A Software Architecture for Cross-Layer Wireless Network Adaptations

Soon-Hyeok Choi, Dewayne E. Perry and Scott M. Nettles
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
{schoi, perry, nettles}@ece.utexas.edu

Abstract

Conventional data networks are based on a layered architecture. The introduction of wireless networks has created a need to violate this layering discipline to create cross-layer designs or adaptations. *Ad-hoc implementations of such cross-layer adaptations reduce the level of modularity and abstraction in the network's implementation, giving rise to a significant increase in complexity. We present a taxonomy of possible cross-layer adaptations that is then used to derive an architecture for their implementation that significantly preserves the networks structure. We present implementation results that validate this architecture in the context of a real wireless network implementation.*

1 Introduction

The IP-based Internetwork has had an impact that its inventors could hardly have imagined. An important underlying key to the Internet's success is that its design and implementation is based firmly on a well established architecture, commonly referred to as the "hourglass model" [8]. The hourglass model defines a set of layers, each of which implements some aspect of the network, while leaving other aspects to higher levels. This architecture is a fundamental software engineering strategy to manage complexity in the design and implementation of a very large distributed hardware and software artifact.

Although, strictly speaking, the Internet is based on the hourglass model, for our purposes it is more useful to consider another layering model for networks, the OSI seven layer model [12], which for the four layers on which we will focus somewhat refines the hourglass model. Fig. 1 shows the layers and how they communicate in a conventional network implementation. The lowest layer is the physical layer, or the *PHY*. The PHY is responsible for actually sending data across a physical link, such as a wire or fiber, and must deal with both the analog physical world

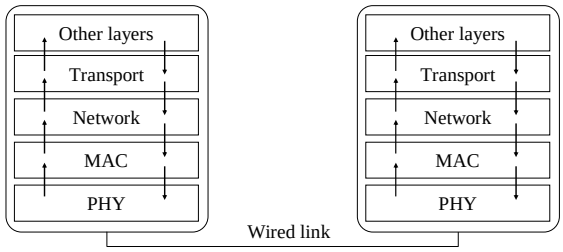


Figure 1. Conventional layered architecture

and the digital world of data communications. The next layer is the link/media access control layer, or the *MAC*. The MAC is responsible for managing communication a single hop in the network, including coordinating which sender is allowed to use a shared medium like the radio frequency (RF) spectrum. The next layer is the Network layer, which is responsible for connecting individual links into a multihop network that can deliver data from a sender to a receiver that are not directly connected. For our purposes the final layer is the Transport layer. The Transport layer is responsible for coordinating end-to-end communication along the connections created by the Network layer. In particular, reliable transport protocols like the Transport Control Protocol (TCP) create reliable communication paths using the inherently unreliable connections provided by Network layer protocols such as the Internet Protocol (IP). The key point is that each layer implements some key functionality with a well defined interface and leaves other functionality for the higher layers to implement. Taken together, we refer to the layers that make up the network as the *stack*. Fig. 1 shows that in the conventional architecture each layer of the stack only communicates with the layer above and below it.

For networks made up of wired links, the networks layered architecture is remarkably successful and the key assumptions and abstraction boundaries work well. However, the introduction of wireless links based on RF communication has revealed that the abstractions are not as cleanly defined as one might expect or hope and that higher lay-

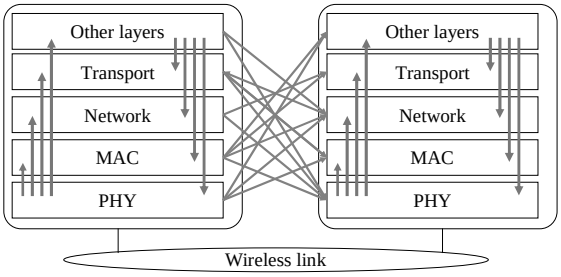


Figure 2. Cross-layer communication paths

ers may make unwarranted assumptions about lower ones. The classic example is when TCP is run over a wireless link [34, 1, 25]. Because wireless links are subject to transmission errors, sometimes they drop a packet. Although TCP has no problem retransmitting the lost packet, it also interprets the drop as a sign that some node in the network is overloaded and dropped the packet to reduce its load. TCP reacts by slowing the rate at which it sends data. This is an incorrect choice when the drop is due to a transmission error and results from an invalid assumption that TCP makes about the reliability of the PHY, a layer that resides several levels down the stack. This is just one example where there needs to be enhanced communication across the layers and of late the area of "cross-layer design" has become a very active one [13].

In general wireless "links" differ from wired ones in many ways. For example, they are lossy and their bandwidth and latency may vary with time. In fact, if the power level or transmission rate is changed, the set of nodes that are directly connected "neighbors" may even change. This leads to possible interactions between nonadjacent layers. For example, by changing power, a PHY property, the network layer might cause different routes to be discovered and used. Even adjacent layers may need to communicate in ways not possible in the current architecture. Later we will present an example where the MAC must obtain information from the PHY that is not part of normal packet processing. Fig. 2 shows just some of the ways that information may need to cross between layers. In fact decision making processes at any layer may need information from any other layer or even a set of other layers. As further shown in Fig. 2, information might also be needed from other layers on other nodes, while the current architecture only allows communication between peer layers. Thus, in general, cross-layer designs and implementations (or as we will refer to them *adaptations*) may need almost arbitrary violation of the basic layering structure. Our goal is to develop an architecture that can accommodate this without destroying the current layered architecture with its advantages of modularity and robustness. Note that although our examples and research prototype focus on interaction between the MAC and PHY, our architecture accommodates other interactions, such as the Transport layer/MAC layer interactions needed to address the TCP over wireless problem.

