A Common Application Requirement Interface for Cognitive Wireless Networks

Janne Riihijärvi, Marina Petrova and Petri Mähönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
email: {jar, mpe, pma}@mobnets.rwth-aachen.de

***Abstract*—We discuss the design of a Common Application Requirement Interface (CAPRI) for cognitive wireless networks. Applications can use CAPRI to express optimization goals and policies they expect the network to follow, which can subsequently be used by cognitive engines in their optimization processes. We believe that CAPRI design fills in an important gap in the current architectural understanding of CWNs, namely the lack of a common interface through which applications can express their requirements to the network. Existing work in this direction has been mainly focusing on hard QoS solutions, lacking the generality and expressiveness needed for CWN applications. We outline the architectural principles behind our design and give a detailed description of the basic functionalities supported by the interface. While the work has been carried out in the context of Cognitive Resource Manager (CRM) development, the design is general and usable in other cognitive engines as well.**

I. Introduction

Cognitive radios and cognitive wireless networks (CWNs) have emerged as a promising approach for optimizing network performance [1]–[4]. We explicitly understand here cognitive radios from Mitola's original point of as intelligent and adaptive terminals, rather than being limited to spectrum sensing and dynamic spectrum reuse as is often done today. Research in this domain has resulted in a number of promising optimization approaches in forms of various *cognitive engines* or *cognitive resource managers* [1], [5], [6]. However, from the architectural point of view there are still several missing technological components that need to be developed before adoption of CWN technologies can be realised in larger scale. One of the most critical of these is the lack of common interface between applications and the cognitive engine for expressing the optimization goals and policies the application expects to be followed. We argue that the the lack of such an interface does not only hinder efficient description of general optimization goals, but also form a roadblock for new applications and business models.

In this paper we give the design of the Common Application Requirement Interface (CAPRI) for CWNs. While being developed as a part of our Cognitive Resource Manager (CRM) architecture, the design of CAPRI is completely general and can be easily modified for other cognitive engines as well. The key approaches adopted in the interface design are *utility-based optimization* [7], [8] and using *policy languages* [9] to express additional application requirements and constraints. From the theoretical point of view, we consider applications giving to the cognitive engine a constrained optimization problem they want the network to solve for them. Utility function described the performance objective in a quantifiable form, and policies are used to give the constraints.

The rest of the paper is structured as follows. In Section II we give a short overview of utility-based optimization and the use of policies to give constraints on the optimization process. We then give a summary of the CRM architecture in Section III followed by a detailed description of the CAPRI design in Section IV. Finally, we give selected examples on the application of utility-based optimization for cognitive radios in Section V before drawing the conclusions and outlining future work in Section VI.

II. Utility Functions and Policies

We shall now go through the concepts behind our design in more detail, especially focussing on the roles of utilities and policies. This requires some formalization of the objectives of a CWN. Specifically, we consider a cognitive network as solving the optimization problem given by

$$\text{maximise} \sum\nolimits_i U_i(\boldsymbol{a}_i) \tag{1}$$

subject to constraint

$$P_i(\boldsymbol{a}_i, \boldsymbol{p}_i) = 1, \tag{2}$$

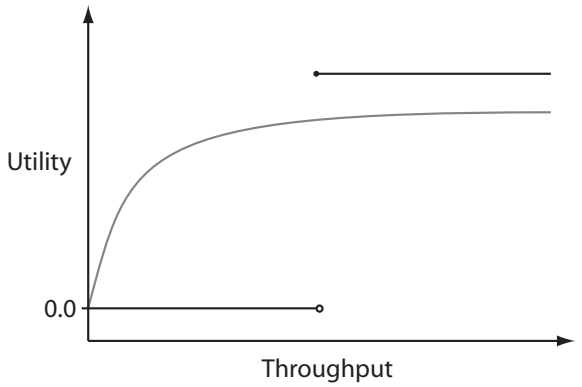

Fig. 1. Examples of one-attribute utility functions. Gray curve corresponds to a utility function of an elastic bandwidth-sharing application (such as file transfer or web-browsing), whereas the black curve illustrates a *simplified* utility function of VoIP client.

