I
Ad Hoc Wireless Networks

The maturity of wireless transmissions and the popularity of portable computing devices have made the dream of "communication anytime and anywhere" possible. An ad hoc wireless network is a good choice for fulfilling this dream. An ad hoc wireless network consists of a set of mobile hosts operating without the aid of an established infrastructure of centralized administration. Communication is done through wireless links among mobile hosts through their antennas. Due to concerns such as radio power limitation and channel utilization, a mobile host may not be able to communicate directly with other hosts in a single-hop fashion. In this case, a multihop scenario occurs, in which the packets sent by the source host must be relayed by several intermediate hosts before reaching the destination host.

Although military tactical communication is still considered the primary application for ad hoc wireless networks, commercial interest in this type of network continues to grow. Applications such as law enforcement operation, commercial and educational use, and sensor networks are just a few possible commercial examples. There are several technical challenges related to the ad hoc wireless network. In ad hoc wireless networks, the topology is highly dynamic and frequent changes in the topology may be difficult to predict. With the use of wireless links, the network suffers from higher loss rates, and can experience higher delays and jitter. In addition, physical security is limited due to the wireless transmission. Finally, as ad hoc wireless network nodes rely on batteries, energy saving is an important system design criterion.

Most of the existing works in ad hoc wireless networks focus on issues related to the network layer, such as routing and broadcasting. Routing protocols in ad hoc wireless networks fall into either *proactive* or *reactive*, although a combination of proactive and reactive is also possible. In proactive routing, routes to all destinations are computed *a priori* and are maintained in the background via a periodic update process. Route information is maintained either as routing tables or as global link state information. In reactive routing, a route to a specific destination is computed "on demand," that is, only when needed. To efficiently use resources in controlling large dynamic networks, hierarchical routing, including cluster based and dominating set based, is normally used.

Many other technical issues are discussed through the use of protocol stacks where at least four layers are used:

1. Physical layer: responsible for frequency selection, carrier frequency generation, signal detection, modulation, and data encryption.
2. Data link layer: responsible for the multiplexing of data streams, data frame detection, medium access, and error control.
3. Network layer: responsible for forwarding the data to appropriate destinations.
4. Application layer: responsible for supporting various applications.

The 19 chapters in this section cover a wide range of topics across multiple layers: MAC (part of the data link layer), network, and applications. One chapter is devoted to the cross-layer architecture for ad hoc wireless networks. Several chapters deal with various efficient and scalable routing, including multicasting and geocasting, in ad hoc wireless networks. One chapter discusses routing in a selfish wireless network. Three chapters present some recent results on topology control while three other chapters are dedicated to energy-efficient design under several different system settings. The security and reliability issues are covered in two separate chapters. MAC protocols are given in one dedicated chapter. Of the three chapters about applications, one discusses ad hoc relaying in cellular networks, one uses ad hoc wireless networks for robust data communication, and one is devoted to the application in Bluetooth. This section ends with a chapter on scalable simulation for ad hoc wireless networks.

1

A Modular Cross-Layer Architecture for Ad Hoc Networks

Marco Conti, Jon Crowcroft, Gaia Maselli, and Giovanni Turi

The success of the cleanly layered Internet Architecture has promoted its adoption for wireless and mobile networks, including ad hoc networks. This has also fostered skepticism toward alternative approaches. However, a strict layered design is not flexible enough to cope with the dynamics of mobile networks and can prevent many classes of performance optimizations. To what extent, then, must developers modify the pure layered approach by introducing closer cooperation among protocols belonging to different layers?

Although the debate on cross-layer versus legacy-layer architecture has been around for a while, we propose a novel solution based on loosely coupled cross-layering, which constitutes a trade-off between the two extremes. Our solution allows for performance optimizations, but at the same time maintains flexibility.

This innovative architecture not only makes room for techniques to design new ad hoc protocols, for which we present specific examples, but also opens up the possibility of research into the usage of cross-layering for the Internet more generally.

1.1 Introduction

The Internet transparently connects millions of heterogeneous devices, supporting a huge variety of communications. From a networking standpoint, its popularity is due to a core design that has made it extensible, and robust against evolving usage as well as failures. Now, mobile devices and wireless communications prompt the vision of networking without a network (ad hoc networking). This brings new challenging issues where the need for flexibility confronts ad hoc constraints. A careful architectural design for the ad hoc protocol stack is necessary to incorporate this emerging technology.