Unfortunately, most of the work on cross-layer design has proceeded in an undisciplined way and has disregarded the design and implementation advantages of the layered network architecture [30, 19]. The result are systems that are basically spaghetti code with limited structure. Thus far there has been no general consideration of how to construct cross-layer adaptations in a systematic and modular manner. Our goal is to remedy this by providing a framework for building cross-layer protocols that maintains to a significant degree the advantages of modularity and abstraction found in the layered design. As such our focus in this paper is not on any particular cross-layer adaptation (except as an example), but rather on the software engineering issues that arise from the need to violate layering in general. Our strategy for achieving this is to first create a taxonomy that allows us to describe the design space of possible cross-layer adaptations. We then use this taxonomy as a framework to define a conceptual software architecture that allows us to implement adaptations within this space in a systematic way that preserves modularity. This architecture further motivates a concrete architecture that allows us to validate our concepts in a working wireless network prototype.

We begin in Section 2 with several example adaptations, which will be used throughout the rest of the paper. Section 3 presents our taxonomy. Section 4 illustrates our architecture and Section 5 contains our validation results. We present some insight into the validity of our framework in Section 6 and discuss related work in Section 7. We conclude in Section 8.

2 Motivating Examples

We developed and validated our taxonomy and architecture by considering a wide variety of example cross-layer adaptations [6]. Here we motivate our discussion using two of these, cross-layer rate control [28] and cross-layer protocol reconfiguration.

For rate control the idea is simple. The rate at which data can be sent depends on how good the RF connection (or *channel*) is between the sender and receiver. Ideally one sends at the highest rate possible, but the quality of the channel may change from packet to packet. One feasible solution arises because it is possible to measure the quality of the channel just before the data is sent. This is because in MACs such as the distributed coordination function (DCF) mode of IEEE 802.11 [16], prior to data transmission there is an exchange of control messages between the sender and receiver to coordinate channel access. The sender first sends a request-to-send (RTS) to the receiver, which, if it is acceptable to send, replies with a clear-to-send (CTS). The

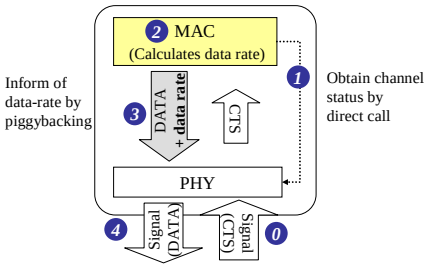


Figure 3. Cross-layer rate control

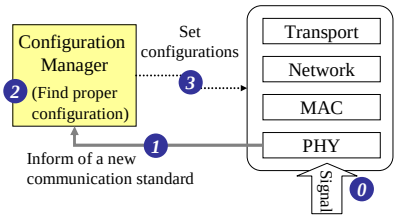


Figure 4. Cross-layer protocol reconfiguration

sender's PHY receives the CTS and as a side effect can determine the quality of the channel from the receiver to the sender. The senders MAC can access this information and use it to set the transmission rate of the immediately following data transmission. The need for cross layering arises because only the PHY can determine the channel state, but only the MAC knows which transmissions are control packets (and thus sent at a fixed low rate) and which are data (and thus candidates for sending at a higher rate).

Fig. 3 shows the process in detail. In step 0, the PHY receives some data and decodes it estimating the channel quality as a side effect. In step 1, if the data was a CTS, the MAC makes a call into the PHY to get the channel information. In step 2, the MAC calculates the correct rate. In step 3, the MAC communicates the correct rate to the PHY by actually attaching the rate to the data packet, a process we refer to as *piggybacking*. Finally, in step 4, the PHY uses the rate to send the packet in the proper manner.

We also consider an additional example [11] that is a slight refinement of the one above. In the case that the channel quality from the sender to the receiver is not the same as in the opposite direction, we can use the RTS to measure the channel quality. In this case, the MAC on the receiver must read the channel quality from the PHY and then piggyback that information on the CTS, which eventually results in the MAC on the sender obtaining the information. Thus this case require internode communication.

The second main example we consider is cross-layer protocol reconfiguration, which allows a node to switch from using one wireless protocol to another. This cross-layer adaptation has a different processing style from rate control. Suppose that we want to allow a mobile device to move from an IEEE 802.11 wireless network to a Bluetooth network. One approach is autonomous reconfiguration of the protocol layers by monitoring the wireless communication environment. Essentially the node listens for other nodes using different protocols and reconfigures itself to use the new protocols as needed. Such reconfigurations are not triggered by or coordinated with packet reception or transmission, but rather occur when a node detects that other nodes in its vicinity are using the different protocol.

Fig. 4 shows the process in detail. The PHY is able to hear all the signals in the wireless environment and thus detects that the alternative communication standard is in use. In this multi-standard environment, the base requirement for the PHY itself is the ability to decode the various waveforms defined by each standard. In step 1, When the PHY detects a new standard, it informs the reconfiguration process about the change. This notification serves as the trigger for reconfigurations. In step 2, the adaptation process finds the proper configurations of protocol layers that meet the new standard and, in step 3, it changes the configuration of the stack. Notice that the actual reconfiguration is coordinated outside of the protocol processing modules. Since the adaptation process requires global information about the protocol layers, using a global manager makes it easy to manage all the configuration parameters for all layers. Further, since the manager is not part of the protocol stack, it can change any layer and is not affected by such changes.

Another example, which is worth mentioning briefly, is the cross-layer extensions for the local agile routing protocol [29]. In this case, nodes monitor the channel condition and exchange information between themselves to assist in creating advantageous routes. For our purpose, the key aspect of this protocol is that it exchanges information between nodes without regard to whether the nodes are carrying data traffic or not.

3 Taxonomy

We use cross-layer rate control and protocol reconfiguration to motivate the development of our taxonomy. Broadly speaking, the goal of developing this taxonomy is to characterize the design space of all reasonable cross-layer adaptations. As importantly, the taxonomy also defines a vocabulary that we can use to describe cross-layer adaptations and their implementation in our architecture. Finally, the taxonomy serves to guide the creation of our architecture itself.

