XIAN: Cross-Layer Interface for Wireless Ad hoc Networks

Hervé Aïache, Vania Conan, Jérémie Leguay, Mikaël Levy
Thales Communications
92704 Colombes cedex, France
{firstname.name}@fr.thalesgroup.com

***Abstract*— In the highly dynamic and unpredictable environment of MANETs, *cross-layer* design is receiving growing interest but lacks experimental validation tools. This paper proposes *Xian*, a Cross-layer Interface for wireless Ad hoc Networks, a generic interface for experimenting cross-layer designs with legacy 802.11 protocols. Xian can be used as a service by other network layers or system components to access information about configuration and performance of MAC/PHY layers. The interface is fully implemented and is available for Linux over the *MadWifi* 802.11 driver. We exemplify its use for the design of various QoS routing schemes and provide experimental demonstration of their potential benefits.**

I. Introduction

Mobile ad hoc networks, called MANETs [1], allow the spontaneous set up of wireless communication systems. A MANET is composed of mobile nodes that share one or more wireless channels without centralized control. Network topology but also its related resources are subject to frequent variations with time. In such dynamic and unpredictable distributed environments, the traditional network conception is challenged. Recent research work and studies explore new promising and more flexible design approaches, that revisit the classical IP stack design, called *Cross-Layer* approaches. This paper presents *Xian*, a Cross-layer Interface for wireless Ad hoc Networks to facilitate cross-layer integrations and experimentations by easing the access to information from MAC/PHY layers. Xian has been implemented and is available for Linux over the *MadWifi* 802.11 driver.

The central idea of cross-layering consists in allowing a more flexible exchange of status or control information between the different components of the communication system. With a better knowledge of available resources from different layers of the ad hoc network stack, the system is expected to be more reactive to the wireless environment and responsive to quality required by applicative-oriented elements.

Different approaches have been investigated. When compared to the usual OSI (Open Systems Interconnection) reference model, existing cross-layer solutions span a wide spectrum of options: some advocate global exchange of information between components (e.g. Conti et al. [2]), others prefer to limit them to adjacent layers (e.g. Kawadia et al. [3]), depending on how they impact or differ from this reference model.

In any case, cross-layering calls for software architectures and implementations that support a more flexible sharing of information and status exchanges between the processes and functional modules of the communication system. Experimenting with cross-layer design for MANETs remains difficult; most ad-hoc testbeds make use of 802.11 cards which lack appropriate API support.

This paper proposes a cross-layer design, called Xian (Cross-layer Interface for wireless Ad hoc Networks), compatible with legacy 802.11 MAC/PHY layers. The interface has been fully implemented and is available for Linux kernel 2.4.X, and build upon the *MadWifi* driver [4]. This implementation aims at encouraging and at facilitating cross-layer studies and experimentations over MANETs testbeds. We also provide an example use-case focused on QoS routing decisions.

The remainder of the paper is structured as follows. Sec. II presents the Xian approach and design. Sec. III describes its implementation on Linux. Sec. IV presents results demonstrating the use of Xian to perform QoS routing decisions. Sec. VI concludes the paper, discussing directions for future work.

II. The Xian design

The Xian approach to cross-layering extends rather than replaces the OSI model, The design enables tight interactions between the 802.11 MAC layer and upper legacy OSI model layers. Fig. 1 presents different examples of cross-layer enhanced services offered by Xian to the wireless node.

Note that the OSI model already integrates the notion of offered services to support protocol dialogs between identical layers of distributed entities. However, the service offered by one layer is only available to the layers directly adjacent to it (e.g. encapsulation, error control or fragmentation).

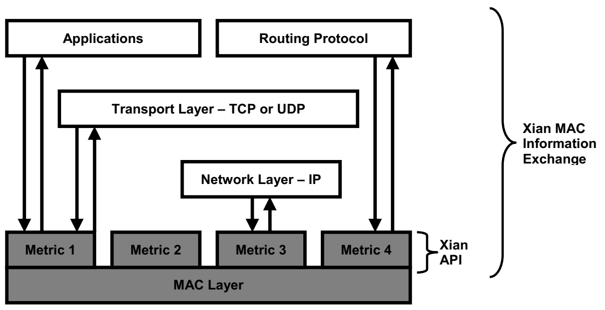


Fig. 1. Xian API and information exchanges.

The major enhancements offered by the Xian implementation is to enable interactions not only between adjacent layers but also between non-adjacent layers. The nature of the exchanged information focuses on metrics representing network resource status and configuration information.

