A reinforcement learning based solution for cognitive network cooperation between co-located, heterogeneous wireless sensor networks

Milos Rovcanin, Eli De Poorter, Ingrid Moerman, Piet Demeester

Ghent University - iMinds, Department of Information Technology (INTEC), Gaston Crommenlaan 8, Bus 201, 9050 Ghent, Belgium e-mail: milos.rovcanin@intec.ugent.be.

Abstract

Due to a drastic increase of the number of wireless communication devices, these devices are forced to interfere or interact with each other. This raises the issue of possible effects this coexistence might have on the performance of each of the networks. Negative effects are a consequence of contention for network resources (such as free wireless communication frequencies) between different devices. On the other hand, a possible cooperation between co-located networks could also improve certain aspects of networking for each one of them. This paper presents a self-learning, cognitive cooperation approach for heterogeneous co-located networks. Enabling cooperation is performed by activating or deactivating services that influence the interaction between wireless devices, such as an interference avoidance service, a packet sharing service, etc. Activation of a cooperative service might have both positive and negative effects on network's performance, regarding its high level goals. Such a cooperation approach has to incorporate a reasoning mechanism, centralized or distributed, able to determine the influence of each symbiotic service on the performance of all the participating sub-networks, taking into consideration their requirements. Coupled with the concept of enabling symbiotic services, a machine learning technique known as the Least Squares Policy Iteration (LSPI), is presented in this paper as a novel network cooperation paradigm.

Keywords:
Network cooperation, machine learning, linear approximation, Least Squares Policy Iteration, policy improvement

1. Introduction

There is a growing need for network solutions that efficiently and dynamically support at run-time cooperation between devices from different sub-nets. Surely, this is a result of a drastic increase of a number of interleaved, heterogeneous wireless networks with different coverage, data rates and mobility capabilities. An illustrative example of the two co-existing wireless networks is given on Figure 1. Once the communication is established between the two networks, the process of cooperation, in terms of sharing the available resources and capabilities, can be initiated.

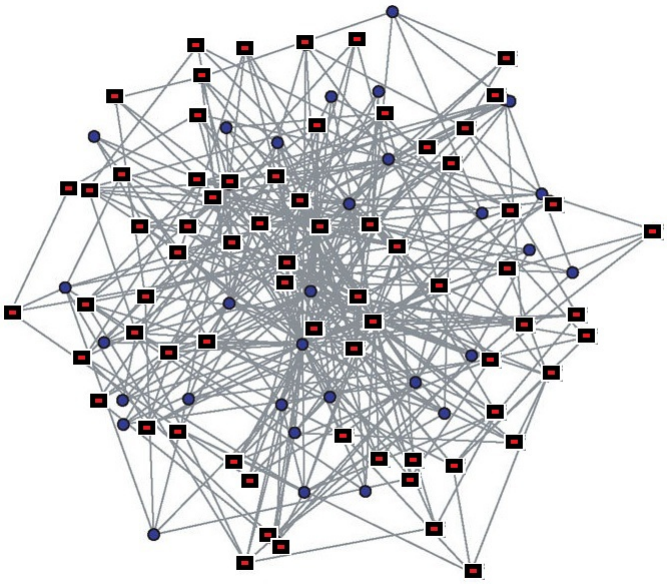

Figure 1: An example of two co-located, heterogeneous, wireless networks, possibly eligible for cooperation. The way nodes are connected (blue dots and red squares) implies that these two networks are unaware of each other at the moment

For the time being, the only way to support connectivity between these co-located devices is to statically group them into different sub-nets, according to their communication technology. This way, the same network policies can be used for a sub-net, regardless of the characteristics of the devices. Although possible, this approach is usually quite complex [26] and inefficient because of the following reasons:

- Manual configuration is time consuming and expensive, especially in a network of a large scale
- Manual approach does not take into account dynamically changing network requirements (e.g. changes in network topology)
- Although not aware of each other, devices from different sub-nets can harmfully interfere with each other

Improvements made by using a dynamic, at run-time management are expected to have an impact on many networking aspects: decreased energy consumption, decreased interference, better coverage, minimized electromagnetic exposure, better bandwidth allocation, increased availability etc. Complexity of the problem is increased by the fact that the management is done over multiple heterogeneous networks, characterized with distinct and different network requirements and capabilities. An utmost attention must be paid to this; otherwise the entire system will suffer from an unacceptable waste of resources. Instead of doing it manually, an intelligent entity - a cognitive engine can be used to initiate and supervise the entire process [25].

Distributed or centralized, it must be capable of a (1) dynamic optimization and decision making and (2) continuous exchange of collected measurements and environmental states. In a cooperation paradigm that is based on activation of symbiotic services, the role of a cognitive engine is to calculate the optimal set of services, activated in each of the participating subnetworks. The ultimate goal is to improve each network's performance, considering its requirements.

In the initial phase of research, described in [2], a linear programming based approach has been utilized for the purpose of coordinating the process of symbiotic service negotiation. In order to calculate the optimal set of services for each sub-net, the CPLEX ILPSolver [1] requires the information about influences each service has on each network incentive (e.g. high reliability, low delay, long network lifetime etc.). This information is extremely difficult to obtain, especially in highly dynamic environments. It can be acquired from a literature, previously published papers dealing with similar issues or as a result of a simulation. In any case, the accuracy of the gathered date is questionable.

Approach described in the following chapters of this paper is an improvement of the above mentioned method. The new methodology does not require any a priori knowledge about the symbiotic service influences on the

network. Being a form of machine learning [3] [4], the Least Square Policy Iteration (LSPI) [5] algorithm gathers knowledge through a number of trial-and-error episodes. It uses *basis functions*, features from the network, to make an assessment about the influence that different service sets pose on each network incentive. Therefore, it does not demand the kind of a priori knowledge needed in the case of an ILPSolver engine. LSPI does not require fine tuning of the initial parameters such as learning rate. It uses linear functions to represent Q-values of each state/action pair, based on the most recently updated information regarding the selected features (hop count measurements, duty cycle, average energy spent per node etc.). In other approaches, agents make decisions directly based on Q-values, which may be outdated, depending on the network and algorithm dynamics.

The remaining of the paper is organized as follows: Section 2 presents the related work. In Section 3, the LSPI fundamentals - mathematical background, convergence and stopping conditions are described in details. Section 4 introduces the use case LSPI is applied to and the evaluation of its performance. All the aspects of the implementation are thoroughly described in Section 5. Results are analyzed and major issues are identified in Section 6. The future course of our research is presented in Section 7. Finally, Conclusion section summarizes the paper.

2. Related work

The following subsections present recent and the most relevant work regarding:

- An application of the reinforcement learning techniques to optimization problems on lower wireless network layers
- Some of the existing and relevant methods for a higher layer network optimization

2.1. Using reinforcement learning as an optimization solution in cognitive radio networks

Cognitive radio [15], [16] enables devices to autonomously reconfigure transmission parameters based on the state of the environment in which they operate.

Reinforcement learning (RL) has been extensively used for solving various optimization problems in cognitive radio networks. Authors of [7] tackled a

problem of an efficient spectrum sharing, using the REINFORCE [**?**]] algorithm. Work presented in [8] and [9] demonstrates how a Q-learning algorithm and a distributed reinforcement learning scheme, respectively, can be used to solve a channel assessment problem. In [10] a real-time, multi-agent RL algorithm - decentralized Q-learning solves the aggregated interference problem [11] in 802.22 based cognitive radio networks.

RL methods have also been used for solving network management problems. An autonomic reconfiguration scheme that enables intelligent services to meet QoS requirements is presented in [12]. Algorithm's efficiency is proved by improving the performance of the original AODV routing protocol in a heterogeneous network environment.

2.2. Higher level network optimization

When parameters of the higher network layers are optimized based on changes in the network environment, the term cognitive networking [28], [29] is used.

Network planning tools [13], [14] optimize network criteria such as coverage or throughput by calculating the optimal placement and transmission power of devices. Unfortunately, they are efficient only in static and predictable network deployments and cannot be used in networks that (i) dynamically change network topology (such as ad-hoc networks), (ii) have network requirements that change over time, or (iii) in mobile environments. In specific cases, planning tools can be used in combination with incentive driven network methodologies. For example, existing planning tools can be used to estimate the influence of network services on the high level goals, which can be used as input for the negotiation phase.

An already mentioned paradigm [2] describes inter-network cooperation through an activation of network services in co-located networks. The process is initiated and controlled by a linear programming based reasoning entity - CPLEX ILPSolver, but requires design-time knowledge of the impact of a network service on the network performance.

2.3. LSPI usage

Least squares policy iteration algorithm have initially been tested initially tested on problems such as "inverted pendulum, bicycle ride" and "chain walk" [5]. In the domain of wireless sensor networks, it has mainly been used to solve routing [21] and link scheduling problems [22].

Our approach utilizes a modified LSPI mechanism to solve a cross-network optimization problem, by detecting and forcing the best performing set of services in each of the participating networks. As to our present knowledge, no similar solutions have been proposed so far.

