Toward Open and Unified Link-Layer API

Tim Farnham, Alain Gefflaut, Andreas Ibing,
Petri Mähönen, Diego Melpignano, Janne Riihijärvi, and Mahesh Sooriyabandara

***Abstract*—This paper describes the motivation and first results from the work carried out in the European GOLLUM-project toward developing a open, extendible, and unified API for accessing link-layer functionality and information. Key features of this API will include a general querying mechanism based on database technologies, and methods for setting up asynchronous notifications regarding changes in link conditions, in a technology-independent manner. Applications for such an interface are numerous and cover domains such as mobility and network cross-layer optimizations.**

***Index Terms*—API, abstraction layer, link-layer events**

I. Introduction

In the present-day wireless world application programmers have to be very mindful of the platform they are writing the applications on. A program designed to work on a cellular phone or on a future smart mobile phone will probably not work without great modifications on a PDA equipped with a Bluetooth connection, or on a laptop using a wireless LAN. Direct portability for even more embedded devices would be almost unimaginable, and in particular most automation and control systems radio link layers are completely proprietary without any support from operating systems and application programmers. In part this is because of the large number of different operating systems in use. While unification is progressing in this sector, great problems remain on the other problem area, namely in the interface used to access the wireless access devices (or air interfaces).

Even when using the same or compatible operating system the methods used to access, say, a Bluetooth or GSM/WCDMA link, and a Wireless LAN differ. This difference becomes even greater when many of the small, embedded radios are considered. The situation becomes even more difficult if the application (or middleware) should somehow intelligently respond to some events or changes in the wireless channel. The necessary mechanisms are usually simply not there, and even in the cases that they are available, they again are different form one technology to the other.

With this background the necessary development seems clear: a unified API of sufficient generality and extendibility and corresponding embedded middleware, together with an embedded software reference implementation should be developed to unify access and information retrieval from various wireless and wired technologies. This is precisely what the GOLLUM-project aims to do.

II. Idea of Unified Link-Layer API

The purpose of the GOLLUM project is to propose and develop a software Application Programmers' Interface (API) that will hide network standards heterogeneity behind a common set of functionality applicable to all types of networks. We call such an API a Unified Link Layer API (ULLA). We believe that such an API is a big step in the direction toward intelligent radio aware software that will be able to accommodate multiple radio standards in a seamless way. The ULLA will help to resolve the complexity and interoperability problem related to the large number of different APIs and methods used for accessing communication interfaces, especially in the embedded domain. It will provide real and useful triggers, handles for different smart, context sensitive and link/network aware applications; enabling the development of "cognitive applications".

As a concept this is a well-known paradigm and goal. The problem is that no really acceptable and useful reference API has been provided in the public domain. ULLA will also provide abstraction and extendibility to permit different underlying wireless interfaces and networking technologies that exist now and will emerge in the future. Important emerging technologies are composite multi-mode radio and Software Defined Radio (SDR), and the flexibility of these devices to operate in different modes of operation and dynamically reconfigure will be made available through ULLA.

The ULLA provides an abstraction from specific link technologies to the applications or other link users (where link users can include any higher layer protocols, middleware or

Manuscript received February 6, 2005. This work was supported in part by DFG, RWTH Aachen, and European Union (GOLLUM-project, IST-511567).

Andreas Ibing, Petri Mähönen and Janne Riihijärvi are with the Department of Wireless Networks, Aachen University (RWTH), Kackertstrasse 9, D-52072 Aachen, Germany (corresponding author is Janne Riihijärvi, phone: +49 2407 575 7034; fax: +49 2407 575 7050; e-mail: jar@mobnets.rwth-aachen.de).

Alain Gefflaut is with the European Microsoft Innovation Center (EMIC), Ritterstrasse 23, D-52072 Aachen, Germany; email: alaingef@microsoft.com.

Tim Farnham and Mahesh Sooriyabandara are with the Telecommunications Research Laboratory, Toshiba Research Europe Ltd, 32, Queen Square, Bristol BS1 4ND, United Kingdom (e-mail: {tim.farnham, m.sooriyabandara}@toshiba-trel.com).

Diego Melpignano is with dept. of Advanced System Technology at STMicroelectronics v. Cardano, 1 - 20041 - Agrate Brianza - ITALY (email: diego.melpignano@st.com).

application software). It achieves this by regarding a link to be a generic means of providing a communication service. Links are made available and configured through link providers to permit abstraction from specific platforms and technologies.

The following high level requirements capture the main functionalities of ULLA.

