Implementation and Demonstration of Unified Link-Layer API

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

Alain Gefflaut
European Microsoft Innovations Center
Ritterstrasse 23, D-52072 Aachen, Germany
Email: alaingef@microsoft.com

Abstract— **The characteristics of wireless links can change abruptly leading to challenges in achieving stable application behaviour. However, no common link-layer independent interface is available to monitor and control wireless links in a convenient and technology-independent way. Instead, only solutions hand-tailored for particular technologies or higher-layer middleware is used today. In this poster we introduce the Unified Link-Layer API (ULLA) that allows technology- and platform-independent access to wireless devices to control configurations, monitor the performance, and register for event notifications. Hereby ULLA enables efficient cross-layering for diverse applications. We also describe two prototype implementations, based on Linux and Windows CE and will demonstrate the latter one together with a network monitoring tool.**

I. Introduction

In recent years support for wireless communication has become more and more commonplace. Additionally, the variety of available technologies has constantly increased leading to more and more heterogeneous environments. It is very common for modern mobile terminals to offer more than two wireless interfaces. The involved complexity increases and, e.g., automatic connection management is not an easy task anymore. Furthermore, APIs to control different network interfaces vary highly between technologies. In this poster we present the *Unified Link-Layer API* (ULLA) as being defined by the GOLLUM project [1] that offers a common interface to access parameter settings as well as performance metrics of different wireless technologies[1].

The characteristics of wireless communication systems is inherently unstable because of the varying nature of the wireless channel. Considerable amount of research has been performed in the area of, e.g., content adaptation or handover control in order to cope with these effects and provide stable application layer performance for good user's experience. However, one major problem in this area is the difficulty of gathering link-layer information in a reliable and consistent way. Due to this, link-aware applications are rare and if available at all specifically tailored to certain scenarios. ULLA offers convenient query-mechanisms to retrieve link-layer information in a technology-independent way. Additionally, applications can register for asynchronous notifications to be informed upon previously defined events in the environment. This way efficient cross-layering can be implemented and applications can react as soon as the wireless interface fires the notification.

Another major advantage of the ULLA approach is platform independence. The same API can be offered on Windows-, and Linux-based systems as shown by the prototypes discussed later. The design scales well and also a version for constrained devices such as wireless sensor nodes is under development.

A. Potential applications

Applications that would benefit from ULLA are diverse. Besides operating system agents offering connection management services, multimedia services using content adaptation are typical examples. However, also ad hoc routing protocols and thus entities not working on the application layer can benefit from ULLA. To the best of our knowledge routing metrics relying on link performance metrics such as error ratios were only examined in homogeneous environments but ULLA would allow to use these enhancements independently of the underlying technology. Also events such as defined in the IEEE 802.21 draft standard for media-independent handover can be implemented using ULLA [2].

II. ULLA Architecture

The ULLA architecture is depicted in figure 1. It consists of two interfaces, the *Link User* (LU) interface and the *Link Provider* (LP) interface. The LUs are applications of all kinds that use the ULLA and LPs are the abstraction of network interface cards offering communication facilities. As these names suggest ULLA works on the abstraction level of *links* that are provided by LPs and used by LUs. The ULLA Core connects both and is the main block in the whole framework managing available links.

In order to interconnect legacy device drivers with ULLA, *Link Layer Adapters* (LLAs) are introduced that implement the technology-independent LP-interface in a technology-specific way adapted to certain driver characteristics. Future newly developed device drivers can implement the LP-interface directly and thus avoid the need for an additional wrapper-entity. The LLA mainly contains technology-specific implementations for computing common abstract attributes such as available bandwidth. Hence, reliable and optimized measurement algorithms can be used but LUs and their programmers do not need any knowledge about these technology-specific details.

[1]Although we purely focus on wireless communication in this poster, the ULLA design also fully supports fixed communication.

