

Link-Layer Abstractions for Utility-Based Optimization in Cognitive Wireless Networks

Janne Riihijärvi, Matthias Wellens and Petri Mähönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
Email: {jar, mwe, pma}@mobnets.rwth-aachen.de

Abstract

We argue that utility functions are very natural tools for formulating optimization problems in cognitive wireless networks. However, their use mandates that the platform used supports well-defined APIs and abstractions for retrieving information necessary to calculate the application utilities. We present the design of a Unified Link-Layer API that offers precisely this functionality in the context of link-layer information. The API developed offers powerful query and notification mechanisms, which considerably simplify the implementation of technology-independent reasoning engines for cognitive networks. Prototype implementations of the API show that despite the rich functionality offered, the implementation can be made very lightweight and fast, enabling optimization decisions even on per-frame basis with standard user terminals.

1 Introduction

Software defined radio (SDR) and more recently cognitive radio (CR) paradigms have become important trends in wireless communication. First SDR-systems have been deployed and CR is one of the most rapidly growing research domains. Although numerous researchers see spectrum-agility as one of the main features of CRs we would like to underline the cognitive aspect bringing machine learning and multi-dimensional optimization techniques to the wireless world. This approach, initially proposed by Mitola [10], will enable mobile devices to take advantage of the new freedom offered by software defined and spectrum-agile radios through context-sensitive adaptation.

Additionally, new and reconfigurable protocol stacks [7] are under development leading to additional flexibility in the overall system. Although several powerful optimization techniques are available that can cope with large parameter spaces and complex interrelations, also those methods re-

quire clear goals, which, in the best case, should be unambiguously measurable. In this paper we follow the proposal by Shenker [13] to use application layer utility as performance metric and optimization goal, which will use several other attributes such as throughput or latency depending on specific application requirements.

The major advantages of this approach are a well defined value that can be used to compare network configurations and the ability to represent differences between applications. The requirements of, e.g., file-download applications compared to a VoIP-call vary a lot and different input metrics should therefore be weighted differently. The first take might be limited to a small number of input variables but the approach itself is easily extensible to cope with complex environments as present in cognitive wireless networks (CWNs). Additionally, multimedia applications will use more complex utility-functions as, for example, today's video encoders are often not able to adapt to arbitrary bitrates but do only support a fixed subset of rates. The utility of a connection will increase only if the bandwidth available for the video conferencing application will increase enough to safely upgrade to a higher encoding quality. Additionally, the use of utility makes the whole approach easier to extend because future applications can specify their utility-function using the same general grammar enabling other entities to understand those requirements.

Present networking research often deals with quality of service (QoS) aspects and issues of QoS-agreements between networks such as Service Level Agreements (SLAs). Though this work is related to the proposal of utility-based optimization we do not consider QoS here. The use of utility as an abstract metric can also be applied in the simple best effort case that leaves enough room for optimizations without giving any guarantees or requiring signalling between network entities.

An optimization algorithm for a CWN should adapt settings of the spectrum-agile radio and the reconfigurable protocol stack in order to improve the application-layer utility. As the input parameter space is definitely too large

to be tested completely, powerful optimization techniques probably combined with machine learning approaches are promising candidates. However, one important prerequisite before the actual optimization can start is a system that offers flexible and technology-independent interfaces to retrieve measurement results. On the way towards completely reconfigurable devices intermediate designs will support a fixed subset of technologies as done by some of today’s mobile terminals. Those rather static systems also use static and often technology-dependent interfaces. The optimization engine has to then support all those proprietary interfaces in order to gather all information needed to start the actual optimization.

In contrast we present the Unified Link-Layer API (ULLA) as an important enabler for CWNs which introduces link-layer abstractions and combines access to different link-layer technologies in one API. ULLA offers one common interface to gather abstract metrics such as bitrate or latency not preventing access to technology-specific details such as the RTS-threshold in IEEE 802.11-based WLANs. In addition to the architecture of the ULLA framework and its implementation design we will explain, taking for simplicity the example of WLANs, how abstracted link-layer metrics can be realised considering technology-specific aspects.

