DESIGN OF A FLEXIBLE CROSS-LAYER INTERFACE FOR AD HOC NETWORKS*

Marco Conti, Gaia Maselli, Giovanni Turi
IIT Institute, CNR, Via G. Moruzzi 1, 56124 Pisa, Italy

Abstract Cross layering has recently emerged as a new trend to cope with performance issues of mobile ad hoc networks. The concept behind this technique is to exploit local information produced by other protocols, so as to enable optimizations and deliver better network performance. However, the need for a new interaction paradigm inside the protocol stack has to face with the legacy aspects of classical architectures (e.g., the Internet), where layer separation allows for easy standardization and deployment. In this paper, we show that cross layering can be achieved maintaining a clean architectural modularity, making protocols exchange information through a vertical interface. Specifically, we present the design of a cross-layer module, and provide a proof of concepts of its "usability" at different layers of the protocol stack, considering two case studies from a design and implementation standpoint.

1. Introduction

Cross layering is generally intended as a way to let protocols interact beyond what allowed by standard interfaces. This clashes with the design principles of classical protocol stacks. Just to provide an example, the Internet architecture layers protocols and network responsibilities, breaking down the networking system into modular components. The resulting "strict-layered" system is composed by modules that are independent of each other and interact through well-defined (and static) interfaces, located between adjacent layers. Although this design principle brings important benefits in terms of flexibility and maintenance costs, it suffers from several characteristics of wireless networks (e.g., node mobility or power constraints), degrading the overall network performance [1]. Hence, the need of introducing stricter cooperation among protocols belonging to different layers. This last point sets the focus of this pa-

*This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies under the IST-2001-38113 MobileMAN project, and by the Italian Ministry for Education and Scientifi c Research in the framework of the FIRB-VICOM project.

per, which aims at investigating cross-layer interactions from an architectural standpoint, in the context of mobile ad hoc networking.

In the ad hoc literature, there are several contributions showing the potential of cross layering for isolated performance improvements [2][3][4][5]. They all focus on specific problems, mainly looking at the joint design of two layers. However, their deployment has to deal with the following issues:

1 *Tight-coupling*: the design of cross-layer optimizations requires direct modification of interfaces, causing the involved protocols to become tightly-coupled, and therefore mutually dependent.

2 *Unbridled stack design*: while an individual suggestion for cross-layer design, in isolation, may appear appealing, combining several of them together could result in a "spaghetti" stack design [6], making architectural maintenance a challenging task. Moreover, an uncontrolled combination of isolated cross-layer optimizations may cause mutual interferences, which could lead single nodes to unstable and degraded behavior, with negative impacts on the entire network.

3 *Correct system implementation*: when introducing new interactions among protocols, special care has to be taken to maintain a correct execution flow, without causing critical problems on the internals of the operating system. In real platforms, network protocols consist of a mixed set of processes executing at both kernel and user levels. For this reason, the implementation of cross-layer interactions should guarantee a correct interleaving of protocols execution, without introducing failure patterns on synchronization and scheduling of local system processes.

This paper addresses cross-layering from an architectural standpoint, providing a basis for tackling the semantic problems of interfering optimizations and correct system implementation. In particular, we claim that new interactions can be realized maintaining the layer separation principle, with the introduction of a cross-layer interface (XL-interface) that standardizes vertical interactions and gets rid of tight-coupling from an architectural standpoint. The key aspect is that protocols are still implemented in isolation inside each layer, offering the advantages of:

- allowing for full compatibility with standards, as the XL-interface does not modify each layer's core functions;
- providing a robust upgrade environment, which allows the addition or removal of protocols belonging to different layers from the stack, without modifying operations at other layers;
- maintaining the benefits of a modular architecture (layer separation is achieved by standardizing the usage of the XL-interface).

Engineering the XL-interface presents a great challenge (Section 2). This component must be general enough to be used at each layer, providing a common set of primitives to realize local protocol interactions (Section 3). To support this novel paradigm, we classified cross-layer functionalities and extended standard TCP/IP protocols in order use them. The result of this effort has been implemented in the *ns*2 Network Simulator (Section 4), realizing a simulative evaluation framework for the usability of the XL-interface at different layers.