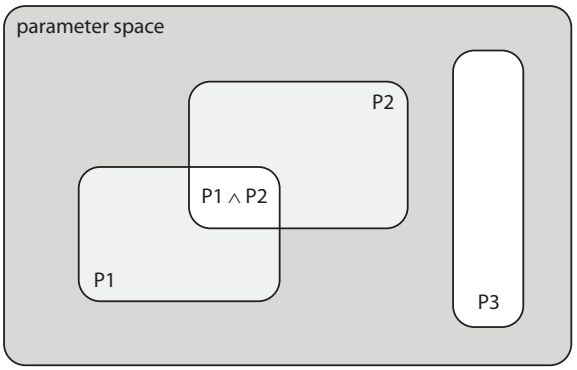

Fig. 2. Illustration of policies as subsets of the parameter space $\{\boldsymbol{p}_i\}$.

where $\boldsymbol{a}_i$ is the vector of network attributes characterising connection used by application i, U_i is the *utility function* of the application, $\boldsymbol{p}_i$ is the set of configurable parameters related to the connections used by the application, and P_i is the boolean *policy enforcement function* for the application. This is a slightly generalised form of the network utility maximisation problem of Kelly [10], [11]. In simpler terms, U_i measures the "goodness" of the connection for application i in terms of attributes such as throughput, delay, jitter and packet loss rate. For example, VoIP client and a web-browser benefit from throughput very differently, which can be encoded by the utility functions sketched in Figure 1. Not that the utility functions are only indicative showing the difference between step-wise and elastic utilities. Especially in the case of VoIP there maybe several steps in the utility function, but the key issue in the sketch is that after some hard point the utility vanishes to zero value. The parameters $\boldsymbol{p}_i$ correspond to all the "knobs" a cognitive engine can turn in an effort to maximise the application satisfaction. Typical components of $\boldsymbol{p}_i$ are choice of transport protocol, link selection, packet size and the radio transceiver configuration. Finally, the function P_i excludes some of the allowable $\boldsymbol{p}_i$ constraining the parameter space of the problem according to application policies. A simple example of an application policy might be to only use free networks (say Wi-Fi hotspots) for downloading email attachments.

In the way we have set up the definitions, policies defined through the functions P_i define subsets of the collection of possible parameter settings $\{\boldsymbol{p}_i\}$ as illustrated in Figure 2. We can aggregate multiple policies together by taking the logical AND of the values of the individual subpolicies as in

$$P_i(\boldsymbol{a}_i, \boldsymbol{p}_i) \equiv P_{i,1}(\boldsymbol{a}_i, \boldsymbol{p}_i) \wedge \cdots \wedge P_{i,n}(\boldsymbol{a}_i, \boldsymbol{p}_i), \qquad (3)$$

which corresponds in terms of Figure 2 into confining the search for optimal parameter settings into an intersection of the areas defined by the subpolicies. Such an approach allowing for multiple simultaneous policy specification is very natural since users are prone to think to constraints such as ones related to security or cost separately. Also policies might originate from a number of sources, including regulatory bodies, operators and terminal manufacturers, and same internal mechanisms can be used to aggregate all of these.

III. The CRM Architecture

In our earlier work we have introduced a Cognitive Resource Manager (CRM) framework for cognitive radios [1], [12]. Since the CRM design and implementation is still ongoing, we would like to give a short overview of the CRM architecture and its integral components in this section. This provides the context in which CAPRI was developed, although it is by design easily portable to other cognitive engines as well.

The framework allows for cross-layer optimization using rich set of information from the protocol layers, external sources of environmental information (e.g. Radio Environmental Maps), measurements as well as regulatory policies or the operating system. This information is fed through a well-defined and generic interfaces as shown in Figure 3. Providing a generic access to information and easy way to configure the protocol parameters is essential for the cognitive radio development.

We have designed Universal Link Layer API (ULLA) to retrieve data link layer and physical layer information independent of the radio technology [13]. For example,