We continue the paper in section 2 with a detailed explanation of the utility as a common performance metric. In section 3 we point out key requirements for our approach and optimization of CWNs in general. Afterwards, we present ULLA in section 4 and give insights to a case study based on WLANs in section 5. We conclude the paper in section 6.

2 Utility-based optimization

As stated in the introduction, utility is a quantitative, numerical expression of the quality of a connection measured at application layer. More formally, we can define utility as a function $U(a_1, \dots, a_n)$ of the various measurable attributes $\{a_i\}$ of the connection. Common examples of these attributes would be maximum achievable bitrate, latency, jitter, frame error rate, and so on. Each of these attributes is itself a function $a_i = a_i(p_1, \dots, p_n; s_1, \dots, s_m)$ of the parameters p_i that can be configured in the (local) network stack, and some stochastic variables $\{s_i\}$ used to model the end-to-end connection. This formulation allows for numerous optimization problems to be formulated, as we shall see in the following. In cognitive wireless networks these optimization problems can in whole, or in part, be solved by application of machine learning techniques. Nevertheless, in simpler cases classical distributed algorithms approach could be usable as well.

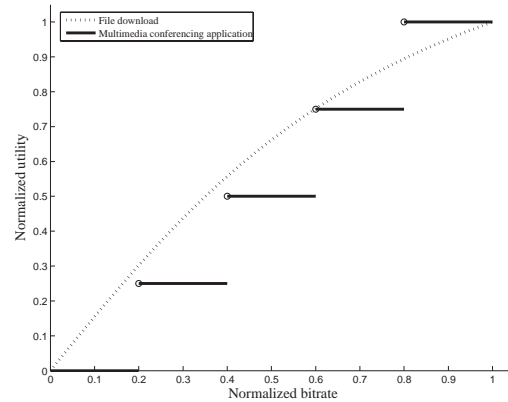


Figure 1. Examples of bitrate utilities for file download and multimedia conferencing applications.

2.1 Examples of utility functions

In classical capacity optimization approaches the quantity to be maximized for each connection has traditionally been the obtained bitrate. However, only specific, “download” types of applications (such as FTP, or nowadays peer-to-peer filesharing applications) obtain strictly increasing utilities as the bitrate is increased. For streamed multimedia content and for interactive sessions the situation is more complicated. First of all, the connection has to support some minimal bitrate and (in the interactive case) latency for it to be usable for the application at all. After these initial conditions are satisfied, the improvement of application-layer utility typically takes place in discrete steps, for example in the case of codec change or reconfiguration becoming possible.

The “traditional” utility functions (see, for example, [4, 5, 13]) corresponding to these two cases are of the generic form given in Fig. 1. Taking into account the detrimental effects of increased latency into the perceived utility, we arrive at the generic form for the two-parameter utility depicted in Fig. 2.

2.2 Utility-based optimization in cognitive networks and radios

We shall now come to the fundamental applications of utility functions in the CWN context. The objective of the “cognitive” software agents in the terminal (and possibly in the network) is to optimize the quality of the connections as perceived by applications. In terms of the specified utilities U_i for different applications i , the corresponding optimization problem becomes finding the collection of parameter

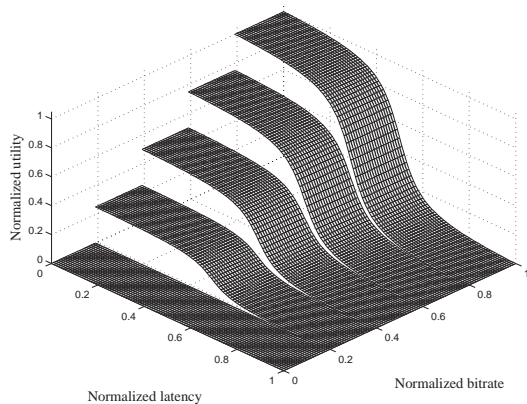


Figure 2. Utility function for interactive session.

settings $p_i = v_i$ such that the overall utility is maximized. More formally, the optimization problem to be solved is

$$\{v_i\} = \arg \max_{p_i} \left\{ \sum_i U_i(a_k(p_i)) \right\}. \quad (1)$$

Usually, in addition to (1) some constraints have to be introduced. Most commonly given examples would be different fairness criteria, to prevent “starving” of some applications to satisfy the needs of very resource-hungry ones. The concept of *utility fairness* first applied in utility-aware congestion control [8] is very relevant here.

