X-Tags: Efficient Data Processing using Cross Layer Hints

Suman Banerjee, Ashok Agrawala
MIND Lab, UMIACS and Department of Computer Science
University of Maryland, College Park, MD 20742, USA
Email: {suman,agrawala}@cs.md.edu

Michael J. Kramer
Aerospace Integration Science Center
Chantilly, VA 20151, USA
Email: Michael.J.Kramer@aero.org

UMIACS-TR 2002-59 and CS-TR 4379
June 2002

***Abstract*— Conventional network stacks define a layered architecture, where each layer implements a set of services and exports a well-defined interface to be used by its immediate upper layer. A key design choice of the layered architecture has been to provide isolation between the functional modules of distinct layers. While such an architecture provides an useful abstraction for system development, the strict isolation of this layered architecture limits the flexibility of tailoring the behavior of the lower layers of the stack to the needs of the application. In this paper we define a new architecture, called X-Tags, which allows flexible interaction between layers for cooperative data processing without impacting the isolation property. In this architecture, applications use special tags to provide semantic hints for data processing to lower layers. We motivate the usefulness of this architecture by describing its applicability to some emerging applications.**

I. Introduction

Conventional network stacks define a layered architecture, where each layer implements a set of services and exports a well-defined interface to be used by its immediate upper layer. A key design choice of the layered architecture has been to provide isolation between the functional modules of distinct layers. Such an abstraction is useful for system development in multiple ways. To quote Clark and Tennenhouse [1], "a major architectural benefit of such isolation is that it facilitates the implementation of subsystems whose scope is restricted to a small subset of the (protocol) suite's layers." However, they also observe that the layered architecture may well conflict with the efficient engineering of data processing at the end systems. Hence, layered engineering should not be thought of as fundamental, but only as one approach, which must be evaluated on the basis of overhead and simplicity against other designs. Therefore, one of the design principles proposed by Clark and Tennenhouse was *Integrated Layer Processing* (ILP). By blurring strict layering boundaries, ILP promotes a greater interaction between the operations performed at different layers and thus facilitates the implementor to define more efficient data processing mechanisms.

In this paper we define a new architecture, which we call X-Tags (pronounced cross-tags), that enable the benefits of ILP while maintaining the isolation property of the layered architecture. In this architecture, as data passes through different layers of the network stack, each layer can add some "tags" that serve as layer-specific hints to other layers. Tags added by one layer are interpreted and used by another layer as processing hints. In this paper, we present two examples of the applicability of the X-Tags architecture that can facilitate more efficient data processing on the network stack.

A. Why X-Tags?

The X-Tags architecture provides significant flexibility to implement efficient data processing mechanisms in the network stack. We explain the key idea of this architecture for a data sender's stack using Figure 1. One of the basic of tenets of traditional network stack implementations is the isolation property. Layer A is able to access services provided by its immediate lower layer, B, using the interface provided by the latter. Layer, B, interacts with its immediate lower layer, C, in the same manner. In the example, layers B and C each implement two different services (B_0 and B_1 for layer B and C_0 and C_1 for layer C). Layer A has two different data packets to transmit through the stack. Data packet 1 requires services B_0 and C_1 from layers B and C respectively prior to packet forwarding, while data packet 2 requires services B_1 and C_0 from the two layers.

In typical network stack implementations, services implemented by lower layers are not exposed to the upper layers. While layer A can choose service B_0 for data packet 1 and service B_1 for data packet 2 through the interface of layer B, there exists no mechanism by which layer A can directly access the services and functionality provided by layer C. This is shown in Panel 0, Figure 1. The isolation between layers is therefore, too restrictive in defining flexible access methods to services at different layers.

The only way layer A can access specific functionality of layer C, is if layer B explicitly exposes the layer

C functionality at its interface. In order for layer A to avail an arbitrary combination of these services, layer B needs to create and expose four different service primitives and access methods in its interface with layer A, one for each of the possible combinations. Layer A then explicitly chooses the appropriate access method of layer B for each of its data packets. The disadvantage of such a design is that it couples the implementation of the two layers (B and C) and thus breaks the isolation property of the layered architecture.