3. Least Squares Policy Iteration

LSPI was first introduced by M.G.Lagoudakis and R.Parr in [5] and further elaborated in [6]. It is presented as a model-free, reinforcement learning technique [24], which combines least Squares Temporal Difference function (LSTDQ) approximation with policy iteration. The fact that it uses linear function approximations poses some significant advantages in the context of reinforcement learning [21]

- Algorithm is easier to implement than other widely used machine learning approaches (e.g. Q learning)
- Behavior of an algorithm is fairly transparent from both analysis and testing point of view
- Suitable for detection of non-linear interactions between protocols. For example, activating protocol A can be beneficial for the network, activating protocol B can also be beneficial, but due to unforeseen interactions, activating both at the same time might hamper the good operation of the network. A self-learning approach such as LSPI can automatically learn about this type of behavior.

3.1. Mathematical background

In LSPI, the state-action value function (Q function), which assigns values (rewards) to every state-action pair from a problem's state-action space, is approximated as a linear weighted combination of k basis functions (features):

$$Q(s, a; w) = \sum_{j} \phi_j(s, a)\omega_j \tag{1}$$

Here, ω is a set of weights, that needs to be recalculated in order to reach a fixed point in value function space. Value function can also be presented in a matrix form as:

$$\mathrm{Q}^{\pi} = \Phi\omega \tag{2}$$

where $|S||A|$ dimensional Φ matrix contains values of basis functions, defined for each state/action (s, a) pair and π designates the decision making policy that is being used. Generally, the number of basis functions is much smaller than the number of state/action values, $k << |S||A|$. Good examples of basis functions are: network's duty cycle, average time spent in a radio receiving/sending mode, residual energy of nodes etc.

If we take into consideration the general outlook of the Q value function, given in a form of a Bellman equation:

$$Q(s,a) = r(s,a) + \gamma \sum_{s'} P(s'|s,a) maxQ(s',a') \tag{3}$$

where the $r(s, a)$ represents the immediate reward for executing action a at state s, while $\gamma \sum_{s'} P(s'|s, a)maxQ(s', a')$ represents the maximum expected future reward. Factor γ is known as the *discount* factor and its purpose is to make sure that a reward given for the same state/action pair is decreasing over time.

Finally, if we combine equations (1),(2) and (3), we get the matrix outlook of the Bellman equation, that considers all the approximations introduced by the LSTDQ algorithm:

$$\Phi\omega = R + \gamma P^{\pi}\Phi\omega \tag{4}$$

where R is the reward given after transferring from one state to another and γ is a discount factor. Assuming that all basis functions are independent (columns of a Φ matrix), we can construct the following equation:

$$\Phi^{T}(\Phi - \gamma P^{\pi}\Phi)\omega^{\pi} = \Phi^{T}R \tag{5}$$

Now, it is relatively easy to calculate the weight factors by solving a system:

$$\begin{gathered} \omega = A^{-1}b \\ \text{where: } \mathrm{A} = \Phi^{T}(\Phi - \gamma P^{\pi}\Phi) \\ \mathrm{b} = \Phi^{T}R \end{gathered} \tag{6}$$

Here, P^π represents a transition probability matrix describing the transitions from pairs (s, a) to pairs $(s', \pi(s'))$. In other words, it contains probabilities of a certain state/action sequence.

Since P^π and R cannot be known a priori, the values they contain must be learned from sampled data in order to determine matrices A and b. Given a set of samples from the environment, $D = (s_{d_i}, a_{d_i}, s'_{d_i}, r_{d_i} | i = 1, 2, ..., L)$, where (s'_{d_i}) is sampled using $P(s'_{d_i} | s_{d_i}, a_{d_i})$, which corresponds to a decision making policy that is used at the moment. The approximate versions of $\Phi, P^\pi\Phi$ and R are generated in the following way:

$$\hat{\Phi} = \begin{pmatrix} \phi(s_1, a_1)^T \\ ... \\ \phi(s_n, a_n)^T \end{pmatrix} \widehat{P^\pi\Phi} = \begin{pmatrix} \phi(s'_1, \pi(s'))^T \\ ... \\ \phi(s'_n, \pi(s'))^T \end{pmatrix}$$
$$\hat{R} = \begin{pmatrix} r_1 \\ ... \\ r_2 \end{pmatrix}$$

(7)

Matrices A and b can now be approximated in the following way:

$$\hat{A} = \hat{\Phi^T}(\hat{\Phi} - \gamma \hat{P^\pi}\Phi)$$
$$\hat{b} = \hat{\Phi^T}\hat{R}$$

(8)

3.2. Convergence and algorithm's stopping condition

As the number of samples, uniformly taken from the state/action space, grows, the consistency between approximated and true values of matrices A and b ($\hat{A}$ and $\hat{b}$) grows. This can be mathematically described in the following manner:

$$\begin{aligned} \mathrm{E}(\hat{\mathbf{A}}) &= \tfrac{L}{|S||A|}\mathbf{A} \\ \mathrm{E}(\hat{b}) &= \tfrac{L}{|S||A|}b \end{aligned} \tag{9}$$

where L is the cardinal number of the training (sample) set. It is important to note that approximations, gathered from any additional set of samples from the same state/action space, $(\hat{A}_1, \hat{b_1}, \hat{A}_2, \hat{b_2})$ can be combined to yield an even better approximation:

$$\begin{aligned} \hat{\mathbf{A}} &= \hat{\mathbf{A}_1} + \hat{\mathbf{A}_2} \\ \hat{b} &= \hat{b_1} + \hat{b_2} \end{aligned} \tag{10}$$

In its original form, LSPI uses a metric describing a difference between consecutive ω parameters as the stopping criterion. The algorithm stops if the difference between two consecutive values of ω factors (in two consecutive episodes, evaluating decision policies π and π') is smaller than a user defined ϵ:

$$\|\omega' - \omega\| < \varepsilon \tag{11}$$

Value of ε can be thought of as a "positive scalar that bounds the errors between the approximate and the true value functions over all iterations". According to [6], LSPI is a stable algorithm that will either converge or oscillate in an area around value function, bounded by approximation error ε. Two factors with the largest influence on ε are the choice of basis functions and the choice of samples, which depends on the decision policy that is being used.

A good choice of basis functions will always depend on the specific nature of the problem that is being solved and this process cannot be generalized. On the other hand, one strong point of the LSPI is that the same set of samples can be utilized in to evaluate different policies and estimate the corresponding state value functions (Q values). This is the major advantage of the algorithm, since LSPI starts off with an initial policy and changes it (improves it) through a number of iterations without the need of collecting additional samples. This means that, in its essence, LSPI is an off-line, off-policy learning algorithm - sample collection and learning are separated from the execution. As explained in [20], the whole approach can be modified into an on-line algorithm, which is more suitable in our use case. More details will be given in the following sections.

4. General LSPI implementation advice

The theory behind LSPI learning engines have been described in several papers. However, there is far less information about how to utilize the LSPI formulas in a practical network optimization problem. As such, based on our experiences, we first propose a number of general guidelines and optimizations for implementing LSPI based decision engines. Afterwards, Section 5 describes in detail how these suggestions were applied to implement our own LSPI engine.

These guidelines aim to answer the following questions:

- How to define system states?
- How to define actions?
- How to define the basis functions?
- How to collect measurements?
- How to calculate the Q-values?
- How to calculate the rewards?

4.1. Defining system states

The first step is to identify what a system state represents. For example, a state can be a combination of currently active symbiotic services in the different networks (see Section 5) or, when using LSPI for the purpose of learning the optimal routing paths [21], the ID of the node where the data packet is currently located. The final number of states depends on the diversity of the properties that define them and should be finite.

Duration of the learning process directly depends on the number of states. Reducing their number will speed up the process. To do this, some states can be discarded a priori by the system architect. In the first example, this relates to a situation where a combination of network services is not allowed. In the case of routing, a forbidden state would be a node that is not used for routing (for example a resource-constrained device or a sniffing device).

The number of states can be reduced during the learning process as well, for example when the performance of a system is way bellow acceptable for a given state. This will strongly depend on the network requirements and metrics that are used to evaluate system's performance. For example, when routing, a forbidden state can be a node that has been identified as malicious or defected.

4.2. Defining actions

Any change of a network property is considered as an action. Consequently, the number of available actions at each state will depend on the number of properties that can be modified, the number of distinct values that can be assigned to them and possible constraints defined by the system architect. An illustrative example of a constraint would be the case when a certain property has a margin above which it cannot be change in two consequent steps(e.g. node's duty cycle cannot be change for more than 20

As imposed by the reinforcement learning paradigm, actions with the highest Q values at a particular state should be taken at each step. Of course, the methodology cannot be used at all times, since the state/action space needs to be *explored* first, in order to be *exploited*. For this purpose, we utilize the above mentioned ϵ *greedy* algorithm [23], which will be explained in detail in the following sections.

A good example of a possible alternative approach would be the Softmax algorithm [19], which uses Boltzmann distribution to define action-selection probabilities:

$$p(s,a) = e^{Q[s,a]/t} / (\sum_{a} e^{Q[s,a]/\tau}) \tag{12}$$

where factor $\tau > 0$ specifies how randomly values should be chosen. High values for τ, mean that the actions will be chosen almost uniformly. As it is reduced, the highest-valued actions are more likely to be chosen and, in the limit as $\tau \to 0$, the best action is always chosen.

From a computational complexity point of view, it is desirable to construct a non-ambiguous system, where $P(s'|s,a) = 1$ for every (s,a,s') tuple. In other words, an action taken at each state can lead to one and only one state.