Within the wireless node, cross-layer communication is implemented on a Request/Response model. Upper layers or communication system components can thus access individual metrics provided by the 802.11 MAC layer.

In the design of the Xian software particular attention was paid to facilitate its use and integration with other operating system components (e.g., IP stack layers or applications). The purpose is to propose an open cross-layer implementation which supports the largest possible class of experimentations in domains such as QoS signalling, ARQ/FEC mechanisms, end-to-end transport protocols, adaptive applications, scheduling algorithms, routing predictions or bit rate adaptations.

Sec. III presents how Xian is implemented for the Linux operating system. The implementation aims at providing concrete re-usable cross-layer mechanisms/design to enable tests, experimentations for real testbeds and trials.

III. The Xian implementation

A first implementation of the Xian approach has been provided for a Linux kernel 2.4.X and experimented with a *MadWifi* driver. This section first presents the main characteristics of the 802.11 Linux driver and then describes the Xian software architecture.

A. MadWifi driver

The *MadWifi* driver results from an open source project called Multiband Atheros Driver for Wireless Fidelity. The Atheros chipset is designed through a real configuration-openness of the associated device. Different Atheros chipset versions are supported by the driver, such as AR5210 dedicated to 802.11a, AR5211 adding 802.11b support and AR5212 complementing the two latter ones with the 802.11g standard.

Thanks to a very active community and based on an open/modular architecture design, different driver versions are supported by Linux:

- The *BSD branch*, which offers a good support of the ad hoc mode and of a monitor mode (802.11 frames sniffing).
- The *WDS branch*, which enables the *Wireless Distribution System* technology use (roaming and bridging management between access points interconnecting different wireless networks).
- The *WPA branch*, which focuses on 802.1x for authentications with Radius servers (through Radius clients known as ”wpa supplicant”).

After couple of tests with several 802.11 Linux drivers, the *MadWifi BSD branch* version was selected for the Xian implementation. This version was the only one combining not only the support of the ad hoc mode, open/modular architecture design choices. It also offers more than 180 state information or statistics. 40 of these measurements are given on a per neighbour basis, the remaining ones being aggregate values for the node.

Sec. III-B explains how the Xian cross-layer software design exposes these information/statuses/statistics to other Linux operating system components.

B. Software design

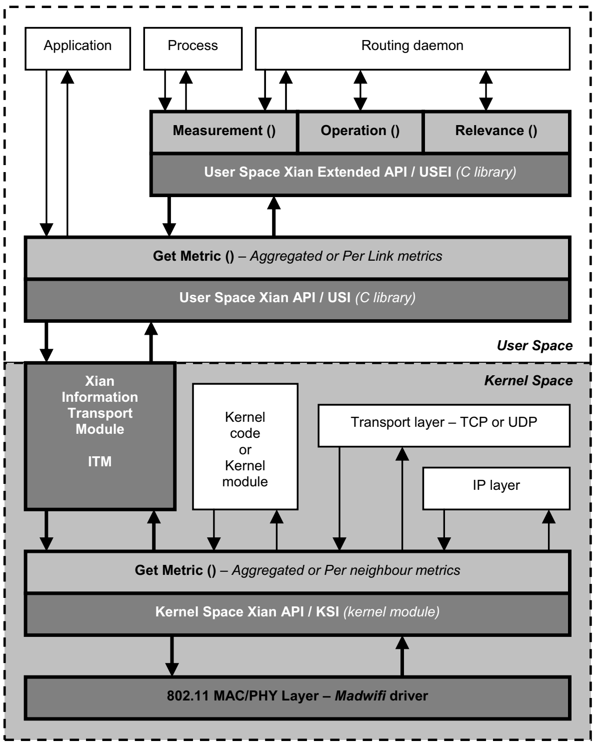
The Xian software is composed of three main components which enable enhancements of the MAC layer in terms of state information accesses implemented at driver level. The particularity of the design consists in proposing driver/MAC internal state information not only to the IP stack layers (i.e. IP, Transport) but also to processes, like applications or routing protocols. These three main components are the following:

- The ***Kernel Space Xian Interface*** (KSI), dedicated to kernel space components (e.g. TCP or UDP implementations) is implemented as a Linux kernel module, which interacts directly with the *MadWifi* driver to retrieve its internal states and statistics.
- The ***User Space Xian Interface*** (USI), which duplicates the Kernel Space Xian API but at the user

space level. This API is implemented as an ordinary C library in order to facilitate its integration with user space programs (e.g. routing daemons or applications).

- The ***Xian Information Transport Module*** (ITM), which allows to pass information and statistics of the *MadWifi* driver from the kernel space to the user space, by connecting the two previous Xian APIs. This module is implemented in this version of Xian as a special character device.