2. Architectural Functionalities

We designed the XL-interface with two models of interaction in mind: *synchronous* and *asynchronous*. Protocols interact synchronously when they share private data (i.e. internal status collected during their normal functioning). A request for private data takes place on-demand, with a protocol issuing a query to retrieve data produced at other layers, and waiting for the result. Asynchronous interactions characterize the occurrence of specified conditions, to which protocols may be willing to react. As such conditions are occasional (i.e. not deliberate), protocols are required to subscribe for their occurrences, and then return to their work. The XL-interface is in turn responsible for delivering eventual occurrences to the right subscribers. Specifically, we consider two types of events: *internal* and *external*. Internal events are directly generated inside the protocols. Picking just one example, the routing protocol notifies the rest of the stack about a "broken route" event, whenever it discovers the failure of a preexisting route. On the other hand, external events are discovered inside the XL-interface on the basis of instructions provided by subscriber protocols. An example of external event is a condition on the host energy level. A protocol can subscribe for a "battery-low" event, specifying an energy threshold to the XL-interface, which in turn will notify the protocol when the battery power falls below the given value. Note that the host energy controller simply provides the current battery level value, but it is not in charge of checking the threshold and notify related events.

As the XL-interface represents a level of indirection in the treatment of cross-layer interactions, an agreement for a common representation of data and events inside the vertical component is a fundamental requirement in order to guarantee loosely-coupling. To this end, the XL-interface works with *abstractions* of data and events, intended as a set of data structures that comprehensively reflect the relevant (from a cross-layering standpoint) information and special conditions used throughout the stack. A straightforward example is the topology information collected by a routing protocol. In order to abstract from implementation details of particular routing protocols, topology data can be represented as a graph inside the XL-interface. Therefore, the XL-interface

becomes the provider of shared data, which appear independent of its origin, and hence usable by each protocol.

How is protocols internal data exported into XL-interface abstractions? This task is accomplished by using *call-back* functions, which are defined and installed by protocols themselves. A call-back is a procedure that is registered to a library at one point in time, and later on invoked (by the XL-interface). Each call-back contains the instructions to encode private data into an associated XL-interface abstraction. In this way, protocol designers provide a tool for transparently accessing protocol internal data.

3. Designing the cross-layer interface

In order to give a technical view of the vertical functionalities, we assume that the language used by the XL-interface allows for an object oriented representation of data structures and functions. We adopt the following notation to describe the XL-interface interface:

$$XL\_object.method : (input) \rightarrow (output)$$

As described in the previous Section, the XL-interface does not generate shared data, but simply acts as intermediary. Protocols synchronize on an abstract representation of internal data (namely XL_data) where one *producer* protocol specifies a call-back function to export its private data to the abstract representation.

$$XL\_data.seize : (callback()) \rightarrow ()$$

On the other hand, *consumer* protocols access the shared data with read only permissions, using

$$XL\_data.access : () \rightarrow (abstractData)$$

Going back to the example on network topology data, the routing agent plays the role of the producer protocol, exporting routing tables into an abstract graph representation. Consumer protocols living in the scope of other layers, could gather network topology information calling the $access()$ method, which in turn invokes the call-back function registered by the routing agent. This makes the interaction between producer and consumer protocols loosely-coupled, avoiding direct protocol dependencies.

The remaining functionalities of the XL-interface cope with asynchronous interactions. In the case of internal events, the role of the XL-interface is to collect subscriptions, wait for notifications, and vertically dispatch event occurrences to the appropriate subscribers. A protocol *subscribes* for a cross-layer event (namely XL_event) by calling the function

$$XL\_event.subscribe : (handler()) \rightarrow ()$$

Note that the subscriber protocol has to specify a handler function, which will be used by the XL-interface to notify occurrences and triggering event handling. So, subscriber protocols play again the consumer role, while producer protocol *notify* event occurrences by calling

$$XL\_event.notify : () \rightarrow ()$$

The XL-interface is in charge of maintaining a subscription list for each kind of cross-layer event, dispatching occurrences to the correct subscribers.

In the case of external events, the XL-interface must additionally act as event notifier. The idea is that some protocols might be interested in conditions that are not directly verified by other protocols. To this end, subscriber protocols instruct the XL-interface on how to detect the event. The detection rules are embedded in a monitor function, which periodically checks the status of the cross-layer abstractions under inquiry. When the monitor detects the specified condition, the XL-interface dispatches the information to the subscriber protocol. A protocol initiates the *monitoring* of an external event by passing a monitor and a handler function to the XL-interface, through the following method of the target data abstraction

$$XL\_data.setMonitor : (monitor(), handler()) \rightarrow ()$$

The XL-interface serves this call by spawning a *persistent* computation that executes the following steps:

while** true **do
  $freshData = XL\_data.access()$
  ***if** $monitor(freshData)$*
    $handler()$
  endif
endwhile