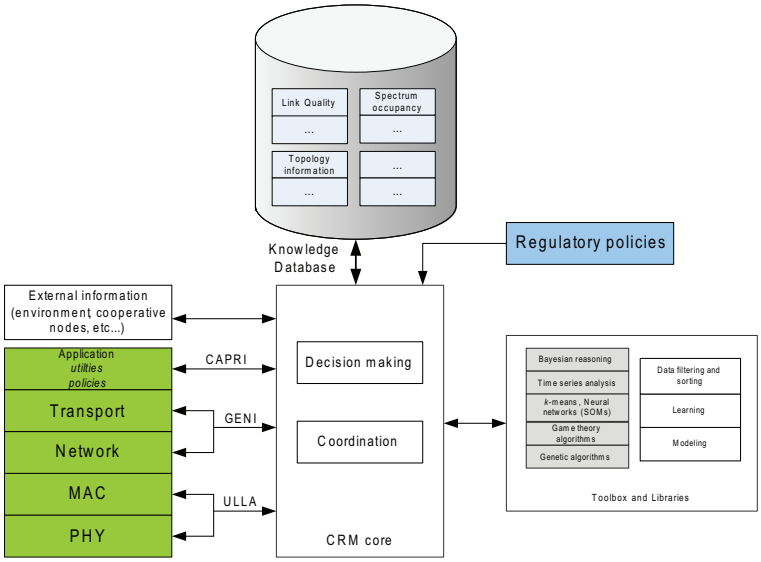

Fig. 3. CRM high level architecture

applications can query or subscribe for bit rate or latency values for particular links. This information can be either used in the learning process or for immediate actions. In similar fashion the Generic Network Interface (GENI) can expose network and transport layer information to the optimization and decision making blocks in the CRM framework. The Common Application Requirement Interface (CAPRI), the subject of the present paper, offers a novel utility-based approach for the applications to express QoS requirements and policies (preferences) which can serve as optimization goals towards the CRM optimizer. Furthermore some of the information form the ULLA and GENI (e.g. network state, quality of the links, connectivity opportunities, etc.) can be used to help the applications define more realistic preferences and express those trough CAPRI. The detailed design of CAPRI will be addressed in the next section.

CRM is designed to have a modular and flexible structure. A toolbox of optimization methods and learning techniques are also an integral part of the framework. Complex multi-parameter and multi-objective optimization problems can be addressed by employing variety of techniques ranging from classical to evolutionary methods. Furthermore the support of learning and AI reasoning mechanisms provide a good base to build a cognitive radio systems.

IV. The CAPRI Design

A. Main function calls

In this section we shall now go into the details of the CAPRI design. As discussed above, key entities to be specified through the API are utilities and policies. In the present CAPRI design we keep the separation between the two, although one might also consider designs in which both specifications are given in an integrated manner. The main CAPRI function calls are

$$\text{register\_utility}(\langle process\_id \rangle, \langle utility\_spec \rangle)$$

for specifying utilities,

$$\text{add\_policy}(\langle process\_id \rangle, \langle policy\_spec \rangle)$$

for introducing policies and, finally,

$$\text{revoke\_policy}(\langle process\_id \rangle, \langle policy\_name \rangle)$$

for revoking previously introduced policies. All functions return a value indicating either success or an error code. The first argument, $\langle process\_id \rangle$, is in each case used to specify the thread or application the utility and policy specifications apply to. A specific macro is used to indicate the calling process, whereas otherwise a platform-specific way to identify the target process is used. This allows legacy applications to be supported without changes to the source code, and also development of dedicated operating system components for managing optimization targets and constraints for applications. We shall now go into the details of the utility and policy specifications, that is, the structure of the arguments $\langle utility\_spec \rangle$ and $\langle policy\_spec \rangle$, respectively.

B. Utility specifications

For specifying utilities we use a simplified version of textual notation for mathematical expressions commonly

used in various computer algebra systems. The language used consists of integers and real numbers (in the commonly used dotted decimal or scientific notations), usual basic arithmetic operations (+ for addition, - for subtraction/negation, * for multiplication, / for division and for exponentiation) together with parentheses. Expressions $\log(a)$ and $\exp(a)$ are included, and evaluate to numerical approximations of the natural logarithm and exponential function with argument a. Additionally, the Iverson bracket notation is supported facilitating the expression of simple conditions, step functions etc. In this notation literal form [⟨*condition*⟩] is used, evaluating to one if the ⟨*condition*⟩ is satisfied, and to zero otherwise.

For expressing the conditions the usual expressions $=$, $!=$, $<$, $<=$, $>$ and $>=$ will be used. Finally, single or multi-letter identifiers are used for the arguments of the utility function, corresponding to the various measurable attributes of the connection. At present these are "t" for throughput, "d" for delay, "plr" for packet loss rate, "ber" for bit error rate and "j" for jitter. The complete description of this mini-language is given in Figure 4.

To make the above more concrete, let us have a look at some common examples. Suppose an application wants to request a minimum throughput of 16384 bits per second. The corresponding utility function would be a step function, expressed by the following specification:

"[t $<$ 16384] * 0 + [t $>=$ 16384]".

Suppose then that the above example application would also be delay-sensitive, but there is not a strict delay requirement that should be fulfilled. The application could express this preference by multiplying the above with an exponential factor involving the delay as follows:

"exp(-d/0.2) * ([t $<$ 16384] * 0 + [t $>=$ 16384])",

which would indicate that the application performance starts to decay as the delay becomes of the order of 200 ms.

