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 layered 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 which is then used to derive an architecture for their implementation that significantly preserves the networks structure. We present some preliminary 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" [6]. 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 the Internet is based on the hourglass model, for our purposes it is more useful to consider another layer model of networks, the OSI seven layer model [24], which for the four layers we will consider 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 wire or radio frequency (RF) link and must deal with both the analog

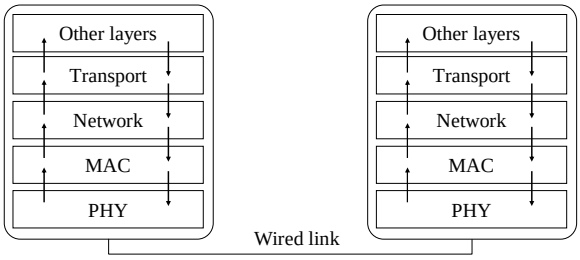


Figure 1. Conventional layered architecture

physical world 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 along one single hop in the network, including coordinating what sender is allowed to use a shared medium like the 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. 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 de-

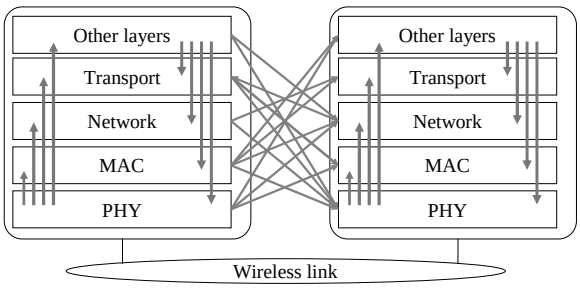


Figure 2. Cross-layer communication paths

fined as one might expect and that higher layers may make unwarranted assumptions about lower ones. The classic example is when TCP is run over a wireless link [1]. 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. Thus 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 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 [9, 20, 13].

In general wireless "links" differ from wired ones in a number of ways. For example, they are lossy and their bandwidth and latency may vary. 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 some of the ways that information may need to cross between layers, but 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 layer 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 should and will accommodate 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 [22, 23, 14]. The result are systems that are basically spaghetti code with limited structure. Thus far there has been essentially no general consideration of how to construct cross-layer adaptations in a systematic and modular manner. Our goal in this paper 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 conceptual framework to define a conceptual software architecture that allows us to implement adaptations within this space in a systematic way that preserves modularity.

We begin in Section 2 with an example adaptation, that will be used throughout the rest of the paper. Section 3 presents our taxonomy. Section 4 illustrates our architecture and Section 5 contains our preliminary validation results. We conclude in Section 6.

2 Example: Cross-Layer Rate Control

We developed and validated our taxonomy and architecture by considering a wide variety of example cross-layer adaptations. Here we motivate our discussion using one of these, cross-layer rate control [19]. 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 DCF mode of IEEE 802.11 [10], 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 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 subsequent data transmission. The need for cross layering arises because only the PHY knows the channel state,

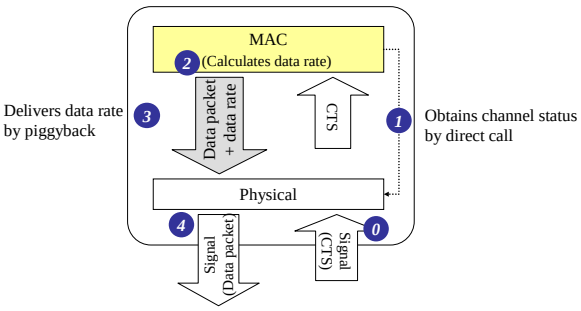


Figure 3. Cross-layer rate control

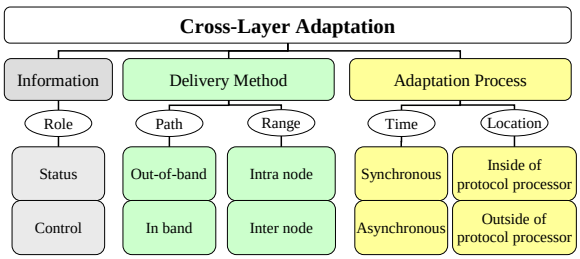


Figure 4. A complete taxonomy of cross-layer adaptation

but only the MAC knows which transmissions are control packets and which are data.