4. Using the cross-layer interface

In order to practise the usage of the XL-interface, we realized a simulation framework, based on the Network Simulator *ns*2 (v. 2.27), and a library of objects and abstractions, called ProtoLib, provided by the Naval Research Laboratory (NRL) [7]. The choice of ProtoLib is motivated by its high flexibility. It provides a set of simple, cross-platform C++ classes that allow the development of network protocols and applications. Currently, the ProtoLib supports several real platforms (e.g., Unix and WIN32), as well as the *ns*2 simulation environment. Another important feature is that the ProtoLib package comes with an implementation of Optimized Link State Routing protocol (OLSR), compliant with the latest specification [8]. In the framework resulting from the integration of *ns*2 with the ProtoLib, it was a "natural" choice to

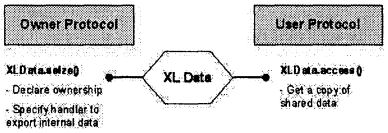

Figure 1. Cross-layer data.

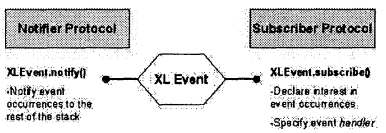

Figure 2. Cross-layer events.

place the objects of the XL-interface inside the ProtoLib. We engineered them as abstract classes that other protocols can implement in order to share data and exchange local events. Specifically, we realized interfaces for *XL_Data* and *XL_Event* objects, respectively for sharing protocol internal data (i.e., synchronous interactions) and for subscribing/notifying internal events (i.e., asynchronous interactions). In the following, we briefly describe the functionalities of the new objects:

ProtoXLData This is a generic class (see Figure 1) that identifies internal data *owned* by a protocol and *shared* to the rest of the network stack. It offers methods to declare ownership of the data and to specify a call-back function for "translating" the internal data format used by the owner, in a cross-layer ontology common to the whole stack. Other protocols access instances of this class with read-only permissions.

ProtoXLEvent This is a generic class (see Figure 2) that identifies conditions or events detected internally to the protocol, which may result of interest for the rest of the stack. It offers methods to *subscribe* interest in events derived from this class, as well as to *notify* occurrences of them.

In the following, we present two examples of cross-layer optimization based on the XL-interface. We show interactions involving network, transport and middleware layers, to highlight how the objects of the XL-interface suites different levels of the protocol stack. The two case studies implement their cross-layer interactions by specializing the base objects presented in the previous Section.

4.1 Improving the performance of data transfer

In this Section, we show how the XL-interface has been used to cope with performance issues of TCP data transfer. TCP performance degrades in ad hoc environments due to losses, which are induced by fault conditions (e.g. network partitions, route failures, and misbehaving nodes), and are erroneously interpreted as effects of congestion. To deal with this problem, we introduce a forwarding mechanism able to improve the performance and reliability of data transfer, also in presence of misbehaving (e.g. selfish) nodes, by means

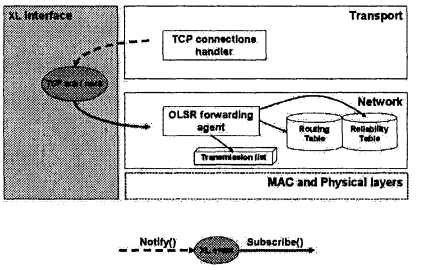

Figure 3. Cross-layer interaction between the network and the transport layer.

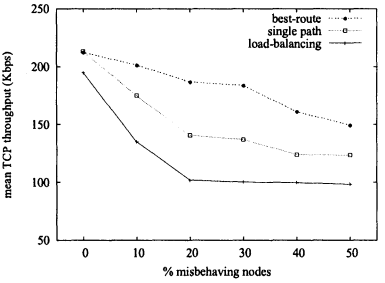

Figure 4. Mean TCP throughput as function of the percentage of misbehaving nodes.

of a cross-layer interaction between the forwarding and transport agents. This mechanism is based on multi-path forwarding and estimates neighbors reliability according to end-to-end acknowledgments. Specifically, in case of reliable data transfer, TCP acknowledgments are used as delivery notifications. The reception of a TCP ack at the transport layer indicates that the corresponding sent packet has been correctly delivered at destination, and hence correctly forwarded by intermediate nodes. Each node estimates only neighbors' reliability, and uses this index to forward packets on most reliable routes, so as to avoid unreliable paths and minimize congestion events. For further details on the forwarding mechanism we point the reader to [9].

