

UNIFIED LINK-LAYER API ENABLING WIRELESS-AWARE APPLICATIONS

Marten Bandholz*, Alain Gefflaut†, Janne Riihijärvi*, Matthias Wellens*, and Petri Mähönen*

*Department of Wireless Networks, RWTH Aachen University
Kackerstrasse 9, D-52072 Aachen, Germany

†European Microsoft Innovations Center
Ritterstrasse 23, D-52072, Aachen, Germany

ABSTRACT

Present mobile devices often support multiple communication technologies. If software developers want to control the configuration or monitor the characteristics of active links their applications will have to use different programming interfaces for each technology and supported platform. We propose an architecture for a Unified Link-Layer API (ULLA) to abstract common metrics and introduce a single interface for retrieving link-layer information. ULLA offers a flexible query interface and a powerful notification mechanism that enable applications to become link-aware. Additionally, special attention was paid to keep the framework extendible for emerging networking technologies. We have implemented prototypes for a Linux notebook and a Windows CE PDA in order to prove the feasibility of the approach. Memory footprint, query duration, and power consumption tests show that ULLA can also be deployed on mobile, resource limited devices.

I. INTRODUCTION

Wireless communication is becoming more and more commonplace. At the same time the number of wireless technologies supported on a device is constantly increasing leading to multi-mode terminals and a need to deal with network heterogeneity. Innovative applications gain from these new possibilities but the involved design complexity gets higher because of several reasons. First, the interfaces used to retrieve link-layer information drastically differ between technologies. Application programmers who want to run their implementation over, e.g., WLAN as well as Bluetooth have to implement the network-related part for each technology from scratch. Second, the flexibility and functionality of these interfaces is severely limited. For example, link-aware applications have to monitor characteristics of the wireless connection themselves. Operating system entities offering a flexible interface to track the frequently changing conditions in the wireless communication are not available today. Third, measurement results available from present device drivers are rarely comparable. Often only very specific and detailed data can be read out such as typical parameter settings but no performance metrics are offered. Finally, interfaces vary not only between technologies but also between products, hardware platforms, and operating systems.

In this paper we propose the Unified Link-Layer API (ULLA) as a solution to these problems. The ULLA is a new API that enables technology-independent configuration and flexible monitoring of lower layer settings and charac-

teristics. The framework abstracts common metrics such as throughput or latency without preventing advanced applications from accessing technology-specific attributes such as the RTS-threshold used in IEEE 802.11-based WLANs. ULLA offers a powerful query mechanism that enables applications to gather link-layer information in a flexible and usable manner. Additionally, applications can register for notifications and define specific conditions depending on the actual use case. ULLA will notify the application when these conditions are fulfilled enabling link-event based programming.

We have implemented two prototypes supporting several link-layer technologies using Linux and Windows CE platforms. The implementations turn out to be lightweight and fast showing that the ULLA approach is also feasible for small and embedded devices, which is important for mobile services. The work was carried out in the context of the European Union GOLLUM research project [1].

Many applications could benefit from an interface such as the ULLA; examples can be categorized into two classes. The first class of applications are Multimedia applications sensitive to varying Quality of Service (QoS), such as video conferencing tools, or VoIP-clients. They can benefit from link-layer information that is abstracted from the underlying link characteristics, thus enabling technology-unaware cross-layer optimizations. The second class of applications have an in-depth knowledge about the available link-layer technology, such as Connection Managers supporting vertical handovers in heterogeneous access networks [2]. Additionally, it is well-known that ad hoc routing protocols should consider technology dependent link characteristics when choosing routes [3]. Also, ULLA makes it easier to implement approaches for link assisted handovers discussed in standardization groups such as IEEE 802.21 [4].

Individual research projects have implemented simple convergence layers to hide the differences between link layers in use (see, for example, [5]). However, existing solutions are often missing important features to enable flexible usage. The Wireless Extensions [6] available in Linux systems offer a configuration interface and a well-defined way to gather performance numbers but they are limited to WLAN devices and extension to other technologies is not foreseen. Linux interfaces for other technologies, such as BlueZ in the case of Bluetooth, are also too specific. In Windows some technologies are grouped so that ethernet-like technologies are interfaced through the Network Device Interface Specification (NDIS) API, modem-like systems use the Telephony Application Programming Interface (TAPI) and Bluetooth devices