Our design also includes a number of *helper function* designed to facilitate the utility specification process. We do not expect all application writers to use the language discussed above directly, but instead construct the specifications in a step-wise fashion. Examples of the specified helpers include

step_utility(⟨*attribute*⟩, ⟨*threshold*⟩)

constructing the utility function specification consisting of a step function at ⟨*threshold*⟩ for ⟨*attribute*⟩, and

elastic_utility(⟨*scale*⟩, ⟨*alpha*⟩)

constructing an instance from the family of the canonical utility functions for elastic traffic with throughput scaled by ⟨*scale*⟩ and with α-parameter ⟨*alpha*⟩. There also exists helpers for adding and multiplying individual utilities, which can be combined with the utility specifications returned by above functions to yield large families of commonly needed utilities.

C. Policy specifications

We shall now turn to the policy specification part of CAPRI, used in the ⟨*policy_spec*⟩ argument of the main function calls. For specifying application policies CAPRI uses an extension of the CoRaL language discussed in [14], [15]. The original application domain of CoRaL is in specification of policies for spectrum access, but the design is general enough to encompass policies related to network configuration as well. Most importantly, the CoRaL framework is completely extendible through the capability of introducing new *ontology specifications* as parts of the policy statements.

For our purposes we need to be able to integrate parameters and attributes accessed through interfaces such as ULLA and GENI into the policy specifications. Due to the extensibility of CoRaL this can be accomplished simply by introducing two additional ontologies, ULLA and GENI, defining the data types of the ULLA and GENI attributes as well as mappings to other ontologies, such as "radio" and "powermask". This is precisely the course we have taken in our design. After making the new ontologies available to the parser and the reasoner, policies including network information can be readily written. For example, policy indicating that only free network connections are to be used would read

policy *free_links_only* **is**
use *ULLA_attributes*;

disallow if
ullaLink.costPerUnit != 0;
end

The first line specifies the name of policy that can be used later when revoking policies as the string ⟨*policy_name*⟩. Names should be unique for a given process, but can in general overlap between different applications. In the second line the use-statements makes the ontology specification *ULLA_attributes* available, providing access to the various ULLA-related classes discussed in Section III. Finally, the last lines of the policy allow the use of only those links that are free.

⟨*attribute-ref*⟩ ::= `t` | `d` | `plr` | `ber` | `j`

⟨*comparison-predicate*⟩ ::= ⟨*scalar-exp*⟩ ⟨*comparison*⟩ ⟨*scalar-exp*⟩

⟨*comparison-bracket*⟩ ::= `[` ⟨*comparison-predicate*⟩ `]`

⟨*scalar-exp*⟩ ::= ⟨*scalar-exp*⟩ `+` ⟨*scalar-exp*⟩
| ⟨*scalar-exp*⟩ `-` ⟨*scalar-exp*⟩
| ⟨*scalar-exp*⟩ `*` ⟨*scalar-exp*⟩
| ⟨*scalar-exp*⟩ `/` ⟨*scalar-exp*⟩
| `+` ⟨*scalar-exp*⟩ | `-` ⟨*scalar-exp*⟩ | ⟨*const-ref*⟩ | ⟨*attribute-ref*⟩ | `(` ⟨*scalar-exp*⟩ `)`
| `exp(` ⟨*scalar-exp*⟩ `)` | `log(` ⟨*scalar-exp*⟩ `)` | ⟨*comparison-bracket*⟩

⟨*const-ref*⟩ ::= ⟨*integer*⟩ | ⟨*float*⟩

⟨*comparison*⟩ ::= `=` | `<` | `>` | `<>` | `!=` | `<=` | `>=`

⟨*integer*⟩ ::= ⟨*digit*⟩ { ⟨*digit*⟩ }

⟨*float*⟩ ::= ⟨*integer*⟩ | ⟨*integer*⟩ `.` ⟨*integer*⟩

⟨*digit*⟩ ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

Fig. 4. Grammar of the language used for utility specifications.

This is done by placing a condition on the attribute *ullaLink.costPerUnit* which contains information about the costs of using any of the available link-layer connections. Similar helper functions as discussed above for utility specifications can of course be defined to assist in construction of the policy statements.

V. Example Applications

In the simplest case the optimization in the wireless communication systems is oriented towards a single goal, for example maximization of the overall throughput, minimization of the latency or the bit error rate. Very often there can be several objectives from different applications, running simultaneously, to be optimized as fairly as possible. Just for illustration, minimizing the packet loss, maximizing the throughput, maximizing the spectral efficiency and minimizing the power consumption could compose a small set of desirable objectives to be achieved at one time instance in the system. The utility function of such a multi-optimization problem can be expressed through CAPRI.

