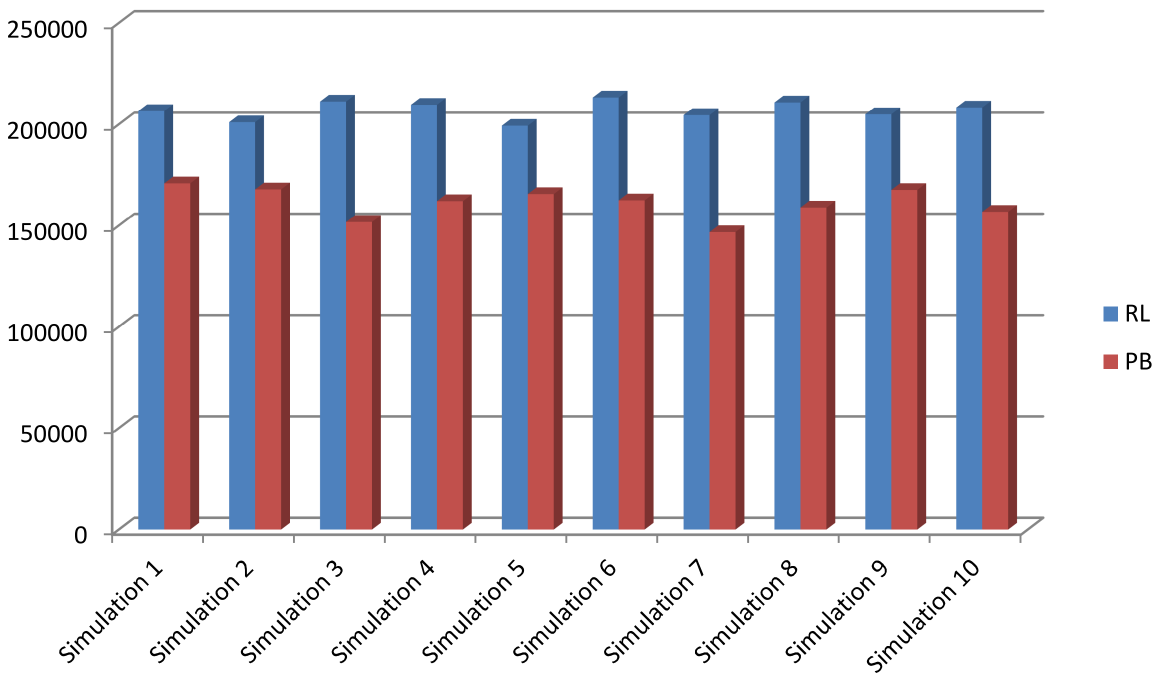


Figure 48 Simulation gains for high load scenario

Then both algorithms were tested on 30 simulations and results were averaged and normalized:

