Enabling Seamless Vertical Handovers Using Unified Link-Layer API

Matthias Wellens, Janne Riihijärvi, Krisakorn Rerkrai,
Marten Bandholz, and Petri Mähönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, 52072 Aachen, Germany
{mwe, jar, kre, mba, pma}@mobnets.rwth-aachen.de

ABSTRACT

Wireless channels vary drastically over time and mobility adds further difficulties to the engineering problem of providing stable and robust communication performance. Handovers occur regularly in mobile environments and the increasing heterogeneity of present networks leads to the requirement for seamless vertical handovers. An important practical problem is the lack of a generic interface that enables applications, such as a vertical handover decision engine, to retrieve comparable link-layer information independently of the deployed technology or the used software and hardware platform. In this paper we propose the Unified Link-Layer API (ULLA) as a solution to this problem. We describe its architecture, how it can be utilized for vertical handover scenarios and introduce our Linux prototype. The prototype performance proves that ULLA is appropriate for mobility management.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms

Design, Performance

1. INTRODUCTION

The capabilities of mobile end user devices constantly increase and recently support for more than one wireless technology has become a standard feature. Additionally, mobile operators deploy new technologies and diversify their services using specific systems. The resulting network heterogeneity leads to several engineering problems with vertical handovers being one of the prominent ones. The communication research community has put considerable effort in solving the problem of seamless vertical handovers between any technologies [6, 9, 12]. However, especially in terms of implementation hand-tailored solutions dominate and more universal solutions are still missing. One of the major practical problems is the diversity of link-layer interfaces used in present-day operating systems.

Vertical handover managing agents have to be able to retrieve link-layer information from all technologies supported on the device. In order to enable universal solutions for multi-parametrical handover decisions these measurement results have to be up-to-date and comparable. Interfaces offered in operating systems such as different flavors of Linux, Windows or also Symbian vary drastically between each other, between wireless technologies, and moreover do not provide comparable results. In this paper we propose the Unified Link-Layer API (ULLA) [10] that enables technology- and platform-independent access to comparable link-layer information and offers common characteristics for all available links without preventing access to technology-specific features. The described API was developed by the European Union research project GOLLUM [1] and as part of the project the API was prototyped on various platforms [11] including embedded systems down to wireless sensor nodes.

Wireless channels are inherently unstable complicating the link management and causing several problems when stable and robust communication performance has to be achieved. Mobility adds further effects because the channel performance varies and interference sources might be present worsening the performance. ULLA is a powerful tool that enables any kind of application[1] to be link-aware and take appropriate actions when the communication environment changes. If the performance of the actually used link decreases below a certain threshold modern communication terminals usually initiate handover actions. The most common handovers are still within one technology but recent standardisation activities (IEEE 802.21 [7] and Unlicensed Mobile Access (UMA), which is now part of 3GPP [2]) document also the increased interest in industry in vertical handovers. We explain why ULLA is an important enabler for efficient heterogeneous and platform independent solutions and describe how ULLA functionality can be used to implement existing proposals.

The remaining paper is structured as follows. We introduce

[1]The term application is used in a broad sense. Not only programs working on the ISO layer seven can benefit from ULLA but also entities working on lower layers such as link-aware routing protocols or vertical handover agents as described in this paper.

the ULLA architecture in section 2. We continue in section 3 with giving insights in how ULLA can be used as important enabler for mobility solutions. In section 4 we describe the Linux prototype together with selected performance figures, and finally conclude the paper in section 5.

2. ULLA ARCHITECTURE

The goal at the beginning of the ULLA-development was to define an API that enables applications to be link-aware without knowing details about the used wireless technology. In addition the API should still allow more sophisticated tools to access detailed technology-specific parameters or configurable settings. The latter case is more probable for mobility management because the basic functionality is required to provide seamless connectivity to end user applications. Although we limit the description in this paper only to wireless technologies ULLA itself can also be used to access information related to fixed networks as well.

Figure 1 shows the ULLA architecture. The major block is the ULLA core in the middle, which acts as a broker between applications, called Link Users (LUs), and device drivers, called Link Providers (LPs). The respective interfaces, Link User interface and Link Provider interface, together form the Unified Link-Layer API (ULLA).