Fig. 2 illustrates how these components interact and how internal driver/MAC states are provided to other Linux system components.



Application
Process
Routing daemon
Measurement ()
Operation ()
Relevance ()
User Space Xian Extended API / USEI (C library)
Get Metric () – Aggregated or Per Link metrics
User Space Xian API / USI (C library)
User Space
Kernel Space
Xian Information Transport Module
ITM
Kernel code or Kernel module
Transport layer – TCP or UDP
IP layer
Get Metric () – Aggregated or Per neighbour metrics
Kernel Space Xian API / KSI (kernel module)
802.11 MAC/PHY Layer – Madwifi driver

Fig. 2. Xian software architecture for Linux system.

From the developer's point of view, the two Xian APIs (i.e. KSI and USI) look exactly the same. As mentioned, the USI allows user space processes to request the KSI thanks to the ITM. The ITM is a messages-oriented module implemented as a character device. This means that the ITM:

- Implements a ”write” function to ask the KSI from the user space level (used by the USI).
- Implements a ”read” function to get KSI answers from the user space level (used by the USI).
- Defines a particular Xian message structure to exchange information between the KSI (i.e. the kernel space) and the user space level (i.e. USI).

```
struct qos_generic_msg      /* Generic Xian message structure */
{
  unsigned long struct_id;    /* Info. type (link metric,
                                           node statistic...) */
  unsigned long type_msg;     /* Info. Exchange status (request,
                                           response or error) */
  char dev_name[10];                          /* Interface name */
  unsigned int id_error;                          /* Error code */
};

struct qos_metric_node_msg    /* Link metric Xian message
                                                   structure */
{
  unsigned long struct_id;    /* Info. type (link metric,
                                           node statistic...) */
  unsigned long type_msg;     /* Info. exchange status (request,
                                           response or error) */
  char     dev_name[10];                      /* Interface name */
  unsigned int id_error;                          /* Error code */
  char macaddr[17];           /* MAC address of neighbour node */
  enum type_metric_t type_metric;
  unsigned long metric_name;                     /* Metric type */
  union metric;                                 /* Metric value */
};
```

Fig. 3. Examples of Xian messages.

The Xian message structures are designed following an object oriented approach: a generic message structure is inherited and then specialized considering the MAC information type (e.g. link metric, node statistic, etc.). In this way, each Xian message is decomposed in two parts:

- A fixed length part, which specifies the Xian information type contained by the message (e.g. link metric), status about the exchanged information (i.e. request, response or error), the network interface name defined by the Linux system and an error code
- A variable length part containing the value of the requested internal driver/MAC state/statistic/metric

Fig. 3 shows two examples of Xian messages: at the top a generic message is presented and at the bottom a Xian message reporting link metric information is illustrated.

Note that the Xian information exchanged through the ITM and reported by the USI and the KSI are simply extracted from specific structures and states maintained by the *MadWifi* driver.

As already mentioned in Sec. III-B, these messages essentially report link metrics (e.g. RSSI for a specific neighbor node) or aggregated statistics (e.g. total of transmitted MAC frames). They can be considered as elementary MAC metrics, which should be combined (thanks to specific operations) to obtain refined and accurate calculated metrics.

The following Sec. III-C describes first how the Xian APIs are designed and, then, introduces a new component of the architecture: an extended Xian interface providing calculated and refined driver/MAC metrics per link for processes running at the Linux user space level.

C. 802.11 metrics and Xian interfaces

The metrics offered by the implementation of the *MadWifi* driver can be clustered in three groups:

- *Configuration states*, which mainly concerns the current configuration parameters of the 802.11 network device, such as the used channel or the number of queues.
- *Aggregated metrics*, similar to counters, this kind of metrics provides global statuses on the use of the 802.11 network interface since it run;s first started. For example, the reported information can be: the number of received frames dropped or with wrong BSSID, the number of transmitted frames with CTS or with RTS enabled, the relative signal strength (RSSI) of the last ACK on transmission, the number of failed receptions (due to queue overrun, bad CRC, PHY errors or decryption problems).
- *Per neighbour/link metrics*, which stores per-neighbour information related to particular transmission at MAC layer. For instance, this kind of metric relates the number of received/transmitted data frames or bytes, the relative signal strength (RSSI) or the number of transmission retries.

The implementation of Xian was mainly concerned by the two last kinds of information provided by the *MadWifi* driver: ***aggregated metrics*** and ***per-neighbour/link metrics***. Moreover, with respect to the Request/Response communication model, the following simple was chosen: implementing a function per metric offered by the *MadWifi* driver.