4.3. Defining basis functions

Basis functions reflect changes in all the relevant network properties during a single learning episode. They are crucial in the process of calculating Q values for each state/action combination. Between two sets, the one that produces stronger response to the same change in network parameters is considered to be the more effective one. It produces a larger difference between Q values, thus making it easier to enforce certain decision making paths.

However, basis functions must be linearly independent to prevent any overlap when calculating the rewards. Increasing the number of basis functions improves the accuracy with which the network performance is modeled, and might be necessary when multiple requirements should be taken into consideration. However, the collection of this additional information typically also introduces additional overhead. As such, as the number of monitored properties of the network increases, a cost-effectiveness analysis in regards to the additional collection overhead is advised.

It is worth noticing that sudden changes in basis functions can give an indication of a sudden change of network conditions. These can trigger alerts to the network administrator and, can require the re-initiation of a learning process.

4.4. Initializing the data

Initially, often no information is available about the performance of each state. As such, the process must be initialized by collecting learning samples, either using a random approach or by going through each (s, a) pair sequentially. It makes sense, if collection is not being done off-line, to make sure that each (s, a) pair is not examined more than once during the process.

During the initialization phase, it is also possible to 'predict' suitable values for a number of (s, a) pairs. The main idea is to try to predict the performance of a system in a certain state by observing results achieved in the neighboring ('similar') states. Of course, this requires the definition of metrics that define which states are "neighbors". Doing this, the duration of the initialization phase can be made significantly shorter, at the cost of initial accuracy.

4.5. Collecting samples

Data samples are collected to calculate the basis functions. However, the process of collecting data might interfere with the network performance, thus giving a skewed vision on the actual performance. As such, it is beneficial to use basis functions that utilize non-intrusive data collection methods that do not interfere with the actual network operation.

4.6. Calculating rewards

Rewards are assigned to reinforce specific state-action pairs. Rewards can be positive or negative. Choosing and defining the rewards can be challenging in some cases, but strongly influences the outcome of the LSPI engine. The most cost-effective way to calculate a reward is to use information gathered for the purpose of calculating values for the basis functions. The simplest way is to simply use combined (weighted) values of basis functions.

Specifically for network optimization problems, a straightforward way of defining the reward function is the difference between the required and the current network performance, which requires an exact definition of the network requirements (incentives), such as delay constraints, audio quality scores (MOS scores), etc. Typically, low-level network metrics (e.g. average

packet loss, number of re-transmissions, etc) are combined into high-level metrics (e.g. end-to-end reliability).

When multiple requirements should be fulfilled, the reward function will typically be a function consisting of multiple (high-level) network metrics, such as end-to-end delay and end-to-end reliability metrics. In this case, it can be useful to enforce an upper limit to the contributions of each network metric. Otherwise, a system that significantly 'overshoots' one requirement but fails to fulfill other requirements can receive a higher reward than one that correctly fulfills all requirements. To prevent this, the associated reward of each network property can be capped, or can be set to increase very slowly once the requirement is met. If the requirements do not describe an upper performance limit, rewards can be unlimited.

4.7. Convergence criteria

Convergence criteria represent a policy iteration stopping rule. The general LSPI's stopping condition is the difference between values of weight factors in two consecutive iterations [5]. Ultimately, the stopping rule yields an optimal policy regarding the problem that is being solved, given the requirements.

How to define a stopping rule in the case when both settings of the problem and requirements are changing? Perhaps a better question would be: should it be defined? In a case of a dynamic network, where the continuous monitoring of the relevant network parameters is needed, the answer is negative. This way we provide versatility to fast changing parameters in unstable environments, while still enforcing the optimal (near-to-optimal) decision making paths.

4.8. Epsilon greedy effect

The epsilon greedy approach is commonly used during the process of state/action space exploration. It enforces sporadic "jumps" to sub-optimal states for the exploration purposes, but also to detect eventual changes of environmental conditions. Whenever a decision is to be made, the one will be picked at random with the ϵ probability [23]. Probability of 1- ϵ is given to the action with the highest Q-value. To avoid time and resource wasting by investigating sub-optimal states, the ϵ greedy algorithm can be made more intelligent. Some examples are the following:

- Rather than selecting a new state at random, constraints can be imposed on which states should be explored first. For example, the engine

can be forced to explore only states that are close to the optimal state. This approach can be generalized by adapting the epsilon greedy algorithm to select with a higher probability those states with high Q-values which are thus likely also quite good.

- Alternatively, the engine can choose not to use epsilon greedy when not necessary. For example, failure to meet the network requirements can trigger the activation of the algorithm.

5. Realizing an LSPI based decision algorithm

The above concepts have been applied to realize a cognitive engine for deciding upon the optimal combination of services in a network, as described in Section 1. Our cognitive engine does not iterate different decision making policies. Instead, it continuously gathers samples from the environment and updates the sample matrices, thus re-initiating LSTDQ on the same policy each time a new sample is collected. This approach does not pose a significant difference to the original LSPI approach, since a decision policy is fully abstracted by its sample set and its basis function values. One policy is used throughout the whole process, but the collected sample set and basis function values that are sent to the LSTDQ are constantly changing. This can be considered as a policy switch and our system is improved in its versatility to notice sudden condition changes in the network environment and modify its current policy accordingly. This will be elaborated in details further on in the paper.

For the proof-of-concept, we reused measurements gathered during the run of a demonstration set-up, previously described in [2]. The set-up consisted of two interleaved, wireless sensor networks, used for temperature monitoring and security purposes, respectively (see Figure 2). Both networks are consist of ultra low power TMoteSky sensor nodes, equipped with 8MHz MSP430 microcontroller, with 10k RAM, 48k flash memory and a 250kbps 2.4GHz Chipcon wireless transceiver. Both networks could activate/deactivate "packet sharing" and/or "packet aggregation" services. The main objective was to find the optimal combination of these services, depending on the set of requirement s (possibly differently prioritized) in each of the sub-nets.

In the original setup, a linear programming based engine, ILPSolver, was used to initiate and control the process of network service negotiation. Since

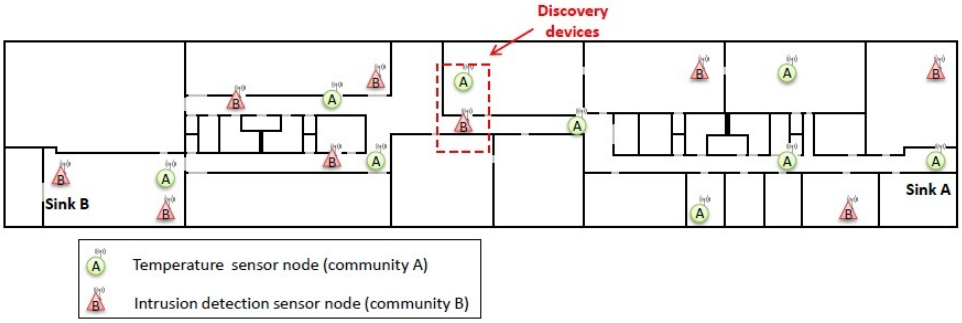

Figure 2: Experimental setup - iLab.t wireless sensor testbed, located at iMinds - Ghent University office building, Belgium. Two networks are interleaved - network A collect temperature readings from the environment while network B is used for the intrusion detection

this engine required a priori knowledge about the influence of different network services, different combinations of services in both networks were tested. Information regarding relevant aspects of networking has been gathered during each testing episode. In the set-up that we used for the proof-of-concept, the most relevant data is the average delay and the number of packet retransmissions during these episodes. Table 1 shows the service combinations that were examined during the testing, while relevant measurements are given on figures 3 and 4.

The rest of this section focuses on testing and verifying the operation capabilities of the new negotiation engine with an emphasis on three main objectives:

- Conduct a cognitive process without an a priori knowledge required by the engine described in [2] and compare the results
- Use decisive values to set up goals regarding each network requirement, other than using major guidelines, described by network requirements
- Enable responsiveness to a change of the network conditions

5.1. Defining possible system states

The state of each particular sub-net is defined as a combination of its both currently activated (see Table 2).