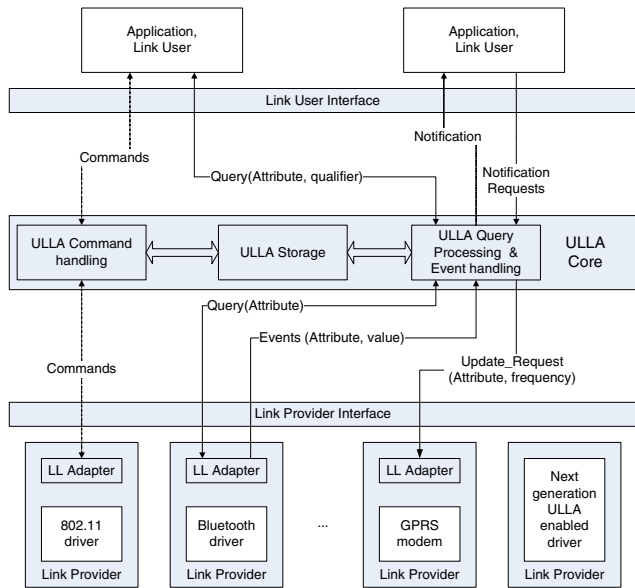


Figure 1: ULLA architecture.

are controlled using the respective Bluetooth-stack. In addition, the Windows Management Interface (WMI) spans over several technologies and is a first step towards technology-independence. However, it still manages specific technologies in slightly different ways and mainly works with Object Identifiers (OIs) which is less comfortable compared to the attribute based ULLA. Besides, these technologies are only available in Microsoft Windows operating systems and thus clearly platform-dependent.

After describing the architecture and the involved design choices in section II we continue in section III with a short description of the prototype implementations based on a Linux notebook and a Windows CE PDA. We evaluate those implementations in section IV and conclude the paper in section V.

II. ULLA ARCHITECTURE

The most important requirement of all applications using ULLA is the ability to acquire link-layer information across the technology boundaries. Therefore, the *ULLA Core* acts as an intermediate entity between *Link Users* (LUs) and *Link Layer Adapters* (LLAs). As shown in Fig. 1 the LUs are the applications¹ taking advantage of ULLA. LLAs are the software entities providing technology specific information about the associated Network Interface Cards (NICs), also called *Link Providers* (LPs), and hosted links.

Hence, the resulting API has two parts, the *LU interface* offered by the ULLA Core and utilized by LUs and the *LP interface* used to coordinate the interaction of ULLA Core and LLAs. The use of one interface is completely independent of the other one enabling, e.g., a LU to transparently access link-layer information from multiple LLAs. Future de-

¹The term application is used in a broad context and not strictly limited to OSI-layer seven. Thus, ad hoc routing agents or other software entities not working on layer seven can also act as LUs.

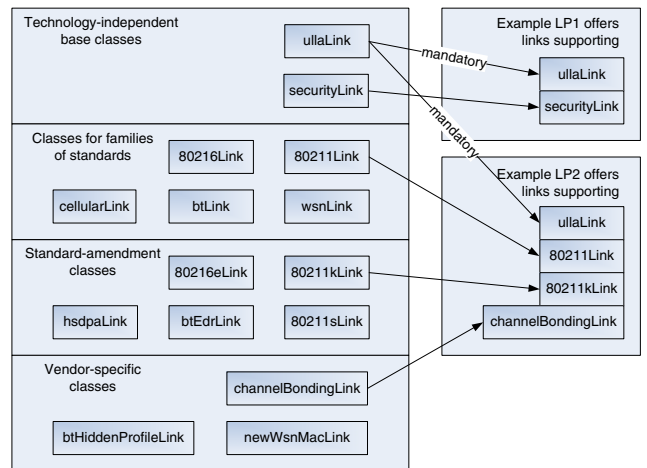


Figure 2: ULLA classes for Links.

vice drivers might inherently support ULLA and integrate the LLA-functionality but during the transition period an additional wrapper entity is used.

A. Data Model

The main entities ULLA works with are *Links*, which are described with classes using an object oriented approach. Member functions are called *commands* and data members are called *attributes*. Fig. 2 shows how different classes are combined to describe the specific features supported by one Link. On top of the hierarchy two base classes, *ullaLink* and *securityLink*, are offering a high abstraction level that enables to work with common attributes such as *rxBitRate* or *encAlgKeySize* describing the received bitrate and the size of the applied encryption key. Supporting *ullaLink* is mandatory for all Links, targeting the requirements of the first class of LUs, which work with abstracted link information.

On the next level of abstraction the Links may provide classes for attributes common to a well-known standard supported, such as IEEE 802.11 or GSM. In addition, several more specific classes are available at lower levels of the class hierarchy. These describe smaller and more specific standard amendments such as support for Radio Resource Measurements as specified in IEEE 802.11k. The lowest level offers classes which are specific to the devices developed by the same manufacturer or even specific for a single product. Out of the second class of LUs Configuration tools that are aware of such classes can configure important details while using the same convenient LU interface.