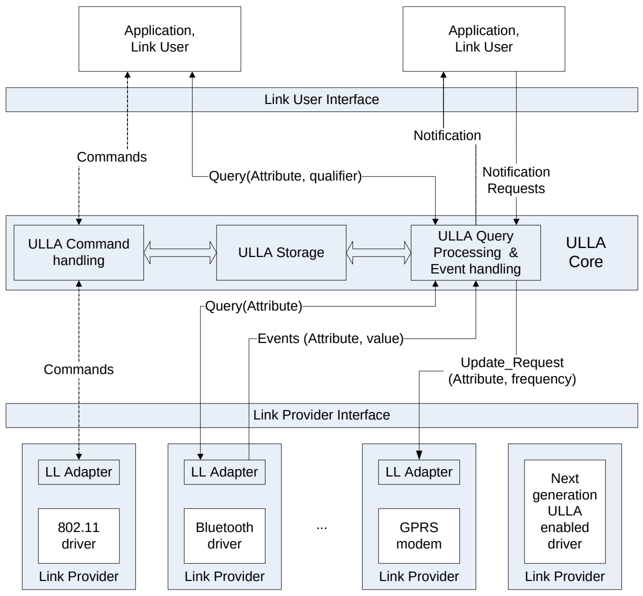

Fig. 1. ULLA architecture.

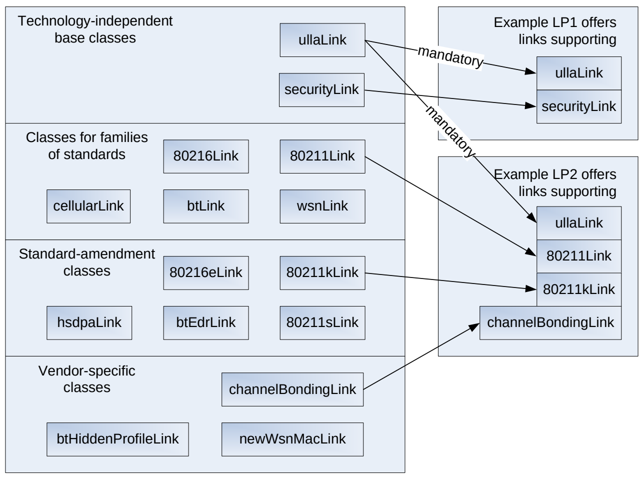

Fig. 2. ULLA link classes.

The interaction between ULLA and LUs is based on three mechanisms which are used to control LPs. Commands are used to trigger actions such as scanning for available links. These can either be synchronous or asynchronous. Queries are used by LUs to gather information related to configuration as well as performance metrics. The used *ULLA Query Language* (UQL) is a subset of SQL. This is natural as the ULLA-internal data modelling is based on a database abstraction of the communication environment. For example, the query *SELECT linkId FROM ullaLink WHERE txAvailableBitrate > 500000* requests the identifiers of all links offering more than 500 kbps.

Using the same kind of UQL-statements LUs can also register for notifications on certain conditions. Such a condition could, e.g., resemble an 802.21-event that later on triggers a handover action. On the other hand content adaptation capable multimedia services could request notifications on changes of delay or throughput. The ULLA notification mechanism is very flexible because the LU can define its own conditions using different attributes and thresholds. However, including reference queries for typical use cases is also feasible.

A. Extendibility

One major requirement for a future-proof API is extendibility which was taken into account during the ULLA-design process from the beginning. Upcoming technologies such as WiMAX, UWB or IEEE 802.20 can be supported by adding a new class to the object oriented abstraction of different technologies. Figure 2 shows the taken approach which is related to the well-known object-oriented principles. Every LP has to support the mandatory base class *ullaLink*. There is another base class for security-related parameters which can optionally be supported.

The further levels as shown in the class hierarchy resemble technology-specific classes that include attributes and commands that are common to all links based on one technology. Recently several standard amendments were standardized to improve some of the existing wireless technologies. Such extensions are not supported by all devices using this technology so that these classes form the next level in the hierarchy. The lowest level of abstraction is used for vendor-specific functionality that is proprietary but still should be controllable via ULLA. A vendor can define its own class and support it via its own LP enabling any LU to take advantage and use this added functionality. We expect this to be particularly useful for configuration and diagnostic tools.

Every LP will implement a subset of all available classes as shown in figure 2 for two examples. However, every link still has to support the ullaLink base class which ensures that basic attributes can be accessed in a uniform way. Applications whose programmers chose to work on this common abstraction level automatically work unchanged in heterogeneous environments and are highly portable also because of the platform independence of ULLA.

III. Implementation and Demonstration

We implemented ULLA on several platforms to evaluate different design options. In this poster we present two representative working ULLA versions, one based on a Linux-platform, the other one using Windows CE. Both use the same API proving the platform independence.