The Internet architecture layers protocol and network responsibilities, breaking down the networking system into modular components, and allowing for transparent improvements of single modules. In a *strict-layered* system, protocols are independent of each other and interact through well-defined (and static) interfaces: each layer implementation depends on the interfaces available from the lower layer, and those exported to the upper layer. Strict-layering provides flexibility to a system's architecture: extensions introduced into single levels do not affect the rest of the system. The separation of concerns brings the added benefits of minimizing development costs by re-using existing code. This design approach relies on "horizontal" communication between peer protocol layers on the sender and receiver devices (the dashed arrows in Figure 1.1). The result is a trend to spend bandwidth (an abundant resource in the Internet) instead of processing power and storage.

Several aspects of the Internet architecture have led to the adoption of this strict-layer approach also for mobile ad hoc networks. Some of these aspects include (1) the "IP-centric" view of ad hoc networks; and (2) the flexibility offered by independent layers, which allows for reuse of existing software. The choice of the layered approach is reenforced by the fact that ad hoc networks are considered mobile extensions of the Internet, and hence the protocol stack must be suitable. However, this design principle clashes with the following facts:

1. Issues such as energy management, security, and cooperation characterize the whole stack and cannot be solved inside a single layer.
2. Ad hoc networks and the Internet have conflicting constraints; and while the former are dynamic, the latter is relatively static.

Some guidelines to approach these problems point to an enhancement of "vertical" communication in a protocol stack (see Figure 1.1),[1,2] as a way to reduce peer (horizontal) communication, and hence conserve bandwidth. Vertical communication, especially between nonadjacent layers, facilitates local data retrieval, otherwise carried out through network communication. The practice of accessing not only the next lower layer, but also other layers of the protocol stack, leads to *cross-layering* to allow performance improvements. The main downside of strict-layering is that it hinders extensibility: a new, higher-level component can only build on what is provided by the next lower layer.[3] Hence, if one layer needs to access functionality or information provided by a nonadjacent layer, then an intermediate extension should be

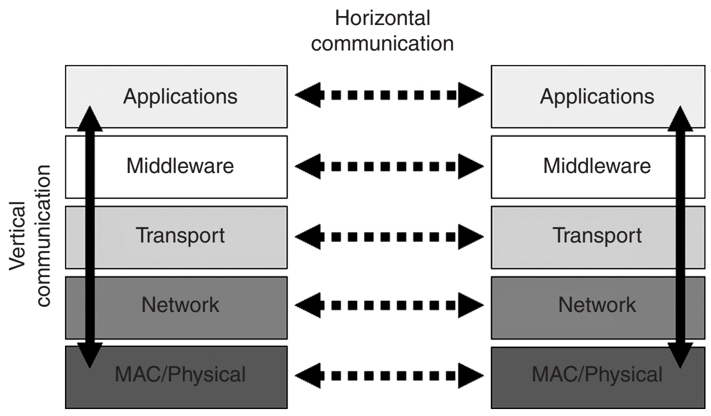

FIGURE 1.1 The Internet emphasizes horizontal communication between peer protocol layers to save router resources, while ad hoc networking promotes vertical interaction to conserve bandwidth.

devised. Cross-layering allows nonadjacent protocols to directly interact, making overall optimizations possible and achieving extensibility at the eventual expense of flexibility.

In the literature there is much work showing the potential of cross-layering for isolated performance improvements in ad hoc networks. However, the focus of that work is on specific problems, as it looks at the joint design of two to three layers only. For example, cross-layer interactions between the routing and the middleware layers allow the two levels to share information with each other through system profiles, in order to achieve high quality in accessing data.[4] An analogous example is given by Schollmeier et al.,[5] where a direct interaction between the network and the middleware layers, termed Mobile Peer Control Protocol, is used to push a reactive routing to maintain existing routes to members of a peer-to-peer overlay network. Yuen et al.[6] propose an interaction between the MAC and routing layers, where information like signal-to-noise ratio, link capacity, and MAC packet delay is communicated to the routing protocol for the selection of optimal routes. Another example is the joint balancing of optimal congestion control at the transport layer with power control at the physical layer.[7] This work observes how congestion control is solved in the Internet at the transport layer, assuming that link capacities are fixed quantities. In ad hoc networks, this is not a good assumption, as transmission power, and hence throughput, can be dynamically adapted on each link. Last, but not least, Kozat et al.[8] propose cross-layer interaction between physical, MAC, and routing layers to perform joint power control and link scheduling as an optimized objective.