Episode	Network A	Network B
1	Aggregation = OFF Packet Sharing = OFF	Aggregation = OFF Packet Sharing = OFF
2	Aggregation = ON Packet Sharing = OFF	Aggregation = ON Packet Sharing = OFF
3	Aggregation = OFF Packet Sharing = ON	Aggregation = OFF Packet Sharing = ON
4	Aggregation = ON Packet Sharing = ON	Aggregation = ON Packet Sharing = ON

Table 1: A reduced set of four system states, examined during the experimental run. Each state is represented as a combination of active and inactive services in both sub-nets during a particular learning episode

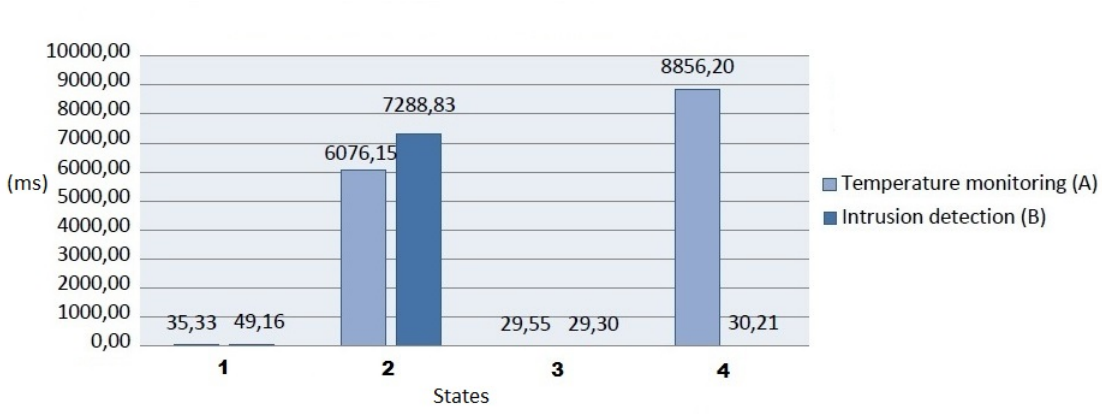

Figure 3: Average end to end delay measured in networks A and B during learning episodes in which the system resided in all 4 defined states: (1) packet sharing and aggregation deactivated in both networks, (2) only packet sharing active in both sub-nets, (3) only aggregation active in both sub-nets, (4) packet sharing active in in both sub-nets and aggregation active only in the sub-net B

The final outcome of the algorithm, in the form of a list of activated and non-activated services, generally differs from one participating sub-net to another. The performance of a certain service set in one network strongly depends on service combination activated in the neighboring communities. Therefore, the states that LSPI engine can go through are defined by all the possible combinations of all the available services in the participating sub-

Figure 4: Average number of transmissions per node measured in networks A and B during learning episodes in which the system resided in all 4 defined states: (1) packet sharing and aggregation deactivated in both networks, (2) only packet sharing active in both sub-nets, (3) only aggregation active in both sub-nets, (4) packet sharing active in in both sub-nets and aggregation active only in the sub-net B

State	Service - AGGREGATION	Service - PACKET SHARING
0	OFF	OFF
1	OFF	ON
2	ON	OFF
3	ON	ON

Table 2: An example how a system state of a single sub-net can be defined as the combination of both active and inactive networks services during a particular learning episode

nets. For example, measurements regarding an average end-to-end delay in network A can vary, for the same combination of active services, depending on whether packet sharing service is enabled in the network B or not.

In our research, we focus on discovering the optimal service set for the temperature monitoring sub-net (Fig.2, network A), while taking into consideration the influence of a co-located security network (Fig.2, network B). Considering there are two services each network can provide, packet aggregation (AGGR) and packet sharing (PS), the LSPI engine can go through 16 different states in total. Each service combination activated in a temperature monitoring network constitutes a distinct state with every combination of services of the security network.

In order to unambiguously meet the conditions set up in [2] and to be able

to make a cross-comparison of the two approaches, we examine a reduced set of four states, given in Table 1. Experimental values, regarding each defined state, are originally gathered during the initial testing and reused for the purposes of this paper. Service combinations (states) that were not examined during the initial experiment are considered to be irrelevant and therefore are not examined in this paper. From now on, whenever the term *state* is used, it will refer to a combination of services set up in network A (Table 2), accompanied with a set of services in network B, as defined in Table 1.

5.2. Defining actions

In each of the four defined states (0, 1, 2, 3), there are four available actions. A system can either decide to stay in a current state or move to any of the remaining three, depending on the current policy (see Figure 5)

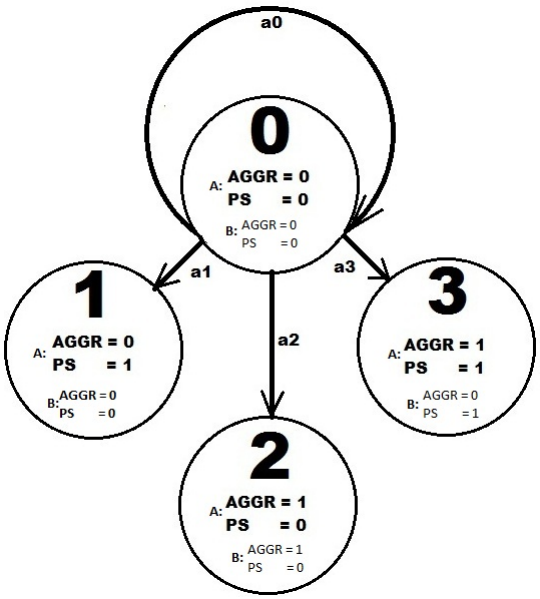

Figure 5: Graphical representation of the reduced set of four distinct system states (see Table 1). System can switch between any two states, depending on the action that is taken. As an illustration, when the system is in state 0, taking any of the actions (a0,a1,a2,a3) will lead it to states 0,1,2,3 respectively

In our use case, after reaching the optimal policy, the maximal Q value will be awarded to a certain state and action that keeps the system in the

same state in each following episode. Whether a policy should be changed or not after this point, will be discussed in one of the following sections.

5.3. Defining basis functions

Two network requirements regarding the high reliability and high network lifetime, are given the same priority. Two basis functions are defined ϕ_1 - average network delay (AND) and ϕ_2 - predicted network lifetime (PNF). The values of these two basis functions are used at the end of each episode to evaluate how well the network performance fulfills the current network requirements

For calculating ϕ_1, the correct data is already available (see Figure 3): the average end-to-end reliability data can be used in a form it is retrieved from the network. However, the experimental setup has no available information about the average energy consumption, needed for ϕ_2. Instead, only the average number of packet transmissions is monitored (see Figure4). To transform the available data into (an approximation of) the second basis function, a number of calculations are performed.

$$\begin{gathered} \mathrm{E}_{trans} = \frac{P_{trans} PacketSize}{TransmissionRate} \\ \mathrm{E}_{recept} = \frac{P_{recept} PacketSize}{TransmissionRate} \\ \mathrm{E}_{perEpisode} = N_{trans} E_{trans} + N_{degree} N_{trans} E_{recept} \end{gathered} \tag{13}$$

Variables E_{trans} and E_{recept} are average amounts of energy spent per single byte transmission and reception, respectively. Similarly, $P_{trans} and P_{recept}$ represent power used for transmitting and receiving a packet. These values are stated in the data-sheet of the user CC2420 radio. Packet size is set to PacketSize = 51 Bytes and a standard CC2420 radio transmission rate of $TransmissionRate = 256 Kbps$ is used. N_{trans} is the average number of transmissions per node, as collected from the network. Since the wireless medium is a broadcast medium, it is assumed that every transmission triggers $NodeDegree$ receptions, where $NodeDegree$ is the average number of neighbors per node. In our case $NodeDegree = 2$.

As mentioned in the previous section, Q-values of each state/action pair are calculated as linear combinations of the basis functions and their respective weights. A pair with a higher Q-value is considered to be "better" than the one with the lower value. In the case of ϕ_1, higher delay must lower the Q-value. Similarly, the increase of the energy spent implies a lower network life time. Required basis functions are calculated in a following way:

$$\phi 1 = \frac{\frac{InitialEnergy}{E_{perEpisode} * T_{episode}}}{T_{goal}}$$
$$\phi_2 = \frac{D_{goal}}{AVG_{delay}} \tag{14}$$

Here, $InitialEnergy$ per node in the network is taken to be $0.5J$. Variable $T_{episode}$ is the duration of a learning episode. The optimal value of the learning episodes will be discussed in detail in the following section. Finally, T_{goal} and D_{goal} represent desired values for both network lifetime and average packet delay. Calculating basis functions this way will ensure that higher Q-values are given to states/action pairs that provide lower network delay and lower energy spent per node in a learning episode.

5.4. Collecting samples

The consistency between true and approximated values of matrices A and b increases with the increase of number of samples (s, a, s', r) (see Section 3.2). Here, s represents a current state, a is the action taken at state s, s' is the next state and r is the immediate reward given for this particular transition. After visiting each state at least once, samples for every state-to-state transition are obtained. This claim is implied from the fact that system's performance at a certain state does not depend on the state it has previously been to.

Since we assume that no a priori information is available, our learning process starts at a random state. During sample collection, state transitions are done in a purely random fashion. In order to minimize the duration of the process, it is forbidden to investigate previously observed states until all the states are examined. This way, the sample collecting time is limited to:

$$\mathrm{t}_{sampling} = N_{states} T_{episode} \tag{15}$$

where N_{states} is the number of defined system states.

Samples collected using the initial, purely random policy, can be used to test future policies. As stated in [5], the same sample set can be used to test any decision policy. However, dealing with highly dynamic environments demands gathering fresh data as frequently as possible. Therefore, new samples are collected during each learning episode and appended to an existing set.

5.5. Calculating Q values

Using a policy that implements ϵ greedy algorithm poses the first step in policy iteration. The random policy, used during the sample collection phase, is considered to be a starting point. New policy operates in a way that an action at each state is picked at random with the ϵ probability. With $1 - \epsilon$ probability, an action with the highest Q-value is taken.

After each new episode, a new sample (s, a, s', r) is collected from the environment, meaning that both Φ and $\widehat{P^{\pi}\Phi}$ matrices are updated. LSTDQ is initiated and ω factors for each basis function are recalculated. According to a statement in the previous section, Q-values for transitions from all the states to a current one are updated.

$$\mathrm{Q(s,a)} = \omega_1 \phi_1 + \omega_2 \phi_2 \tag{16}$$

Behavior of a symbiotic network, in this case, does not suffer from a 'memory effect' (i.e. the performance of a state does not depend on which previous state was selected), after spending one episode in a state S_i, collected measurements can be used to calculate Q-values for transitions from any state (including S_i) to S_i, thus significantly speeding up the convergence time.