Of course, how this prescription translates to concrete configuration changes depends on the capabilities of the terminal. In the full software radio case the problem becomes extremely complicated to solve, as the configuration space of the terminal is very large (essentially the cartesian product over the space of all physically distinct waveforms and scheduling strategies). Simplifying heuristics would clearly be necessary to deal with such problems. In case of less futuristic radios, much simpler approaches are possible. The simplest extreme case would be that of access selection without reconfiguration, that is, simply choosing which of the wireless interfaces available on the terminal to use.

2.3 Role of the first wireless hop

The discussion in the previous section was still rather generic, and applies as well to network-wide optimization, as for the typical single wireless hop case. We shall focus on the latter, assuming that the changes in the conditions of this first-hop link dominate the changes in utility compared to the events taking place in the fixed network. This

assumption greatly simplifies the problem, as terminals can make optimization decisions based on local information (or information gathered only from one hop away). While certainly not applicable in all cases, this approximation should also hold significant amount of time. Fixed access and core networks tend to be lightly utilized (with the exception of some brief congestion periods), and there are no indications that this would change in the near future [12].

3 Abstractions as key requirement for utility-based approaches

After presenting the utility as application layer metric and its usage for network optimization we continue with describing fundamental technologies that are required on the way towards a real implementation of utility-based network optimization.

The utility should be calculated based on detailed and up-to-date measurement results. In today’s systems the bottleneck usually is the first wireless hop, which concentrates the collection of such measurements on the end user device and the respective base station or access point. However, retrieving measurement results is not an easy task. The interfaces vary between communication technologies, such as Bluetooth, WLANs, or cellular systems. In the case of upcoming cognitive radios cross-layer information exchange is clearly required and the present approach of retrieving link-layer information in technology-specific way will not scale due to the increased complexity. Common interfaces, such as the proposed Unified Link-Layer API (ULLA), allow the calculation of the utility independently of which wireless interface is actually used.

Generalized metrics made available through universal interfaces will have to be carefully calculated, e.g., throughput cannot be instantaneously determined. Instead, the amount of successfully transmitted data is divided by the monitored time period. Therefore, the chosen amount of time will change the reported value and longer times leading to more smoothed values will delay possible reactions on changes in the throughput. On the other hand, small spikes should not trigger any action so that smoothing out those is useful. Hence, the optimal way of smoothing depends on the application requirements.

Portability is another closely related aspect because more complex optimization tools should be reusable on other devices using different hardware or running different operating systems. Again, present interfaces of embedded platforms or real computers look different and especially lower layer networking interfaces vary between operating systems. A solution to this problem is an additional requirement to enable CWNs.

The description of the utility function should be handled in an abstract and platform-independent way. A uni-

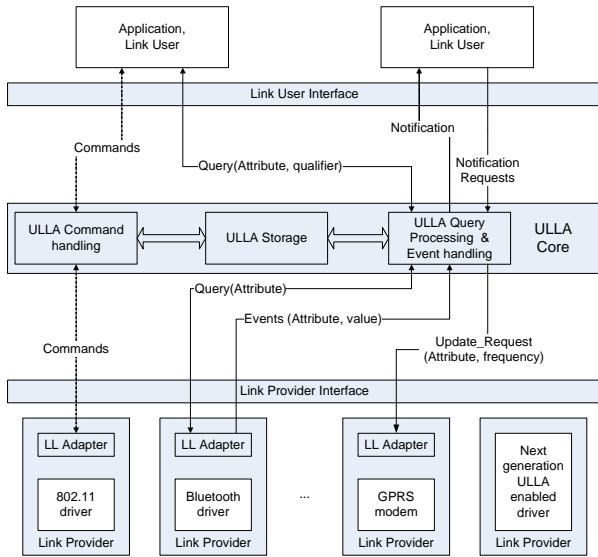


Figure 3. ULLA architecture.