The realization of the forwarding mechanism in our evaluation framework involves the introduction of a class of cross-layer events of type *Recv TCP-ack/nack*, to which the forwarding agent subscribes for notifications coming from a local TCP agent (see Figure 3). Specifically, the TCP agent notifies a TCP-ack event to the forwarding agent whenever it receives a valid acknowledgment. Instead, TCP-nack events are caused by packets retransmissions. An event notification causes the forwarding agent to update the reliability index associated to the neighbor through which the packet passed. The update is positive for TCP-ack and negative for TCP-nack. Reliability indexes are used to send packets on most reliable routes. Specifically, we implemented a forwarding policy which chooses the route with the smallest route-length/reliability ratio, namely *best-route* forwarding. As the simultaneous use of multiple paths (i.e., *load-balancing*) degrades TCP performance [10], we compare our best-route forwarding policy with the conventional case in which packet forwarding is based on single path routing (like in OLSR), where the shortest route is always chosen (i.e., *single path*).

The simulation study investigates TCP performance by varying the percentage of misbehaving nodes. The simulated network is composed of 20 nodes, with 5 active Telnet sessions. Connection endpoints are generated randomly, and each simulated scenario is characterized by an increasing number of misbehaving nodes that cooperate to routing, but do not forward TCP traffic. Figure 4 shows how best-route outperforms single path forwarding whenever misbehaving nodes are present, while the two method are comparable in cooperative networks. Specifically, the performance gain achieved with our forwarding mechanism grows up to 50% in the case of 20% and 30% of misbehaving nodes. Results also confirms the poor performance achieved by the load-balancing policy, that increase the chances of encountering misbehaving nodes.

4.2 Improving the quality of unstructured overlays

In this Section, we show how the XL-interface has been used to improve the performance of Gnutella, a well-known unstructured overlay platform. Although we used Gnutella as a case study, a similar approach could be used for other peer-to-peer (p2p) platforms, so as to make them more reactive and usable in ad hoc environments. Full details about this application of the XL-interface are reported in [11].

By simulating a fully-fledged Gnutella system in ad hoc environments, we identified peer discovery as a critical issue. In summary, discovery procedures based on application layer flooding generate overhead, and decrease the capacity of building the overlay. Moreover, as peer selection is random, Gnutella overlays are significantly sensitive to nodes mobility, and fail to react promptly in scenarios with partitioning or heavy churn rates. Under these observations, we re-designed peer discovery and link selection in order to interact with the routing agent at the network layer. The fundamental idea is to exploit node discovery procedures provided by routing agents, so to jointly perform peer discovery together with gathering topology information. For example, in a proactive routing protocol like OLSR, nodes periodically issue Hello and Topology Control messages, containing information about the neighbors that they currently sense. This information could be enriched with *Optional Information* (OI), containing peer credentials (e.g., IP address and port number of the Gnutella service). This approach saves the network resources consumed by an explicit peer discovery protocol.

We modeled the cross-layer interactions using events between Gnutella peers and OLSR agents (see Figure 5). We initially extended the NRL implementation of OLSR to handle new messages for *optional information*, and afterward specialized two classes of cross-layer events: i) *Spread OI* events, to which routing agents subscribe, receiving notifications from Gnutella peers. These

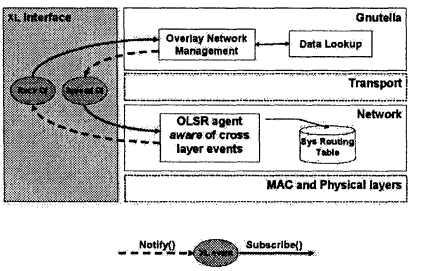

Figure 5. Event-based cross layer interaction for Gnutella peer discovery.