We will give a more concrete example of an optimization problem, in order to discuss more closely the practical impact and importance of CAPRI. We have been solving with CRM Framework several different cross-layer optimization problems. Let us take a case where an application require low BER to ensure a reliable transmission and the same time the system needs to minimize the transmission power to prolong the battery life. We have used genetic algorithms [16] to find the best suited solution. We connect these two objectives in a single utility function by simply making a weighted sum

$$\mathrm{U} = w_{\mathrm{BER}} f_{\mathrm{BER}} + w_{\mathrm{power}} f_{\mathrm{power}}, \tag{4}$$

where the the weights satisfy

$$w_{\mathrm{BER}} + w_{\mathrm{power}} = 1. \tag{5}$$

By changing the value of the weight coefficients the we can steer the search of the best solution in a particular direction and give more importance to the first or the second objective, thus having the flexibility to adapt to different scenarios. For most applications the utility functions are expected to be more complex, but already this simple example is sufficient to illustrate the basic concept and power of the utility-based approach.

In Figure 5 and Figure 6 we depict the different values we get for the BER and power levels per subcarrier as a function of the weighting coefficients after performing GA-based optimization. We chose an OFDM system model with 100 data sub-carriers and 16 different transmit power levels for our simulation. The population size of the GA is set to 10 and the optimization was carried out per subcarrier. We performed the simulations taking into account AWGN channel. The differences in the optimization outcomes for different choices of the utility function are evident, showing that knowledge on such application requirements can indeed be used to optimize

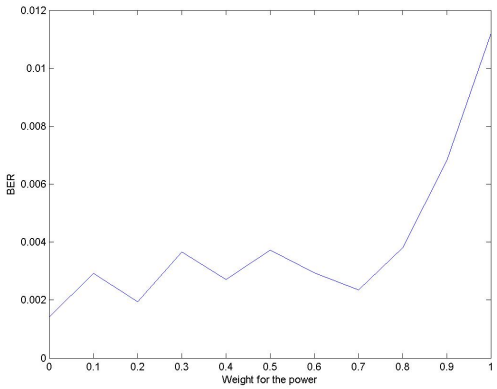

Fig. 5. Dependence of the BER values from the weighting coefficients in the utility function used in GA-based optimization.

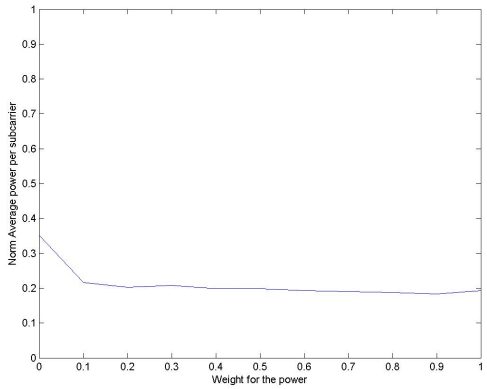

Fig. 6. Dependence of the power level per subcarrier values from the weighting coefficients in the utility function used in GA-based optimization.

the radio configuration. In our simple example we only considered radio parameters, and taking into account adjustable parameters higher in the protocol stack as well even more opportunities for optimization emerge.

Similar optimization problems have been solved by several groups in the context of cognitive engines and CRM. However, the utility functions and their values have been directly hand-coded to most of the implementations. Our initial research on CAPRI was strongly affected by the practical problems that clearly indicated that it is difficult to make cognitive resource managers an industrial reality without flexible and *run-time* capability to change and register application requirements and utilities.

One of the reasons to keep application requirements and policies separate in our interface is that it allows also different logical sources for the information. In a simplified case requirements are generated by (user) applications, but policies can be enforced, e.g., by network service providers or terminal manufacturer. It should be also noted that architecturally CAPRI and its policy extension are not simple functional extensions of SLA (Service Level Agreement) mechanisms. CAPRI has three functional parts. First, the interface structure itself to exchange data between applications and CRM including the API (Application Programming Interface) itself. Second, the query and parsing engine itself that is able to efficiently parse constraints, utility functions and policies and to pass this information to appropriate optimizers and control loops. Third, the Cognitive Resource Manager itself is required to facilitate the conflict resolution and a sensible combination of different objectives. This is a part, where also the learning and inference becomes an integral part of CARPI and CRM interplay. As can be seen from Figure 5 and 6, the system performance may be sensitive for different parameter values and ranges of values. Codifying or learning this information helps system not only to adjust its performance faster, but also to chose how to handle and combine different objectives. Sometime the objectives might be *formally* or *functionally* competitive, but the real combined situation might be solved rather easily due to different sensitivity to the parameter values.