Although these solutions are clear examples of optimization introduced by cross-layering, the drawback on the resulting systems is that they contain tightly coupled, and therefore mutually dependent components. Additionally, while an individual suggestion for cross-layer design, in isolation, may appear appealing, combining them all together could result in interference among the various optimizations.[9] From an architectural point of view, this approach leads to an "unbridled" stack design, difficult to maintain efficiently, because every modification must be propagated across all protocols. To give an example of interfering optimizations, let us consider an adaptation loop between a rate-adaptive MAC and minimal hop routing protocol (most ad hoc routing protocols are minimum hop). A rate-adaptive MAC would be able to analyze the quality of channels, suggesting higher layers on the outgoing links, which provide the higher data rates in correspondence with shorter distances. This conflicts with typical decisions of a minimum hop routing protocol, which chooses a longer link (for which the signal strength and data rate are typically lower) to reach the destination while using as few hops as possible.

We claim that cross-layering can be achieved, maintaining the layer separation principle, with the introduction of a vertical module, called *Network Status** (NeSt), which controls all cross-layer interactions (see Section 1.2). The NeSt aims at generalizing and abstracting vertical communications, getting rid of the tight coupling from an architectural standpoint. The key aspect is that protocols are still implemented in isolation inside each layer, offering the advantages of:

- Allowing for full compatibility with standards, as NeSt does not modify each layer's core functions
- Providing a robust upgrade environment, which allows the addition or removal of protocols belonging to different layers from the stack, without modifying operations at other layers
- Maintaining the benefits of a modular architecture (layer separation is achieved by standardizing access to the NeSt)

In addition to the advantages of a full cross-layer design, which still satisfies the layer separation principle, the NeSt provides full context awareness at all layers. Information regarding the network topology, energy level, local position, etc. is made available by the NeSt to all layers, to achieve optimizations, and offers performance gains from an overhead point of view. Although this awareness is restricted to the node's local view, protocols can be designed so as to adapt the system to highly variable network conditions (the typical ad hoc characteristic).

*This term indicates the collection of network information that a node gathers at all layers. It should not be confused with a concept of globally shared network context.

This innovative architecture opens research opportunities for techniques to design and evaluate new ad hoc protocols (see Section 1.3), but also remains compliant with the usage of legacy implementations, introducing new challenging issues concerning the usage of cross-layering for the Internet more generally (see Section 1.4).

1.2 Toward Loosely Coupled Cross-Layering

One of the main problems caused by direct cross-layer interactions (as already discussed in Section 1.1) is the resulting *tight coupling* of interested entities. To solve this problem, the NeSt stands vertically beside the network stack (as shown in Figure 1.2), handling eventual cross-layer interactions among protocols. That is, the NeSt plays the role of intermediary, providing standard models to design protocol interactions. While the new component uniformly manages vertical exchange of information between protocols, usual network functions still take place layer-by-layer through standardized interfaces, which remain unaltered. This introduced level of indirection maintains the *loosely coupled* characteristic of Internet protocols, preserving the flexible nature of a layered architecture.

The idea is to have the NeSt exporting an interface toward protocols, so as to allow sharing of information and reaction to particular events. In this way, cross-layer interactions do not directly take place between the interested protocols, but are implemented using the abstractions exported by the NeSt. This approach allow protocol designers to handle new cross-layer interactions apart, without modifying the interfaces between adjacent layers. The work described by Conti et al.[10,11] introduces this idea in the context of pure ad hoc networking. This work extends the definition of the NeSt interaction models, presenting the exported interface. This is to evolve toward a general-purpose component, eventually suitable for cross-layering in a future Internet architecture.

1.2.1 Overview of NeSt Functionalities

The NeSt supports cross-layering implementation with two models of interaction between protocols: *synchronous* and *asynchronous*. Protocols interact synchronously when they share private data (i.e., internal status collected during their normal functioning). A request for private data takes place on-demand, with a protocol querying the NeSt to retrieve data produced at other layers, and waiting for the result. Asynchronous interactions characterize the occurrence of specified conditions, to which protocols may be willing to react. As such conditions are occasional (i.e., not deliberate), protocols are required to subscribe for their occurrences. In other words, protocols subscribe for events they are interested in, and then return to their work. The NeSt, in turn, is responsible for delivering eventual occurrences to the right subscribers. Specifically, we consider two types of events: *internal* and *external*. Internal events are directly generated inside the protocols. Picking just one example, the routing protocol notifies the rest of the stack about a "broken route" event, whenever it discovers the failure of a preexisting route. On the other side, external events are discovered inside the NeSt on the basis of instructions provided by subscriber protocols. An