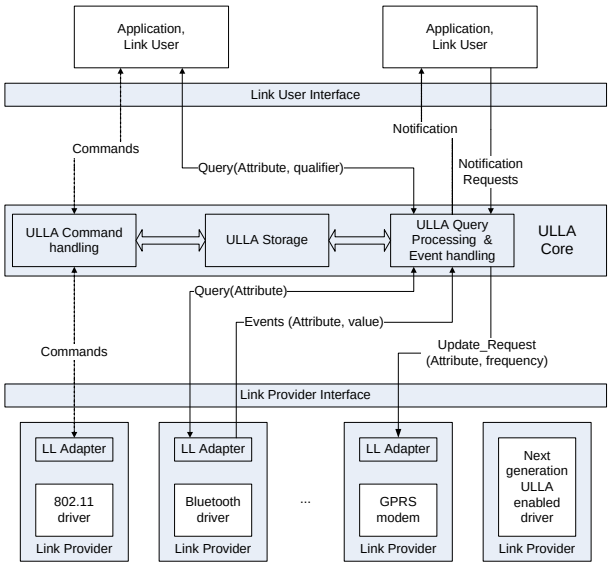

Figure 1: ULLA architecture.

ULLA core itself consists of smaller blocks that implement the main functionalities as further described in subsection 2.2 and 2.3 and of the ULLA storage where link layer information is cached in order to avoid unnecessary requests towards the LPs. LUs can benefit from ULLA by simply including the `ulla.h` header-file and calling the respective LU-functions. LPs are abstractions of network interface cards (NICs) and mostly implemented as device drivers, although legacy drivers have to be extended using Link Layer Adapters (LLAs), that wrap existing interfaces and implement the LP interface. LLAs can be done in different ways, for example Linux WLAN LLAs could simply wrap the wireless extensions API or extend the actual device driver open source code to implement the LP interface directly. The former one is limited to the available functionality and its only main benefit is the unification of interfaces. The latter one is more complex but opens possibilities to add further features and, e.g., adapt the way the signal strength is smoothed over time. Future ULLA-enabled device drivers are expected to directly support the LP-interface.

2.1 Data model

The main entities ULLA works with are links. As the names tell, these are offered by Link Providers and utilized by Link Users. The data model applied in the ULLA framework is following an object-oriented approach, exposed to the LU through a database abstraction. Links are described using different classes structured in a class hierarchy as shown in figure 2. Each class consists of attributes, such as received signal strength or the length of the used encryption key, and of commands, which resemble the methods in the object oriented view. Commands can be used to start the scanning process, connect a link, or trigger other more complex actions on the NIC. On top of the class hierarchy two mandatory base classes are placed, `ullaLink` and `ullaLinkProvider`, where the latter one includes characteristics of the LP. Both classes have to be supported by any ULLA-compliant implementation. Support for further classes is optional but will be crucial for technology-specific solutions or further added functionality. Another base class `securityLink` describes security features that can also be abstracted over wireless technologies. The next hierarchical level describes technology-specific attributes and commands, such as the RTS/CTS threshold used in WLANs. More specific standard amendments, e.g. IEEE 802.16e or UMTS HSDPA (High Speed Downlink Packet Access), build up the third level and the most specific level are classes consisting of vendor-specific additions that could, for example, be used by diagnosis tools.

The object-oriented abstraction enables extensibility and flexibility. Upcoming technologies such as IEEE 802.20, WiMAX, or UWB can be supported by the ULLA framework by adding new classes and providing ULLA-enabled LPs. ULLA core itself is completely technology-independent because it gets to know new classes when LPs register first time. ULLA core can query the whole class structure using a reflection interface and thus can learn during runtime about new technologies.

ULLA storage, which can be implemented either as database backend, such as the well-known MySQL, or as proprietary solution, maintains one table per class[2]. When LPs find new links during scanning they will register those with ULLA core and ULLA storage will add another row (class instance) to all tables that are supported by the respective link. LUs can later on request information from one table without any need to specify in which specific link they are interested in. Using this database model LUs can flexibly choose one out of all available links or compare characteristics of different links.

[2]As each table represents the data registered for one class these two terms are used equivalently in the remainder of the paper.

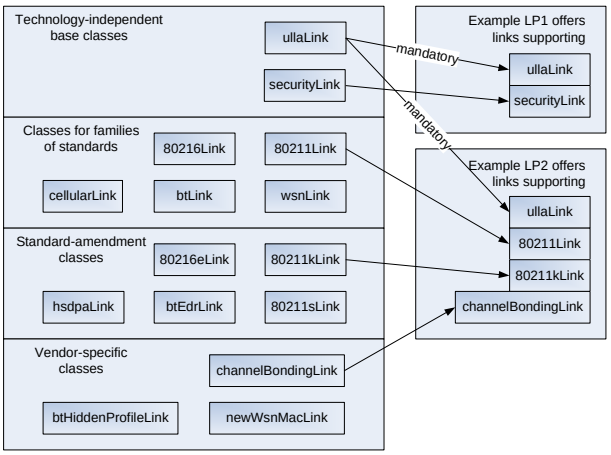

Figure 2: ULLA classes for links.