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 [8] 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 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.

Two other examples that we considered are worth mentioning briefly. One is cross-layer protocol reconfiguration [2], which allows a node to switch from using one wireless protocol to another. Such reconfigurations are not triggered or coordinated with packet reception or transmission, but rather occur when a node detects that other nodes in its vicinity are using different protocol. The second, is the implementation of cross-layer extensions for the local agile routing protocol [21]. In this case, nodes monitor the channel and exchange information between themselves to assists in creating advantageous routes. For our purposes, 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 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 will serve to guide the creation of our architecture itself.

Considering rate control, we see that there are three primary constituents involved [18]. 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 channel condition. Second, some information is communicated across layers, in this case, the channel condition and the rate itself. 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. 4 shows our complete taxonomy, with these three basic categories, Information, Delivery Method, and Adaptation Process, making up the top-level.

Using our 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, our 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. Our 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, our example only illustrates just 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 a additional distinction for the Delivery Method based on *range*. For the basic protocol, the range is *Intra node* and for the internode version it is *Inter node*. These distinctions are important because any mechanism that communicates information between nodes is inherently more expensive and failure prone than one that does not.

The final distinctions present in our taxonomy are not found in our current example, but are present in other examples we have considered. They concern the adaptation process. The timing of the process can also be *Asynchronous*, which occurs when the timing of the adaptation is not coordinated with packet transmission or reception. It could be for example triggered detecting that a new protocol is currently being used triggering a protocol reconfiguration. Finally, the location of the adaptation process can be *Outside the protocol processor*. This means that the adaptation is not part of the packet processing flow. This, for example, might occur when the process needs to coordinate between a number of protocol layers.

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 [12], our architecture is a conceptual one, although in practice it can serve as a concrete one as well. Further, since it describes a range of systems, in the sense of [17], it is a generic architecture.

We begin by presenting a series of high level goals and requirements for the architecture [5]. We then present some key architectural decisions. We then use the rate control example to flesh out the details of the architecture. Finally we motivate aspects of the architecture that were not covered by the example.

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 had the basic underlaying structure required.

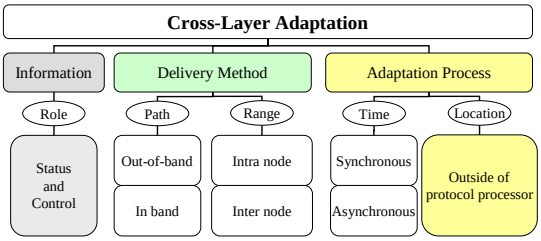


Figure 5. Refinement of our taxonomy based on architectural decisions

4.1 Key Architectural Decisions

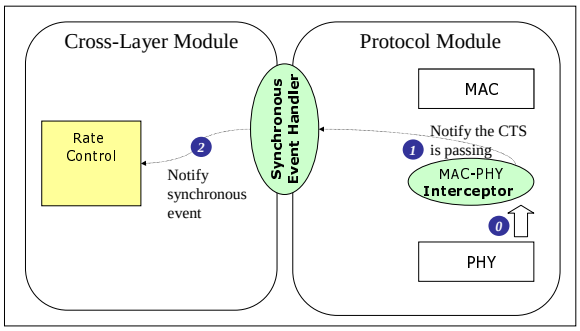
Fig. 5 shows our taxonomy after we have applied two key architectural decisions. The first decision is simple. Although functionally different, the implementation of cross-layer information does not vary based on whether the data is used as status or control. Thus we can merge these two categories for 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.