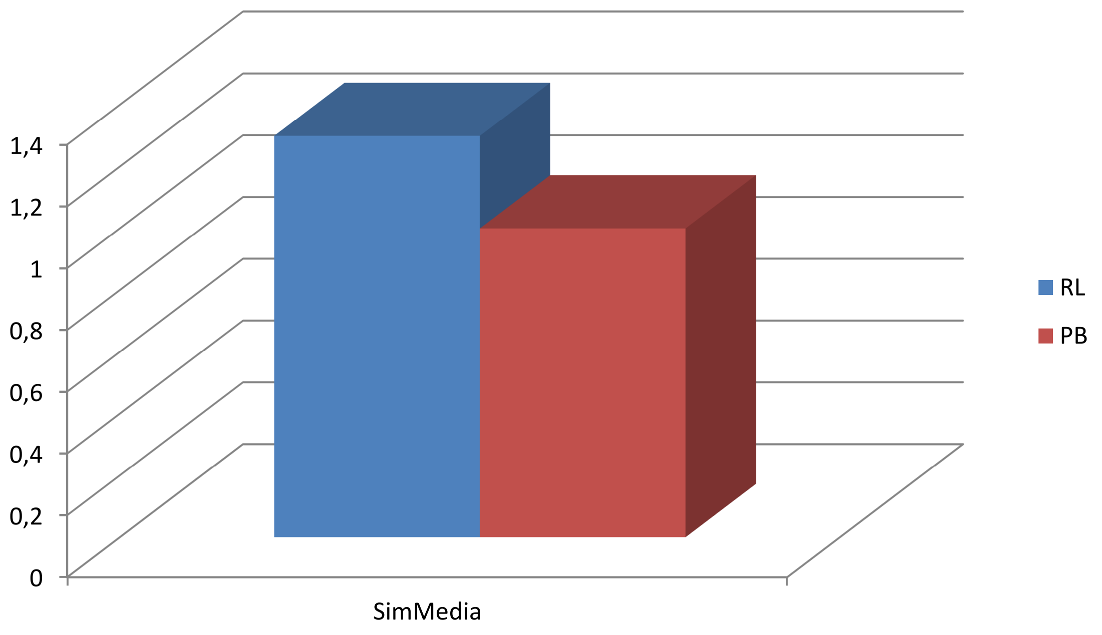


Figure 49 Mean gain comparison in high load scenario

7.5.4. Scenarios summary

Collecting all these results in a bar-chart, is it clear that, in terms of network's operator revenue, Reinforcement Learning strategy behaves exactly as Peak-Based when connection arrival rates are low and link is far from saturation condition; when arrival rates grow up, revenue of Reinforcement Learning is higher than Peak-Based due to the fact that RL strategy encourages those calls that are more profitable, when available bandwidth on the link starts to lack.

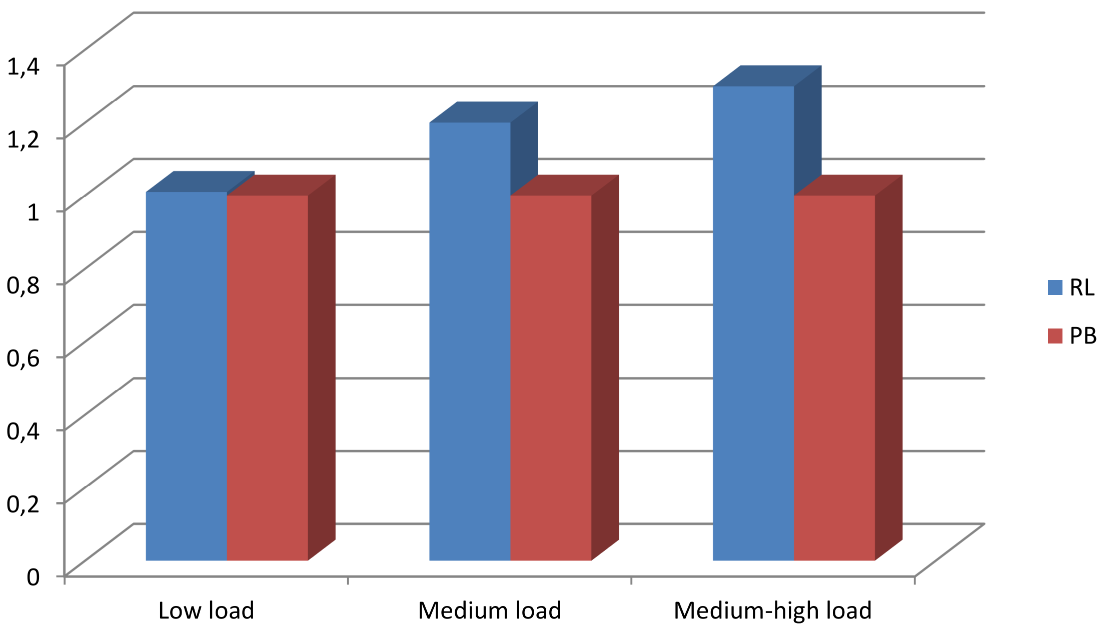


Figure 50 Gain summary

Summarizing:

- In the first scenario, both strategies have approximately the same network's operator revenue (RL algorithm earns just 1% more than peak-based one).
- In the second one, CAC-RL earns about 20% more than Peak-Based.
- In the third case, CAC-RL earns about 30% more than Peak-Based.

7.6. *Comparison between two RL approaches*

In order to ease the process of comparison of the two RL approaches, the algorithms where tested on the same semi-high load scenario (which is the one where advantages of RL algorithms are more evident), as indicated in the following table:

CoS	BitRate [Kbit/s]	$\lambda\ [s^{-1}]$	MHT[s]	reward [€/s]
1	53	0,75	124	1
2	87	0,43	97	5
3	111	0,12	213	7

Table 15 Traffic parameters of medium-high load scenario

500 simulation, using a list of 500 randomly generated different seeds, were ran, first using peak-based algorithm, then the first RL approach (without penalties and probabilities control) and, finally, the second RL one. Results were averaged and then normalized with respect to the peak-based income; the resulting bar chart is depicted below:

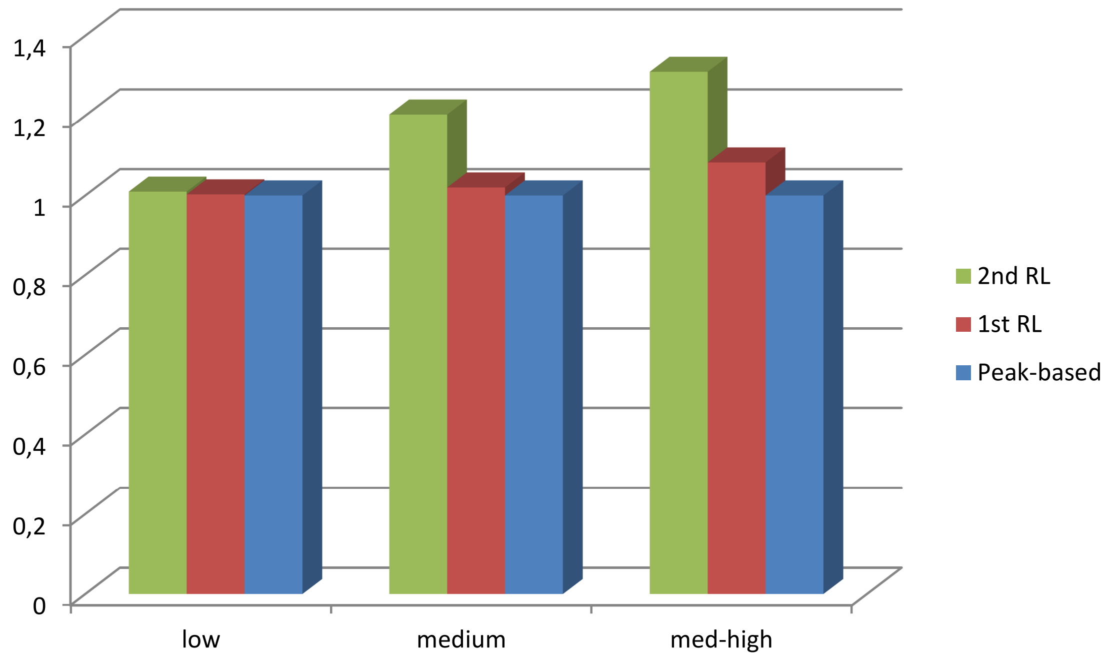


Figure 51 Comparison of results of Peak-based and RL algorithms in three different bandwidth load scenarios

As it is possible to see, results are aligned with those obtained with couple comparisons: the first RL approach earns about 2% and 8% respectively in medium and med-high load, while the second one about 20% and 30% respectively. While higher gains of RL approaches with respect to peak-based one are now clear, maybe it is not clear the different behavior of the two RL approaches.

The behavior of the first approach, the one with sigmoids, is strictly connected to sigmoids and their parameters; in other words, such sigmoidal functions highly impact on RL learning rules. Opportunely tuning sigmoid and other parameters could result in higher incomes in some scenarios, but lower in others, so that, unless having a precise knowledge of the actual scenario where to use the first RL approach, it could happen it will not work at its best. The second RL approach, instead, is much simpler, and is able to well adapt to all situations, just thanks to RL properties: so that this second RL approach is easier of the first one to be applied with good results.

Conclusions

Aim of this PhD. thesis was to ideate and create new CAC algorithms, based on the Reinforcement Learning theory, able to guarantee network operators higher revenues with respect to currently adopted peak-based solutions, while guaranteeing a certain degree of Quality of Service, in order to improve users' experience.

After a documentation regarding existing CAC solutions, different control and optimization theories, existing simulation tools and APIs, I've decided to adopt a Reinforcement Learning approach to solve the CAC problem. This solution is due the fact RL seemed to be a really adaptive solution to CAC problem, especially in the scenario I've decided to consider, i.e. the assumption to not have any measurement system or knowledge about network topology.

The tools adopted for simulation where OMNeT++, a free, open source, plug-in-based, network and traffic simulator, the Java language, for its portability, and the Piqle Java API to reproduce RL algorithms.

The thesis work started with the study of a possible state representation for the CAC problem in RL algorithms, then with the individuation of an appropriate reward function able to guarantee RL agent to take right decisions at calls' arrivals. Several different state representation and reward functions were considered, but, after some preliminary tests, two of them seemed to be more productive.

The first solution mainly adopted the number of ongoing calls for each class of service as state representation and an opportune sigmoid function as reward function. The so represented state was opportune to depict current situation to the RL agent; the reward function, instead, was able to guarantee the "inversion" of decision from accept to reject new calls when reaching certain critic conditions. First of all the sigmoid function was created to guarantee bandwidth limit respect, then, some additive contributions were introduced in order to guarantee either blocking probability and drop probability control.

This first approach demonstrated to be effective: starting from medium-load scenarios, this RL approach earned always more than peak-based solution on the same scenarios and in mean; this was true also when introducing probabilities control, which had, as counter-part, a relative diminution of the total income.