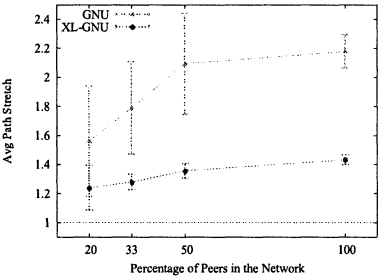

Figure 6. Comparison of the average path stretch produced by Gnutella and XL-Gnutella under increasing peer densities.

events are used to ask OLSR agents to advertise local peer credentials around, along with the next Hello or Topology Control message (see OLSR RFC [8] for details on the protocol); ii) *Recv OI* events, to which Gnutella peers subscribe, in order to receive notifications from underlying OLSR agents. These events are used to notify Gnutella peers about incoming credential advertisements of remote peers. This allowed us to realize peer discovery by making each peer periodically advertise its credentials, and reacting to events of advertisements reception. The overall discovery procedure became simpler and easier to control. On receiving cross-layer events, peers were able to fill up a local table of advertisement generated by foreign agents. Moreover, as advertisements travel the network along with routing control packets, it was possible to get accurate estimates of peers physical distances (in number of hops). This topological information enriched the advertisement table, and allowed us to play a smarter overlay formation protocol, introducing a link selection policy based on physical distances. The rational behind was to simply prioritize closer connections over further ones, with the goal of building an overlay topologically closer to the physical network. In order to give a flavor of the benefits introduced by the XL-interface, Figure 6 shows the results obtained by studying the *path stretch* generated by the legacy and the cross-layer version of the protocol, defined as the ratio of the number of hops (in the physical network) along the path connecting two peers in the overlay, to that along the direct unicast path. This metric measures how far (from a topological point of view) the overlay is from the physical network, and characterizes the overhead induced by the former on the latter. By configuring an increasing percentage of peers in a network of fixed size, we observed that cross-layer Gnutella (XL-Gnutella) produces better path stretches (e.g., respectively 1.35 against 2.1 of legacy Gnutella with a 50% of peers), exhibiting a stable behavior with smaller variances.

5. Concluding Remarks

Cross layering represents a trendy solution to overcome performance limitations of mobile ad hoc environments. Current proposals testify the effectiveness of cross-layering in delivering better protocol performances, but they tackle single cases without prospecting any form of coexistence from an architectural standpoint. The contribution of this paper is the design of an interface able to support several cross-layer solutions, using common interaction models. This approach decouples interacting entities, and preserves the flexibility and modularity features of legacy architectures.

In order to evaluate the usability of the proposed interface, we considered two case studies, verifying that the cross-layer primitives could be used at different layers of the stack, for different purposes. The next step is to engineer and deploy an implementation of the cross-layer interface on a real platform, and verify that the interaction primitives guarantee a clean execution pattern, without introducing mutual interferences.

References

[1] M. Conti, J. Crowcroft, G. Maselli and G. Turi. "A Modular Cross-Layer Architecture for Ad Hoc Networks," in *Handbook on Theoretical and Algorithmic Aspects of Sensor, Ad Hoc Wireless, and Peer-to-Peer Networks*, CRC Press, July, 2005.

[2] K. Chen, S. H. Shah, and K. Nahrstedt. "Cross-Layer Design for Data Accessibility in Mobile Ad Hoc Networks," in *WPC*, vol. 21, no. 1, pp 49-76, 2002.

[3] R. Schollmeier, I. Gruber, and F. Niethammer. "Protocol for Peer-to-Peer Networking in Mobile Environments," in *Proceedings of the 12th IEEE ICCCN '03*, Dallas, Texas, USA, Oct. 2003.

[4] M. Chiang. "To Layer or not to Layer: Balancing Transport and Physical Layers in Wireless Multihop Networks," in *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004.

[5] U. C. Kozat, I. Koutsopoulus, and L. Tassiulas. "A Framework for Cross-layer Design of Energy-efficient Communication with QoS Provisioning in Multi-hop Wireless Networks," in *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, Mar. 2004.

[6] V. Kawadia and P. R. Kumar. "A Cautionary Perspective on Cross Layer Design", in *IEEE Wireless Communications*, Feb. 2005.

[7] "PROTEAN Research Group", http://cs.itd.nrl.navy.mil/5522/

[8] T. Clausen and P. Jacquet. "Optimized Link State Routing Protocol (OLSR)", *RFC 3626*, Oct. 2003.

[9] M. Conti, E. Gregori, and G. Maselli. "Improving the performability of data transfer in mobile ad hoc networks", to appear in the *2nd IEEE SECON '05*, Sept. 2005.

[10] H. Lim, K. Xu, and Mario Gerla. "TCP Performance over multipath routing in mobile ad hoc networks", *Proceedings of the 38th IEEE ICC '03*, Anchorage, Alaska, May 2003.

[11] M. Conti, E. Gregori, and G. Turi. "A Cross Layer Optimization of Gnutella for Mobile Ad hoc Networks", in *Proceedings of the 6th ACM MobiHoc '05*, Urbana-Champaign, Illinois, USA, May 2005.