Therefore, equal to the number of selected metrics, about 180 functions have been developed and integrated in the Xian APIs (i.e. in the KSI and in the USI, the duplicated interface at the user space level). Depending on the type of the reported metric (aggregated or per-neighbour/link), the prototype of the function (for a given metric named *metric_name*) was elaborated as follows:

- For per-neighbour/link metrics:

```
u_int32_t                         /* metric value */
get_node_metric_name(
        u_int8_t * macadd,        /* neighbor's MAC address */
        char * dev_name,          /* device name */
        unsigned int * code_err); /* error code */
```

- For aggregated metrics:

```
u_int32_t                         /* metric value */
get_metric_name(
        char * dev_name,          /* device name */
        unsigned int * code_err); /* error code */
```

These metrics, obtained directly from the *MadWifi* driver, can be considered as *elementary metrics* which should be combined or refined in order to be meaningful or at least more useful for specific system components. For example, the number of transmitted MAC frames (in bytes) would not be an interesting metric if no time-correlation is introduced to reflect how this metric evolves during the life-time of the wireless communication system. In other cases, the average value of a given metric is more meaningful than an immediate measurement. Therefore, we have decided to complement the Xian interfaces with another API developed on top of the USI. This new component of Xian is called ***User Space Xian Extended Interface*** (USEI). The USEI interacts directly with the USI and can be decomposed in three main set of functions:

- *Measurement functions*, which provides calculated metrics resulting from the combination of several elementary metrics taken directly via the USI or from the refinement of an elementary metric (e.g. average values).
- *Operation functions*, which implements the corresponding mathematical operator required by the new defined calculated metrics (e.g. min or max functions).
- *Relevance functions*, which implements the corresponding comparator allowing to indicate if a significant difference occurs between two calculated metrics (typically between the previous and the new ones).

Note that, like the USI, the USEI is implemented as a usual C library. The function prototypes are defined as follows:

```
struct qos_metric_measurement /* Returned metric structure */
get_qos_metric   (
  unsigned int type,          /* metric type to be measured */
  char * macaddr,             /* MAC address of neighbour node */
  char * dev_name,            /* Interface name */
  unsigned int  * code_err);  /* Error code */

unsigned int                                 /* Error code */
metrics_operation (
  struct qos_metric_measurement qos_metric_1,  /* First operand */
  struct qos_metric_measurement qos_metric_2, /* Second operand */
  struct qos_metric_measurement * qos_metric_result); /* Result */

unsigned int                                /* Error code */
is_relevant_metric (
  struct qos_metric_measurement qos_metric_new,  /* Compared metric */
  struct qos_metric_measurement qos_metric_old); /* Reference metric */
```

The following Sec. IV presents the results obtained with the implemented Xian components on a real 802.11 ad hoc platform and demonstrates their expected benefits for QoS routing.

IV. QoS ROUTING: A USE CASE

We present in this section two example use cases in which 802.11 MAC/PHY metrics are measured through the Xian API and we discuss how they may be used to improve the quality of the routes selected by routing protocols.

As shown by De Couto et al. [5], the use of the hop count may lead to poor quality routes that follow long range links, suffering from high packet error rate, or traversing heavily loaded areas, presenting a high level of radio interference or a high level of congestion. QoS routing uses metrics from other layers that can take these parameters into account. A number of such metrics have been proposed in the literature: the *expected transmission count* (ETX) proposed by De Couto et al. [6] measures the bidirectional packet loss ratio of links and the *medium time metric* (MTM) proposed by Awerbuch et al. [7] selects high throughput paths, the *available bandwidth* introduced by Déziel et al. [8], the metric presented by Iannone et al. [9] combines the packet success rate, the interference level, and the physical bit rate. Note that a number of contributions have been made to integrate those metrics in routing protocols. Badis et al. [10] proposed a QoS routing extension to OLSR[11] and Renesse et al. [12] one for AODV[13].

As pointed out by Sec. III, a number of raw and calculated metrics can be accessed easily via simple APIs. The following section shows how to use Xian to perform the integration of these metrics into routing protocols and reports the main points and results of our experimentations.

A. Experimental ad hoc platform and parameters

To highlight the benefit of QoS routing we set up two experiments: the first one with four nodes (quad topology) and the second one with three nodes (triangle topology). We used 4 GigaByte GN-WMAG cards with Atheros chipsets. On each machine involved in the experiments, we run *iperf* [14] to generate and measure performance of TCP and UDP. We also run a simple program that logs, for each link, the metrics accessed through the Xian API. We logged:

- The *RSSI* (Relative Signal Strength Indicator): the wide-band received power within the used channel.
- The *delta occupancy*: the sum of total data bytes received and sent at the MAC layer within the time-interval of length δ.