In addition to the notion of Links ULLA also works with the registered *Link Providers* (LPs), which are described using a similar kind of class hierarchy. The same approach as described for Links is followed demanding a second mandatory base class *ullaLinkProvider*. Included attributes refer to the NIC itself. For example, the firmware version is an attribute of *ullaLinkProvider*, whereas the time used for scanning the IEEE 802.11 channels is an attribute of *80211LinkProvider*.

B. Extendibility

Each LLA informs the ULLA Core during its registration process about all supported classes enabling the ULLA Core to check the conformance of any LU-request. Beforehand, the ULLA Core implementation does not have to be aware of the internal structure of any class making extensions easy. This feature enables a legacy ULLA Core to interwork with future LLAs offering upcoming technologies such as Ultrawideband or even completely new technologies.

C. Link User Interface

The LU interface is implemented by ULLA Core and offers three main features:

- Queries
- Notifications
- Commands

All features provide a database abstraction of the already described data model. Instances of the described classes are seen as tables with one entry per available Link or LP supporting the class. Following this approach queries are formulated using the ULLA Query Language, UQL for short, which is a subset of the popular Structured Query Language (SQL).

LUs call the *requestInfo* function when asking for an attribute value and provide a UQL-string amongst other parameters to specify the query. For example, *SELECT linkId FROM ullaLink WHERE txLatency < 400* will result in a list of Link identifiers of all available links with an up-link latency smaller than 400 ms. Combining different classes in one query is supported by joined queries. The query *SELECT 80211Link.dot11RtsThreshold FROM ullaLink, 80211Link WHERE ullaLink.linkId = 4* will report the RTS-threshold used by the Link with identifier four. LUs can also include a *validity* in queries specifying the minimum requirement in terms of up-to-dateness of the requested values. Queries asking for a validity of zero will always be forwarded to the LLAs but less strict constraints may be answered with values cached in the *ULLA Storage*.

The ULLA notification mechanism offers powerful features when changes in the network should be detected, which is a typical use case for multimedia services or handover management. The *requestNotification* call requires amongst others a UQL-string and a callback function pointer as parameters. When the given condition, specified in the WHERE-clause of the request, is fulfilled the callback function is invoked enabling the LU to take respective actions. If a multimedia conferencing application requires at least 1 Mbps throughput and uses Link number three it will register a notification using the query *SELECT rxBitRate FROM ullaLink WHERE (linkId = 3) AND (rxBitRate < 1000000)*. If the bitrate goes below 1 Mbps the application will be notified and can, for instance, lower the video frame rate. In addition to such event-based notifications ULLA supports periodic notifications that are fired for a certain number of times separated by a configurable inter-notification period. LUs can follow the changes of one attribute for more detailed historical analysis or other monitoring purposes.

Commands are the third feature offered via the LU interface. LUs use the *doCmd* call to request a command such as *enable* for LPs or *connect* for Links. Using *lpId* or *linkId* the LU addresses the command and the ULLA Core forwards it after checking the syntax with the classes registered by the corresponding LLA.

D. Link Provider Interface

The class model is utilized differently for the LP interface. In order to organize the work, an LLA has to do for one query, efficiently, the functions of the LP interface are based on single attributes. The ULLA Core can use *instantaneous data retrieval*, *event-based data retrieval*, or *command execution* to communicate with the LLAs.

Queries are broken down to a sequence of *getAttribute* calls. Notifications are usually based on a condition, which the ULLA Core disassembles to use the *requestUpdate* function for each relevant attribute. The LLA then reports changes in the attribute value to the ULLA Core². Upon reception of an attribute update the ULLA Core will evaluate all related pending notifications and fire if the conditions are fulfilled. Commands are forwarded using the *execCmd*-call when the mentioned checks have been successfully performed.

At last, the LLA calls *registerLp* to inform the ULLA Core about newly available LPs and their supported classes. Upon deregistration of LPs *unregisterLp* is used and the ULLA Core deletes all stored values for those entities. Respective functions for Links complete the LP interface.

III. EXAMPLE IMPLEMENTATIONS

In order to evaluate the ULLA architecture on multiple platforms we implemented prototypes for Windows CE and Linux. Due to the similarities in the APIs the Windows CE implementation has also been ported to Windows XP. In addition to the ULLA Core we developed a WLAN LLA for Linux and WLAN, GPRS and Bluetooth LLAs for Windows CE. For Windows XP we have a WLAN LLA and an Ethernet LLA.