VI. Conclusions

In this paper we introduced an interface design for expressing application requirements towards a cognitive engine. The Common Application Requirements Interface (CAPRI) supports utility-based optimization constrained by application policies on allowable network settings. The utility-based optimization paradigm is a very powerful one, and especially appropriate for cognitive wireless networks. However, existing application programming interfaces have not provided support for it. We have also been careful in making the CAPRI design independent on the used cognitive radio framework it would interface to. In the examples given we have referred to our CRM design for concreteness, but mapping these examples to other implementations and architectures is very straightforward.

Acknowledgment

The authors would like to thank the RWTH Aachen University and the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) for providing financial support through the UMIC research center. We

would also like to thank the European Union for providing partial funding of this work through the ARAGORN project.

REFERENCES

[1] P. Mähönen, M. Petrova, J. Riihijärvi, and M. Wellens, "Cognitive wireless networks: your network just became a teenager," in *Proceedings of the 25th Conference on Computer Communications*, 2006, pp. 23–29.

[2] J. Mitola III and G. Maguire Jr, "Cognitive radio: making software radios more personal," *IEEE Personal Communications*, vol. 6, no. 4, pp. 13–18, 1999.

[3] S. Haykin, "Cognitive radio: brain-empowered wireless communications," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 2, pp. 201–220, 2005.

[4] B. Fette, *Cognitive radio technology*. Newnes, 2006.

[5] T. Newman, B. Barker, A. Wyglinski, A. Agah, J. Evans, and G. Minden, "Cognitive engine implementation for wireless multicarrier transceivers," *Wireless Communications and Mobile Computing*, vol. 7, no. 9, 2007.

[6] T. Rondeau, C. Rieser, B. Le, and C. Bostian, "Cognitive radios with genetic algorithms: intelligent control of software defined radios," in *Software Defined Radio Forum Technical Conference*, 2004.

[7] J. Riihijärvi, M. Wellens, and P. Mähönen, "Link-Layer Abstractions for Utility-Based Optimization in Cognitive Wireless Networks," in *Proceedings of CROWNCOM'06*, Mykonos, Greece, June 2006.

[8] S. Shenker, "Fundamental design issues for the future Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1176–1188, 1995.

[9] G. Denker, D. Elenius, R. Senanayake, M. Stehr, D. Wilkins, S. Int, and M. Park, "A policy engine for spectrum sharing," in *2nd IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, 2007. DySPAN 2007*, 2007, pp. 55–65.

[10] F. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, vol. 8, pp. 33–37, 1997.

[11] F. Kelly, A. Maulloo, and D. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research society*, vol. 49, no. 3, pp. 237–252, 1998.

[12] M. Petrova and P. Mähönen, *Cognitive Resource Manager: A cross-layer architecture for implementing Cognitive Radio Networks*. Cognitive Wireless Networks (eds. Fittzek F. and Katz M.), Springer, 2007.

[13] M. Sooriyabandara, T. Farnham, C. Efthymiou, M. Wellens, J. Riihijärvi, P. Mähönen, A. Gefflaut, J. Galache, D. Melpignano, and A. van Rooijen, "Unified Link Layer API: A generic and open API to manage wireless media access," *Computer Communications*, vol. 31, no. 5, pp. 962–979, 2008.

[14] D. Wilkins, G. Denker, M. Stehr, D. Elenius, R. Senanayake, C. Talcott, S. Int, and M. Park, "Policy-based cognitive radios," *IEEE Wireless Communications*, vol. 14, no. 4, pp. 41–46, 2007.

[15] D. Elenius, G. Denker, M. Stehr, R. Senanayake, C. Talcott, and D. Wilkins, "CoRaL–Policy Language and Reasoning Techniques for Spectrum Policies," in *Proceedings of the Eighth IEEE International Workshop on Policies for Distributed Systems and Networks*. IEEE Computer Society Washington, DC, USA, 2007, pp. 261–265.

[16] A. de Baynast, P. Mähönen, and M. Petrova, "ARQ-based cross-layer optimization for wireless multicarrier transmission on cognitive radio networks," *Computer networks*, vol. 52, pp. 778–794, March. 2008.