Secondly a different RL approach was investigated. A similar representation state was adopted, while, this time, using a much more simple reward function: to reward the agent with the mean reward of current calls of the incoming class of service in case of call acceptance, to penalize it of the same amount in case of rejection. This simple solution was tested on the same scenarios of the previous one, and has demonstrated to be much more effective than the first one. This was a little bit surprising, but a deeper analysis indicated that first RL approach is very sensible to sigmoid function parameters which, if not well tuned, could highly negatively impact RL incomes.

Regarding future deployments of the proposed algorithms, future studies on both algorithms will be developed, in particular to reduce sensibility to parameters in the first one, in order to introduce block and drop probabilities control in the second.

Regarding finally their possible usage, both algorithms are being proposed as solution to the CAC problem within the OMEGA FP7 project, to be integrated into OMEGA components requiring CAC functionalities.

Ringraziamenti

Potrà sembrare assurdo, ma questa è forse la parte più difficile da scrivere dell'intera tesi! Infatti, durante questi quasi tre anni di tesi, moltissime persone, a vario titolo, mi hanno aiutato, nella sua realizzazione.

Innanzitutto un ringraziamento sentito va al prof. *Francesco Delli Priscoli* che mi ha proposto questa meravigliosa opportunità di applicare la mia passione per l'informatica e l'innovazione a qualcosa di concreto e, auspicabilmente, utile per le reti del domani. Inoltre devo sempre a lui la crescita individuale e l'esperienza fattiva che questa tesi mi ha permesso di effettuare e, inoltre, la possibilità di continuare a lavorare nel progetto OMEGA nell'ambito dell'Università.

Poi, ma non stiamo facendo un ordine di importanza dei vari contributi, sento il dovere, oltre che il piacere, di ringraziare il dott. Vincenzo Suraci e l'ing. Alessandro Di Giorgio, che si sono dimostrati ottimi compagni di lavoro, attenti alle varie problematiche che via via si delineavano e sempre pronti a far valere la loro esperienza al fine di dipanare qualche ostica matassa.

Un ringraziamento va anche a tutti i "ragazzi" del *labreti* (chiamarlo "laboratorio di reti" sembra quasi volerne parlare in modo distaccato) ed in particolare i tesisti che, a vario titolo, hanno contribuito alla realizzazione di questo lavoro: Giorgio, Ivan ed Alessandra.

Durante questo periodo di tesi ho avuto modo di stringere amicizia o conoscenza con le persone del "*labreti*" (dottori, collaboratori, tesisti) che, essendo estremamente numerose, non posso ricordare qui per esteso: fra i tanti, e non me ne abbiamo quanti per dimenticanza non citerò, vorrei ringraziare Antonio, Tiziano, Emiliano, Gianfranco, Ilaria, Claudia, Marco, Andrea, Laura, Marco, Gabriele, Fabio, Erasmo, Andrea, Anna e Giulia, che hanno reso per me il lavoro molto più dolce. Ma in tutti questi anni di studio ho avuto al mio fianco amici eccezionali, spalle su cui appoggiarmi nei momenti del bisogno, risa nel momento dello svago, menti ed intelletti svegli e attenti nel momento dello studio e mi piace qui ricordarli tutti singolarmente (nell'ordine in cui mi vengono in mente, non di importanza o altro): Marco, Simone, Dario, Fabrizio, Diego, Alessia, Antonio. Mi piacerebbe citarne altri che per me molto hanno avuto valore, specie in passato, ma occorrerebbe troppo spazio e non è dunque il caso.

Vanno inoltre ringraziate tutti i parenti che mi sono stati vicini e mi hanno permesso di portare a termine il mio corso di studi: i miei genitori, mia sorella, zii e cugini, e, con caloroso affetto, i miei nonni, Balilla e Forisena, Michele e Rosa.

Infine, ma, nel mio cuore, necessariamente e gioiosamente per prima, voglio ringraziare Romina: oltre ad essere diventata mia moglie, povera lei, ora sta per darmi la più grande gioia che un uomo possa sperare… una figlia! Flavia: sicuramente il miglior risultato che io abbia mai ottenuto!

References

[1] 3GPP TR 21.905: “Vocabulary for 3GPP Specifications”

[2] 3GPP TS 23.002: “Network architecture”

[3] 3GPP TS 23.207: “End-to-end Quality of Service (QoS) concept and architecture”

[4] 3GPP TS 29.207: “Policy control over Go interface”

[5] ETSI TS 282 001: “NGN Functional Architecture”

[6] ETSI TS 282 004: “TISPAN Network Attachment Subsystem”

[7] ETSI TS 282 003: “TISPAN Resource and Admission Control Subsystem”

[8] ITU-T Rec. Y.1541: “Network performance objectives for IP-based services”, May 2002

[9] TISPAN Doc. Nb. TR 180 001 Ver. 1.1.1 DTR/TISPAN-00001-NGN-R1

[10] B. Doshi, et al., “*Integrated Network Design Tools (INDT): A suite of network design tools for current and next generation networking technologies”*, Proceedings of the 2nd IEEE Symposium on Computers and Communications (ISCC `97), 1997