Randomness in a new policy will lead the system to a sub-optimal state. With a proper choice of the ϵ parameter, with a high probability system will return in the most optimal state at the next hop.

By constantly collecting new samples, the LSPI stays aware of the network's dynamics. Depending on certain factors, which will be discussed in later sections, a sudden change in network conditions will be noticed sooner or later.

5.6. Calculating rewards

Rewards given after each episode are calculated according to the difference in network's performance and user defined goals. These values are set for both the high network lifetime and average network delay requirements of the temperature measuring network.

The following reward formula is used:

$$\mathrm{r} = \sum_i C_{p_i} max R_i \tag{17}$$

where C_{p_i} is the (normalized) priority of a high level network goal and R_i is the individual reward given for the performance associated with a particular goal. In our case, individual rewards are equal to the values of the respective basis functions, ϕ_1 and ϕ_2. This is the simplest form of calculating rewards. Priorities are set to be equal, $C_{p_i} = 0.5$.

In the case when priorities are different, exceptionally good performance regarding a high priority requirement will have a significantly higher impact on the total reward, thus overshadowing a possible poor performance regarding other network requirements. Therefore, it makes sense to set up an upper limit to each individual reward.

6. Implementation and evaluation

This section evaluates the LSPI engine described in Section 5. The following aspects are discussed:

- Implementation of the LSPI decision engine
- Choosing the optimal epsilon value
- Dealing with dynamic networks
- Choosing the optimal weight factors
- Convergence and stopping rules

6.1. Implementation

The program runs on a dedicated server that is connected with sink nodes of each participating sub-net (see Fig. 2). During the learning process, these links are used to distribute new combinations of services (to be activated) towards the devices of the participating networks. To calculate the rewards, network measurements are periodically collected by the server. It is assumed that the communication between co-located sub-nets is already established and lists of available services are collected by the engine. Detailed descriptions of the configuration and data-gathering processes are given in [2] and are out of the scope of this paper. To evaluate the convergence speed and the ability to adopt to network condition changes, a set of pre-collected measurements (from [2]) is used as described in Section 5.

The cognitive engine is implemented in Java, but could equally well be implemented in any other programming language. It compiles to 92 KBytes, with no more than 3.7MBytes of heap memory usage during run-time. It can easily be optimized to run also on embedded devices. Processing the measurements and selecting the next optimal state currently requires less than a millisecond on a 2.4GHz processor.

6.2. Choosing the optimal ϵ value

As described in Section 4.8, the epsilon greedy mechanism enforces state transitions to enforce (i) the exploration of unknown states and (ii) detect network changes. The following charts present results collected during the experiments, in respect to different values of the discovery factor - ϵ (ϵ greedy algorithm). Q values, describing a transition to a designated state are given

on the y-axis (marked with green, blue, red or purple). The number of learning episodes that have passed are depicted on the x-axis.

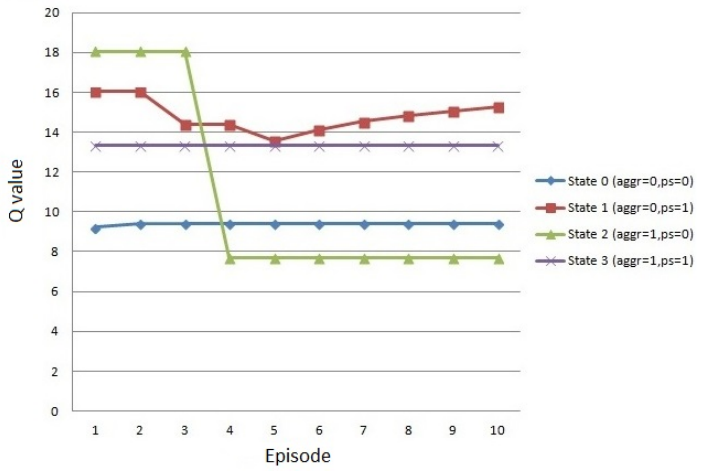

Figure 6: Behavior of the state value functions over 10 learning episodes, with the discovery factor ϵ set to 0.1. System performance in a particular state is not affected by the state the system was previously in. Therefore, state value function describes a transition from any given state to the one it is assigned for

First four episodes, utilizing a random walk to gather samples for all the possible state transitions, are not depicted on graphs. The ones that are shown start with the initiation of the ϵ greedy algorithm. In the case when $\epsilon = 0.4$ the highest Q-value is given to transitions to state 3, implying that the optimal conditions for the temperature measuring network is to keep both aggregation and packet sharing services active. At the same time, security network needs to keep aggregation switched on. These results correspond with the results presented in [2], where state 3 was determined to be optimal.

It is noticeable that calculated Q values,for the same state/action combinations, differ slightly from case to case, which is a consequence of the state investigation methodology. This issue will be addressed in more details in the following section. However, the general conclusion, after observing all the graphs, is that states 1 and 3 produce similar performances, significantly higher in comparison to states 0 and 2. State 2, in particular, is marked as the worst performing one, witch once again matches the outcome of the experiments presented in [2].

In cases when high *epsilon* values are used, the decision making policy

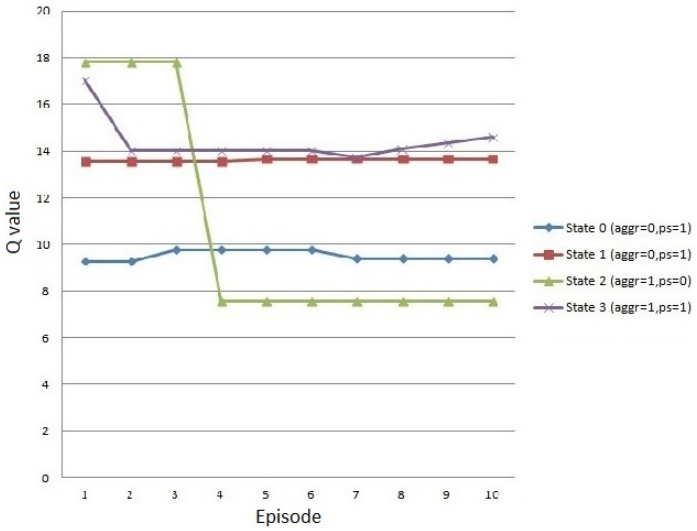

Figure 7: Behavior of the state value functions over 10 learning episodes, with the discovery factor ϵ set to 0.4

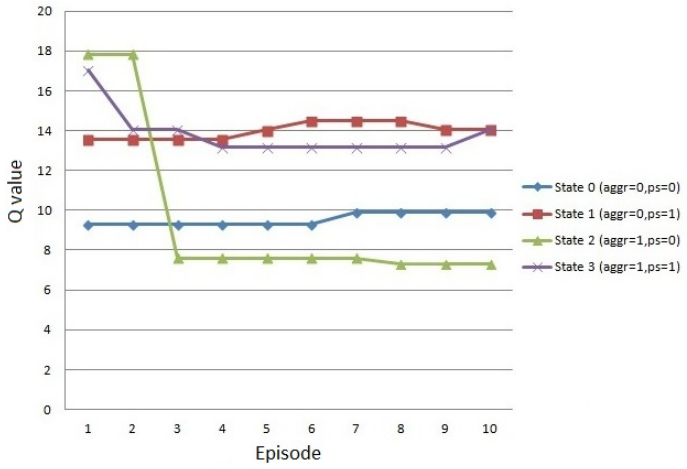

Figure 8: Behavior of the state value functions over 10 learning episodes, with the discovery factor ϵ set to 0.7

can be considered as a near-random one. This is notable in Table 3 for the cases where the ϵ values are set to 0.9 and 0.7. On the other hand, setting the *epsilon* to a low value, such as $\epsilon = 0.1$, makes the process highly dependent

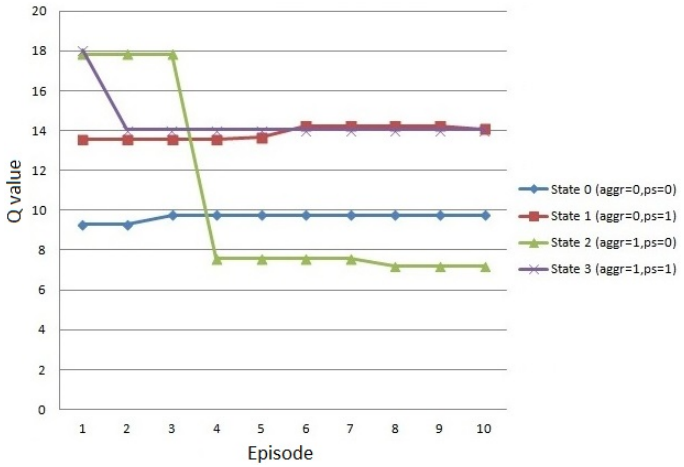

Figure 9: Behavior of the state value functions over 10 learning episodes, with the discovery factor ϵ set to 0.9

on the initial (random) decisions, made during the first couple of episodes. This is demonstrated for $\epsilon = 0.1$, where state 1 was kept on throughout the rest of the experiment. Such a behavior could lead to an enforcement of sub-optimal states, since the optimal ones will get a small chance to be examined, which can ultimately result in a very long convergence time.