Application programmers using Linux or Windows can access the LU interface by dynamically linking to the respective library. This provides the set of functions to perform queries or register for notifications. Also, all helper functions to unfold the query result are implemented in the library on both platforms. On Linux, the library also hosts the UQL-parser, which is generated by flex and yacc using a grammar close to the well-known Backus-Naur-Format (BNF). On Windows the parser is contained in the main part of the ULLA Core.

On both platforms ULLA core is implemented as a driver. On the Linux platform a character driver is utilized, which is a kernel module dynamically linked to the kernel. On Windows the respective driver is a stream driver, which is running in the context of a specific process, the Device Manager³. Synchronous communication between ULLA Core and LU is car-

²Depending on the level of integration between LLA and device driver *requestUpdate* might also implement polling to inform ULLA Core about the latest values.

³On Windows XP the ULLA Core was implemented as a Component Object Model (COM) service.

ried out using ioctl-calls. Asynchronous communication used for notification messages requires netlink sockets on Linux and Message queues on Windows CE, both providing a datagram (message) abstraction.

LLA developers need to access the LP interface in a platform specific way because the implementation of device drivers depends highly on the operating system in use. Linux device driver vendors offering an LLA call the LP interface functions directly from within their kernel modules by the means of the dynamic kernel symbol table. These LLAs have direct access to the underlying hardware of the NIC via, e.g., register I/O [7]. On Windows the LLA programmers link to a Dynamic Link Library (DLL). These LLAs are using technology specific APIs to retrieve the attribute values via IOControl (WLAN), TAPI and AT commands (GPRS), or Bluetooth sockets (BT).

IV. PERFORMANCE EVALUATION

The main objective of the performance evaluation was to test the feasibility of the ULLA approach and the scalability down to resource constraint mobile devices and embedded systems. We present the hardware platforms in use in subsection A, an overview about the memory footprint in subsection B, and the observed query duration in subsection C. Finally, we comment on the power consumption in subsection D.

A. Prototyping hardware platforms

The Linux prototyping platform is based on a standard notebook using an Intel Pentium M clocked at 1.4 GHz and 512 MB of RAM. The Netgear MA 401 PC-card, based on the IEEE 802.11b Prism2-chipset, was the only wireless interface switched on during the measurements. The Linux kernel version 2.4.26 was deployed as part of a Red Hat 9 distribution using ACPI-support. For Windows CE, we use an HTC Himalaya Pocket PC Phone Edition 2003 device. The device uses an Intel XScale CPU (PXA263) with a clock frequency of 400 MHz, 128 MB of RAM and 32 MB of flash. For communications, the PDA also comes with an onboard triband GSM/GPRS modem and Bluetooth support. As the device does not natively support 802.11, we use an external SDIO wireless LAN card from Socket Communications.

B. Memory footprint

Table 1 lists the memory occupied by the evaluated prototype implementations, which is especially limited on small embedded devices. The envisaged functionality of the ULLA Core is fully implemented, whereas the LLAs feature only a subset of the possible amount of attributes and commands.

Altogether, the ULLA Core, the user library and the LLAs represent a static memory occupation of less than 200 kB. As mentioned in section III the Linux user library implements the UQL parser, whereas in the Windows implementation it is moved to the ULLA Core. The resulting difference in the respective memory footprint is listed in table 1.

Table 1: Memory footprint of ULLA in prototype implementations.

| ULLA component | Operating system | |
|----------------|------------------|----------|
| | Linux | WinCE |
| User library | 87.4 kB | 19.5 kB |
| ULLA Core | 57.7 kB | 72.0 kB |
| 802.11 LLA | 40.1 kB | 43.5 kB |
| Bluetooth LLA | N/A | 34.0 kB |
| GPRS LLA | N/A | 30 kB |
| Total | 185.1 kB | 195.0 kB |

C. Query duration

In order to estimate the additional delay ULLA introduces to the process of the retrieval of attribute values, we evaluated the *requestInfo* call. For both platforms we utilized a WLAN setup to be able to query the same attributes. We measured the duration of 2000 requests and obtained the averaged duration per call. The standard deviation is calculated from the averaged results of 10 experiments. The complexity is increased from 2 to 8 attributes in the UQL-statement and the number of integer and string attributes was chosen to be the same for all queries. The impact of the size of the returned result is evaluated by the number of WLANs present by the time of the request made (2 or 5 links).

In the next step we evaluated the performance gain of utilizing the ULLA Storage. If the LU uses a validity time greater than zero, the ULLA Storage is checked for a valid attribute value before getting it from the LLA.