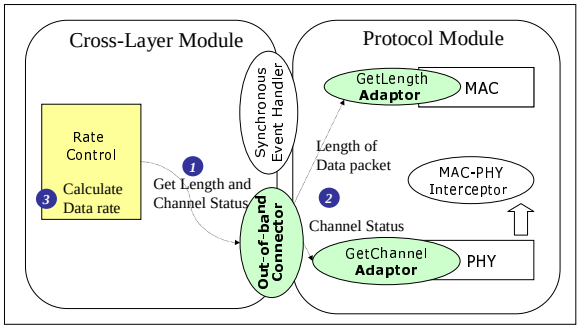
4.2 Example Driven Architecture

Fig. 6 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.

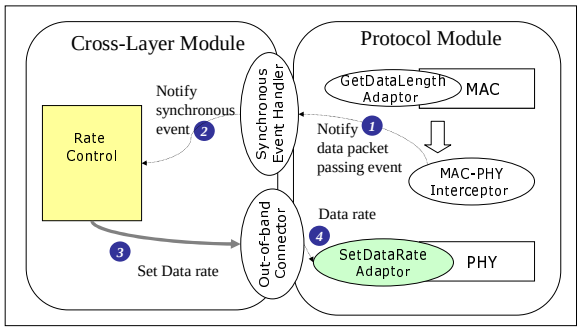
The first stage (Fig. 6(a)) shows the mechanisms needed to support a synchronous adaptation process outside of the protocol module. Note that the rate control adaptation has been placed in a separate cross-layer module. A key requirement is that when the packet moves from the PHY to the MAC, the adaptation process must be notified if that



(a) Components for synchronous process



(b) Components for out-of-band delivery



(c) Reusing the components for the rest of process

Figure 6. Architectural solutions for cross-layer rate control

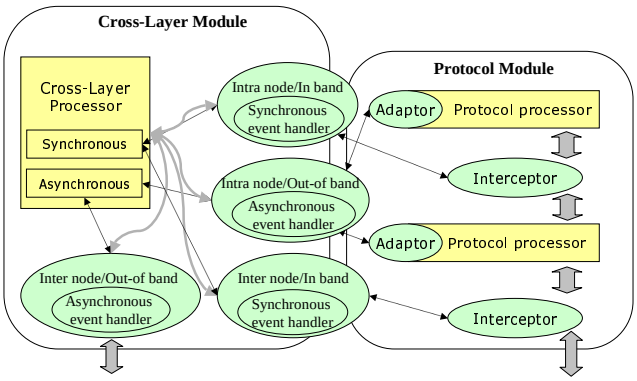


Figure 7. A generic architecture for cross-layer adaptations in wireless networks

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. 6(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. 6(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.

4.3 Completing the Architecture

Fig. 7 shows all the details of our architecture. Many 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 [15]. Disregarding the event handler aspect for now, we see four kinds of connectors, corresponding to the four delivery mechanisms in the taxonomy. The Intra-node connectors are used inside a single node to integrate existing protocol modules with our architecture [7]. 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 Inter-node 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 not triggered by packet passage, but are generated when the event handler detects a change of information within a protocol processor.

4.4 Further refinements

If we wish to actually implement an adaptation in a loosely coupled way, our conceptual architecture can serve as a concrete one as well. However, in the interest of performance, we may wish to use a concrete architecture that has less overhead. For example, we might merge the interceptors into the layer above or below them, thus reducing the amount of packet handling. Similarly, the event handlers may be merged into the protocol processing. For asynchronous handlers this might eliminate the need to poll for changes. We might even merge the cross-layer processing into a particular protocol processor, thus eliminating a substantial amount of communication. 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 has been done by careful consideration and paper design of the examples found in Section 2, as well as others. We are beginning a more substantial validation using the basic strategy of implementing 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 will be done in the context of our Hydra testbed. 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 [16]. Click also provides network support and interfaces to the Linux TCP/IP stack allowing full end-to-end application to application experiments.

The current Hydra implementation is similar to 802.11a [11]. 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 Rate Adaptation

We have implemented the inter-node version of rate control, both using our fully decoupled architecture and in the ad-hoc manner that might be considered "conventional." Both of these implementations are operational inside of Hydra, but our experience with them thus far is limited.