[11] IMAGES Deliverable 2.4: “End to End QoS Architecture”

[12] IMAGES Deliverable 3.1: “Transport Plane Analysis and Design”

[13] EuQoS Consortium: Deliverable “*D1.2.1: EuQoS exploitation cookbook – Intermediate*”, April 30th, 2006

[14] DAIDALOS II: Deliverable 2.3.1 “Architecture and design: Quality of service”

[15] E. Angori, E. Borcoci, S. Mignanti, C. Nardini, G. Landi, N. Ciulli, G. Sergio, P. Neves “*Extending WiMAX technology to support End to End QoS guarantees”*, WEIRD Workshop - Coimbra, 22 May 2007

[16] OMEGA project homepage: http://www.ict-omega.eu/

[17] Goldstein, S., Remarks on the Global Markov Property. Comm. Math. Phys. 74 (1980), no 3, 223-234

[18] E. Levin, R. Pieraccini, and W. Eckert, "Using Markov Decision Process for Learning Dialogue Strategies," Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP-98),Vol. 1, pp. 201-204, Seattle, U.S., May 1998

[19] Z.Liu, M. El Zarki, “SIR-Based Call Admission Control for DS-CDMA Cellular Systems”, IEEE Journal on Selected Areas in Communications, Vol. 12, No. 4, May 1994.

[20] F.Y. Li, N. Stol “A Priority-oriented Call Admission Control Paradigm with QoS Re-

negotiation for Multimedia Services in UMTS” Proc. IEEE Vehicular Technology Conference 2001, pp 2021-2025.

[21] W. Burakowski, M. Diaz, O. Dugeon, A. Pietrabissa, F. Racaru, G. Santoro, H. Tarasiuk, “On Multi-Domain Connection admission Control in the EuQoS System”, IST Summit 2006, submitted.

[22] C. Bruni, F. Delli Priscoli, G. Koch, I. Marchetti, "An Optimal Approach to the Connection Admission Control Problem", International Journal of Control, Elsevier Science Pub., Vol. 79, No. 10, October 2006, 1237-1250.

[23] C. Bruni, F. Delli Priscoli, G. Koch, I. Marchetti, "Connection Admission Control in Cellular Networks: a Discrete Time Optimal Solution", Proceeding of IEEE INFOCOM Barcellona (Spain), April 2006.

[24] Judd, S. *Neural Network Design and the Complexity of Learning*. Cambridge, MA : MIT Press, 1990

[25] Michael L. Littman. *Markov games as a framework for multi-agent reinforcement learning*. In Proceedings of the Eleventh International Conference on Machine Learning, pages 157--163. Morgan Kaufman, 1994

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

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press Cambridge, Massachusetts, 1988 London, England

[28] Piqle homepage: http://piqle.sourceforge.net/

[29] Santamaria, J.C., Sutton, R.S., Ram, A. (1998). Experiments with reinforcement learning in problems with continuous state and action spaces, Adaptive Behavior 6(2): 163-218

[30] A. Varga, "*The omnet++ discrete event simulation system*," in Proc. of the European Simulation Multiconference (ESM'01), Prague, Czech Republic, June 6--9 2001

[31] A. Varga, "Omnet++," IEEE Network Interactive, vol. 16, no. 4, 2002

[32] Andr Maurits, George van Montfort, and Gerard vande Weerd. OMNeT++ extensions and examples. Technical report, Technical University of Budapest, Dept. of Telecommunications, 1995

[33] Varga, Andrs. Using the OMNeT++ discrete event simulation system indication

[34] D.M. Beazley, SWIG: An Easy to Use Tool for Integrating Scripting Languages with

C and C++, 4th Annual Tcl/Tk Workshop, Monterey, CA (1996)

[35] Barto, A. G. (1995b). Reinforcement learning. In Arbib, M. A., editor, Handbook of Brain Theory and Neural *Networks*, pages 804-809. The MIT Press, Cambridge, MA.

[36] Bellman, R. E. (1957a). Dynamic Programming. Princeton University Press, Princeton, NJ.

[37] Bellman, R. E. (1957b). A Markov decision process. Journal of Mathematical Mech., 6:679-684.

[38] Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ.

[39] Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. Journal of Artificial Intelligence Research, 2:287-318.

[40] Dayan, P. (1992). The convergence of TD(λ) for general λ. Machine Learning, 8:341-362.

[41] Doya, K. (1996). Temporal difference learing in continuous time and space. In Touretzky, D. S., Mozer, M. C., and Hasselmo, M. E., editors, Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pages 1073-1079, Cambridge, MA. MIT Press.