In reporting the results of the measured metric values in the following subsections, we used the following convention: the plots in black represent the sampled values of the metric measured at every δ time-interval, and the plots in gray correspond to averaged values computed over the N past sampled values. Note that sampled and averaged values are computed within Xian and returned to the logging tool via the APIs of Sec. III-C. Table I shows the different parameters we used.

Parameter	Value
δ	10ms
N	60
UDP sending rate	100 KBytes/s
Packet size	100 bits
Interval between iperf reports	0.5s
802.11 bitrate	11Mbits
RTS/CTS	off

TABLE I
EXPERIMENT PARAMETERS.

B. Quad topology use-case

The first scenario is composed of 4 nodes (see Fig. 4): node S is the source and sends traffic to the destination node D. A concurrent UDP flow is generated between R_2 and D.

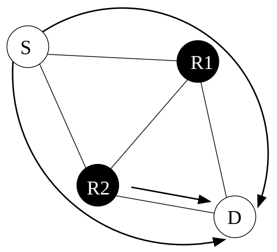


Fig. 4. Quad topology.

The purpose of the scenario was to compare the performances of similar UDP and TCP flows for the two possible 2-hop count routes from S to D: the one going via R_1, and the one going via R_2.

All measurements are synchronised on the same time line by a dedicated tool that handles distributed scenarios. For traffic generation we used the following sequence:

1) $[0s : 100s]$: S sends UDP traffic to D via R_1.
2) $[100s : 200s]$: S sends TCP traffic to D via R_1.
3) $[200s : 300s]$: S sends UDP traffic to D via R_2.
4) $[300s : 400s]$: S sends TCP traffic to D via R_2.

Figures 5(a) to 5(d) show the RSSI values for each of the unidirectional links taken by data packets. Note that RSSI measurement for link $S \rightarrow R_1$ is carried out at R_1 and that its value is updated by the driver only when packets are received. We can see that the route going from S to D via R_2 suffers from links having lower RSSI than the one going via R_1. Figures 5(e) to 5(h) plot the observed link occupancy. These curves allow to follow the global experiment and show the different occupancies that result from the sent traffic. Note that the concurrent UDP flow generated between R_2 and D is visible on Fig. 5(h).

Fig. 6 shows the performance results. It plots the throughput achieved by UDP and TCP flows when relayed by R_1 and by R_2, and the delay measured for UDP packets. Regarding UDP traffic, we measured a throughput of 667.67 Kbits/sec via R_1 and of 599.98 Kbits/sec via R_2 with respectively a standard deviation of 50.37 Kbits/sec and 85.47 Kbits/sec. The route via R_2 provides better performance with higher and more stable throughput. The same holds for the delay since we measured an average delay of 2.13 ms via R_1 and of 2.24 ms via R_2 with respectively a standard deviation of 1.67 ms and 1.63 ms. Similar results are obtained for the TCP flows, with an average throughput of 2116.03 Kbits/sec via R_1 and 1809.25 Kbits/sec via R_2.