Considering rate control, we see that there are three primary constituents involved [27]. First, at its heart there is some process that actually effects the cross-layer adaptation, in this case, the process that decides the rate given a

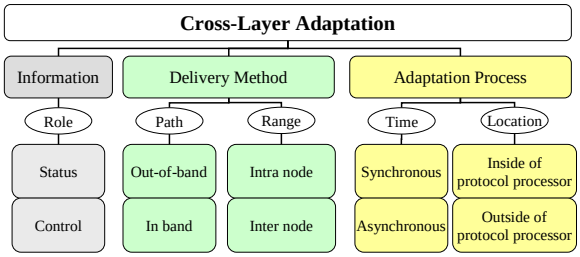


Figure 5. Our taxonomy of cross-layer adaptation

channel condition. Second, some information is communicated across layers, in this case, the channel condition and the rate. Third, there are the delivery mechanisms that are used to communicate the information to and from the process, in this case, a direct call to gather the channel state and piggybacking the rate on the data packet. Fig. 5 shows our complete taxonomy, with these three basic categories, Information, Delivery Method, and Adaptation Process, making up the top-level.

Using our rate control example, we can partially refine each category. For information, there is one refinement based on the *role* of the information. The two subcategories are *Status* and *Control*, the roles of which are obvious. For Delivery Mechanism, this example illustrates a distinction based on the *path* the information takes. *Out-of-band* information, such as the channel status, takes a path that is different from the actual packet data, in this case a procedure call from the MAC to the PHY. *In-band* data, such as the rate, takes the same path as the packet data and can in general be piggybacked on the packet, as it is in this case. Finally, we see two attributes of the Adaptation Process. The first is based on the *time* that the adaptation is performed. This example illustrates just one possibility in which the adaptation is *Synchronous*. This means that the adaptation is synchronized with the reception or transmission of a packet. In our example, the rate calculation is triggered by receiving the CTS and must take place before transmitting the actual data. The second attribute is based on the *location* of the adaptation process. Again, this example only illustrates one possibility in which the adaptation is actually part of the MAC implementation itself. In general, we classify this adaptation as being *Inside the protocol processor*.

The internode version of rate control gives rise to an additional distinction for the Delivery Method based on *range*. For the basic rate control protocol, the range is *Intranode* and for the internode version it is *Internode*. These distinctions are important because any mechanism that communicates information between nodes over the RF link is inherently more expensive and failure prone than one that does not. For Intranode delivery, the distinction of In-band and Out-of-band is obvious as discussed above. But this classification is also applicable for Internode delivery. When the Internode version piggybacks the channel condition on the CTS, this is In-band information that is piggybacked on an 'existing' packet that is already being delivered to another node. In contrast, in some cases, we might create a new packet to deliver Out-of-band information using an additional delivery path dedicated to cross-layer information.

The protocol reconfiguration example allows us to complete our taxonomy by further refining the Adaptation Process. As a complement to *Synchronous* adaptations, the reconfiguration process is an *Asynchronous* adaptation. This reconfiguration process is not coordinated with packet transmission or reception, but rather is triggered asynchronously when the PHY detects a new standard being used in its vicinity. Further, while rate control needs to finish its adaptation before the data transmission, the reconfiguration process can achieve its goal even if the actual time of adaptation is somewhat delayed after detection. Another significant refinement is that the reconfiguration process occurs *Outside the protocol processor*. Such adaptations are not part of the packet processing flow and thus might occur when the process needs to coordinate between a number of protocol layers, as might be needed in the control of quality of service or energy consumption.

4 An Architecture for Implementing Cross-layer Adaptations

Developing the taxonomy allowed us to describe the possible cross-layer adaptations succinctly. Our goal in developing an architecture is fundamentally to provide a set of mechanisms that can be used to implement a wide variety of cross-layer adaptations. In the sense of [18], our main architecture is a conceptual one, which shows in general a set of components and their relationships in a system at a high-level abstraction. Thus our conceptual architecture helps us to understand how we can implement the desired adaptation processes using our framework and in practice serves as a reference model from which a variety of concrete architectures can be derived. This concrete architecture shows more detailed implementation issues that arise when we implement our framework on a specific wireless system. Thus, in the sense of [26], our architecture is a generic architecture, which can describe a range of cross-layer architectures.

We begin by presenting a series of high level goals and requirements for the architecture [7]. We then present some key architectural decisions. We then use the rate control and protocol reconfiguration examples to flesh out the details of the architecture. Finally we motivate a few aspects of the architecture that were not covered by the examples.

The most important goal of our architecture is to provide a set of mechanisms that support the implementation

of all reasonable cross-layer adaptations described by our taxonomy. There are a number of secondary goals, which are fundamentally motivated by a desire to maintain the advantages of the existing layered architecture to the extent possible. The first goal is to preserve the modularity of existing protocol modules to the greatest extent possible. This is key, because otherwise we would be free to simply implement any cross-layer adaptation in an ad-hoc manner. The next goal is to allow cross-layer adaptations to be implemented in as flexible and extensible a manner as possible as well as to facilitate implementing multiple adaptations in a single system. Finally, we want to allow our implementations to be portable to a variety of protocol implementations. For example, ideally if we implement rate adaptation for one particular MAC, it would be easy to move this implementation to some other MAC implementation as long as it have the same underlaying structure.

4.1 Key Architectural Decisions

Fig. 6 shows our taxonomy after we have applied two key architectural decisions. The first decision is simple. Although functionally different, the implementation of cross-layer information does not vary based on whether the data is used as status or control. Thus we can merge these two categories for the purpose of the architecture.

The other change, the elimination of the "Inside the protocol processor" location for the Adaptation Process requires more discussion. The motivation is simple, if we implement an adaptation as part of a protocol module, we will by necessity make changes that compromise the modularity of our system. Furthermore because these changes will be intertwined with the implementation of the base protocol, flexibility, extensibility, and portability will also be compromised. Thus the key challenge in creating our architecture becomes a question of whether we can achieve our goal of comprehensive cross-layer adaptation support, without allowing substantive changes to the protocol modules themselves.

4.2 Example Driven Architecture Development

Fig. 7 shows the progression of high level stages that are required to map our rate control example to our proposed architecture. We consider each stage in turn, explaining the architectural features required.