1) Notify of appropriate link changes and statistics

Only inform the link users of events that the link users are interested in and with appropriate timeliness and granularity. For example, an application could be interested in the periodic link statistics or significant performance change events. Examples of statistics are average packet loss rate over a specific period of time or average latency for packet transmissions. Significant events to one application could be a specific increase or decrease in bandwidth or error rate. To other applications latency variations may be more important, such as the disruption caused by handover events.

2) Process link events

Events from the link providers need to be processed in order to determine whether to forward them to link users or store them and/or perform statistical operations on the event information.

For example, the link events could be generated frequently and many events may not be of interest to the applications, but statistics regarding the events may have been requested, such as averages or repeated events (for example, packet loss burst) lengths. Therefore, the event information needs to be stored and processed to form the notifications to the applications.

3) Provide link information and configure links

The commands to configure links need to be processed to determine which link provides the corresponding command, and to issue a "not supported" response if necessary. It is also necessary to determine which link provider operations to invoke, for example, to connect or disconnect a specific link. It should therefore be possible for link providers to register with ULLA in order to specify which operations and attributes can be accessed.

Applications may require detailed information prior to selecting and configuring specific links and therefore, the selection and retrieval of link related information (attributes) should be made possible by the ULLA.

III. State-of-the-art

There were, in the past, several attempts to provide a uniform and simplified access to link layers. Through the Linux Wireless Extensions (LWE) [8], Linux proposes a uniform interface to control and configure wireless network devices. The LWE is implemented as an extension of the socket interface and features the ability to deliver network events to user level applications. LWE is, however limited to 802.11 based network drivers and does not provide any way to be dynamically extended. Also the set of supported events is constrained to simple things such as Link up or down. The Windows NDIS (Network Driver Interface Specification) extends this idea to all network drivers by providing a generic framework and interface allowing applications to query and set Object Identifiers (OIDs) representing specific attributes of a network, such as connection status or bandwidth. Still the NDIS interface does not deal with link layers that are not implemented as NDIS network drivers (Bluetooth, modems). Moreover, the interface provided to applications is low level since it can be compared to I/O controls allowing attributes to be queried and set.

To provide a uniform access model to modems embedded in mobile phones, manufacturers have extended the basic set of commands called "Hayes commands" or "AT commands", originally specified to allow applications to use PSTN lines to interchange information and to reach remote information resources. With the rise of GSM/GPRS system, the AT commands specification has been enriched with the aim of allowing the control and management of the capabilities being provided by the new devices. The AT interface now supports configuration as well as event notifications and has become the de-facto standard to manage modems embedded in phones. However due to the number of supported commands the AT interface stays cumbersome to use and is limited to modem like devices. Some higher level APIs such as the Windows Telephony API (TAPI) have been built over the AT command set in order to simplify the access to modem like devices. These interfaces are still dedicated to a single type of devices, though.

Similarly, several research projects have tried to provide an interface enabling applications to retrieve information or configure link layers in a way simpler than the existing interfaces. Odyssey [6] is a framework built on NetBSD and aims to enable *application-aware adaptation* to resources availability and modification. Applications collaborate with the Odyssey framework by communicating resource expectations that are expressed in term of lower and upper bounds. The Odyssey framework is then responsible for monitoring the resources and for notifying the application as soon as the resources have left the requested bounds.

Even if Odyssey shares some of the goals defined for the ULLA (notify application of appropriate link changes), it suffers from a non portable interface since it is based on the Virtual File System interface of NetBSD. It also does not provide any function to control link layers. Finally, the number of available monitored resources is limited (6) and the extensibility to other resources is poor. CME [7] is another middleware architecture for network-aware adaptive applications where application notifications are available. The proposed architecture takes, however, another approach by delegating the control of the link layers to a centralized Connection Controller whose decisions are based on policies registered by applications. This is different from the proposed ULLA since the ULLA is not meant to decide which link layer should be used and how. Also CME does not address issues such as extensibility or the ability to control the link layers directly from application level. It finally provides a very limited set of notifications for applications.

IV. Example Application Domains

In the following we discuss some application areas that would benefit from the existence of the ULLA API.

A. Connection Manager (Always Best Connected)

At present, due to the limitations in the IP stack, most mobile devices can only handle a single active connection at a time (per link). Connection manager is typically understood as the agent responsible for deciding on the user of the link, but at present the existing connection managers have to operate on a very limited amount of information (essentially driven by profiles and simple inferences from URIs, for example).

The existence of ULLA would allow development of much more intelligent connection management schemes based on information inferred from both the link-layer, and end-to-end connection. This could even include implementation of actual hand-off schemes, for switching connection from one interface to another, either using Mobile IP (see below for further discussion), or any of the various transport or application layer schemes. Also, using ULLA, the connection management implementation could be independent of particular link-layer technologies, and also partially transportable between operating systems.