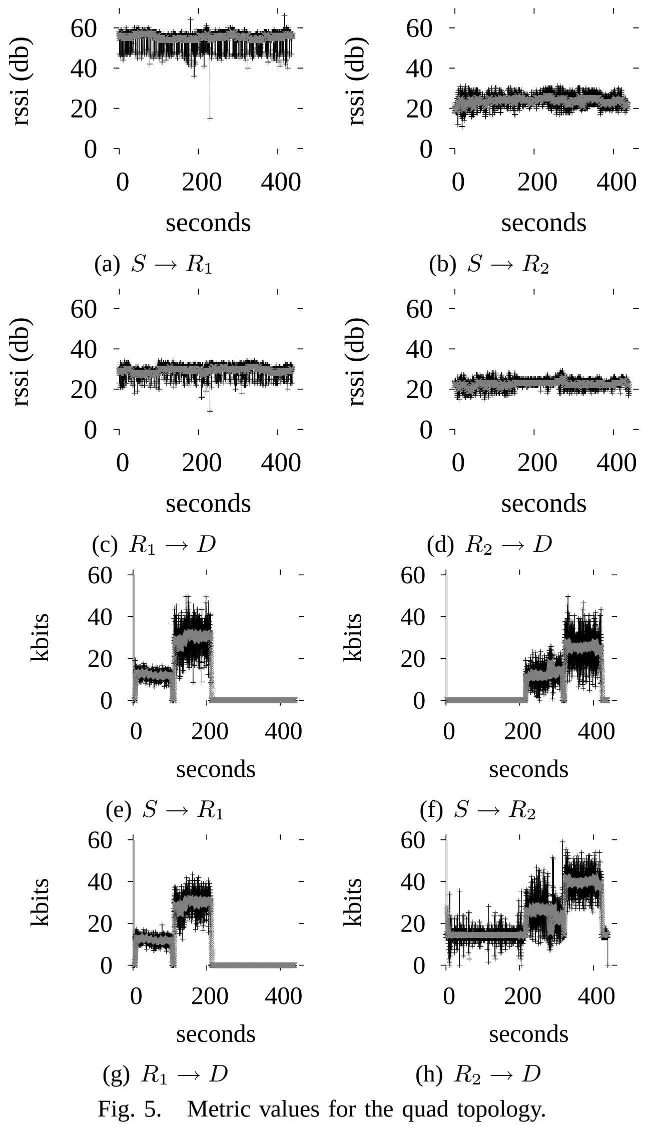

(a) $S \rightarrow R_1$ (b) $S \rightarrow R_2$ (c) $R_1 \rightarrow D$ (d) $R_2 \rightarrow D$ (e) $S \rightarrow R_1$ (f) $S \rightarrow R_2$ (g) $R_1 \rightarrow D$ (h) $R_2 \rightarrow D$

Fig. 5. Metric values for the quad topology.

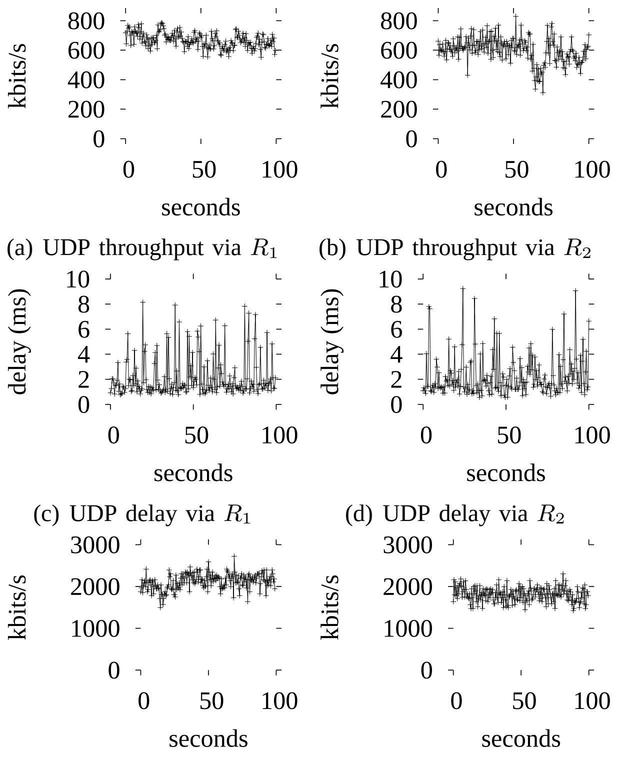

(a) UDP throughput via R_1 (b) UDP throughput via R_2 (c) UDP delay via R_1 (d) UDP delay via R_2 (e) TCP throughput via R_1 (f) TCP throughput via R_2

Fig. 6. Route performance for the quad topology.

The experimental results highlight the potential benefits of QoS routing. Indeed, in the presence of two routes of identical length, one would have preferred in that case the one passing via R_1. The remaining difficulty, not addressed here, consists in chosing the appropriate metric that would be able to cope with all cases and topologies encountered in real life.

C. Triangle topology use-case

The second scenario is composed of only three nodes (see Fig. 7). The two options to route packets from S to D are either to go via R_1 or to use the direct connection between source and destination. The purpose of the scenario was here to compare TCP and UDP performances for a 2 hop-count route with good quality links and for a 1 hop-count lossy link.

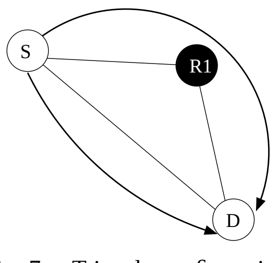


Fig. 7. Triangle configuration.

We used the following sequence of generated traffic:

1) $[0s : 100s]$: S sends UDP traffic to D via R_1.
2) $[100s : 200s]$: S sends UDP traffic to D directly.