2.2 Link User interface

The LU interface offers three main functions: Queries, Notifications and Commands. Queries enable LUs to retrieve any attribute from any available class. Following the database abstraction introduced above LUs use database queries to describe the requested information. The ULLA Query Language, or UQL for short, which is a subset of the well known Standard Query Language (SQL), enables LUs to flexibly construct their queries. The LU will call `requestInfo()` and provide among other parameter the query string. The example string `SELECT linkId, rxSignalStrength, txBitRate FROM ullaLink` results in the identifier, the received signal strength and the maximum achievable bitrate in transmit direction of all links that were registered with ULLA core. Moreover, LUs can also use conditions to specify more in detail what kind of links they are interested in: `SELECT linkId FROM ullaLink WHERE txBitRate > 1000000` will only report indentifiers of links offering at minimum 1 Mbps transmit bitrate.

Notifications based on the call `requestNotification()` also use the UQL syntax but take a callback function as another parameter which is called when the given UQL-condition is fulfilled. The LU can register for periodic notifications specifying an inter-notification period if he wants to monitor certain attributes. Furthermore, he can register for event-based notifications when choosing a period of zero. Event-based notifications inform the LU only when the specified conditions occur and enable the applications to react as soon as the environment changes. Examples might be video-streaming applications that change the stream resolution or the vertical handover management as described in section 3. As UQL also supports joined queries the UQL statements used in notifications can include attributes from multiple classes and thus specify rather complicated conditions.

The last feature, commands, allows LUs to start actions such as connecting a link. ULLA supports synchronous commands using the `doCmd()` function and also asynchronous commands using the `execCmd()` function.

2.3 Link Provider interface

In contrast to the LU interface that works on the level of classes, the LP interface deals with single attributes. A query asking for the signal strength of all registered links will be implemented using separate `getAttribute()` calls addressed to single LPs who registered links before. Additionally, LPs might be asked several times if they registered more than only one link. This can easily happen when, for example, a WLAN device is not connected but receives beacons from several APs. ULLA core will save the values provided by the LPs to ULLA storage and forwards the information to the LU.

Notifications can be implemented using `requestUpdate()` asking LPs to provide any updated value for one attribute. LPs will call the `handleEvent()` function offered by ULLA core in order to inform ULLA core about the availability of new measurement values. The implementation of the requestUpdate call depends on the type of LLA. A simple wrapper unit may only be able to periodically poll the existing legacy interface. Modern ULLA-enabled device driver will inherently know when new values are available and fire events only in these cases. Commands, the last major ULLA feature, are forwarded towards the LP after syntax checking was successfully performed by ULLA core.

3. MOBILITY SOLUTIONS USING ULLA

We shall now explain how to apply the Unified Link-Layer API in different types of mobility solutions. Our focus will be on enabling vertical handovers, as these have traditionally suffered the most from the lack of standard technology-independent APIs to access parameters related to handover decisions. Nevertheless, ULLA can also be used to simplify the design and implementation of handover agents for a single technology as well.

3.1 Multiparameter handovers

The traditional handover algorithms use the received signal strength as the main trigger for handover decisions. Other performance metrics could also be monitored to enhance the handover decision process. Examples for such attributes are listed below:

- Error rates measured in the down link based on different units such as bits, blocks, frames or packets.
- Number of retransmissions and respective ratios measured in the uplink.
- Level of interference and possible identification of interfering technologies.
- MAC congestion level.

Today's proprietary implementations already use several of those metrics not only for handover management but also for resource optimization. ULLA can be used to generalize this process by using platform- and where possible also technology-independent metrics in handover decisions. The flexibility of the ULLA and the applied class hierarchy allow very general but also hand-tailored and optimized solutions for single platforms. Link users can query for different metrics but can also combine several attributes in the

WHERE-condition and use the resulting query to register for an event-based notification. Such a notification could trigger the analysis and possibly the initiation of a handover. This way, ULLA easily enables multi-parametrical handover evaluations that could also be vertical handovers as described below.