The first stage (Fig. 7(a)) shows the mechanisms needed to support a Synchronous adaptation process outside of the protocol module. Note that the rate control adaptation has been placed in a separate cross-layer module. A key requirement is that when the packet moves from the PHY to the MAC, the adaptation process must be notified if that

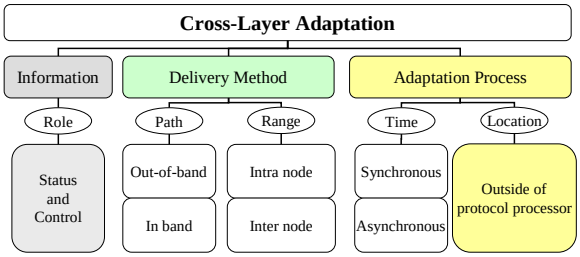


Figure 6. Refinement of our taxonomy based on architectural decisions

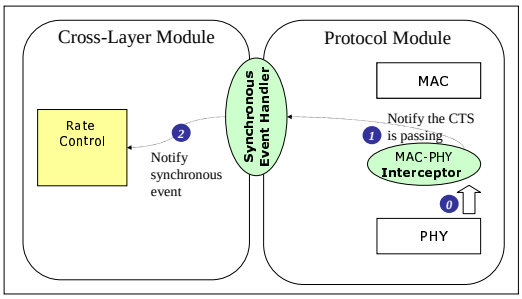
packet is a CTS. Thus we see that in step 0, we have added a MAC-PHY interceptor module. This module is inserted between the two existing layers and provides each with the same interface and thus does not compromise our modularity goal. In general, this interceptor is a kind of connector, but it will be implemented as a "shim" layer in the stack and so we do not group it with the other connectors discussed below. In step 1, the interceptor has detected a CTS and notifies the Synchronous event handler that connects the protocol module to the cross-layer module. Finally, in step 2 the event handler notifies the rate control process itself.

The second stage (Fig. 7(b)) shows the support needed for Out-of-band delivery. In step 1, the rate control process communicates to the Out-of-band connector that it needs the length of the packet and the channel status. Notice that unlike the case where the process is part of the MAC, it needs to access MAC as well as PHY information. In step 2, the connector communicates with the getLength and GetChannel adaptors attached to the MAC and PHY. The adaptation requires that we be able to query the protocol modules, by structuring these queries in terms of special adaptors we are able to minimize (but not eliminate) changes to the protocol modules. Finally, in step 3, the rate control process calculates the new rate.

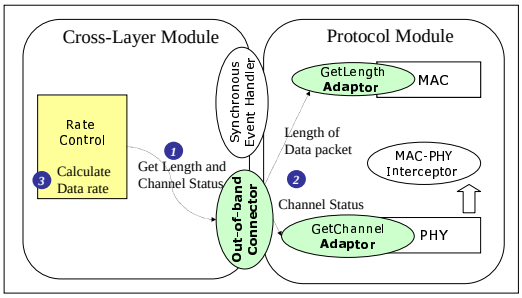
The final stage (Fig. 7(c)) shows how we use the existing mechanisms to complete the rate control process. In step 1 and 2, the interceptor notifies the rate control process that the data packet is being sent. In step 3 and 4, the rate control process sets the rate in the PHY using the setDataRate Adaptor.

To implement the protocol reconfiguration process, we can use most of the mechanisms introduced for rate control. In our example, a global configuration manager already performed the reconfiguration process outside of protocol module. Further, the existing Out-of-band connector allows the manager to read and update information inside protocol modules such as the communication standard stored in the PHY and the configurations of the protocol layers.

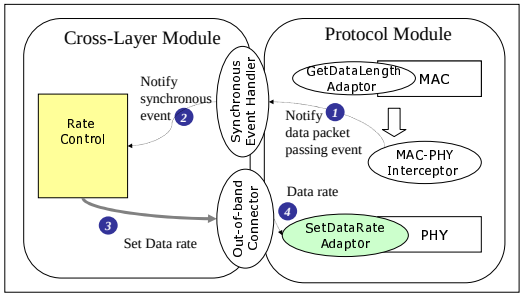
The only new requirement is triggering the reconfiguration process when the PHY detects a new communica-



(a) Components for Synchronous process



(b) Components for Out-of-band delivery



(c) Reusing the components for the rest of process

Figure 7. Architectural solutions for cross-layer rate control

tion standard, which is an Asynchronous adaptation process. One solution is to allow the PHY to notify the manager of the detected standard information, as described in our example. The problem with this active notification is however that the PHY implementation needs to be changed to be aware of the manager. This makes the PHY dependent on the manager. To address this problem, we introduce an Asynchronous event handler that periodically polls the standards information stored in the PHY and triggers the manager when it detects a change. Such a polling mechanism only requires an additional Adaptor attached to the PHY and thus maintains the modularity of the PHY. Although the periodic polling may cause some bounded delay before detecting changes, the Asynchronous adaptation process is a process that can tolerate such delays according to the definition in our taxonomy and so this is not an issue.

4.3 Completing the Architecture

Fig. 8 shows all the details of our architecture. Most aspects of this diagram have already been presented, the main refinement is in the connectors presented and their relation to whether the cross-layer processer is synchronous or asynchronous. These all are typical software connectors [23]. Disregarding the event handler aspect for now, we see four kinds of connectors, corresponding to the four delivery mechanisms in the taxonomy. The Intranode connectors are used inside a single node to integrate existing protocol modules with our architecture [10]. The In-band connector accesses the data stored in a packet's internal structure when the packet passes though an interceptor, while the Out-of-band connector uses adaptors to access data in the protocol modules themselves. The Internode connectors require that any information must be placed in a packet and sent from one node to another. In the In-band case the information can be piggybacked on the protocol packet. Thus this case is shown intercepting the data in the packet delivery path. In the Out-of-band case, the information must be formatted into its own packet and sent independently. Thus this is shown as a separate communication path. Returning to the event handlers, we see that the asynchronous versus synchronous nature of the processes is fundamentally captured by the type of the event handler. Synchronous event handlers are driven by the passage of packets through the interceptors. However, Asynchronous events (and thus processes) are triggered, when the event handler inside Intranode version of Out-of-band connector detects a change of information within a protocol processor or when the Internode version of event handler receives a new information from counterpart in another node.