Fig. 8 shows the RSSI measured for the different links. We can see that the signal received by D from S is significantly low compared to that of the other links. Note that we observed that for an RSSI lower than $10dB$, the link is reported as broken to upper layers.

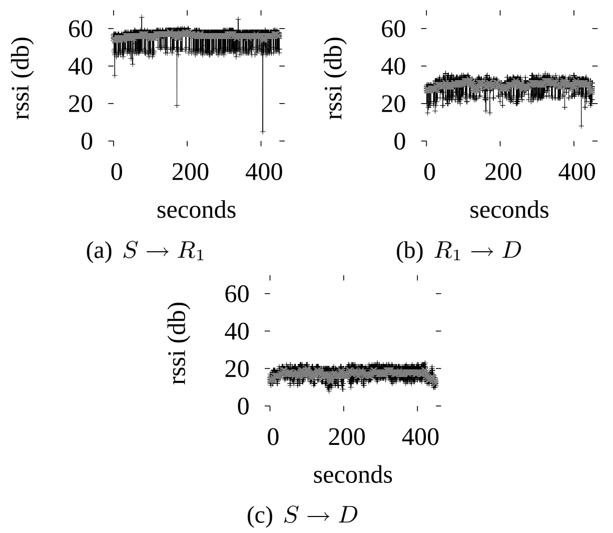


Fig. 8. Signal strength (RSSI) on links.

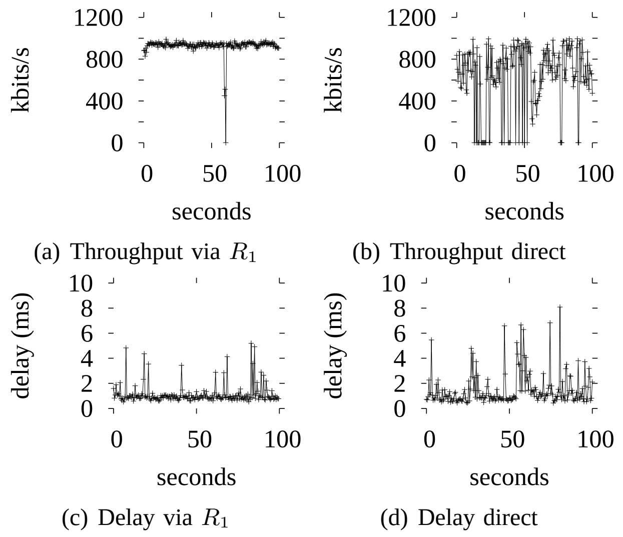


Fig. 9. Route performance for the triangle topology.

The performances of the UDP flows are plotted on Fig. 9. We measured an average throughput of 927.51 Kbits/sec via R_1 and of 625.885 Kbits/sec with a direct route with respectively a standard deviation of 82.24 Kbits/sec and 313.74 Kbits/sec. Regarding the delay, we measured 1.11 ms in average via R_1 and 1.47 ms by the direct route. This experiment shows that in some cases, even when the source and the destination are within range, relaying traffic by a node in between can provide a route with better performances.

V. Related work

The literature discussing cross-layer issues for wireless systems is quite large, so we restrict here to cross-layer simulation, implementation, experimental work on the use by other layers of metrics from MAC/PHY layers.

Xian allows higher layers to have a better access and knowledge of the Wi-Fi MAC status. These higher layers may be the IP layer as we have seen in Sec. IV but any layer can benefit from available information. Local information provided by Xian can be used for instance:

- To support seamless vertical hand-offs between WLAN and other access technologies such as in ECLAIR presented by Raisinghani et al. [15].
- To adapt video transcoding function of traffic conditions as shown by Setton et al. [16].
- To perform load balancing by estimating the local load as in CrossTalk introduced by Winter et al. [17].
- To gather topological information such as link ups/link downs as mentioned by Marrón et al. [18].

The Xian implementation can also be used to validate cross-layer architectures, such as the ones proposed by Conti et al. [2] or by Wang et al. [19].

What is not provided by Xian is the means to distribute globally over the network the information that it provides access to locally. A number of work in that direction have been published, such as the use of the link-state routing protocol OLSR to disseminate QoS information (see QOLSR [20] or CrossTalk [17]). Additionally the Xian implementation does not allow either the setting of internal parameters of the MAC Wi-Fi driver such as the size of the contention window or waiting times (e.g. BIFS or SIFS).