versal interface for applications is required that allows to describe the QoS requirements and the complete utility-function without limiting the usage of the applications to certain devices or technologies. This way applications can still be portable although, for example, the resource optimization is done as proprietary part of future operating systems.

4 Unified Link-Layer API

We shall now present in detail the Unified Link-Layer API (ULLA for short) offering the link-layer abstractions needed for utility-based optimization for cognitive radios. The API itself is a result of the European GOLLUM-project [1]. The overall architecture of ULLA is depicted in Fig. 3. The main entry point for entities using the ULLA (called *link users* in the ULLA architecture) is the *link user interface*. Three classes of functionality can be accessed there:

- **Queries** can be issued by the link users to obtain values of the various attributes (such as latency, capacity and jitter, to name a few) of the available wireless interfaces.
- **Commands** can be used to change the configuration of the interfaces in question. Common examples of using commands would be connecting and disconnecting links, setting credentials for authentication purposes, or configuring the power saving mode of the interface.
- **Notifications** can be requested to inform the link user

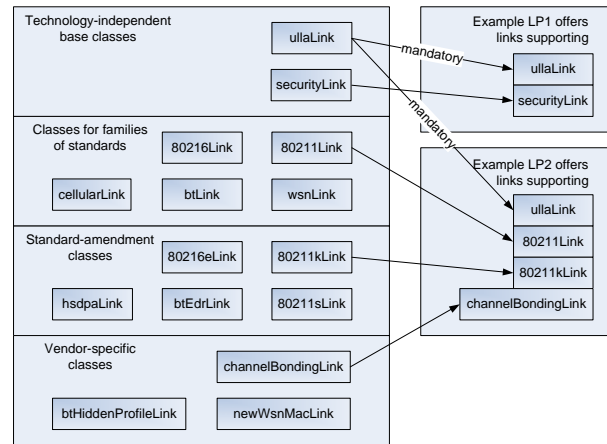


Figure 4. Class hierarchy used by ULLA.

if a change occurs either in availability of the connection, or the corresponding attributes.

The data model visible to the link user through the ULLA is a combination of object oriented programming paradigm, and database abstraction. Links (understood as potential connections between two wireless interfaces) are defined by abstract classes consisting of attributes and commands, the instances of which are visible to the link user as rows in a database. A lightweight subset of SQL, called ULLA query language (UQL), is used to specify the queries and the conditions for notifications.

An extensive class hierarchy (shown in Fig. 4) is specified for different link types to allow the link user to choose the abstraction level to operate on. For the optimization purposes which we focus on here, the highest abstraction level offered by the technology-independent base classes would be most appropriate. The most fundamental of these classes is *ullaLink* containing the basic attributes that must be present in all links. Other types of applications, such as technology-specific configuration and diagnostic tools would, on the other hand, most likely utilize classes offering less abstraction. This class hierarchy is extensible, and dynamically formed at runtime. Devices can give descriptions of the extension classes they support, and link users can query for these extensions without being limited to compile-time knowledge of the available classes.

The API is also very easy to use in practise, as the following code samples demonstrate. To find all links that have (estimated) highest achievable receive bitrate of higher than 128 kbit/s one would issue the call

```
requestInfo("SELECT linkId, rxLatency FROM ullaLink
WHERE rxBitrate > 128000", &myres);
```

where *myres* is a handle to ULLA result structure, in this case to be filled in with a sequence of identifiers of the links

together with their latencies matching the query. This list of attribute/value tuples can then easily be parsed using simple accessor functions.