In Panel 2, Figure 1, we present the design of the X-Tags-enhanced network stack. In this design there is a single demultiplexer at the interface of each layer. Layer A specifies a set of tags with each data packet, which provide semantic hints to the lower layers for packet processing. In each layer, these tags are mapped to the set of service primitives that are implemented in this layer. By specifying the appropriate combination of hints, the data packet will be processed using the desired service primitives at the lower layers. In the example in Panel 1 of Figure 1, layer A specifies two tags: r and e for data packet 1. Tag r is mapped to service primitive B_0 in layer B and tag e is mapped to service primitive C_1 in layer C. This allows each layer to independently apply the appropriate service primitive for this packet as using the hints provided by the originator of the data source. In particular, the application layer can use this architecture to explicitly recommend packet processing mechanisms at different layers.

Thus, the X-Tags architecture allows a flexible interaction between the different layers without breaking the isolation property of the layered architecture. Note that unlike the simple enhancement to the traditional network stack (Panel 1), each layer still can independently implement the set of services that it provides.

Use of this tag-based mechanism to exchange data processing hints is beneficial to the system in two different ways: (a) implementation of each layer remains independent of the other layers and (b) intelligent packet processing can be performed at each layer using additional semantic information from other layers. An interesting feature of this architecture is that different layers may apply different service primitives to the same packet based on the same tag. In fact, interpretation of the same tag by a layer may differ between network stacks of two different devices, depending on the capabilities of the layer for that device.

We illustrate these aspects of the architecture in the next two sections using two examples, one for energy-constrained wireless devices and the other for data security for mobile devices. In Section IV we present details of the X-Tags architecture. In Section V we present some of the related work and we finally conclude and present future directions of work in Section VI.

II. Example I: Energy Efficient Packet forwarding for Wireless Devices

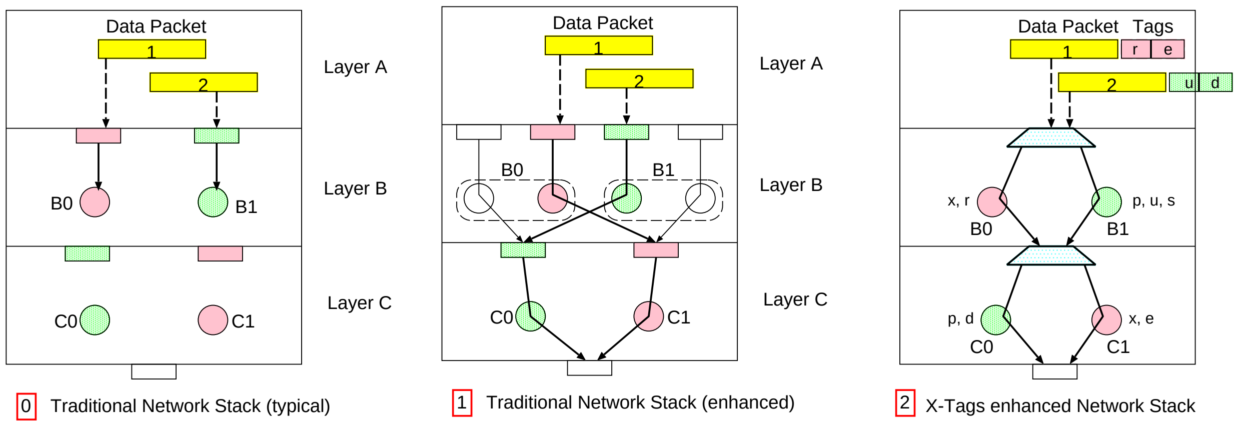
Energy efficient mechanisms for computing, signal processing and communication are key to the success of different wireless devices. Recent research has defined different energy-efficient link-layer forwarding techniques for such wireless devices [6], [2] However, different applications have different forwarding requirements from the link-layer. We now describe how the flexibility of the X-Tags architecture allows applications to choose appropriate mechanisms and services from different layers in the context of power-constrained wireless devices.

Wireless channels are inherently noisy in nature. The noise of the channels is time-varying and typically the bit error rate of the wireless channel is a monotonically decreasing function of the received power level for the signal.

We can, therefore, describe two different forwarding mechanisms for successful frame delivery in the link layer by varying the following choices:

- Choice of transmission power: If a high transmission power is used for the data, then the error rate on the channel is reduced. Thus one scheme for successful packet delivery across a noisy channel is to transmit packet with a high power. Clearly this would mean fewer packet corruptions due to noise and hence, fewer retransmissions. An advantage of this scheme is that the end-to-end latency of packet delivery is also reduced. However since packets are transmitted with higher power, the energy cost is high.
- Choice of transmission data rate: This is an alternative technique to combat channel noise. This technique is based on the observation that in many channel coding schemes, the energy required to transmit a packet can be significantly reduced by choosing a low transmission power and by transmitting the packet over a longer period of time [6]. Some of the existing wireless MAC layer protocols make use of this technique in choosing data transmission rates. The IEEE 802.11b standard [4] specifies three different data rates for transfer of data packets (1 Mbps, 2 Mbps and 11 Mbps). In different implementations of the IEEE 802.11b wireless LAN standard, the devices do not increase their transmission power. Instead they choose a lower bit rate for data transfer in presence of high channel errors to reduce packet losses.