[42] Howard, R. (1960). Dynamic Programming and Markov Processes. MIT Press, Cambridge, MA.

[43] Peng, J. (1993). Efficient Dynamic Programming-Based Learning for Control. PhD thesis, Northeastern University, Boston, MA.

[44] Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the Dyna framework. Adaptive Behavior, 1(4).

[45] Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In Proceedings of the Tenth International Conference on Machine Learning, pages 298-305. Morgan Kaufmann.

[46] Sutton, R. S. (1988). Learning to predict by the method of temporal differences. Machine Learning, 3:9-44.

[47] Tsitsiklis, J. N. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. IEEE Transactions on Automatic Control.

Annex A. The WEIRD Project

A.1. Introduction

It is common knowledge that over the last decade there has been a major boost in communication networks. In fact, the development of high-performance backbone networks was immediately followed by the rapid dissemination of broadband wired access technologies, such as leased lines based on fibre-optic links, cable modems using coaxial systems, and digital subscriber line (xDSL) access networks. This gave users a whole new class of services that exploit the increasing number of available network resources. Many new services are based on multimedia applications, such as voice over IP (VoIP), video conferencing, video on demand (VoD), massive online gaming, and peer-to-peer. Unlike traditional TCP/IP services, multimedia applications usually require strict network guarantees such as reserved bandwidth or bounded delays.

A.2. The general context

The increasing of wireless Metropolitan Area Networks is due to the need to reach more and more user communities – in case isolated – by overcoming the cost barriers of wired technologies. This trend paved the way to the use of mainly proprietary solutions, some of them based on updated and empowered Wi-Fi systems, others focused on point-to-point wireless connections based on RF technologies. This sub-optimal progression stimulated the relevant standardization bodies to work for the introduction of new open standards, facilitating large scale economies and wide market acceptance: in this context the IEEE 802.16 (also known as WirelessMAN) and the ETSI HiperMAN started to be defined and

Worldwide Interoperability for Microwave Access (WiMAX) consortium was established to support certifications of the IEEE 802.16-2004 standard.

In the meanwhile, most of the worldwide research initiatives started to focus on IP network architectures able to decouple the Application and Control Planes from the underlying Transport Plane. The main objective of these studies and developments is the seamless end-to-end integration of the various network technologies, and this is commonly achieved through special "convergence layers" in the most advanced network architectures. With respect to the Transport Plane, these convergence layers are able to cope with the different underlying technologies by means of special technology-dependent drivers; towards the Control Plane, they provide special functionalities for QoS/resource management, access authentication, L3 and L2 mobility, security, etc. The control mechanisms enabled by the entities in the "convergence layers" proved to enhance the network performance, both in terms of resource utilization/consumption and of end users' satisfaction, because they simplify the provisioning of the best network configuration for each incoming service request.

The WEIRD project aims to exploit and enhance the WiMAX technology in a convergence layer heterogeneous network architecture, in order to cope with future needs of research user communities and to build a test-beds allowing European research network GÈANT, GÈANT2 and relevant National Research Networks, to be reachable from isolated or remote areas.

A.3. Objectives

The WEIRD project primarily addresses the objective IST-5-2.5.6 Research networking Testbeds: it aims at validating actual wireless state-of-the-art technology, but also at upgrading and integrating it in order to prepare for the deployment of next generation Information and Communications networks across Europe.

Basically, the WEIRD project proposes broadband connectivity based on a wireless technology providing a flexible, cost-effective, standards-based means of filling existing gaps in broadband services not envisioned in a “wired” world. The project is in support of the activities carried out in the area of Research Infrastructures on high-capacity and high-speed communication networks for all researchers in Europe (GÉANT), offering a proper connection technology in charge of adding new NRNs to GÉANT. For instance, a wireless backhaul is definitely the best solution, in terms of costs and required deployment time, in the presence of physical obstacles, compared to the wired one. With the proposed wireless technology, a NRN that is actually isolated from a “wire” point of view, or that belongs to a developing country, can easily be integrated in the GÉANT research network. This provides open test infrastructures to third party researchers and includes demonstrator environments, resulting in research synergies and also by facilitating their exploitation.

Research networks community is not the only subject that the WEIRD project addresses. There are several different real-world scenarios and production activities that can take advantage from the proposed technology. First of all residential broadband customers and underserved areas: practical limitations prevent cable and DSL technology from reaching many potential broadband customers, so that many urban and suburban locations may not be properly served. Deploying a wire has a significant cost that is not successively covered if the broadband service is offered in an area with a low subscriber density. A wireless solution would seem best suited, but unfortunately, the current generation of wireless systems is relatively expensive for mass deployments because, without a standard, it’s difficult to achieve economy of scale. This cost inefficiency will be changed by the promotion of standard-based systems, as supported within the WEIRD project. Standards are important for the wireless industry because they enable economies of scale that can bring down the cost of equipment, ensure interoperability and reduce investments risks for operators.