B. Unified Cross-Layer Optimization

In recent years, a variety of solutions for wireless cross-layer optimization (CLO) have been proposed. With CLO multiple parameters at different layers of the protocol stack are jointly optimized to meet application requirements. However, these solutions are often developed in an ad hoc fashion limiting their widespread use as standard mechanisms. Therefore, the ability to control radio "knobs" ([1]) in an intelligent way requires that interfaces are defined to expose controllable parameters. By providing a uniform way to access a wide variety of link-layer parameters in a technology-independent way, ULLA would be an important enabling technology for CLO. We believe that ULLA could help optimizing network operations across the protocol stack.

At the network layer ULLA could be used to optimize mobility solutions by providing preliminary notifications when the link quality falls below a threshold, so that the handover process can be anticipated. This would help to reduce handoff latency. In ad-hoc networks link-layer information could be used for routing optimization by selecting the network interface to use in multi-homed terminals.

Transport protocols like TCP can be adversely affected by the wireless link error and delay patterns, leading to throughput reduction and energy waste in a mobile terminal [1]. Error control at the link layer may interfere with end-to-end flow control at the transport layer. A joint tuning of parameters seems therefore desirable. Information provided by ULLA related to handoffs, network disconnection and packet losses could be used to differentiate congestion problems from packet losses, and improve timer management in TCP (see, for example, [4]) and other protocols sensitive to jitter and other changes in network characteristics.

Adaptation to changing link characteristics can be effective at the application layer, as shown in [1]. In multimedia streaming scenarios, source coding parameters can be dynamically varied, based on link statistics; PHY/MAC parameters can also be controlled accordingly. In [2] a CLO example is presented that uses real-time transcoding in a Wireless LAN environment. Delay constrained applications (like wireless VoIP) may want to enforce robustness by adding FEC streams ([3]), when the packet error rate goes above an acceptable level. Applications could also use the ULLA notifications to delay their network requests until the required network characteristics are fulfilled (bandwidth, latency…).

C. Context-sensitive Applications

Considerable research effort has emerged for creating frameworks for context-sensitive applications. In general, context sensitivity can be taken to be adaptation from the part of the terminal or application to changes in environment, network conditions, location of the user (logical or physical), and so on. However, actual implementation of context-sensitivity has turned out to be extremely difficult, and it has even been argued [5] that "generic artificial intelligence" is necessary to make the concept work.

We see ULLA as being one enabling technology for building generic context management systems. It would allow receiving information about the user context that the network(s) could provide through a unified interface, something that is impossible with present-day technologies. The context ULLA could provide would, for example, include location information (either in terms of absolute or relative coordinates in case of cellular systems or ultra-wideband links, or in logical terms using the information about the networks detected in the case of WLANs, for example), information about user mobility, and about other devices surrounding the user.

V. Preliminary Architecture

On top of providing a uniform API, the architecture of the ULLA has been designed to fulfil the following requirements. (1) Extensibility: the proposed architecture should be able to easily integrate new link layer technologies, possibly providing new features. (2) Platform independence: the proposed architecture should be able to be integrated on multiple software and hardware platforms (platform independence). (3) Scalability: the proposed architecture should be light enough to be used on very limited platforms such as sensor devices. (4) Battery life friendly: the proposed architecture should not be a major source of battery drain.

The approach taken in the design of the ULLA is that it should provide an abstract view of the link layers to the ULLA clients. Using an abstract representation allows the ULLA to provide a uniform way to access the wide range of existing link layers independently of their implementation. In order to manipulate these abstractions, a specific query

language should be used. In the following, a ULLA query specifies a request made by an application to retrieve information about a link layer. A notification request is used to specify a condition that should trigger an asynchronous notification. Finally a command is a request specifying an action that should be executed to modify a link layer state. Queries and request notifications both use the ULLA query language. The following paragraph gives an overview of the basic elements composing the ULLA architecture.

A. Detailed Description of the Architecture

Figure 1 describes the ULLA architecture. The architecture is composed of four main components.