Now consider two different applications on the wireless device that are transmitting packets with different require-



Data Packet
1
2
Layer A
Layer B
Layer C
B0
B1
C0
C1
0 Traditional Network Stack (typical)
1 Traditional Network Stack (enhanced)
Data Packet Tags
r e
u d
x, r
p, u, s
p, d
x, e
2 X-Tags enhanced Network Stack

Fig. 1. In a traditional network stack, the upper layer, A, interacts with the lower layer, B, only through the well-defined interface exported by B. Therefore, A can access the functionality of an even lower layer, C, if and only if B exports this functionality to A. In the X-Tags enhanced stack, data passes between neighboring layers though a demultiplexer and data processing is performed in each layer based on the tags and are independent of the other layers.

ments, namely a real-time audio application and a bulk data transfer application. A real-time audio application has no reliability requirement but is delay-sensitive and would require low delay for its packets. On the other hand, the bulk data transfer application requires end-to-end reliability but has no delay constraints. In power-constrained wireless devices, bulk data transfers should be performed in the most energy efficient mechanism that is possible. Under the X-Tags architecture, both applications are able to indicate their requirements that its data packets desires from the wireless infrastructure as application-level hints using tags that are added in the application space.

We can instantiate this example in Panel 2, Figure 1. We consider layers A, B and C in the figure to be the application, transport and the link layers respectively.

The primitives B_0 and B_1 at the transport layer are *reliable* and *un-reliable* data delivery mechanisms (e.g. TCP and UDP respectively). The primitive C_0 at the link layer implements low-delay forwarding, i.e. when the channel noise is high, a high transmission power is used to reduce the bit-error rate on the channel and consequently the end-to-end latency of data packets. Although this is inefficient use of the scarce power resources, it is necessary to reduce the end-to-end latency to provide a good audio quality. The primitive C_1 implements energy-efficient forwarding by reducing the transmission bit rate in presence of high channel noise.

Data packet 1 shown in the figure belongs to the bulk data transfer application — it requires reliability and at the same time is not delay-sensitive. The data processing services for packets for this flow can be *composed* by the application by using the X-Tags architecture as follows: It uses the two tags, r and e, where tag r corresponds to reliability requirement (note that it is mapped to primitive B_0 at the transport layer) and tag e corresponds to delay-insensitive energy-efficient link-layer forwarding (and is mapped to primitive C_1 at the link layer).

Data packet 2 belongs to a real-time audio application. For packets belonging to this flow, the application uses the tags, u for reliable transfer and d for delay-sensitive link-layer forwarding. As the packets pass through the different layers, they are appropriately processed and can avail the desired set of services.

This example demonstrates the flexibility of the X-Tags architecture in which an application can compose the services desired by its packets by just providing application-level hints. Note that it is not necessary that all stacks interpret these hints in an uniform manner. For example, energy efficiency is not a concern in wired networks and devices. Therefore in the link layer of a wired device, both tags e and d will be mapped to the same default link-layer forwarding technique.

III. Example II: Mobility-Aware Data Security

We now consider another example application of the X-Tags architecture. Mobility support has slowly become an important component of the Internet. As the number of mobile devices have increased, new techniques have emerged to provide users with a seamless experience as they change their network point of attachments. This implies that the mobile device should be able to change its network point of attachment without changing its IP address.