Figures 3 and 4 show the first results. Most times are below 1 ms showing that the approach taken during the development of the ULLA performs well and does not introduce long delays. Query durations of 1 ms are also sufficient for applications such as multimedia content adaptation or handover management. Additionally, the caching of attribute values can clearly speed up the query process if the trade-off between timeliness and duration is well chosen by the LU.

D. Power consumption

A long battery lifetime is crucial for mobile devices. We evaluated the impact of the usage of ULLA on the battery level by deploying a simple test application requesting two attributes every 10 ms. We compared the resulting trace of battery degeneration with the idle trace just leaving the WLAN interface enabled. In spite of this high level of activity we were not able to conduct a significant difference in the recorded traces.

V. CONCLUSIONS

This paper introduced the Unified Link-Layer API offering a common interface to configure communication links and mon-

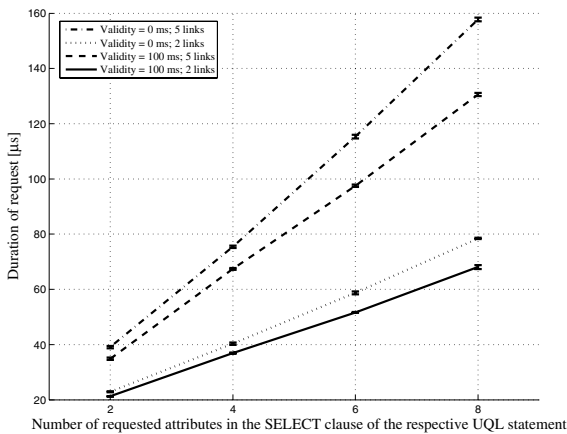


Figure 3: Duration of a query done using *requestInfo* measured on the Linux prototype.

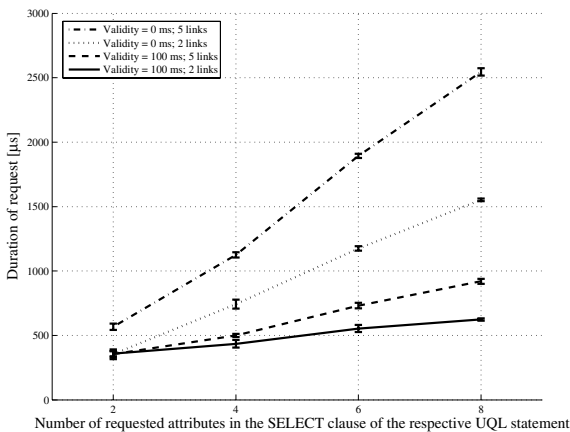


Figure 4: Duration of a query done using *requestInfo* measured on the Windows CE prototype. In this test the ULLA was implemented as a driver.

itor their characteristics. Using an object oriented representation ensures extendibility and enables applications to choose the abstraction level they want to work on. The query and notification features allow software developers to write link-aware applications without knowing details about networking or the finally used technology. The prototype implementations support several wireless interfaces and three operating systems proving the platform- and technology-independence of ULLA. The query duration tests show that introducing the ULLA framework does not add considerable overhead and that the use of ULLA for applications with soft realtime requirements is reasonable. The memory footprint and power consumption tests also indicate that ULLA is usable on mobile and embedded devices.

The described Linux implementation is available as open source from the sourceforge platform [8]. The release also includes an extended Helix Player from Real Networks [9] adapting its settings based on attributes read out from ULLA and utilizing the notification feature to be link-aware.

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: 15th of March 2006], 2004.
- [2] 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," Telecommunication Networks Group, Technische Universität Berlin, Tech. Rep., July 2002.
- [3] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris, "Performance of Multihop Wireless Networks: Shortest Path is Not Enough," in *Proc. of HotNets-I*, Oct. 2002.
- [4] IEEE Computer Society LAN MAN Standards Committee, "Draft IEEE Standard for Local and Metropolitan Area Networks: Media Independent Handover Service," In IEEE P802.21/D00.01, July 2005.
- [5] P. Mähönen *et al.*, "Platform-Independent IP Transmission over Wireless Networks: the WINE Approach," *IEEE Personal Communications Magazine*, December 2001.
- [6] *Linux Wireless Extensions*. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html [Cited on: 15th of March 2006]: Jean Tourrilhes, 2002.
- [7] 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.
- [8] *Open source release of the Unified Link-Layer API*, <http://ulla.sourceforge.net/> [Cited On: 4th of April 2006], 2006.
- [9] *Helix Community*, <https://player.helixcommunity.org/> [Cited On: 4th of April 2006], 2006.