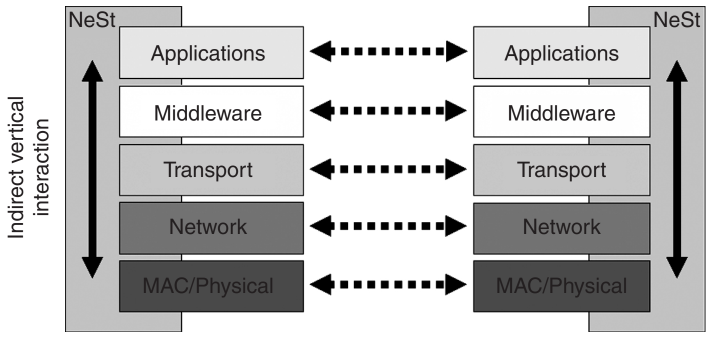

FIGURE 1.2 An architectural trade-off for loosely coupled vertical protocol interactions.

example of an external event is a condition on the host energy level. A protocol can subscribe for a "battery-low" event, specifying an energy threshold to the NeSt, which in turn will notify the protocol when the battery power falls below the given value.

As the NeSt represents a level of indirection in the treatment of cross-layer interactions, an agreement for common-data and events representation inside the vertical component is a fundamental requirement. Protocols must agree on a common representation of shared information, in order to guarantee loose coupling. To this end, the NeSt works with *abstractions* of data and events, intended as a set of data structures that comprehensively reflect the relevant (from a cross-layering standpoint) information and special conditions used throughout the stack. A straightforward example is the topology information collected by a routing protocol. To abstract from implementation details of particular routing protocols, topology data can be represented as a graph inside the NeSt. Therefore, the NeSt becomes the provider of shared data, which appear independent of its origin and hence usable by each protocol.

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

1.2.2 The NeSt Interface

To give a technical view of the vertical functionalities, we assume that the language used by the NeSt to interface the protocol stack allows for declaration of functions, procedures, and common data structures. We adopt the following notation to describe the NeSt interface:

$$\textit{functionName} : (\textit{input}) \rightarrow \textit{output}$$

Each protocol starts its interaction with the NeSt by *registering* to the vertical component. This operation assigns to each protocol a unique identifier (PID), as shown by step *a* in Figure 1.3. The registration is expected to happen once for all at protocol bootstrap time by calling

$$\textit{register} : () \rightarrow \textit{PID}$$

As described in the previous section, the NeSt does not generate shared data, but acts as an intermediary between protocols. More precisely, a protocol *seizes* the NeSt abstractions related to its internal functionalities and data structures. The example of the network topology suggests the routing protocol to acquire ownership of an abstract graph containing the collected routing information. This operation requires a protocol to identify itself, providing the PID, and to specify the abstraction's identifier (AID) together with the associated callback function (see step *b* in Figure 1.3). When invoked, the callback function fills out the

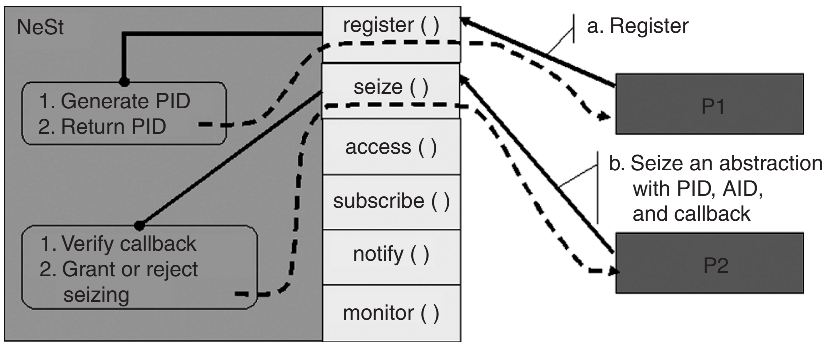

FIGURE 1.3 NeSt functionalities: register and seize.

abstraction, encoding protocol internal representation in NeSt format. Note that the callback invocation takes place asynchronously with the seizing operation, every time a fresh copy of the associated data is needed inside the NeSt. The entire process begins by calling

$$\textit{seize} : (PID, AID, \textit{readCallBack}()) \rightarrow \textit{result}$$

The result of a call to $seize()$ indicates the outcome of the ownership request.

Once an abstraction has been seized, the NeSt is able to satisfy queries of interested entities. A protocol *accesses* an abstraction by calling

$$\textit{access} : (PID, AID, \textit{filter}()) \rightarrow \textit{result}$$

This function shows that the caller must identify itself with a valid PID, providing also the abstraction identifier and a *filter* function. The latter parameter is a container of instructions for analyzing and selecting only information relevant to the caller's needs. The NeSt executes this call by spawning an internal computation that performs the following steps (see Figure 1.4):

1. Invoke the callback installed by the abstraction's owner (if any).
2. Filter the returned data locally (i.e., in the context of the NeSt).
3. Deliver the filtering result to the caller.