3.2 Implementing IEEE 802.21 using ULLA

The IEEE working group 802.21 (WG 802.21) standardizes approaches to enable Media Independent Handover (MIH). The work focuses on wired and wireless solutions as specified in the IEEE 802-family of standards. Additionally, handovers to cellular systems are evaluated.

At present the IEEE 802.21 consists of three main services that are offered and implemented by the MIH function:

1. Media Independent Event Service (MIES)
2. Media Independent Command Service (MICS)
3. Media Independent Information Service (MIIS)

The MIES is the core functionality required to enable MIH and specifies several technology-independent link layer triggers such as Link Down or Link Going Down that have to be supported by every standard-compliant MIH function. Applications, such as the MIH function itself, can register for those triggers and will be alerted if the predefined conditions occur.

We have specified an implementation of the MIH on top of ULLA. The basis of the implementation is a new `ieee80221Link` class which includes several string variables that provide requests how the generic 802.21 events should be implemented for the given LP. The LU would at first query for the string-variables and later on use these strings as conditions when registering for notifications.

The final set of standardized events is not decided, yet, but in the following we list some examples for events that are described in the draft standard [7]:

1. Link Up
2. Link Down
3. Link Going Down
4. Link Parameters Change
5. Link Handoff Complete

The last example event Link Handoff Complete includes higher layer information and thus would have to be implemented by the MIH function running on the mobile device. The former four events can easily be included in the new `ieee80221Link`-class using attributes such as `linkUpClass` or `linkGoingDownCondition`. As IEEE 802.21 events need rather complex queries we implement those as joined queries and include conditions as well as required classes as string-attributes to the newly introduced class. For example, to obtain the (technology-specific) condition clause for the Link Going Down event, the link user makes the call (assuming "C" programming language)

```
sprintf(query,
  "SELECT ieee80221Link.linkGoingDownCondition
   FROM ullaLink, ieee80221Link
   WHERE ullaLink.linkId=%d;", linkId);

error = requestInfo(query, &myResult);

ullaResultStringValue(myResult, 1, &condition, LEN);
```

At this point the string `condition` (of length `LEN`) contains the appropriate condition clause the LU can use when registering for the notification corresponding to the Link Going Down event. The classes to be used in the FROM clause can be retrieved in a similar manner:

```
sprintf(query,
  "SELECT ieee80221Link.linkGoingDownClass
   FROM ullaLink, ieee80221Link
   WHERE ullaLink.linkId=%d;", linkId);
error = requestInfo(myId, query, &myResult);
ullaResultStringValue(myResult, 1, &class, LEN);
```

The actual registration is performed using the following function calls. First, we compose the final condition string:

```
sprintf(query,
  "SELECT ullaLink.linkId, ullaLink.rxSignalStrength
   FROM ullaLink, %s
   WHERE ullaLink.linkId=%d AND %s;",
     class, linkId, condition);
```

Then, we perform the actual request for notification:

```
RnDescr_t myNotification;
myNotification.count = 0; // valid indefinitely
myNotification.period = 0; // event-based
// handler is a pointer to the LU function that
// is called when notification is triggered:
myNotification.handler = &handleLinkGoingDown;
myNotification.privdata = NULL;
int rnId;

error = requestNotification(&rnId,
          query, myNotification);
```

In the case of IEEE 802.11, the returned condition attribute could, e.g., have the value `ieee80211Link.txRetryRatio>0.4`, and the involved classes are `ullaLink` and `ieee80211Link`. Since `ullaLink` is always included we simply add `ieee80211Link`.

Besides the MIES the draft standard also describes the Media Independent Command Service (MICS). Although the name indicates similarities to the ULLA Command Processing (UCP) the MICS is not as powerful as the respective UCP features. The MICS is mostly designed to enable the MIH functions to request lower layers to perform certain measurements or simply provide latest measurement results.

Such functionality is covered by the ULLA Query Processing (UQP) and simple configuration updates can be realized using the LU interface function `setAttribute()`.

The MIH function uses the third 802.21-service, the MIIS, to gather information about available networks and their capabilities, which are described using an XML-based RDF scheme. ULLA could easily be extended to support also the MIIS. The description of the available networks and their capabilities is already supported by the ULLA framework as it is defined now. The additional support for the RDF-syntax can be added to the `ieee80221Link`-class introduced above. New attributes in this class could provide all the needed information using the syntax and semantic finally standardized by the IEEE WG.

ULLA is very appropriate for implementing 802.21-based MIH. All required features are supported by the latest ULLA or could easily be added when the final 802.21-standard is available. The proposed new class `ieee80221Link` would include all needed extensions but cannot be defined before the final standard is ratified.