Mobile IP [5] is one of the main proposals that allow transparent routing of IP packets to mobile destinations in the Internet. Each mobile device has a long term IP address in the address space of its home network. In the Mobile IP solution, each mobile device registers itself with an

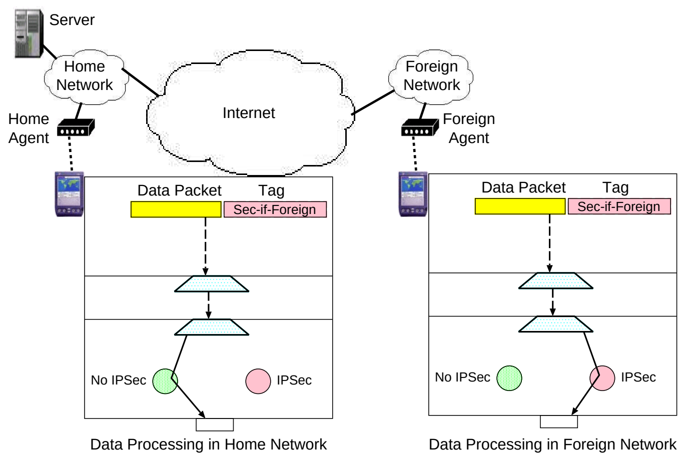


Fig. 2. Conditional data security for Mobile IP-enabled data stack using the X-Tags architecture.

entity, called the Home Agent (HA), in the home network. When away from its home network, a temporary "care-of address" is associated with the mobile node and reflects the mobile node's current point of attachment. Data packets destined for the mobile device will typically reach the home network of the device, and will be tunneled by the HA to the care-of address.

A typical corporate user on the road will continue access sensitive corporate data from the servers located at the home network. However seamless experience of Mobile IP ensures that such underlying changes to the network point of attachment are not visible at the application layer. The integrity and privacy of sensitive corporate data is a major concern when being accessed from a foreign untrusted network and so the data needs to be appropriately encrypted. However, this may not be an issue when the mobile user is accessing the servers directly from the home network. Data encryption and decryptions are computationally expensive operations (specially for some of the palm devices) and are avoided when the perceived threat is low (i.e. connecting directly via the home network).

The X-Tags architecture can be used to put into place such a security policy as shown in Figure 2. The application will add a "secure-if-foreign" tag to all data packets. Let us assume for this example that data security is provided by invoking IPSec functionalities [1]. Therefore, the IP layer of the mobile device will invoke IPSec mechanisms if and only it is roaming in a foreign network. This information is not available to the application in traditional network stacks due to the isolation property of the layered architecture. However, by allowing the application to provide such hints to the IP layer, such a security policy can be implemented in an efficient manner without violating the clean implementation of a layered stack.

[1] See IP Security Protocol (ipsec) charter at http://www.ietf.org/html.charters/ipsec-charter.html

As both these example applications show, an important feature of the X-Tags framework is the flexibility available to specific applications to tailor the services of the different layers to its various needs.

IV. Architecture

In this section, we first describe the basic X-Tags architecture for a single host and the tags are exchanged between different layers of the host. If all the communicating hosts are X-Tags-capable, then the entire end-to-end interaction can be X-Tags-aware. Later in this section we briefly outline some extensions required for an end-to-end implementation of X-Tags.

A. Basics

In the X-Tags architecture, each separate unit of self-contained code that interacts with other such units through a well-defined interface is considered to be a layer. Apart from the traditional layers of a network stack, multiple application-layer protocols can exist in a host and will be considered to be separate layers.

We call the layer which generates data in the X-Tags architecture the *principal* for the data. The principal typically is aware of the semantics of the data and therefore, generates the data processing hints, in form of tags. However, any other layer can also append and/or update tags for the data.

The format of a X-Tags data unit is shown in Figure 3. A X-Tags header is created by the principal along with the data. The X-Tags header consists of a *version number* field, a *count* field that indicates the number of tags that follow, and the *total length* of all the tags (in bytes) in the X-Tags header. This is followed by the individual tags. Each tag consists of a *tag type*, a *tag source* which indicates the entity that added this tag to the data and *tag length*, which indicates the total length of this tag (in bytes). Finally, each tag can carry an optional variable length data field, that can be used to convey additional semantic information about the data. The length of each of these fields are shown in Figure 3. As the data passes down the stack, layer-specific headers are added to the data. We extend each of the layer headers to have a pointer to the beginning of the X-Tags header. This makes the X-Tags header directly accessible to the layer processing logic.