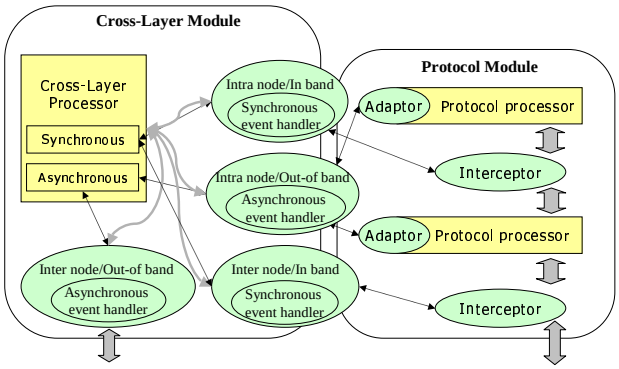


Figure 8. A conceptual architecture for cross-layer adaptations in wireless networks

4.4 Further Refinements

Thus far our architecture is a conceptual one, which describes at a high-level abstractions the key mechanisms that are required to support a wide variety of cross-layer adaptations. If we wish to actually implement an adaptation in a loosely coupled way, our conceptual architecture can serve as a concrete one as well, by simple refinement of the framework to conform to the implementation environment of a target wireless system. One exemplary refinement will be shown in the next section.

However, in the interest of performance, we may wish to use a concrete architecture that has less overhead. We might merge the Interceptors into the layer above or below them. For example, the MAC in our rate control example can be changed so that, when the MAC receives a CTS, it actively notifies of the Synchronous event, thus reducing the amount of packet handling caused by the Interceptor. Similarly, the Asynchronous event handling may be merged into the protocol processing. The PHY in our protocol reconfiguration example can notify the event handler as soon as it detects a change without periodic polling process. We might even merge the cross-layer processing into a particular protocol processor, thus eliminating a substantial amount of communication. Such implementation techniques might introduce further changes in the protocol processing. Never-the-less, we believe the existence of the conceptual model should allow us to make such optimizations in a systematic and disciplined manner.

5 Validation

Initial validation of our taxonomy and architecture was done by careful consideration and paper design of the examples found in Section 2, as well as others. We are conducting a more substantial validation using the basic strategy of implementing our architectural framework and a number of our examples in a realistic wireless network testbed. We expect this experience to allow us to refine our approach, in particular with respect to what concrete architectures are desirable.

5.1 Hydra

Our implementation has been done in the context of our Hydra testbed [21]. Hydra is a prototype multihop wireless network, which is designed to allow experimentation with implementations of PHYs, MACs, and cross-layer adaptations, using functional hardware and software, rather than simulation.

Hydra uses an RF frontend, the universal software radio peripheral (USRP) [4] from Ettus Research, which allows experimentation with various frequency bands and which allows a limited amount of signal processing to be done using a field programmable gate array. The USRP connects to the Hydra PHY over USB 2.0. The PHY is implemented using the GNU Radio framework [3] and all signal processing is done using the general purpose processor. Hydra's MAC interfaces to the PHY using interprocess communication and is implemented using the Click modular router infrastructure [24]. Click also provides network support and interfaces to the Linux TCP/IP stack allowing full end-to-end application to application experiments. Both GNU Radio and Click allow us to implement flexible network protocols. Using the GNU Radio framework, we create a set of "signal processing blocks", each of which implements a signal processing algorithm. Then we can compose a PHY protocol by connecting these small blocks into a signal processing flow graph. Similarly, the Click modular router allows us to create a set of "packet processing elements", each of which implements some task required for packet processing and to compose a new protocol by connecting the elements. Thus Hydra allows us to flexibly configure a protocol by changing the connection graph that is composed of a set of small components.

The current Hydra implementation is similar to 802.11a [17]. It supports orthogonal frequency division multiplexing (OFDM) at the physical layer, with support for multiple transmission rates. The MAC is essentially the 802.11 DCF MAC briefly discussed in Section 2. Because both the MAC and PHY are primarily implemented in C++, modification of each is straight forward. Hydra is currently operational. In addition to experiments on cross-layer adaptations, the major next implementation step is to add support for multiple antenna algorithms, principally multiple input multiple output (MIMO).

5.2 A Concrete Architecture for Hydra

We refined our conceptual architecture to implement it inside Hydra. The key challenge was that protocols in Hydra are implemented using three different protocol modules. The MAC and Network layers are implemented using Click while the PHY uses GNU Radio. Further the TCP/IP protocol stack implements the Transport layer. This meant that the Interceptors and Adaptors needed to be implemented differently to conform to each implementation environment. To address this problem, we divided all of the connectors into two levels, except the Internode version of the Out-of-band connector that is not connected to the protocol modules. Each local connector is implemented conforming to implementation environment provided by each protocol module and thus easily manages the Interceptors and Adaptors that are implemented using the same environment. Then the local connectors communicate with global connectors that provide cross-layer processors with event notification and data delivery. Thus this concrete architecture still allows the cross-layer adaptations to be independent of the existing protocol implementations.

5.3 Rate Adaptation

We have implemented both the Intranode and Internode versions of rate control, discussed in Section 2, using our fully decoupled architecture. We have also implemented both versions of rate control in the ad-hoc manner that might be considered "conventional" to compare both the implementation techniques.

Fig. 9 shows an experiment result using the Internode version of rate control. (Note, this result was first presented in our paper describing Hydra [21].) The X-axis is the packet sequence number and the Y-axis on the left is for the received signal to noise ratio (SNR), which was used to estimate the channel condition. A higher SNR fundamentally represents a better channel status and thus allows the rate control process to select a higher data rate. Each packet is plotted at the SNR threshold for the rate at which it was transmitted. In this experiment, the transmit power (the Y-axis on the right) was decreased and then increased in steps to control the received SNR. We see that in general when changes in power cause the received SNR to cross a threshold, rate control process adapts the data rate of the transmission as expected. We also see instances of packets that are transmitted at higher or lower data rates than most packets at the same power level. These show the rate control process can adapt to short term fluctuations in the channel.