Furthermore, the WEIRD project aims at integrating, testing, validating and demonstrating the WiMAX wireless access technology in charge of solving the difficulties in last-mile implementations. WISPs (Wireless Internet Service Providers) have been asking for wireless technologies that make metropolitan area access possible. The three key deployment types that make up metropolitan area access are backhaul, last-mile and large area coverage (referred to as hot zones). Wireless last-mile and large-area coverage typically uses a properly modified standard (IEEE 802.11), but the need for a specific standard is evident and the WEIRD project contributes for its provision: open standard radio technologies offer advantages to WISPs and users; industry-wide support and innovation are driving broadband wireless networking technologies. The aim of WEIRD is to integrate WiMAX in a heterogeneous networking environment, by defining the interfaces with the convergence layers. Authentication, Authorization, Accounting, roaming, security, QoS and resource management entities will be the main blocks where the WEIRD project will focus on, and the approach that will be followed to implement new algorithms will be based on advanced mathematical and control theories, setting the grounds for the achievement of a high degree of network autonomy.

Finally considering scientific environments, the WEIRD project supports a technology that will be extremely useful in all those scenarios where human presence cannot be continuously granted or moving is not easy. These scenarios include all the monitoring activities in remote or dangerous areas, such as a volcanic sites monitoring, fire prevention and monitoring systems, or simply the communication between isolated areas, such as sea platforms. Thus the WEIRD project promotes interoperability of solutions across different scientific and industrial disciplines and this also means the possibility of large scale experimentation in real settings to promote interoperability across heterogeneous technology domains, with particular attention to the wireless technologies belonging to the state-of-the-art.

A.4. Main innovations

The foreseen deployment of WiMAX technology in a heterogeneous network powered with the tools and mechanisms of a convergence layer will constitute the main field of innovation of the WEIRD project.

The project will try to impact as little as possible each mature architectural module, by focusing in these cases on the interworking aspects needed to build a coherent modular system. Consequently, deep interlacing and dependencies will be highlighted among the different procedures: QoS at layer 2 and layer 3, security with authentication and authorization, accounting with continuous network measurements, integration and coexistence of different hierarchical virtual private networking services (e.g. VLAN, VPN, etc.).

The design and development of this interfacing, as well as of the convergence layer drivers, are expected to bring significant improvements and openness, still within the range of standard specifications. The demonstration in a distributed and enriching test-bed of the designed and implemented functionalities spanning the overall project life-time will provide a powerful and gradual validation of the WEIRD solutions. Moreover, the planned demonstrations will contribute to spread the culture and application of easily managed and self-controlled networks with evident impact on a range of possible stakeholders that will benefit of the WEIRD “proof-of-concept”.

WEIRD project is expecting to impact and improve the quality of the technology and boost the market consensus of WiMAX technology. The main innovation activities that will be part of WEIRD project and that will have a potential impact in this way encompass:

- Definition of generalized Network Interface semantics and mechanisms (i.e. for the interactions between the Control Plane layer and the convergence layer), which enables the automatic SLAs negotiation and service invocation at the different network boundaries, based on the interoperation of existing signalling mechanisms.
- Define an integration framework for seamless end-to-end security and AAA procedures, at the Control and Management Plane, in the considered network sections.
- Simulation studies on the possible enhancements to the standardized WiMAX technology for the selected deployment scenarios. These simulations will be aimed to evaluate performance not only in terms of throughput (as often present in the state of the art), but also in terms of fulfilment of the service requirements claimed by the applications (e.g. VoIP, videoconference and video streaming, telemedicine, e-learning, distributed classrooms and tele-engineering).
- Integration between fixed and mobile environments will be improved by the WEIRD project. This will be achieved by the integration of extended signalling functions in the control plane and by cross-layer optimizations.
- Application adaptation in the WEIRD project will improve application awareness. This will be achieved by the integration of location management and environment awareness mechanisms in the applications to be used in the project testes and trials.
- WEIRD will implement a test-bed which will demonstrate full functionalities for WiMAX, interacting with higher layers.

- WEIRD includes emblematic user communities that will drive system requirements obtaining a WiMAX solution able to extend GEANT to solve their problem.
- Conception, design and validation of RoF photonic subsystems for massive deployment of WiMAX networks at low cost. Possibility to extend coverage of WiMAX networks to difficult propagation environments, such as the underground.
- WEIRD will improve the technological offer to support emergency applications and remote area coverage. The project will provide broadband connection and mobility to experimenting the fire prevention and volcano monitoring enabling the use of these applications in emergency scenarios.