B. Data Processing and Control Plane

The data generated by the principal is passed down the layers and is processed at each layer based on the tags associated with the data. It is also possible that a layer, other

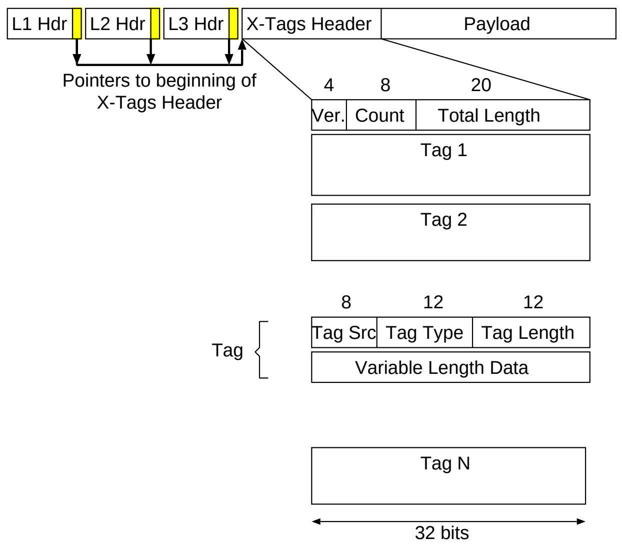


Fig. 3. The X-Tags data format. Each of the layer headers has a pointer to the X-Tags header.

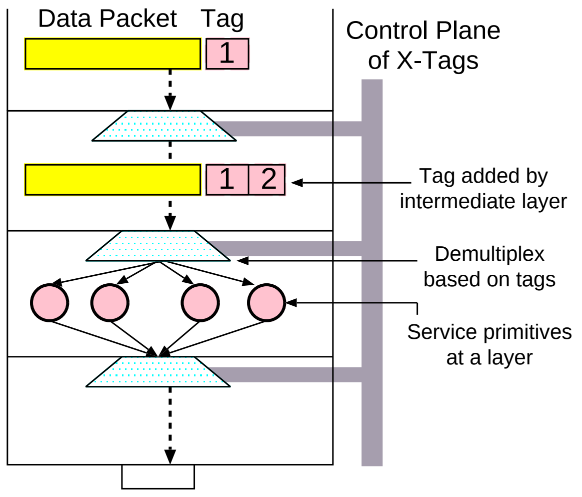

Data Packet
Tag
1
Control Plane of X-Tags
1
2
Tag added by intermediate layer
Demultiplex based on tags
Service primitives at a layer

Fig. 4. Architecture of X-Tags.

Function	Annotation
SetId(i)	Sets i as the identifier for the layer (boot-up)
AddTag(*tag*)	Maps a tag to a service primitive
DeleteTag(*tag*)	Deletes the mapping above
QueryTags(i)	Finds the list of tags that are supported by layer i
TestConflict(i,*TagSet*)	Tests and identifies tag conflicts at layer i

TABLE I
CONTROL PLANE FUNCTIONS

than the principal, generates and adds one or more tags to the data. Such a flexibility is useful in certain scenarios. Consider a case in which the application adds a "Secure" tag to the data (i.e. encryption is needed prior to data transmission). Depending on the configuration of the host, secure services can be provided at different layers. The first layer that provides this service encrypts the data and removes the "Secure" tag, so that no subsequent layer attempts to perform any redundant re-encryption of the data.

Each layer of the stack implements a set of different services. The interface of the layer uses the tags in the data packet to choose one or more services that are applied to the data. In some of the applications that we have described, the choice of an appropriate service primitive would depend on the *tag type*. In some other cases (as shown in Section III) it will depend on the *tag type* as well as state information local to that layer (e.g. whether the mobile device is currently attached to the home network or not). However, in a more general case, the demultiplexing may be based on the *tag type*, the state information at the layer as well as specific semantic information in optional data field.

Each layer maps each possible tag to a subset (including an empty subset) of its service primitives. Again in the most typical case, a tag will be mapped to a single service primitive. Each layer also identifies one of its primitives as the default primitive, which is used by a data packet if none of its tags are mapped to any specific primitive. It is possible that a data packet carries a "contradicting" tag set, i.e. the set of service primitives chosen by the tag set of the data packet are conflicting operations. For example, a data packet might erroneously have both "reliable" and "unreliable" tags (which correspond to the reliable and unreliable transport layer services). One way to prevent such conflicts is to define a set of rules when tags are assigned such that no conflicts are created at any layer. This approach is called *conflict avoidance*. Instead we use a *conflict resolution* approach. In this scheme each layer defines a priority ordering of its service primitives. If the tag set of a data packet chooses conflicting service primitives, then only the service primitive higher in the priority order is applied to the data. This approach is simpler to implement and can be done locally at each layer.