The conventional implementation required a set of changes to the existing MAC and PHY implementations. To implement both the Intranode and Internode version of rate control processes:

1. A set of Click elements were created and modified:
   - to allow the MAC to obtain the channel status information,
   - to allow the MAC to perform rate adaptation, and
   - to conform to the new CTS packet format that delivers the channel information from the receiver to the transmitter.

2. A GNU Radio block was changed:
   - to allow the MAC to access the channel status information.

3. The interfaces between the MAC and PHY were changed:
   - to allow cross-layer information piggybacked on packets to be marshalled and unmarshalled when they move between the MAC and PHY.

The key problem was that such changes cause individual protocol processors to become dependent on others. At one point we changed the rate control in the MAC to use a different type of channel information, from an integer valued received signal strength indication (RSSI) to a floating point valued SNR. This required that the interfaces between the MAC and PHY and the signal processing block in the PHY all needed to change to deal with the new type of channel information.

We then implemented the rate control processes based on our loosely coupled architecture. These implementations were encouraging in that they did not introduce any significant changes into the existing protocol processors. After implementation of the global and local connectors, to implement both the Intranode and Internode version of rate control processes:

1. A set of Click elements were created:
   - to add the Interceptors into the MAC, and
   - to attach an Adaptor that accesses data length information.

2. A GNU Radio block was created:
   - to attach an Adaptor that accesses channel status information.

3. Rate control processor was created:
   - to execute rate control.

Although a set of protocol processors were created and inserted into Click and GNU Radio to implement our architectural components, these components did not introduce any significant changes in the existing protocol implementations. An Interceptor notifies the Synchronous event handler of the passage of a CTS packet and transparently changes the format of the CTS packet. Further the Adaptors augmented the interfaces of the MAC and the PHY without

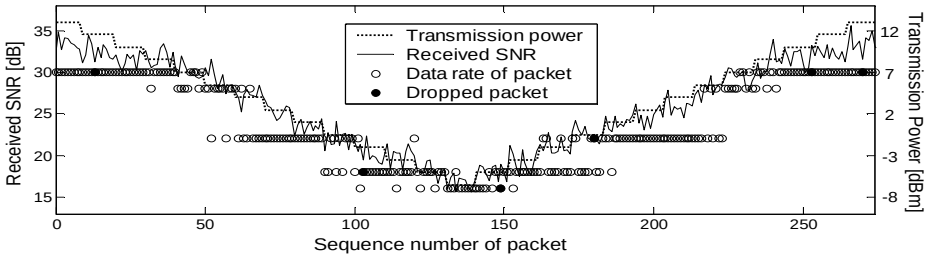


Figure 9. Trace for experiment with cross-layer rate control.

affecting core functionality of the existing protocol processors. The only change caused by the Adaptor was to expose some variables inside the protocol processors to allow the Adaptor to access the information.

We implemented the Internode version of rate control by extending the Intranode version. To extend the adaptation process in a conventional implementation, we needed to modify a few more packet processing elements in Click. The problem was that the dependency between the Click elements changed. However, the implementation based on our architecture only required extension of the existing rate control processor after inserting a few more Interceptors into Click. This shows that our architecture allows rate control to be independent of the infrastructure and to freely change its operation without significant impact on existing protocol implementations.

5.4 Frequency Selection

To explore the issues in Asynchronous event handling and Out-of-band delivery, we have implemented a cross-layer frequency selection process, which is similar in operation to our protocol reconfiguration example. Our preliminary conclusion based on this implementation is that our architecture substantially meets its goals.

In most of wireless communication standards, a large frequency band is divided into a set of smaller frequency bands, each of which can be used independently of the other bands. A possible problem with such multiple bands is that a set of nodes may use the same band and suffer from high load while another band is not used by any node. This causes an inefficiency in utilizing the scarce wireless bandwidth resource. To address the problem, our frequency selection process monitors the traffic load on a frequency band, which can be accurately measured by the MAC. Then, when the load becomes high, it changes the frequency used by the PHY to another one to balance the load among multiple bands. Thus this adaptation is another example that performs Asynchronous processing using the Out-of-band communication path.

We have implemented frequency selection both using our loosely coupled architecture and in the ad-hoc manner. First, to implement the adaptation process in the ad-hoc manner:

1. A Click element was created:
   - to allow the MAC to monitor the load on a frequency, and
   - to allow the MAC to select and change the frequency.

2. The interfaces between the MAC and PHY were changed:
   - to create an additional communication path dedicated for the frequency information,
   - to allow the frequency information to be marshalled and unmarshalled, and
   - to allow the MAC to access the frequency information in the PHY.

In contrast to rate control, the frequency selection process does not need to be coordinated with existing packet processing. Thus the implementation required insignificant changes to the existing protocol processors. However, the new communication path introduces interdependency between the MAC and the PHY. Additional communication paths and message formats need to be defined to allow the frequency information to move between the MAC and PHY. Thus, a change of information and communication requires the modification both the MAC and the PHY.

We then implemented the adaptation process based on our architecture. After implementing the Asynchronous event handler that periodically polled for the relevant information, to implement the adaptation process:

1. Two Click elements were created:
   - to allow the MAC to monitor the load of a frequency, and
   - to attach an Adaptor that accesses the load information.

2. Two GNU Radio blocks were created:
   - to attach the Adaptors that access the frequency information.

3. A frequency selection processor was created:
   - to select and change the frequency.

As in the rate control example, our architecture did not introduce any significant changes in the existing protocol implementations. It only added a set of Adaptors to access the load information stored in the MAC and frequency information in the PHY. Thus the implementation did not introduce interdependencies between the MAC and the PHY. Another benefit of using our architecture was that we were able to implement the adaptation by reusing the Out-of-band connector implemented for rate control. We expect many of the mechanisms implemented for these examples to be reusable across other adaptations including protocol reconfiguration.

5.5 A Final Example

The remaining aspect of our architecture that needs validation is the Internode version of the Out-of-band connector. We plan to implement cross-layer extensions for the local agile routing protocol [29] to explore these aspects. This adaptation acquires channel conditions for the *local* area to build robust routes that reach multihop neighbor nodes and thus requires periodic exchanges of channel information between nodes regardless of whether packets are being delivered. Thus implementing cross-layer extensions for the local agile routing protocol will allow us to validate Internode version of Out-of-band communications.