The remaining functions of the NeSt interface cope with asynchronous interactions. In the case of internal events, the role of the NeSt is to collect subscriptions, wait for notifications, and vertically dispatch occurrences to the appropriate subscribers, as shown in Figure 1.5. A protocol *subscribes* for an event by identifying itself and providing the event's identifier (EID), calling the function

$$\textit{subscribe} : (PID, EID) \rightarrow \textit{result}$$

To *notify* the occurrence of an event, a protocol must specify in addition to the event identifier (EID), information regarding the occurrence. This happens by calling the function

$$\textit{notify} : (PID, EID, \textit{info}) \rightarrow \textit{result}$$

After the notification of an event, the NeSt checks it against subscriptions, and dispatches the occurrence to each subscriber.

In the case of external events, protocols subscribe by instructing the NeSt on how to detect the event. The rules to detect an external event are represented by a monitor function that periodically checks the status of a NeSt abstraction. When the monitor detects the specified condition, the NeSt dispatches the information to the subscriber protocol. As shown in Figure 1.6, a protocol delegates the *monitoring* of an

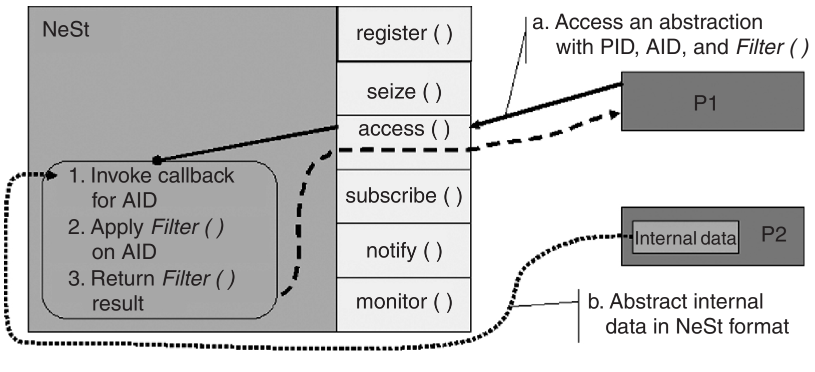
NeSt
register ( )
seize ( )
access ( )
subscribe ( )
notify ( )
monitor ( )
1. Invoke callback for AID
2. Apply Filter ( ) on AID
3. Return Filter ( ) result
a. Access an abstraction with PID, AID, and Filter ( )
P1
Internal data
P2
b. Abstract internal data in NeSt format

FIGURE 1.4 NeSt functionalities: access an abstraction.

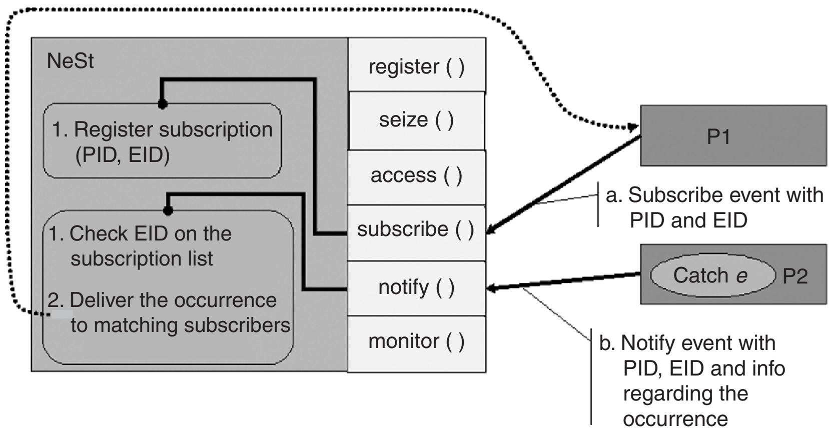
NeSt
1. Register subscription (PID, EID)
1. Check EID on the subscription list
2. Deliver the occurrence to matching subscribers
register ( )
seize ( )
access ( )
subscribe ( )
notify ( )
monitor ( )
P1
a. Subscribe event with PID and EID
Catch *e*
P2
b. Notify event with PID, EID and info regarding the occurrence

FIGURE 1.5 NeSt functionalities: management of internal events.

external event by passing to the NeSt a monitor function and the identifier of the target abstraction. This happens by calling

$$monitor : (PID, AID, monitor()) \rightarrow result$$

The NeSt serves this call by spawning a *persistent* computation (see Figure 1.6) that executes the following steps:

1. Verify the monitor (e.g., type checking).
2. While (true):
   a. Refresh the abstraction invoking the associated callback.
   b. Apply the monitor to the resulting content.
   c. If the monitor detects the special condition, then notify the requesting protocol.