The conventional implementation required numerous changes to the protocol layers. The MAC was modified to allow it to perform the rate adaptation and to conform to the new CTS packet format that delivers the channel information from the receiver to the transmitter. The interfaces between the MAC and the PHY were changed because of data format changes. This was required because the MAC and PHY are in different address spaces and so packets must be marshalled and unmarshalled when they move between them.

Within our architecture, we extended the example shown in Section 4.2 to the inter-node case. Rate adaptation outside the MAC only requires the adaptors for the MAC and the PHY that allow the connectors to access data length and channel information and to set the rate. Interceptors transparently change the format of the CTS packet. The key challenge was that the MAC is implemented using Click while the PHY uses GNU Radio. This meant that the interceptors and adaptors needed to be implemented differently to conform to each implementation environment. Further the connectors are divided into two levels. Connectors in Click and GNU Radio manage their interceptors and adaptors and communicate with global connectors that provide cross-layer processors with event notification and data delivery. This allows rate adaptation to be independent of the infrastructures and to freely change its operation without significant impact on existing layers. Many of the mechanisms implemented for this example are reusable across other adaptations. Our preliminary conclusion is that our implementation confirms that our architecture substantially meets its goals.

5.3 Other Examples

The remaining aspects of our architecture that need validation are the asynchronous event handler and internode version of the out-of-band connector. We plan to implement cross-layer protocol reconfiguration and cross-layer extensions for the local agile routing protocol to explore these aspects.

Cross-layer protocol reconfiguration [2] can be used to explore the issues in asynchronous event handling. Since the adaptation changes the configuration of the layers as the wireless communication standard in an area changes, it requires asynchronous processing that is triggered when a node detects a change of standard. Further reconfiguration of both the protocols and cross-layer adaptations will be used to show how our architecture addresses flexibility, extensibility and portability of systems, our main secondary goals.

Implementing cross-layer extensions for the local agile routing protocol [21] will allow us to validate internode version of the out-of-band connector. It 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.

6 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 some preliminary 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 ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 2–11, Berkeley, CA, Nov. 1995.

[2] L. Berlemann, R. Pabst, M. Schinnenburg, and B. Walke. Reconfigurable Multi-mode Protocol Reference Model Facilitating Modes Convergence. In *Proceedings of the 11th European Wireless conference 2005 (EW 2005)*, pages 280–286, Nicosia, Cyprus, Apr. 2005.

[3] E. Blossom. GNU Radio, Aug. 2006. http://www.gnu.org/software/gnuradio/index.html.

[4] E. Blossom. UniversalSoftwareRadioPeripheral, Mar. 2006. http://comsec.com/wiki?UniversalSoftwareRadioPeripheral.

[5] 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 17 Workshop on Software Architecture (WOSS)*, pages 31–43, Seattle, WA, Apr. 1995.

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

[7] 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 (ASE)*, pages 223–231, San Diego, CA, Nov. 2001.

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

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

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

[11] 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, Sept. 2000.

[12] 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 (ICSE'99)*, pages 555–563, Los Angeles, CA, May 1999.

[13] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J.-C. Chen. A Survey of Energy Efficient Network Protocols for Wireless Networks. *Wireless Networks*, 7:343–358, Apr. 2001.

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

[15] N. R. Mehta, N. Medvidovic, and S. Phadke. Towards a Taxonomy of Software Connectors. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'00)*, pages 178–187, Limerick, Ireland, June 2000.

[16] 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, Dec. 1999.

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

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

[19] 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, Oct. 2002.

[20] V. T. Raisinghani and S. Iyer. Cross-Layer Design Optimizations in Wireless Protocol Stacks. *Computer Communications*, 27:720–725, May 2004.

[21] C. A. Santivanez, R. Ramanathan, and I. Stavrakakis. Making Link-State Routing Scale for Ad Hoc Networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking and computing (MobiHoc)*, pages 22–32, Long Beach, CA, Oct. 2001.

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

[23] Q. Wang and M. A. Abu-Rgheff. Cross-Layer Signalling for Next-Generation Wireless Systems. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC'03)*, pages 1084–1089, New Orleans, LA, Mar. 2003.

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