6 Discussion

This paper presents some of the examples we used to develop and validate our taxonomy and architecture. They capture the key issues in implementation of cross-layer adaptations and classify them into a compact framework by considering a wide variety of cross-layer adaptations. For example, the rate control techniques discussed in Section 2 are only two examples among a set of rate control processes [28, 11, 14, 2, 20, 15] we investigated. We also took into account a wide variety of adaptation processes that occur between other than between the MAC and PHY, including the aforementioned local agile routing protocol, and TCP and applications that optimize the their performance over wireless links.

We believe that our taxonomy and architecture is generic enough to cover a wide variety of cross-layer adaptations. The three top-level categories in our taxonomy (Information, Delivery Method, and Adaptation Process) capture the three basic elements that comprise software systems [27] and thus should hold in general. Further, the subcategories capture the key issues in implementation of adaptation process using high-level abstractions. But, as our understanding of cross-layer adaptation improves, our current framework might need to be extended. Our taxonomy allows such extensions by allowing new subcategories to be added to each top-level category. This will lead us to add a new component to our architecture in a systematic way. Our taxonomy also allows further refinement of each subcategory based on detailed implementation issues. For example, the MAC protocol is generally runs on a network interface card while the routing protocols run on the general purpose processor. To allow data exchange between those layers, Intranode delivery requires interprocess communication mechanisms. Thus we can further divide Intranode delivery into one that occurs within the same address space and one that crosses different address spaces. This detailed refinement allows us to consider what concrete architecture is desirable.

Another use of our taxonomy and architecture is to allow us to synthesize new cross-layer adaptation processes in a systematic way. For example, we were able to design a new cross-layer routing protocol using the proposed framework. The main goal of this network algorithm is to improve throughput of multihop wireless network by mitigating interference between nodes. Mitigating interference requires a node to cooperate with a set of local neighbor nodes, each of which can be reached directly or reached by a few intermediate hops. Further this adaptation requires complex interactions across the multiple layers including the MAC, PHY, and Network layers. Our framework allowed us to decompose the issues in design and implementation of such a complex adaptation process into manageable pieces. We designed an adaptation process to exchange a set of cross-layer information with the local neighbor nodes using the Internode version of the Out-of-band and the In-band delivery mechanism. We then refined the adaptation to communicate across the multiple layers using the Intranode version of delivery and event handling mechanisms.

7 Related Work

A set of software architectures [30, 32, 9, 5, 31, 33, 22, 35] have been proposed to coordinate the implementation of both the protocol layers and cross-layer adaptation processes. The unique aspect of our architecture that distinguishes it from the preexisting architectures is its general consideration of how to implement cross-layer adaptations in a systematic way starting from an understanding of the complex and wide design space of cross-layer adaptations.

The cross-layer architecture that is most similar to ours is Efficient cross-layer architecture for wireless protocol stacks (ECLAIR) [32]. In ECLAIR, a tuning layer (TL) attached to a protocol layer provides interfaces, which allows a protocol optimizer (PO) to read and update data inside the protocol layer and also to be notified of certain events generated by the protocol layer. Then a PO implements a cross-layer adaptation outside of the protocol stack by using TLs. Thus the TLs serve as the Intranode version of the

Out-of-band connector and the Asynchronous event handler, while the PO can be mapped to our cross-layer processor. However, ECLAIR does not consider the In-band and Internode delivery cases. Further ECLAIR does not support the Synchronous event handler, since it views that the Synchronous adaptation process can block the operations of a protocol processor and can introduce performance degradation in existing protocol process. However, our taxonomy shows there are cases where the adaptation process needs to be synchronized with packet processing and we believe that optimizations of the connectors can to a significant degree reduce the possible performance degradation.

The mobile metropolitan ad-hoc networks (MobileMAN) system [9] preserves the modularity of the existing protocol implementation by providing standardized interfaces, which allows a protocol layer to indirectly communicate with the other layers. Thus MobileMAN allows a protocol layer to be maintained or replaced with a new release without affecting other protocols. However, MobileMAN disregards the advantages of modular design and implementation of cross-layer adaptations and thus merges the adaptation processes into the protocol processors. The changes to the cross-layer adaptation process can cause modification or replacement of existing protocol implementations. In MobileMAN, a protocol layer delivers data and notifies an event to another protocol by using the Out-of-band delivery path using standardized interfaces. However, MobileMAN does not pay much attention to the In-band and the Internode delivery cases, since its main concern is the indirect communications between layers within a node over a global connector.

To our best knowledge, ECLAIR is the most sophisticated architecture among ones that coordinate the adaptation processes with the existing protocols outside of the protocol module. Further, MobileMAN shows a typical model of a set of another existing architectures, which maintain the modularity of existing protocol implementations by indirect communication between layers. Thus describing the architectures using our framework shows that our architecture can cover a variety of preexisting cross-layer architectures and thus validates our architecture as a generic model that allows us to derive a wide set of concrete architectures.

8 Conclusion

In a wireless network, it is useful for a wide variety of adaptations to violate the networks traditional layered architecture. Unfortunately doing so in an undisciplined way is likely to result in a poorly structured system and to greatly increase the complexity of an already complex system. We presented a taxonomy that describes the design space of such cross-layer adaptations. Based on this taxonomy, we derived an architecture that supports the implementation of cross-layer adaptations in a controlled disciplined manner. Perhaps the most important design decision in this architecture is for the cross-layer adaptations to be decoupled from the implementation of the basic protocols, thus minimizing changes to the basic layered structure of the network. Finally, we presented a validation of our approach using a wireless networking prototype. This validation suggests that our architecture can indeed achieve its goals.

References

[1] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz. Improving TCP/IP Performance over Wireless Networks. In *Proceedings of the 1st International Conference on Mobile Computing and Networking*, pages 2–11, Berkeley, CA, November 1995.

[2] J. C. Bicket. Bit-rate Selection in Wireless Networks. Master's thesis, MIT, February 2005.