Using commands is likewise simple. The relevant API call (`doCmd()`) takes as an argument a command descriptor, containing the identifier of the link the command is to be issued on, and the command name together with an argument string. Most commands do not require any arguments at all, as most of the necessary ones are already implicitly present in the data structure for the particular link.

In our point of view the notification support is the most novel part of ULLA. The relevant API call is of the form

```
requestNotification(&notId, query_string, &notDesc);
```

The first argument is used to return the unique identifier of this notification request for cancelling or modification purposes. The second argument is again a UQL-string similar to the simple query example, but this time the WHERE-clause is used as a criterion for triggering the notification. Final argument is a notification descriptor, detailing, for example, how many times the notification is allowed to be fired.

The notification mechanism has obvious application in the utility-based optimization. As the application or any “cognitive” software agent performing the optimization functions knows the expression for the application utilities, it can always convert threshold values expressed in terms of utility into conditions on attributes that then can be used as conditions in the corresponding UQL statement. Furthermore, connection manager like applications can use ULLA to monitor link and spectrum usage. Thus, ULLA makes it easy to implement cognitive radios that optimize the spectrum usage in conjunction with the application utility. For example, the notification mechanism can be configured based on current policies to alert the reasoning engine on significant changes in spectrum usage such as appearance of the primary licensee.

The parts of ULLA architecture residing below the link user interface are, of course, completely hidden from the link users. They consist of the ULLA core functionalities, offering the said database abstractions and processing necessary for commands, queries and notifications, and of *link providers* (LPs), which can be thought of as ULLA-enabled device drivers. Legacy drivers can be used as LPs by the means of a *link-layer adapter* (LLA). A well-defined *link provider interface* is given to maintain interoperability and compatibility between link provider and core implementations coming from different vendors or programmers.

Despite the rich feature set offered by ULLA, our prototyping activities have shown that it can be implemented in a very compact space, and that excellent performance in terms of overhead and query and notification latencies can be achieved. For example, the memory footprint of our

Linux prototype implementation is on the order of 185 kB and typical query durations are in the range of 25–50 microseconds. Also the additional processing delay introduced by the notification mechanism is normally well under half a millisecond, showing that ULLA can be used for very time-critical adaptations as well. Using a real-time computing platform would naturally improve these results still.

5 Case study in IEEE 802.11 technologies

ULLA is a powerful tool that allows applications to easily access link-layer information. However, the major part of the link-layer abstraction has to be done in the LP, which has to map the technology-independent performance metrics, for example attributes in the `ullaLink`-class, to a technology-dependent implementation. In the following we describe parts of this process by using the example of IEEE 802.11-based WLANs.

The maximum achievable throughput is one of the most important performance metrics for applications. In the case of WLANs only the actually used throughput is practically measurable in an easy way simply by aggregating the successfully acknowledged amount of data over time using standard sliding window techniques.

One possible approach is to assume a perfect channel and derive from theoretical overhead-calculations and the used PHY-rate the available capacity [6]. The proposal is practically feasible but neglects important effects because it assumes a perfect channel and only indirectly considers transmission failures because the used PHY-rate will decrease. More accurate results also considering effects of hidden nodes can be achieved by using active probing [11]. However, the measurement requires about 30-40% of the channel capacity and relies on support on both peers making it inappropriate for general usage.

More promising is the algorithm suggested in [9] which monitors the channel usage and filters out transmissions of other links. Estimating the capacity used by other nodes leads to the fraction which is left for the measuring node itself. Simulations proved the feasibility but practical implementations face further challenges. The node has only a local view of the network and hidden nodes might seriously degrade the performance. Additionally, reporting traffic addressed to other nodes to the device driver requires the WLAN interface to be set to promiscuous mode which prevents it from transmitting data frames. Therefore, a possible demonstration requires a second interface used purely for monitoring purposes.