The control plane of the X-Tags architecture is used for out-of-band interaction between the layers. At system boot-up, it assigns an unique identifier to each layer. A layer can bind and un-bind a tag to one or more of its service primitives and uses the *AddTag* and *DeleteTag* functions to signal this information. These functions also update the priority order information in which tags will be processed in case of conflicts. Each layer can find out what tags are explicitly mapped to different service primitives in another layer using the *QueryTags* function. Additionally the control plane also supports queries about conflicts between a set of tags and the priority order used by a layer to resolve such conflicts. This is done using the *TestConflict* function. The principal of data can use this function to evaluate how the data will be processed at the queried layer. The relevant set of functions on the control plane that interact with the layers is specified in Table I.

C. Extensions for End-to-end Support

If the receiving end-host of data communication does not support X-Tags, then the X-Tags header information associated with the data packet is stripped out at the sender before it is transmitted to the receiving counterpart(s). However, if all the communicating end-hosts implement X-Tags, then these semantic tags associated with the data can be retained in the data transmission and can be used for efficient and flexible data processing at the receiver(s). Clearly it requires specific coordination between the end-hosts so that the semantics of the data tags are mutually understood. The control plane functionality needs to be extended so that the sending host can query the set of tags that are supported by different layers at the receiving host and vice versa.

V. Related Work

The idea of using tags to exchange hints between entities is not a new concept. Feldmeier [3] defined a data labeling technique that added application-specific information to data fragments. The goal of the technique was efficient fragmentation and re-assembly of data. In particular, Feldmeier observed that the data fragments (called chunks) could be mis-ordered in the network, and the data-labeling technique allowed, in particular, out-of-order chunks to be processed by the entire protocol stack without depending on the arrival of any other chunk. This data labeling technique is one instantiation of Application Level Framing (ALF) as proposed by Clark and Tennenhouse [1]. In the ALF proposal, Application Data Units (ADUs) are application-specified aggregates of data that should be manipulated by lower layers without further fragmentation. An ADU is thus the smallest unit of data which the application can independently deal with out of order. The Axon architecture [7] also proposed application-specific labeling and framing of data for efficient re-assembly purposes. In contrast, in X-Tags any layer in general, and the application in particular, can add semantic hints to the data, which can be used by other layers to apply suitable service primitives for efficient data processing.

Integrated Layer Processing (ILP) [1] is another approach for efficient data processing in network stacks. ILP blurs strict layering boundaries and promotes greater interaction between operations performed at different layers. The X-Tags architecture also promotes such interaction between layers. However, unlike ILP, we still maintain the layering boundaries. The goal of ILP is to improve protocol performance by appropriately re-ordering operations from different layers. In contrast the goal of X-Tags is to provide more flexibility to the applications in composing the functionalities available at different layers of the network stack.

VI. Conclusions and Future Work

We believe that the X-Tags architecture exposes a richer data processing interface to applications and can be used to tailor the network stack behavior to application-specific needs. Our continued work in this framework is two-fold:

- *Implementation of the architecture:* We are currently implementing the X-Tags framework for Unix-based systems. We have chosen the two applications described in this paper to demonstrate the flexibility of the system.
- *Instantiation of general-purpose tags and identification of the dependencies:* Ideally we would want a minimal set of tags that can be used to define all possible functionalities available in the network stack. Additionally, given a set of tags and the functionalities that they enable, we also need a generalized framework to express the priority order among the service primitives at each layer for tag conflict resolution.

References

[1] D.D. Clark and D.L Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of ACM Sigcomm*, 1990.

[2] A. El Gamal, C. Nair, B. Prabhakar, E. Uysal-Biyikoglu, and S. Zahedi. Energy-efficient Scheduling of Packet Transmissions over Wireless Networks. In *Proceedings of IEEE Infocom*, June 2002.

[3] D.C. Feldmeier. A Data Labelling Technique for High-Performance Protocol Processing and its Consequences. In *Proceedings of ACM Sigcomm*, 1993.

[4] IEEE. Wireless LAN medium access control (MAC) and physical layer (PHY) specification, Standard 802.11, 1999.

[5] C.E. Perkins. IP Mobility Support. RFC 2002, IETF, October 1996.

[6] B. Prabhakar, E. Uysal-Biyikoglu, and A. El Gamal. Energy-efficient Transmission over a Wireless Link via Lazy Packet Scheduling. In *Proceedings of IEEE Infocom*, April 2001.

[7] J. Sterbenz and G. Parulkar. Axon: A High-Speed Communication Architecture for Distributed Applications. In *Proceedings of IEEE Infocom*, June 1990.