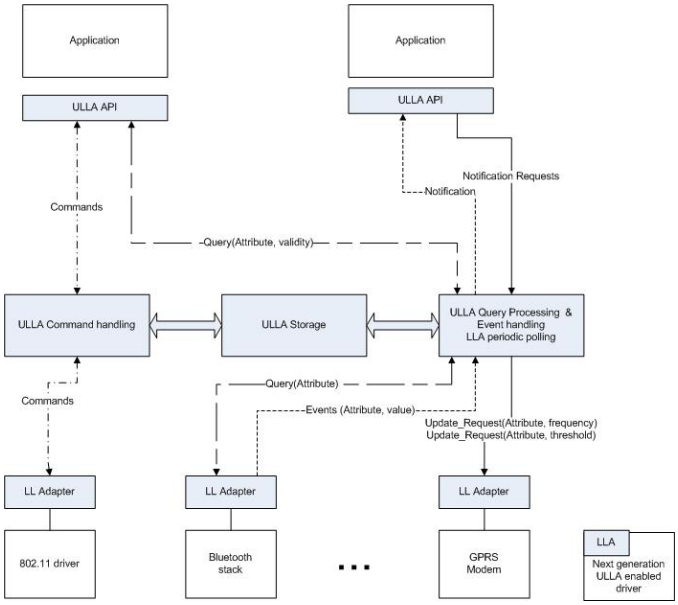

Fig. 1: Basic ULLA architecture.

The ULLA Query Processing Engine (UQPE) is responsible for parsing the requests made by ULLA clients (queries, notification requests and commands). The handling of a request depends on its type. If the request is a query, the UQPE parses the request and uses the ULLA storage to return the requested information. If the request is a notification request, the UQPE stores the requests in the ULLA storage so that it can be later on evaluated. Finally, if the request is a command, the request is forwarded to the corresponding ULLA Link Layer Adapter.

To provide abstraction of the existing link layers, we use an object oriented design. The ULLA storage is used to store class definitions (representing link definitions) as well as instances of these classes (representing discovered links). Link definitions are written with an Interface Definition Language (IDL) and expose a set of attributes that can be queried (in this sense, a class definition is similar to a DB schema). A link definition also specifies a set of methods available to modify the state of a link layer (commands). Instance of class definitions are created when new link layers are discovered.

The ULLA archive storage is an optional component of the architecture. Its purpose is to provide a way to store historical information about the link layers in order to provide statistical values for ULLA clients. Access to the ULLA archive storage is realized through the UQPE.

Obviously today's operating systems won't support the ULLA expected interface. To keep the ULLA platform independent and enable a quick integration in existing systems, we introduce the notion of ULLA Link Layer Adapters (LLAs). A LLA is a proxy interface on the existing driver that implements an interface known by the ULLA (methods and queries), enabling the ULLA to forward queries and method calls to the driver through the LLA. Upon starting, a LLA registers, to the ULLA, the interface that it implements. In addition, LLAs are used to send update requests resulting from notification queries made by applications (see paragraph on battery life). LLAs push modifications of attributes through events that are sent to the UQPE. LLAs are also used to report link discovery to the ULLA. In such a case, a new class using the class definition supported by the LLA, is created in the ULLA storage. Because LLAs are tightly coupled with the link layer implementation, they will probably have to be hand written. In the future we expect to see ULLA enabled drivers that will integrate the functionality provided by the LLAs.

B. Extensibility

Easy extensibility is probably the most important requirement on the ULLA, in particular in a field where new standards appear at a rapid pace. The ULLA provides extensibility by supporting link class inheritance. We envisage that a default ULLA_Link class will provide all common attributes and methods generic for all link layers. More specific attributes can be defined in daughter classes inherited from the ULLA_Link class. From an application point of view, it should always be possible to access a link layer using only the attributes defined by the ULLA_Link class. If an application wishes to use more advanced feature of a link, it needs to get the inherited class describing this link.

C. Battery life

As presented in the architecture description, the ULLA storage maintains a cache version of the attributes provided by known link layers (bandwidth, Signal Strength…). The main issue we have now is updating these values while still limiting battery consumption due to polling or code execution. To reach this goal, the following strategy is used. All attributes maintained in the ULLA storage have an associated timestamp indicating the last time they were updated. When a query request is performed, the validity of the requested attribute(s) is specified in the query and passed to the UQPE. For each requested attribute, the UQPE checks with the associated attribute timestamp the validity of the attribute. If the attribute is too old, a query is forwarded to the corresponding LLA to retrieve the current version of the attribute. This "lazy update" strategy enables the ULLA to keep the number of LLA queries to the bare minimum required by applications.

Dealing with notification request is a little trickier.