The result of a call to *monitor*() only returns the outcome of the monitor's installation, while the notification of external events takes place asynchronously.

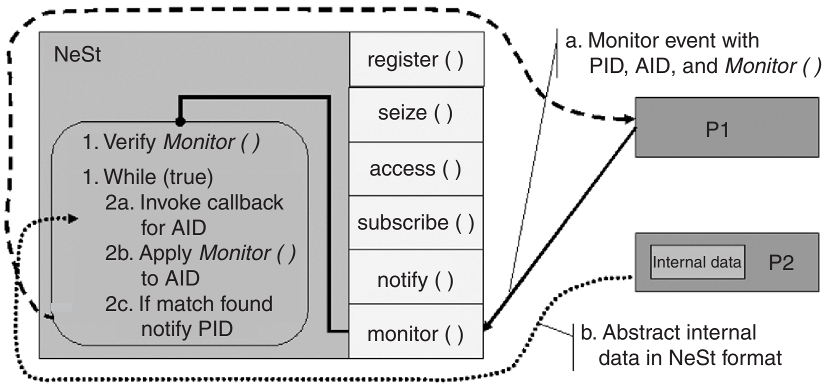
NeSt
1. Verify *Monitor ( )*
1. While (true)
2a. Invoke callback for AID
2b. Apply *Monitor ( )* to AID
2c. If match found notify PID
register ( )
seize ( )
access ( )
subscribe ( )
notify ( )
monitor ( )
a. Monitor event with PID, AID, and *Monitor ( )*
P1
Internal data
P2
b. Abstract internal data in NeSt format

FIGURE 1.6 NeSt functionalities: management of external events.

1.2.3 Design and Implementation Remarks

It is difficult to find comparisons to the proposed architecture as, to the best of our knowledge, there are no similar approaches in the organization of a protocol stack. However, there are important observations and remarks to be given.

First of all, the NeSt is a component dedicated to enabling optimization. If on the one hand it helps maintain the layering principle allowing loosely coupled interactions, on the other hand it must guarantee the appropriate level of real-timing. That is, when subjected to a heavy load of cross-layering, the NeSt should be responsive, avoiding making protocol efforts fruitless. For example, in the case of synchronous interactions where call-backs are employed, the NeSt should not degrade the performance of both the requestor and provider protocols. For these reasons, it advisable to pre-fetch and cache exported data (when possible), serving a series of accesses to the same abstraction with fewer callback executions. However, this approach also requires the presence of cache invalidation mechanisms, which protocols can use to stale pre-fetched or cached abstractions.

As presented here, the NeSt should come with an *a priori* set of abstractions for data and events, to which protocols adapt in order to cross-interact. A more mature and desirable approach would reverse the adaptation process, having the vertical component adapt to whatever the protocols provide. For example, this adaptation issue could be solved through the use of *reflection*, a characteristic of some modern programming languages[12] that enables introspection of software components, allowing for dynamic changes in behavior. A NeSt reflective API would allow each protocol to define its contribution to cross-layer interactions, providing an initial registration of profiles describing the data and the events it is able to share. The resulting data and event sharing would be more *content-based* than the presented *subject-based* mechanism. With this approach, the sole agreement between the two parties would regard the representation of protocol profiles. A solution could be the usage of a language that provides rules to define both profiles data and metadata, as for example the eXtensible Markup Language (XML). This solution would restrict the agreement on the set of tags (i.e., the *grammar*) to use in building profiles. Note that such use of higher-level programming languages would interest only initial negotiation phases between the NeSt and the protocols, without affecting the runtime performance.

One might argue that the NeSt exhibits some conceptual similarities with a management information base (MIB). An MIB is a collection of network-management information that can be accessed, for example, through the Simple Network Management Protocol (SNMP). SNMP facilitates the exchange of information between network devices and enables network administrators to manage performances, find and solve problems, and plan for network growth. Some NeSt functionalities could be realized through a local MIB (storing protocols information), to which other protocols can access in order to read and write data. However, the NeSt and MIBs target different goals. MIBs are designed for network statistics and *remote* management purposes, while the NeSt aims at overall *local* performance improvements. Furthermore, the MIB's nature makes it unsuitable for the real-time tasks typical of NeSt optimizations, which require only local accesses and fine-grained time scales (e.g., in the order of single packets sent/received).