3.3 Further applications

ULLA can also be used to enhance other handover types considered especially in the research community. Location-based or location-assisted handovers can be facilitated by using ULLA to create technology-independent location services. This can either be accomplished by utilizing classical tracking and localization algorithms based on, for example, received signal strengths from multiple fixed nodes of known locations, or other physical or link-layer related positioning information. Especially technologies based on Ultra-Wideband principles could prove interesting in the latter respect.

Mesh networks are another active area of research related to mobility solutions. In mesh networks the change of point of attachment is typically not done via separate handover procedure, but is performed by the same routing algorithm that is used throughout the mesh infrastructure. Based on the experiences gathered by the ad hoc routing community, routing solutions for meshes typically use more advanced routing metrics than simple hop counts. However, implementing such routing agents in portable manner has so far been impossible due to the differences between the programming interfaces discussed earlier. The abstraction of different routing and handover -related characteristics offered by ULLA can be used to correct this, and would enable existing mesh solutions such as IEEE 802.11s [4] be implemented in technology-independent manner. For further information about mesh networking the reader is referred to the review by Raffaele Bruno et.al. in [5].

4. PROTOTYPE IMPLEMENTATION

In order to evaluate the suitability of ULLA architecture for mobility solutions, we have developed the prototype on the Linux operating system based on the Red Hat kernel 2.4.26. The ULLA is implemented in both user-space and kernel-space as shown in Figure 3. The ULLA core module resides in the kernel-space and uses the LP interface provided by the LLAs whereas applications residing in the user-space use the LU interface via a shared library. This split is completely transparent to both LUs and LPs. The rationale of splitting the ULLA core is to deal with the high number of function calls on the LP interface. For instance, the getAttribute() method may be used multiple times if the UQL string carries more than one attribute. A context switch for each attribute would introduce a high latency in the query processing. This is particulary crucial when a notification with long conditions has to be evaluated periodically. The `getAttribute()` method can be called from interrupt context, but the evaluation of the notification is scheduled as a task.

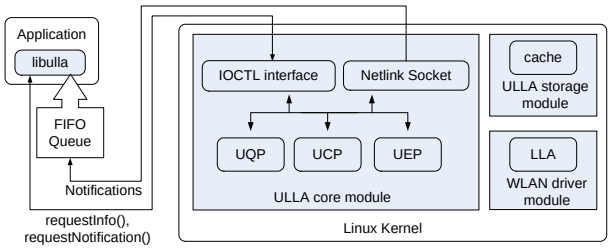

Figure 3: Architecture of Linux implementation.

Application programmers can access the LU interface by dynamically linking to the respective library. This provides the set of functions to perform queries (`requestInfo()`) or register for notifications (`requestNotification()`). The UQL-parser contained in the library is used to analyse these query strings requested by the LUs and to generate an abstracted tree-representation that is forwarded to the ULLA kernel module. The parser is generated by using flex and yacc with a grammar similar to the well-known Baccus-Naur-Format (BNF). Furthermore, it also implements helper functions to enable the application to retrieve the query result later on. The actual evaluation of the queries and notifications is left for the kernel module.

Communication between the ULLA library and the ULLA kernel module which is hidden from any LLA and LU is provided via two types of Linux kernel-to-user space mechanisms. Asynchronous and synchronous mechanisms are utilized and the implementation uses a specific ULLA-netlink socket and an ULLA character driver, respectively. The ULLA netlink socket is derived from the basic netlink socket support in the kernel, which provides the possibility to send netlink messages between user- and kernel-space via a FIFO queue per application. These messages are used for asynchronous communication as needed for the notifications from kernel to user-space. The synchronous ioctl mechanism of the character driver is used for queries and commands. Finally, a proc file interface can be used to observe known classes or registered notifications of the ULLA core.

The internal structure of the ULLA core is neither exposed to the LLAs nor to the LUs. For instance, LUs can register a callback function for notifications by the means of the address of their user-space function. This handler is called from within the library code once a notification is fired. From the LLA perspective the user-space part is not visible. The `getAttribute()` method allows the LLA to

write its data into a kernel-space buffer without any concerns about memory accessibility.

The optional storage of the ULLA core is a proprietary data structure which is used to cache attribute values. It was implemented as an extra kernel module. It does not use any data base backend but the ULLA design allows ULLA developers to choose different implementation options because the final choice is transparent to LUs and LPs. Additionally, it allows for future extensions towards a historical storage.