Notification requests specify a condition that should be met before notifying an application. A notification condition can span over several attributes of several link layers. When such a condition is received, the UQPE breaks it in a series of update requests sent to the involved LLAs. Each LLA update request specifies an attribute as well as a frequency or a threshold that should be used to send an event back to the UQPE. For example, the UQPE could request a LLA to send, an update for the signal strength attribute, every 2 seconds or if it changes more than x db. Internally the LLA is free to implement this update request through polling or with the support of the underlying driver that could provide some form of asynchronous notification. Note that the UQPE is not aware of the internal implementation of LLAs. It only receives events from the LLA and these events trigger a re-evaluation of the pending notification requests stored in the ULLA storage.

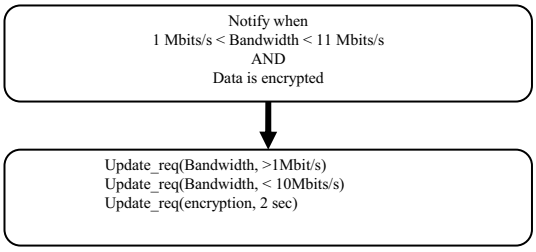

Fig. 2: Mapping of request notification to LLA update requests.

D. *Example Scenario*

Potentially a large number of applications could utilise information passed through ULLA to perform specific adaptations to realise performance enhancements. For example a video application could consider information about changing bandwidth passed through ULLA to adapt and optimise its performance. During start-up, the application could request for notifications about bandwidth changes. Notification from ULLA to the application may also contain additional information about availability of alternate transmission modes. Based on this information, application could either adapt its operation by changing its coding schemes and playback buffers, instructing ULLA to select another mode of operation or performing both adaptations to maintain user perceived quality. The final adaptation may depend on the user preferences (on quality, economy, security etc) and also application may consider and optimise for power consumption as well as user perceived quality.

VI. Conclusions and Future Work

In this paper we have presented the motivation, and basic architecture of a Unified Link-Layer API. We believe that such an API is an essential enabling technology toward software radio, and more intelligent applications. Although several attempts to design such an API have been documented in the literature, none of the existing approaches really fulfils the requirements we have derived. Especially simultaneously achieving extendibility and technology-independence while retaining simplicity has proved to be a hard problem.

The work in the project is now focusing on the specification of the API exposed to the applications in terms of the "high-level" commands. This will set out the generic frameworks for making queries, subscribing to and for receiving notifications, and for issuing commands to the link-layers. Our present "best guess" is that the API offered to the application will be primarily text-based. Queries, commands and notification subscriptions would be performed via simple, fixed high-level function calls that take as one of the arguments a specification written in a specific query language that will be developed in the next phase of the project. The reason for embracing such a text based approach is the easy extendibility.

The exact structure of the query language is still under discussion, though we believe something like a well-defined subset of SQL will most likely be used as a basis. Parsers for such query languages with very small footprint are already available. Our viewpoint is to think the collection of link-layer information essentially as a database, with additional labelling of data with regard to staleness, update frequency, and other such quantities.

Further steps to be taken in the near future are the possible refinement of the architecture, and clarification of where some of the functionalities reside. Toward the end of the project, our aim is to have a fully working prototype implementation of ULLA for a variety of platforms, including a scaled-down implementation for sensor-type platforms.

Acknowledgment

The authors would like to acknowledge discussions with all the members of the GOLLUM consortium.

References

[1] M. Zorzi and R. Rao, "Perspectives on the Impact of Error Statistics on Protocols for Wireless Networks", IEEE Personal Communications, vol. 6, pp.32-40, Oct. 1999.

[2] G. Convertino, D. Melpignano, E. Piccinelli, F. Rovati, F. Sigona, "Wireless Adaptive Video Streaming By Real-time Channel Estimation And Video Transcoding", in *Proc. IEEE Inter. Conf. on Consumer Electronics,* Jan 2005.

[3] J. Rosemberg, H. Schulzrinne, "An RTP Payload Format for Generic Forward Error Correction", *IETF RFC2733*, Dec. 1999.

[4] W. T. Raisinghani, A. Kr. Singh and S. Iyer, "Improving TCP performance over Mobile Wireless Environments using Cross-Layer Feedback", in *Proc. ICPWC-2002*, New Delhi, 2002, pp.81-85.

[5] T. Erickson, "Some Problems with the Notion of Context-Aware Computing," ACM Commun., vol. 45, no. 2, Feb. 2002, pp. 102–04.

[6] B.D. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn, and K.R. Walker, "Agile Application-Aware Adaptation for Mobility", *in Proc ACM Symposium on Operating Systems Principles*, Saint Malo, France, Oct 1997.

[7] J. Sun , J. Tenhunen and J. Sauvola, "CME: a middleware architecture for network-aware adaptive applications", In *Proc. 14th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications,* Beijing, China, 1:839 – 843

[8] http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html; visited on 4.02.2005.