A final implementation will also combine other metrics specifically chosen for the WLAN-case in order to estimate the maximum achievable throughput. Important candidates are the ratio of frames received with the retransmission bit set and the average number of tries needed to suc-

cessfully send a frame in the other direction. Both metrics will increase with worse SNR-values as well as higher contention and thus more collisions on the channel. The latter one can also be measured following the details of the IEEE 802.11 MAC-algorithm. A node will defer its transmission when it senses the channel busy and present device drivers, e.g. [3], offer a counter tracking the number of such deferral events per transmitted frames. Careful calibration-measurements will allow to retrieve direct relationships between these measurable metrics and the unused capacity.

Hence, the LP should combine in a first step directly accessible metrics to abstract technology-independent attributes which later on can be used to determine the application utility and to choose and optimize the whole network stack configuration. Machine learning capabilities in future cognitive systems could also be used to adapt such abstractions based on specific conditions in the surroundings. Part of our future work is the detailed analysis of these approaches.

6 Conclusions

In this paper we elaborated on the use of utility-based optimization for cognitive wireless networks. Utility functions offer a clean way to formalize application requirements and how well they are fulfilled by a given connection. However, to enable the use of utility in this manner requires that relevant network performance measurements are available through sufficiently abstracted and technology-independent interfaces. We presented such a solution for accessing link-layer information that also allows to change and configure PHY- and MAC-layer settings in technology-independent manner. Complete reference implementation of this Unified Link-Layer API (ULLA) has been developed. Performance tests have shown that efficient cross-layer optimization using ULLA is very feasible even without resorting to technology-specific low-level programming. The public release of the ULLA Linux implementation presented here is available from the sourceforge project page [2].

Acknowledgment

We would like to thank DFG (Deutsche Forschungsgemeinschaft), European Union (the GOLLUM-project) and RWTH Aachen University for the financial support. We would also like to thank the GOLLUM research team for fruitful discussions.

References

- [1] *The GOLLUM-project website*, <http://www.ist-gollum.org> [Cited on: 8th of February 2006], 2004.

- [2] *Open source release of the Unified Link-Layer API*, <http://ulla.sourceforge.net/> [Cited On: 4th of April 2006], 2006.
- [3] Absolute Value Systems Inc. (AVS). wlan-ng driver implementation. <http://www.linux-wlan.org/>.
- [4] Z. Cao and E. W. Zegura. ABR service for applications with non-linear bandwidth utility functions. In *Proc. of 5th IEEE ICNP*, page 15, Washington, DC, USA, 1997.
- [5] Z. Cao and E. W. Zegura. Utility max-min: An application-oriented bandwidth allocation scheme. In *Proc. of IEEE INFOCOM (2)*, pages 793–801, 1999.
- [6] M. Deziel and L. Lamont. Implementation of an IEEE 802.11 link available bandwidth algorithm to allow cross-layering. In *Proc. of WiMOB'2005*, pages 117–122, Montreal, Canada, August 2005.
- [7] T. Farnham. Radio link enhancement using an open flexible protocol stack framework. *Wireless Communications and Mobile Computing*, 5(4):379–395, June 2005.
- [8] T. Harks and T. Poschwatta. Utility fair congestion control for real-time traffic. In *Proc. of 8th IEEE Global Internet Symposium, co-located with IEEE INFOCOM*, pages 85–90, Miami, FL, USA, March 2005.
- [9] M. I. Kazantzidis. MAC Intelligence for Adaptive Multimedia in 802.11 Networks. *IEEE Journal on Selected Areas in Communications*, 23(2):357–368, February 2005.
- [10] J. Mitola. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*. Ph.D. Thesis, KTH (Royal Institute of Technology), 2000.
- [11] M. Nielsen. Demo of residual bandwidth estimation in an 802.11 ad hoc network. In *Proc. of REALMAN '05*, pages 144–146, Santorini, Greece, June 2005.
- [12] A. Odlyzko. Data networks are lightly utilized, and will stay that way. *Review of Network Economics*, 2(3):210–237, 2003.
- [13] S. Shenker. Fundamental design issues for the future internet. *IEEE Journal on Selected Areas in Communication*, 13(7), September 1995.