A.5. Role of University of Rome

The University of Rome through the "Dipartimento di Informatica e Sistemistica (DIS)" (Department of Computer and System Sciences) of the Faculty of Engineering is involved in the tasks related to the Quality of Service in WEIRD Project.

This project will give opportunity to the University of Rome to reinforce the already existing co-operations and to create new links between the University of Rome and the manufactures and operators both in the surrounding area and in remote areas with the goals, on one hand, to stimulate these companies towards advanced research topics and, on the other hand, to create new employment opportunities especially for the young people.

In particular University of Rome expects to collaborate with Datamat to create a common research laboratory for control and management of WiMAX technology. University of Rome plans to validate the research that will be made in the area of QoS and resource management for WiMAX in a real test-bed at Wind premises

In addition, the University of Rome intends to exploit the results of this project for didactic and teaching purposes. In particular, many master degree theses are expected to profit from the documentation and the background coming from the project in question. Moreover, project results will be exploited to upgrade and update the programs of several courses and to hold thematic seminars on these matters, the courses of the telecommunication graduate and postgraduate program will be improved with courses in management and control of WiMAX. In particular, participation to this project will allow new generation engineers to acquire know-how on telecommunications and informatics and more specifically on WiMAX, on Next Generation Networks and on QoS management.

Finally, dissemination will be also assured by extensive publications especially on the major international reviews and conferences and by the participation to the main events organized by the European Union as well as by other institutions.

Annex B. List of publications

B.1. Accepted

Silvano Mignanti, Alessandro Di Giorgio, Vincenzo Suraci, "*A Model Based RL Admission Control Algorithm for Next Generation Networks*", The Eighth International Conference on Networks (ICN 2009), March 1-6, 2009 - Gosier, Guadeloupe (France)

Silvano Mignanti, Ilaria Marchetti, Kostas Pentikousis, Fausto Andreotti, Antonio Cimmino, Giada Landi, Gabriele Tamea, Pedro Miguel Neves, MariaRita Spada, Paulo Simoes, Mario Castellano "*WEIRD Testbeds with fixed and mobile WiMAX technology for user applications, telemedicine and monitoring of impervious areas*" Tridentcom 2008

Silvano Mignanti, P. Neves, M. Castellano, V. Augusti, C. Mambretti; G. Martufi, F. Andreotti "*WEIRD – Real Use Cases and Applications for the WiMAX Technology*" CCNC'2008/2nd IEEE BWA Workshop

Silvano Mignanti, Vincenzo Suraci, Massimiliano Tamburriello, Augusto Silvani "*CDQL: a language for multi-protocol content discovery architectures*" STreaming Day 2007

Vincenzo Suraci, Silvano Mignanti, Augusto Silvani, Massimiliano Tamburriello "Context-awareness in Content Discovery architectures" STreaming Day 2007

E. Angori, E. Borcoci, S. Mignanti, C. Nardini, G. Landi, N. Ciulli, G. Sergio, P. Neves *"Extending WiMAX technology to support End to End QoS guarantees"*, WEIRD Workshop - Coimbra, 22 May 2007

Silvano Mignanti, Vincenzo Suraci, Anna Aiuto "*Context-aware Semantic Service Discovery*" IST Mobile & Wireless Communications Summit 2007

Silvano Mignanti, Vincenzo Suraci, Carmine Di Menna "*An Ontology-Based Multi-Protocol Service Discovery Framework*" IST Mobile & Wireless Communications Summit 2007

Carmine Di Menna, Silvano Mignanti "*Enhancing Multiprotocol Service Discovery Framework in Pervasive Computing Networks*" IST Mobile & Wireless Communications Summit 2007

S. Mignanti, V. Suraci, T. Inzerilli "*Enhancing Service Location Protocol with a OWL-based Service Description Model for Service Discovery in Pervasive Computing Networks* ", IST Mobile & Wireless Communications Summit 2006

Silvano Mignanti, Vincenzo Suraci "*Toward Peer-to-Peer Service Sharing in Pervasive Systems*", IST Mobile & Wireless Communications Summit 2006

S. Mignanti, V. Suraci, T. Inzerilli "*Design and Implementation of a Service Discovery Architecture in Pervasive Systems*", IST Mobile Summit 2005.

Co-author of DAIDALOS deliverables: D412 e D352

Co-author of DAIDALOS II deliverables: D121, D 321, D351, D421,…

Co-author and responsible of WEIRD deliverables: D6.1, D6.2, D6.3, D6.4, D6.5, D6.6, D6.7, D6.8

B.2. Submitted

Silvano Mignanti, Vincenzo Suraci, Alessandro Di Giorgio, "*A Model Based RL Admission Control Algorithm for Next Generation Networks*", IEEE Wireless Communications Magazine (Notification of acceptance: March 15, 2009)