A. Linux

The Linux-implementation is based on kernel version 2.4.26 and runs on a standard notebook. The ULLA core is mostly placed in kernel space but some smaller parts such as the UQL-parser are left in user space. Instead of an existing database core we designed our own small-footprint solution. The prototype supports WLAN-devices based on the IEEE 802.11b standard. The respective LLA is based on the wlan-ng open source device driver [3] for Prism2 and related chipsets. Implementing the LLA as part of the device driver proofs the feasibility of future ULLA-enabled drivers.

The memory footprint of the ULLA core including the LLA is about 185 kBytes which would also be acceptable for mobile devices. The power consumption read out via standard ACPI-functions offered by the Linux operating system did not show any difference during an one hour test no matter whether ULLA was enabled and running or disabled.

During first performance evaluations we measured the duration of one single query of increasing complexity. The UQL supports the definition of a validity of the requested attributes. If the value available from the optional ULLA storage is newer than the specified validity ULLA will return those cached values. Otherwise ULLA will forward the query to the LLA and retrieve up-to-date information. For the latter case the performance clearly depends on the type of the attribute. Requests asking for, e.g., 8 attributes that can be answered by the device driver take up to only 50 μs. In contrast, requests which require firmware access need up to 4 ms for 8 attributes. Even this time is completely acceptable for any foreseen application.

B. Windows CE

The second implementation runs on various versions of Windows CE, namely Windows Mobile 2003 and 2005. The ULLA core was successfully tested on PDAs as well as smartphones. On both hardware platforms the setup includes three wireless interfaces, GPRS, Bluetooth and WLAN, showing the technology-independence of the ULLA design.

The footprint of the implementation is approximately the same size as the Linux-implementation and also the power consumption test showed that ULLA does not overload the system. The additional power consumption over half an hour added by an ULLA evaluating notifications every 10 ms was near to the measurement accuracy of the Windows CE power management. A simple query asking for, e.g. 2 attributes, takes approximately 500 μs which shows that using ULLA for multimedia applications is a realistic solution.

C. Demonstration

We would like to show the Windows CE prototype during the demonstration session. It is based on a mobile device making it easy to show interested conference participants the scalability of the chosen approach and allows guests to use the device themselves. In addition to the PDA and the smartphone we would require space for one notebook to offer a WLAN ad hoc network for demonstration purposes. We do not need Internet access although it would be preferable. In case of depleting batteries standard continental European power plugs for both devices would be helpful.

The demonstration application is a network monitoring tool shown in figure 3. It retrieves information about all available links from ULLA and offers this information via graphical user interface. As part of the demonstration we walk the interested attendees through the source code of the implementation step by step to further highlight how easy ULLA is to use. Additionally, we show a preliminary version of a connection manager that connects the link offered by the notebook-PC and configures higher layer protocol settings.

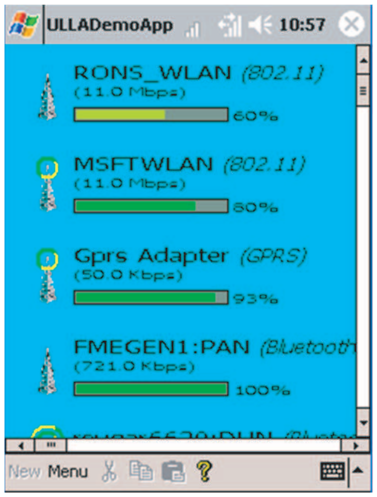

Fig. 3. Screenshot of the network monitoring demonstration tool using the Windows CE based ULLA.

IV. Conclusions

In this poster we presented the Unified Link-Layer API (ULLA) for easily retrieving information about available wireless links in mobile environments in a technology-independent manner. The ULLA also supports configuring links and registering for notifications on changes of link conditions. We described the design and architecture of ULLA and gave details about two of the existing ULLA prototypes. We will demonstrate a network monitoring tool based on the Windows CE implementation to show parts of ULLA functionality. The presented Linux implementation is available for download as open source from [4].

Acknowledgment

We would like to thank DFG (Deutsche Forschungsgemeinschaft), European Union (the GOLLUM-project, [1]) and RWTH Aachen University for the financial support. We would also like to thank the GOLLUM research team for fruitful discussions.

References

[1] *IST GOLLUM project website*, http://www.ist-gollum.org [Visited Jan. 17, 2006], 2006.
[2] 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.
[3] Absolute Value Systems Inc. (AVS), “wlan-ng driver implementation,” http://www.linux-wlan.org/ [Visited Jan. 18, 2006].
[4] *Open source release of the Unified Link-Layer API*, http://ulla.sourceforge.net/ [Visited Mar. 29, 2006], 2006.