States	0	1	2	3
$\epsilon = 0.1$	10%	80%	10%	0%
$\epsilon = 0.4$	20%	10%	10%	60%
$\epsilon = 0.7$	20%	30%	20%	30%
$\epsilon = 0.9$	20%	40%	30%	10%

Table 3: Percentage of the number of episodes that system have spent in each state during a particular experimental run. Each run is characterized by a different value of the discovery factor - ϵ = (0.9, 0.7, 0.4, 0.1)

Since performances in states 1 and 3 are quite similar (see Figures 6, 7, 8, 9), as long as the system resides in any of the two states, its performance can be considered optimal. Therefore, percentages that correspond to these states (see Table 3) can be joined into a single value. We conclude the following:

- In cases when ϵ was set to 0.1 and 0.4, the ratio between the number of episodes spent in optimal and sub-optimal states was 4:1, meaning that in 4 out of 5 episodes, our system will reside in one of the two optimal states.

- In cases when ϵ was set to 0.7 and 0.9, the ration decreases to 3:2 and 1:1, respectively. This behavior is expected since the decision policy is almost random.

For practical reasons, the aforementioned conclusions are made after examining a relatively small number of episodes. It is safe to claim that, with the increase in the number of learning episodes, statistics will change in favor of the optimal states.

6.3. Weight factor recalculation, convergence and stopping rules

The main purpose of recalculating weight coefficients is to precisely determine the influence of each basis function on the respective Q values. It is done repeatedly after each learning episode.

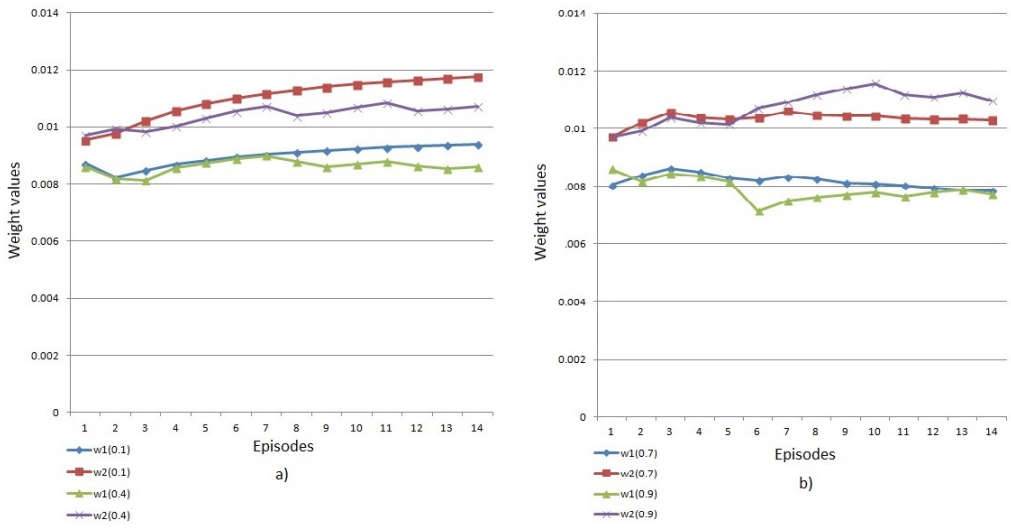

Figure 10: Behavior of the weight factor values during an experimental run. Figure 10.a depicts cases where ϵ is set to 0.1 and 0.4. Figure 10.b depicts cases where is set to 0.7 and 0.9

As the decision making policy is being shaped, weight factors are expected to monotonically converge to specific values. This is illustrated in Figure 10.a and Figure 10.b. The difference between consecutive values of weight factors is given on Figures 11.a and **??**.b.

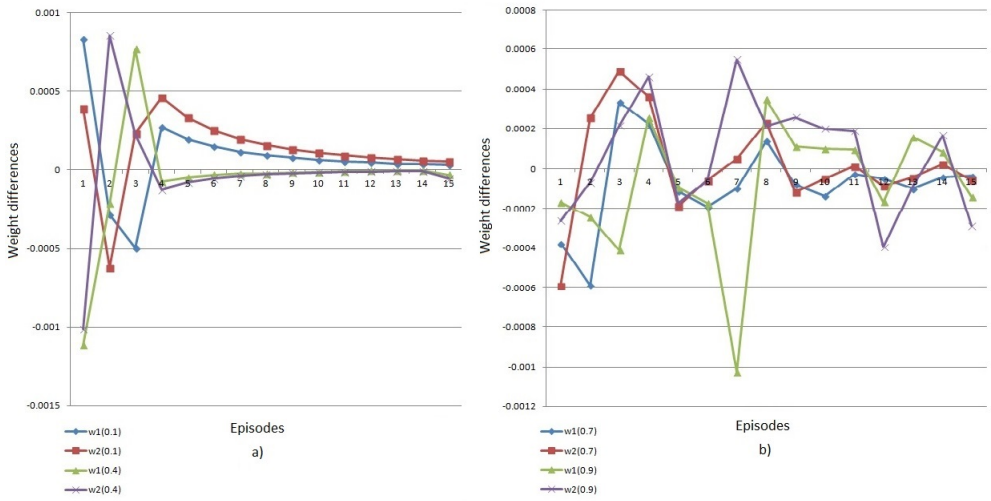

Figure 11: Differences between weight factor values in consecutive episodes (generally used as the algorithm's stopping criteria). Experiments are run with ϵ set to 0.1 and 0.4 (Figure 11.a), while Figure 11.b depicts cases where ϵ is set to 0.7 and 0.9

With higher ϵ values (Figure 10.b), less optimal system states are visited more frequently, which produces fluctuation in basis function values, ultimately causing noticeable oscillations of the weight factors (see Figure 11.b.

Results of using different values for the general stopping criteria, are presented in Table 4 and Table 5. As presented, the difference $||\omega' - \omega||$ for both basis functions, is set to 0.0002 and 0.0004, in two consecutive experiment runs. The final outcome of each of the experiments remains the same. However, the time (number of episodes) needed to obtain results is shorter for the second experiment. Further increase of the stopping factor does not produce meaningful results. On a contrary, it prevents the initiation of the algorithm since the stopping condition is immediately met.

Choosing the an appropriate exploring factor is a case specific problem. Generally, it demands balancing between the effects it has on processes of establishing/keeping the system in the optimal state and adjusting to possible

Exploration factor	0.1	0.4	0.7	0.9
stopping condition	$\lvert\omega_1' - \omega_1\rvert < 0.0002$ $\lvert\omega_2' - \omega_2\rvert < 0.0002$	$\lvert\omega_1' - \omega_1\rvert < 0.0002$ $\lvert\omega_2' - \omega_2\rvert < 0.0002$	$\lvert\omega_1' - \omega_1\rvert < 0.0002$ $\lvert\omega_2' - \omega_2\rvert < 0.0002$	$\lvert\omega_1' - \omega_1\rvert < 0.0002$ $\lvert\omega_2' - \omega_2\rvert < 0.0002$
Terminal episode	7	10	7	12
Optimal state	1	3	1	1

Table 4: Number of episodes needed for the algorithm to meet a predefined stopping condition set to 0.0002. The same condition is set for both weight factors

Exploration factor	0.1	0.4	0.7	0.9
Stopping rule	$\lvert\omega_1' - \omega_1\rvert < 0.0004$ $\lvert\omega_2' - \omega_2\rvert < 0.0004$	$\lvert\omega_1' - \omega_1\rvert < 0.0004$ $\lvert\omega_2' - \omega_2\rvert < 0.0004$	$\lvert\omega_1' - \omega_1\rvert < 0.0004$ $\lvert\omega_2' - \omega_2\rvert < 0.0004$	$\lvert\omega_1' - \omega_1\rvert < 0.0004$ $\lvert\omega_2' - \omega_2\rvert < 0.0004$
Terminal episode	7	8	5	10
Optimal state	1	3	1	1

Table 5: Number of episodes needed for the algorithm to meet a predefined stopping condition set to 0.0004 for the both weight factors

environmental changes. One solution might even be to dynamically change its values over time. This possibility is considered as a part of the future improvements and will be elaborated in details in the future research.

6.4. Dealing with dynamic networks

In theory, once the optimal state-action pairs have been found, the LSPI learning engine can be turned off, since the basis functions will not change anymore. However, in dynamic networks with changing network conditions change, the values of the features (basis functions) change as well. With the ϵ greedy switched off, LSPI engine will not be able to detect any of these changes, thus it will preserve the whole system in a possibly sub-optimal state.

One solution would be to lower down the ϵ value during the learning process.

To illustrate this, an experiment was performed showing the LSPI behavior for different epsilon values when the worst state suddenly switches place with the optimal one after the 10th episode (Figures 12, 13, 14, 15) . Although not impossible, this extreme scenario is used strictly for the purpose of demonstration and not considered to be highly probable.