1.3 The Need for Global Evaluation

The NeSt architecture, as described in previous sections, is a *full* cross-layer approach where protocols becomes adaptive to both application and underlying network conditions. Such an approach brings the stack as a whole to the best operating trade-off. This has been highlighted by Goldsmith and Wicker,[13] where the authors point at global system requirements, like energy saving and mobility management, as design guidelines for a joint optimization. Our approach opens up different perspectives in the evaluation of network protocols. We claim that in a full cross-layer framework, the performance of a protocol should not only be evaluated by looking at its particular functionalities, but also by studying its contribution in cross-layer activities. Therefore, a stack designed to exploit joint optimizations might outperform a "team" of individually optimized protocols.

To give an example, let us consider ad hoc routing, which is responsible for finding a route toward a destination in order to forward packets. With reference to the classifications reported by Royer and Toh[14] and Chlamtac et al.,[15] the main classes of routing protocols are proactive and reactive. While reactive protocols establish routes only toward destinations that are in use, proactive approaches compute all the possible routes, even if they are not (and eventually will never be) in use. Typically, reactive approaches represent the best option: they minimize flooding, computing and maintaining only indispensable routes (even if they incur an initial delay for any new session to a new destination). But what happens when we consider the cross-layer contribution that a routing protocol might introduce in a NeSt framework?

To answer this question, we provide an example of cross-layer interaction between a routing protocol and middleware platform for building overlay networks, where the former contributes exporting the locally collected knowledge of the network topology. Building an overlay network mainly consists of discovering service peers, and establishing and maintaining routes toward them, as they will constitute the backbone of a distributed service. The overlay network is normally constituted by a subset of the network nodes, and a connection between two peers exists when a route in the underlay (or physical) network can be established. The task of building and maintaining an overlay is carried out at the middleware layer, with a cost that is proportional to the dynamics of the physical network. Overlay platforms for the fixed Internet assume no knowledge of the physical topology, and each peer collects information about the overlay structure in a distributed manner. This is possible because the fixed network offers enough stability, in terms of topology, and bandwidth to exchange messages. Of course, similar conditions do not apply for ad hoc environments, where bandwidth is a precious (and scarce) resource and the topology is dynamic. In ad hoc networks, cross-layering can be exploited, offering the information exported by the network routing to the middleware layer. The key idea is that most of the overlay management can be simplified (and eventually avoided) on the basis of already available topology information.[16] In this case, the more information available, the easier the overlay management; and for this reason, a proactive routing approach becomes more appealing. To support this claim, let us look at what is described by Schollmeier et al.[5] This article describes a cross-layer interaction between a middleware that builds an overlay for peer-to-peer computing and a dynamic source routing (DSR) at the network layer. In this work, the DSR algorithm is forced to maintain valid routes toward the overlay peers, even if these routes are not in use. That is, a reactive routing is forced to behave proactively, with the additional overhead of reactive control packets. The same cross-layer approach with a proactive protocol would probably represent the best joint optimization.

Another example of joint optimization is the extension of routing to support service discovery. A service discovery protocol works at the middleware layer to find out what kind of services are available in the network. As the dynamics of the ad hoc environment determines frequent changes in both available services and hosting devices, service discovery is of fundamental support. The IETF proposes the Service Location Protocol (SLP)[17] to realize service discovery in both Internet and ad hoc networks. Recently, they also underlined the similarity of the messages exchanged in SLP, with those used in a reactive routing such as the Ad hoc On-demand Distance Vector (AODV) protocol.[18] This proposal discusses an extension of AODV to allow service request/reply messages in conjunction with route request/reply. In this proposal, there is a background cross-layer interaction that allows SLP to interface directly with AODV, asking for service-related messages, providing local service data, and receiving service information coming from other nodes. The proposed joint optimization would work even better in the case of a proactive routing protocol such as DSDV or OLSR (see Chlamtac et al.[15] for details). In case of proactive routing, the service information regarding the local services offered on each node could be *piggybacked* on routing control packets and proactively spread around the network, together with local connectivity information. The service discovery communication could be significantly reduced, at the expense of broadcasting routing control packets a few bytes longer. This optimization would result in a proactive service discovery, where a component such as the NeSt supports the exchange of service information from the service discovery protocol, at the middleware, with the routing protocol, and vice versa.