[3] E. Blossom. GNU Radio. http://www.gnu.org/software/gnuradio/index.html.

[4] E. Blossom. UniversalSoftwareRadioPeripheral. http://comsec.com/wiki?UniversalSoftwareRadioPeripheral.

[5] G. Carneiro, J. Ruela, and M. Ricardo. Cross-Layer Design in 4G Wireless Terminals. *IEEE Wireless Communications*, 11:7–13, Apr. 2004.

[6] S.-H. Choi. A Software Architecture for Cross-layer Wireless Networks. Proposal report, January 2007.

[7] L. Chung, B. A. Nixon, and E. Yu. Using Non-Functional Requirements to Systematically Select Among Alternatives in Architectural Design. In *In Proceedings of ICSE Workshop on Software Architecture*, pages 31–43, Seattle, WA, April 1995.

[8] Computer Science and Telecommunications Board, National Research Council. *Realizing the Information Future: The Internet and Beyond*. National Academy Press, Washington, D.C., 1994.

[9] M. Conti, S. Giordano, G. Maselli, and G. Turi. MobileMAN: Mobile Metropolitan Ad Hoc Networks. *Lecture Notes in Computer Science*, 2775:169–174, 2003.

[10] A. Egyed and R. Balzer. Unfriendly COTS Integration - Instrumentation and Interfaces for Improved Plugability. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, pages 223–231, San Diego, CA, November 2001.

[11] N. V. Gavin Holland and P. Bahl. A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks. In *Proceedings of International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001.

[12] H. Zimmerman. The OSI Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4):425–432, April 1980.

[13] Z. J. Haas. Design Methodologies for Adaptive and Multimedia Networks. *IEEE Communications Magazine*, 39:106–107, November 2001.

[14] I. Haratcherev, K. Langendoen, R. Lagendijk, and H. Slips. Hybrid Rate Control for IEEE 802.11. In *ACM International Workshop on Mobility Management and Wireless Access Protocols*, Philadelphia, PA, October 2004.

[15] I. Haratcherev, J. Taal, K. Langendoen, R. Lagendijk, and H. Sips. Automatic IEEE 802.11 rate control for streaming applications. *Wireless Communications and Mobile Computing*, 5:421–437, June 2005.

[16] IEEE 802.11 Working Group. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification, November 1997.

[17] IEEE 802.11 Working Group. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band, September 2000.

[18] R. C. H. Ivan T. Bowman and N. V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In *Proceedings of the 21st International Conference on Software Engineering*, pages 555–563, Los Angeles, CA, May 1999.

[19] V. Kawadia and P. R. Kumar. A Cautionary Perspective on Cross Layer Design. *IEEE Wireless Communications*, 12:3–11, February 2005.

[20] M. Lacage, M. H. Manshaei, and T. Turletti. IEEE 802.11 Rate Adaptation: A Practical Approach. In *Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 126–134, Venice, Italy, Oct. 2004.

[21] K. Mandke, S.-H. Choi, G. Kim, R. Grant, R. Daniels, W. Kim, R. Heath, and S. Nettles. Early Results on Hydra: A Flexible MAC/PHY Multihop Testbed. In *IEEE 65th Vehicular Technology Conference*, Dublin, Ireland, April 2007.

[22] P. J. Marrón, A. Lachenmann, D. Minder, J. Hähner, R. Sauter, and K. Rothermel. TinyCubus: A Flexible and Adaptive Framework for Sensor Networks. In *Proceedings of the 2nd European Workshop on Wireless Sensor Networks*, pages 278–289, Istanbul, Turkey, Jan. 2005.

[23] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a Taxonomy of Software Connectors. In *Proceedings of International Conference on Software Engineering*, pages 178–187, Limerick, Ireland, June 2000.

[24] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek. The Click Modular Router. In *Symposium on Operating Systems Principles*, pages 217–231, Kiawah Island Resort, SC, December 1999.

[25] K. Pentikousis. TCP in Wired-Cum-Wireless Environments. *IEEE Communications Surveys*, pages 2–14, Fourth Quarter 2000.

[26] D. E. Perry. Generic Architecture Descriptions for Product Lines. In *Proceedings of the 2nd International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families*, pages 51 – 56, Las Palmas de Gran Canaria, Spain, February 1998.

[27] D. E. Perry and A. L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 17:40–52, October 1992.

[28] D. Qiao, S. Choi, and K. G. Shin. Goodput Analysis and Link Adaptation for IEEE 802.11a Wireless LANs. *IEEE Transactions on Mobile Computing*, 1:278–292, October 2002.

[29] C. A. Santivanez, R. Ramanathan, and I. Stavrakakis. Making Link-State Routing Scale for Ad Hoc Networks. In *Proceedings of International Symposium on Mobile Ad hoc Networking and Computing*, pages 22–32, Long Beach, CA, October 2001.

[30] V. Srivastava and M. Motani. Cross-Layer Design: A Survey and the Road Ahead. *IEEE Communications Manazine*, 43:112–119, December 2005.

[31] T. Stevens, B. Davies, and A. Fapojuwo. Cross Layer Signaling in Wireless Ad-hoc Networks Using Embedded Computers. In *Wireless 2005*, pages 62–67, Calgary, Alberta, Canada, July 2005.

[32] V. T.Raisinghani and S. Iyer. Cross-Layer Feedback Architecture for Mobile Device Protocol Stacks. *IEEE Communications Manazine*, 44:85–92, Jan. 2006.

[33] R. Winter, J. H. Schiler, N. Nikaein, and C. Bonnet. CrossTalk: Cross-Layer Decision Support Based on Global Knowledge. *IEEE Communications Manazine*, 44:93–99, Jan. 2006.

[34] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC Protocol Work Well in Multihop Wireless Ad Hoc Networks? *IEEE Communications Manazine*, 39:130–137, June 2001.

[35] W. Yuan, K. Nahrstedt, S. V. Adve, D. L. Jones, and R. H. Kravets. Design and Evaluation of a Cross-Layer Adaptation Framework for Mobile Multimedia Systems. In *Proceedings of Multimedia Computing and Networking*, Santa Clara, CA, Jan. 2003.