The time (number of episodes) necessary to detect network conditions increases for lower ϵ values. The quickest response is noticed in the case when $\epsilon = 0.9$ (5 episodes). The trade-off for using such high values is that sub-optimal states will be selected periodically even when no changes are

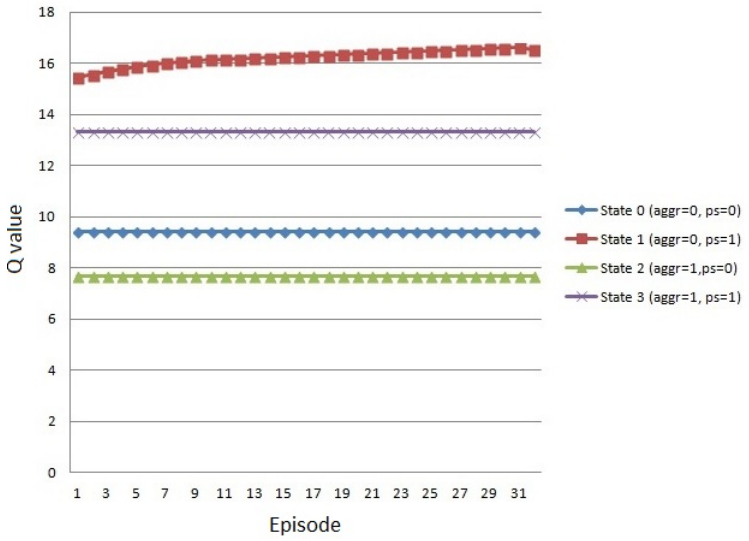

Figure 12: Illustration of the system's capability to notice sudden and drastic network changes. The system is tested in the "worst case" scenario, with the ϵ factor set to 0.1

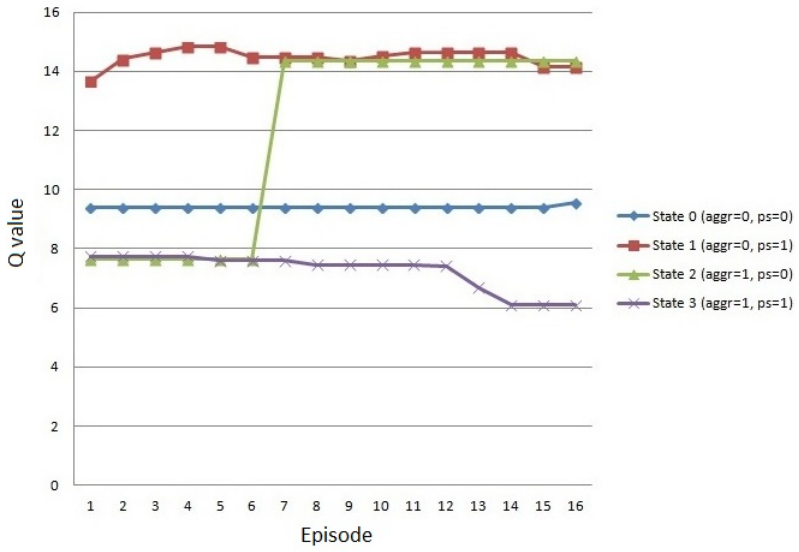

Figure 13: Illustration of the system's capability to notice sudden and drastic network changes. The system is tested in the "worst case" scenario, with the ϵ factor set to 0.4

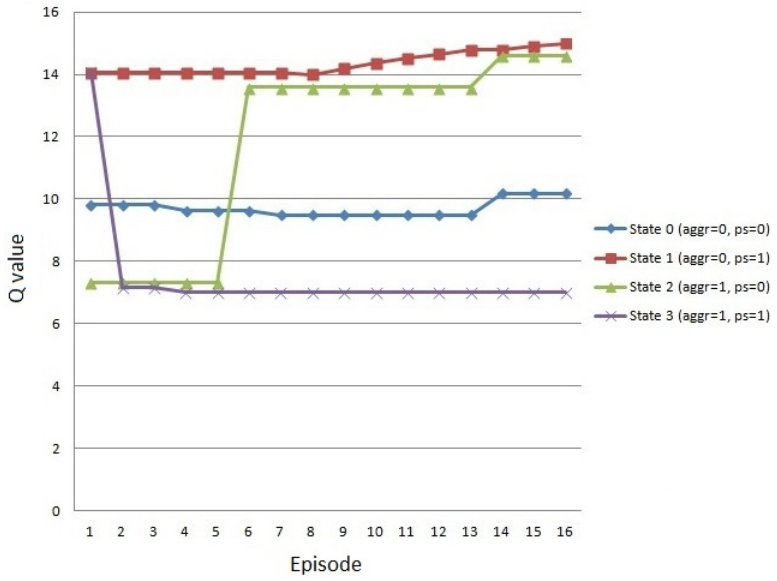

Figure 14: Illustration of the system's capability to notice sudden and drastic network changes. The system is tested in the "worst case" scenario, with the ϵ factor set to 0.7

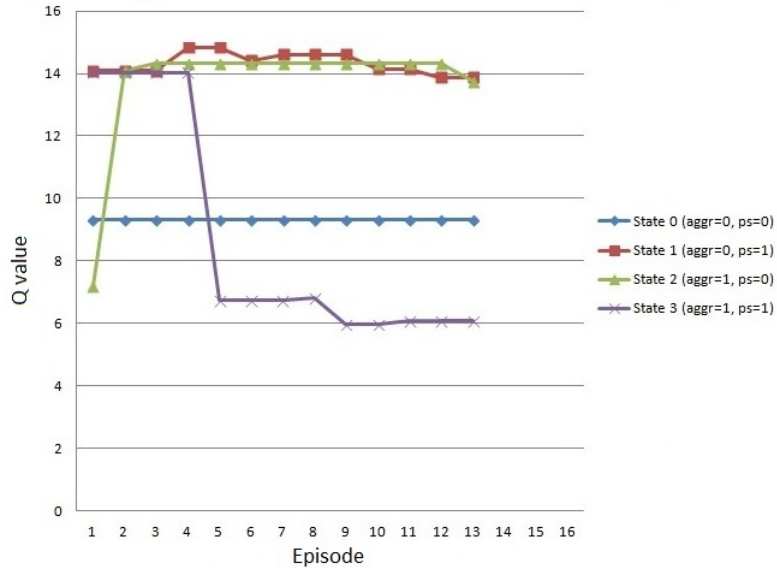

Figure 15: Illustration of the system's capability to notice sudden and drastic network changes. The system is tested in the "worst case" scenario, with the ϵ factor set to 0.9

detected. Particularly interesting result is depicted on Figure 12, where no changes are detected. There has to be a compromise regarding the ϵ values used during the learning process and after the optimal state has been determined. Being case specific, no general guidelines can be given regarding this matter. The following Table 6 summarizes the results presented in this section:

States	2	3	Total
$\epsilon = 0.1$	Not detected	Not detected	Not detected
$\epsilon = 0.4$	7	1	1
$\epsilon = 0.7$	6	2	6
$\epsilon = 0.9$	2	5	5

Table 6: Number of episodes needed for network condition changes to be notified. Performances in states 2 and 3 have been switched. Numbers of episodes needed by the engine to notice each change are given in the respective columns.

7. Future work

The results above demonstrate that the use of an LSPI based cognitive engine (that requires no a priori knowledge about the network) can both learn the best network configuration and adapt to changing network conditions. Future work will extend the current use case with additional states and further investigate the influence of the ϵ parameter to both duration of a learning process and responsiveness of the engine to a change of network conditions. Choosing an optimal ϵ value will pose a crucial challenge, since the duration of the learning process directly depends on it. Different stopping criterion might be necessary. Instead of comparing consecutive ω values, other metrics for evaluating the optimality of a system state can be designed. Once their values are in the proper vicinity of a user defined performance goals, the learning process will be considered complete. Finally, new mechanisms of calculating rewards might be needed, to more precisely evaluate system accomplishments during each episode.

There are also plans to expand the application of the LSPI engine. The engine will be applied both for:

- Optimizing a performance of a single network protocol

- Optimizing a negotiation process between multiple networks and coordinating the cooperation

It is reasonable to believe that, as the time goes on, due to the wide variety of wireless network conditions, network protocols will become increasingly adaptive and configurable. The main goal of optimizing a single protocol would be to determine the optimal set of protocol settings. Combinations of settings represent system states. Large number of settings will lead to a large number of states, of course. However, the learning process can be extended by implementing deduction techniques that enable the LSPI engine to recognize certain behavioral patterns. The predictions, for example, can be used to detect sub-optimal settings combinations, before inspecting them, which would speed up the learning process significantly.

To optimize multiple co-existing networks, some compromises will be inevitable. Metrics that will precisely describe whether certain compromise is justified or not, from each sub-net's and the entire network's point of view, will have to be designed.

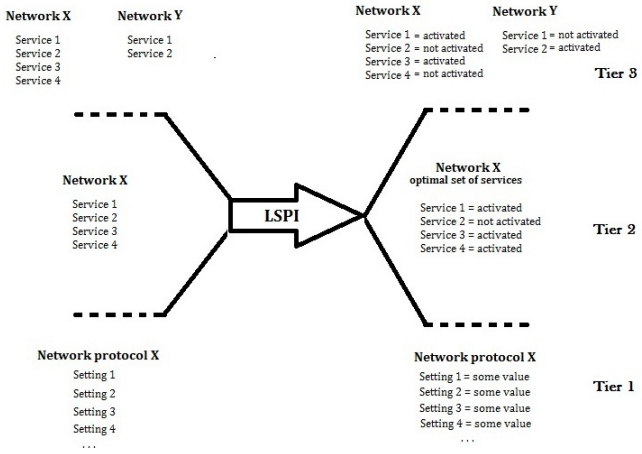

Figure 16: Scalability of the LSPI algorithm - a) optimization of a single network protocol by tuning up its settings b) optimization of a homogeneous network through defining an optimal set of services that needs to be activated c) optimization of a symbiotic (heterogeneous) network through the process of service set negotiation

As an alternative, in fully-configurable networks (with multiple configurable network services and multiple configurable protocols), we propose to

reduce the optimization complexity by using a tiered system of network optimization (see Fig. 16).