VI. Conclusions and future work

This paper presented Xian, a cross-layer interface implementation that may be applied to build experimental set-ups for validating a large variety of use cases of Wi-Fi cross-layering. Among the contributions of this work, we described the Xian design and detailed the API offered to other network layers or system components. We also presented one of the possible use cases through a real deployment highlighting the benefits of QoS routing. And finally, we released our code that can be downloaded [21].

Future work along these lines would include the development of interfaces working in a publish/subscribe manner. This kind of interface may improve further the integration of the MAC and routing layers as it would allow, for instance, reporting of link up and link down events and help the system react more quickly to topology changes. Weighted average can also been introduced to favour latest measurements or compound metrics can be obtained by combining elementary ones. Other metrics, that should be implemented inside the MadWifi driver or on top of the MadWifi stripped driver [22], can be added as well. Finally, one could wish to extend the generic APIs to support other chipsets than Atheros in the spirit of the Wireless Tools.

In addition to efforts around improvements of Xian, we plan to conduct work on integration with other system components. We intend to use Xian for cross-layer integration with audio/video streaming components and QoS routing extensions for MANETs protocols.

Acknowledgments

We would like to thank the funders of this work: Thales Communications, Euronetlab, E-Next, the LIP6 and the ANRT, which provided the CIFRE grant 135/2004.

References

[1] S. Corson, "RFC 2501: Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations," IETF, January 1999.

[2] M. Conti, G. Maselli, G. Turi, and S. Giordano, "Cross-layering in mobile ad hoc network design," *IEEE Computer*, pp. 48–51, february 2004.

[3] V. Kawadia and P. R. Kumar, "A cautionary perspective on cross layer design," *IEEE Wireless Communication Magazine*, july 2003.

[4] "MadWifi," http://www.madwifi.org.

[5] 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. HotNets*. ACM SIGCOMM, 2002.

[6] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," in *Proc. MobiCom*, 2003.

[7] B. Awerbuch, D. Holmer, and H. Rubens, "High throughput route selection in multi-rate ad hoc wireless networks," in *Proc. WONS*, 2004.

[8] L. L. M. Déziel, "Implementation of an IEEE 802.11 link available bandwidth algorithm to allow cross-layering," in *Proc. WiMob*, 2005.

[9] L. Iannone, R. Khalili, K. Salamatian, and S. Fdida, "Cross-layer routing in wireless mesh networks," in *Proc. ISWCS*, 2004.

[10] H. Badis and K. A. Agha, "Internet draft draft-badis-manet-qolsr-01.txt: Quality of service for ad hoc Optimized Link State Routing Protocol (QOLSR)," IETF MANET working group, September 2005, work in progress.

[11] T. Clausen and P. Jacquet, "RFC 3626: Optimized link state routing protocol (OLSR)," IETF, October 2003.

[12] R. de Renesse, M. Ghassemian, V. Friderikos, and A. H. Aghva, "Qos enabled routing in mobile ad hoc networks," in *Proc. IEEE International Conference on 3G Mobile Communication Technologies*, 2004.

[13] C. Perkins, E. Belding-Royer, and S. Das, "RFC 3561: Ad hoc on-demand distance vector (AODV) routing," IETF, July 2003.

[14] "Iperf," http://dast.nlanr.net/Projects/Iperf/.

[15] V. T. Raisinghani and S. Iyer, "ECLAIR: An efficient cross layer architecture for wireless protocol stacks," in *Proc. World Wireless Congress*, 2004.

[16] E. Setton, T. Yoo, X. Zhu, A. Goldsmith, and B. Girod, "Cross-layer design of ad hoc networks for real-time video streaming," *IEEE Wireless Communications Magazine*, vol. 12, pp. 59–65, august 2005.

[17] R. Winter, J. Schiller, N. Nikaein, and C. Bonnet, "CrossTalk: Cross-layer decision support based on global knowledge," *IEEE Communications Magazine*, vol. 44, pp. 2–8, january 2006.

[18] P. J. Marrón, D. Minder, A. Lachenmann, and K. Rothermel, "TinyCubus: An adaptive cross-layer framework for sensor networks," *it - Information Technology*, vol. 47, no. 2, pp. 87–97, 2005.

[19] Q. Wang and M. A. Abu-Rgheff, "Cross-layer signalling for next-generation wireless systems," in *Proc. WCNC*, 2003.

[20] H. Badis, A. Munaretto, K. A. Agha, and G. Pujolle, "QoS for ad hoc networking based on multiple-metric: Bandwidth and delay," in *Proc. MWCN*, 2003.

[21] "Xian," http://sourceforge.net/projects/xian/.

[22] "Madwifi stripped driver," http://pdos.csail.mit.edu/~jbicket/madwifi.stripped/.