In addition to the ULLA core implementation, an 802.11 LLA is also developed for Linux. It is a wrapper of the wlan-ng driver for the prism2 chipset [8]. In order to test the LLA functionality, the LLA is ported into the prism2cs driver module as an ULLA-enabled driver and tested with the Netgear MA-401 PC-card. It abstracts the NIC with a link provider class to ULLA and reports links with the according link class description to ULLA if other peers or access points are visible via the supported NIC. The selected driver is capable of supporting multiple NICs at the same time, resulting in the multiplexing of multiple LPs and corresponding links. The LLA is loaded when a NIC is inserted by triggering the Linux PC-card services routine to load the ULLA-enabled wlan-ng driver. It also automatically loads the ULLA core if not exist yet. The LLA module also features an LP for testing purposes, capable of accepting dummy data sets via a test library from user-space without any use of hardware.

Even though the prototype was not extensively optimized for embedded use, the performance obtained is completely satisfactory for mobility applications. The memory footprint of the prototype is on the order of 185 kB, which is minimal compared to even the basic linux kernel overhead. In a laptop environment, typical queries for 2-8 attributes take less than 0.2 ms and added latencies for notification processing are also in the millisecond range. These results show that even when extrapolating to less powerful mobile terminal, the perfomance of the prototype implementation is already completely satisfactory to serve as a basis for portable handover agent implementations.

5. CONCLUSIONS

One major practical problem for enabling seamless vertical handovers is the diversity of link-layer interfaces between different technologies as well as operating system platforms. In this paper we proposed the Unified Link-Layer API (ULLA) as a solution and showed how it can efficiently be used to implement vertical handover solutions that are currently under standardisation. The performance of the Linux prototype implementation proves to be well acceptable for mobility management use cases. Moreover, the flexible and extensible design makes ULLA an important enabler for more robust networks and innovative services. ULLA scales down to wireless sensor devices showing that the usage of ULLA on mobile and resource-limited devices is feasible. The ULLA Linux implementation is available as open source from [3].

6. ACKNOWLEDGMENTS

We would like to thank DFG, 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.

7. REFERENCES

[1] *The GOLLUM-project website*, http://www.ist-gollum.org [Cited on: 5th of May 2006], 2004.

[2] *UMA Technology website*, http://www.umatechnology.org/ [Cited on: 5th of May 2006], 2005.

[3] *Open source release of the Unified Link-Layer API*, http://ulla.sourceforge.net/ [Cited on: 4th of May 2006], 2006.

[4] *Status of IEEE task group 802.11s ESS Mesh Networking*, http://grouper.ieee.org/groups/802/11/Reports/tgs_update.htm [Cited on: 5th of May 2006], 2006.

[5] R. Bruno, M. Conti, and E. Gregori. Mesh Networks: Commodity Multihop Ad Hoc Networks. *IEEE Communication Magazine*, 43(3):123–131, March 2005.

[6] L. Dimopoulou, G. Leoleis, and I. S. Venieris. Fast Handover Support in a WLAN Environment: Challenges and Perspectives. *IEEE Network*, 19(3):14–20, May/June 2005.

[7] IEEE Computer Society LAN MAN Standards Commitee. Draft IEEE Standard for Local and Metropolitan Area Networks: Media Independent Handover Service. In IEEE P802.21/D00.01, July 2005.

[8] Intersil Corp. Prism Driver Programmers Manual RM025.3, Version 3.0. `http://calradio.calit2.net/calradio1/rfboard/datasheets/RM025-prism-dpm-ver3d0.pdf`, July 2003.

[9] N. Montavont, E. Njedjou, F. Lebeugle, and T. Noel. Link Triggers Assisted Optimizations For Mobile IPv4/v6 Vertical Handovers. In *Proc. of the 10th IEEE Symposium on Computers and Communications (ISCC 2005)*, 2005.

[10] J. Riihijärvi, M. Wellens, P. Mähönen, and A. Gefflaut. Implementation and Demonstration of Unified Link-Layer API. In *Proc. of INFOCOM'06*, Barcelona, Spain, 2006.

[11] T. Farnham *et al*. Unified Link Layer API: Design and Initial Implementation Results. In *Proc. of IST Mobile Summit*, Mykonos, Greece, June 2006.

[12] A. Wolisz (editor) and A. Festag. Optimization of Handover Performance by Link Layer Triggers in IP-Based Networks; Parameters, Protocol Extensions, and APIs for Implementation. Technical report, Telecommunication Networks Group, Technische Universität Berlin, July 2002.