1. In tier 1, LSPI is used to first optimize the parameters of each network protocol of the network.
2. Next in tier 2, LSPI is used to identify a number of service sets that offer acceptable network performance.
3. Finally, in tier 3, LSPI selects amongst the acceptable service sets those that are also beneficial for co-located networks.

Each tier is essentially based on the same - LSPI learning concept.

8. Conclusion

Optimizing multiple co-located networks, each with a variable number of network functionalities that all influence each other, is a complex problem that will become increasingly relevant in the future. In this paper, we propose to use a reinforcement learning technique, LSPI, for the purpose of selection and composition of the high-level networking optimization in heterogeneous networks. It main advantages, compared to previous work, is that the methodology (i) does not require a priori knowledge or configuration and (ii) is able to cope with changing network dynamics. The critical parameters of the mechanisms have been detected, their effects examined and elaborated in details. Future plans, that include several improvements of the existing implementation, as well as its extensions to other areas of interest, are also presented. Most of the proposed improvements aim at increasing the speed of exploration of the parameter space, thus decreasing learning cycle duration.

An implementation of the cognitive engine is described, and the outcome results are compared to previous results obtained using a linear programming based reasoning engine. It is shown that more detailed results can be obtained with less pre-deployment requirements. The new engine also demonstrates high versatility in dynamic environments, which was not the case with the previous solution.

We strongly believe that the problem of interfering co-located networks will only increase. As such, innovative cross-layer and cross-network solutions that take these interactions into account will be of a great importance to the successful development of efficient next-generation networks in heterogeneous environments.

Acknowledgment

This research is funded by the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWTVlaanderen) through the IWT SymbioNets project, by iMinds through the QoCON project and by the FWO-Flanders through a FWO post-doctoral research grant for Eli De Poorter

9. References

[1] http://www-01.ibm.com/software/commerce/optimization/linear-programming/

[2] E. De Poorter, P. Becue, M. Rovcanin, I. Moerman and P. Demeester, "A negotiation-based networking methodology to enable cooperation across heterogeneous co-located networks", Ad Hoc Networks, Available online 8 December 2011, ISSN 1570-8705, 10.1016/j.adhoc.2011.11.007.

[3] T. G. Dietterich, and O. Langley, (2007) "Machine Learning for Cognitive Networks:Technology Assessment and Research Challenges in Cognitive Networks: Towards Self Aware Networks", John Wiley and Sons, Ltd, Chichester,UK. doi: 10.1002/9780470515143.ch5

[4] A. Forster, "Machine Learning Techniques Applied to Wireless Ad-Hoc Networks: Guide and Survey", 3rd International Conference on Intelligent Sensors Sensor Networks and Information (ISSNIP), Melbourne, Australia, 3-6 Dec. 2007.

[5] M. Lagoudakis and R. Parr, "Model-free least-squares policy iteration". In Proc. of NIPS, 2001.

[6] Michail G. Lagoudakis and Ronald Parr, "Least-Squares Policy Iteration, Journal of Machine Learning Research, 4, 2003, pp. 1107-1149.

[7] N. Vucevic, I.F. Akyildiz, J. Perez-Romero, "Cooperation Reliability Based on Reinforcement Learning for Cognitive Radio Networks", Fifth IEEE Workshop on Networking Technologies for Software Defined Radio (SDR) Networks, 21st of June 2010, Boston, MA, USA

[8] M. Yang and D. Grace, "Cognitive Radio with Reinforcement Learning Applied to Multicast Downlink Transmission with Power Adjustment," Wireless Personal Communications, vol. 57, pp. 73-87, 2011.

[9] H. Li, "Multi-agent Q-learning of channel selection in multi-user cognitive radio systems: A two by two case," in Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on, 2009, pp. 1893-1898.

[10] A. Galindo-Serrano, L. Giupponi, "Decentralized Q-Learning for Aggregated Interference Control in Completely and Partially Observable Cognitive Radio Networks", Consumer Communications and Networking Conference (CCNC), 7th Annual IEEE Conference , 9-12 Jan. 2010, Las Vegas, Nevada, USA

[11] S. Shankar and C. Cordeiro, Analysis of aggregated interference at DTV receivers in TV bands, in Proc. Int. Conf. CROWNCOM, Singapore, May 1517, 2008, pp. 16.

[12] Minsoo Lee, Dan Marconett, Xiaohui Ye, S. Yoo ,"Cognitive Network Management with Reinforcement Learning for Wireless Mesh Networks", IP Operations and Management (2007), pp. 168-179, doi:10.1007/978-3-540-75853-$2_1$5

[13] Y. Wu, P. Chou, Q. Zhang, K. Jain, W. Zhu, S.-Y. Kung, Network planning in wireless ad hoc networks: a cross-layer approach, Selected Areas in Communications, IEEE Journal on 23 (1) (2005) 136 150. doi:10.1109/JSAC.2004.837362(410) 23.

[14] Y. Wu, P. Chou, Q. Zhang, K. Jain, W. Zhu, S.-Y. Kung, Network planning in wireless ad hoc networks: a cross-layer approach, IEEE Journal on Selected Areas in Communications Vol. 23 (2005) pp. 136 150.

[15] S. Haykin, Cognitive radio: brain-empowered wireless communications, Selected Areas in Communications, IEEE Journal on 23 (2) (2005) 201 220. doi:10.1109/JSAC.2004.839380

[16] F. K. Jondral, Software-de
ned radio - basics and evolution to cognitive radio, EURASIP Journal on Wireless Communications and Networking vol. 2005, no. 3 (2005) pp. 275283.

[17] R. W. Thomas, D. H. Friend, L. A. Dasilva, A. B. Mackenzie, Cognitive networks: adaptation and learning to achieve end-to-end performance objectives, Communications Magazine, IEEE 44 (12) (2006) 51 57. doi:10.1109/MCOM.2006.273099.

[18] R. W. Thomas, D. H. Friend, L. A. Dasilva, A. B. Mackenzie, Cognitive networks: adaptation and learning to achieve end-to-end performance objectives, Communications Magazine, IEEE 44 (12) (2006) 51 57. doi:10.1109/MCOM.2006.273099.

[19] M. Tokic and G. Palm, "Value-difference based exploration: Adaptive control between epsilon-greedy and Softmax", In J. Bach and S. Edelkamp, editors, KI 2011: Advances in Artificial Intelligence, volume 7006 of Lecture Notes in Artificial Intelligence, pages 335-346. Springer Berlin / Heidelberg, 2011

[20] Lucian Busoniu, Robert Babuska, Bart De Schutter, Damien Ernst, "Reinforcement Learning and Dynamic Programming Using Function Approximators", 2010 by Taylor and Francis Group, LLC, ISBN 978-1-4398-2108-4 (Hardback)

[21] P. Wang, T. Wang, "Adaptive Routing for Sensor Networks using Reinforcement Learning", CIT '06 Proceedings of the Sixth IEEE International Conference on Computer and Information Technology, October 22-25, 2006 Charlotte Convention Center Charlotte, NC

[22] Z. Ye and A. A. Abouzeid, "Layered Sequential Decision Policies for Cross-layer Design of Multihop Wireless Networks", Information Theory and Applications Workshop (ITA'10), San Diego, CA, Feb 2010.

[23] John Myles White, "Bandit Algorithms for Website Optimization", Published by O'Rilley Media, December 2012, Inc.

[24] L. P. Kaelblign, M. L. Littman, A. W.Moore,"Reinforcement learning: A Survey", Journal of Artificial Intelligence Research 4 (1996) 237-285

[25] R. W. Thomas, L. A. DaSilva and A. B. MacKenzie, "Cognitive networks", Proc. IEEE DySPAN 2005, pp.352-60

[26] Wakamiya, N.; Arakawa, S.; Murata, M.,"Self-Organization Based Network Architecture for New Generation Networks", 2009 First International Conference on Emerging Network Intelligence, pp.61-68, 11-16 Oct. 200

[27] M. Rovcanin, E. De Poorter, I. Moerman, P. Demeester, " An LSPI based reinforcement learning approach to enable network cooperation in cognitive wireless sensor network ", The 27th IEEE International Conference on Advanced Information Networking and Applications, March 24-28, 2013 Barcelona, Spain

[28] http://www.iminds.be/en/develop-test/ilab-t

[29] L. Tytgat, B. Jooris, P. De Mil, B. Latr, I. Moerman, P. Demeester, Demo abstract: Wilab, a real-life wireless sensor testbed with environment emulation, published in European conference on Wireless Sensor Networks, EWSN adjunct poster proceedings (EWSN), Cork, Ireland.

Ad hoc on-demand distance vector (aodv) routing. networking group request for comments (rfc): 3561, http://tools.ietf.org/html/rfc3561 (July2003).