1.4 Discussion and Conclusions

Typically, cross-layering is emphasized as a way to work around the TCP/IP implementation limits, as it introduces direct interaction between protocols to enable smarter adaptation or better performance, at the cost of a "spaghetti-like" code.[9] We believe that cross-layering is possible while keeping the layer separation principle, and that the Internet community is incrementally moving toward cross-layering. The simplest step in this direction is represented by layer triggers, which are predefined signals that notify events between protocols. An example is given by the Explicit Congestion Notification (ECN) mechanism, which notifies the TCP layer about congestion detected by intermediate routers (IP layer). In this way, the source can be informed of congestion quickly and unambiguously, without resorting to inferring mechanisms based on retransmit timer or repeated duplicate ACKs. Another example is given by L2 triggers,[19] added between the data link and IP layers to efficiently detect changes in the wireless links' status. A further step toward cross-layering is presented in by Waldvogel and Rinaldi.[20] This work proposes a way to build topology-aware overlay networks, where logical neighbors are also close in the physical network, according to metrics coming from different levels. These include metrics typically used in routing protocols, such as the physical distance and the bandwidth achieved by a TCP stream.

An open question is to understand if a NeSt-like approach can support cross-layering in the future Internet architecture. Considering the above examples, the NeSt could easily handle the described interactions in the following way:

- The signaling of the ECN bit might correspond to an internal event generated by the IP protocol, previously subscribed by the TCP. Analogously, the MAC layer could generate link-related events to notify the IP layer.
- The metrics used in the construction of the topology-aware overlay network could be associated to NeSt abstractions, seized by the routing and transport protocols, and accessible through the NeSt API.

The NeSt could also be employed to realize optimizations, proposed for the Internet architecture, which are not cross-layered but sidestep the standard layer interfacing. For example, Application Level Framing (ALF)[21] aims at minimizing retransmissions due to data loss, enabling the application level to break the data (to transmit) into suitable aggregates, and the lower level to preserve these frame boundaries when processing the data. Thus, the data segmentation functionality is moved from the transport layer to the application layer. Although the vertical data pipeline inside the protocol stack is not altered, the ALF approach needs a customized implementation, which complicates the maintainability of the overall stack and disagrees with standard interfacing. In the NeSt architecture, the same goal could be achieved keeping the data segmentation functionality at the transport layer and allowing the application layer to instruct, through information sharing, the transport protocol on the way to break data.

Finally, cross-layering provides an effective step toward the mobile Internet. The NeSt architecture is a building block for context-aware computing and networking, a novel paradigm in which a system shows the ability to discover and take advantage of contextual information. Context can be defined as the set of environmental states and settings that determines a system's behavior, and in which system events of interest for the user occurs.[22] While current distributed Internet applications and middleware platforms tend to provide a transparent representation of the underlying execution environmen,[23] context awareness fits well in the area of mobile computing, where applications and software components have to cope with dynamic environments, determining changes in context. In mobile networks, applications and middleware platforms need to be aware of context details, leaking out information such as device location, network bandwidth, or surrounding environment, to and from adjacent layers. The NeSt approach goes toward full awareness of networking context: if applications and network protocols are mutually aware of their operating conditions, then they can adjust their behavior to achieve functional trade-offs and deliver the best end-to-end performance.[13]

Awareness is an important requisite for extending the mobile Internet toward 4th Generation wireless technology. The computing world is experiencing a seamless integration of mobile ad hoc networks with other wireless networks and the fixed Internet infrastructure. The global system presents different characteristics, depending on both physical constraints (e.g., bandwidth, energy, processing power) and usage patterns. The key requirement for the operation of this *heterogeneous* Internet is the protocol's ability to globally adapt to application requirements and underlying network conditions. The need for adaptive networking becomes a challenging issue, the solution of which requires context-awareness.

Acknowledgments

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

References

1. J.P. Macker and M.S. Corson. Mobile ad hoc networking and the IETF. *ACM Mobile Computing and Communications Review,* 2(3):7–9, 1998.
2. M.S. Corson, J.P. Macker, and G.H. Cirincione. Internet-based mobile ad hoc networking. *IEEE Internet Computing,* 3(4):63–70, 1999.
3. C. Szyperski. *Component Software,* pp. 140–141. Addison-Wesley, 1998.
4. K. Chen, S.H. Shah, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. *Wireless Personal Communications,* 21(1):49–76, 2002.
5. R. Schollmeier, I. Gruber, and F. Niethammer. Protocol for peer-to-peer networking in mobile environments. In *Proc. 12th IEEE Int. Conf. Computer Communications and Networks,* Dallas, TX, 2003.
6. W.H. Yuen, H. Lee, and T.D. Andersen. A simple and effective cross layer networking system for mobile ad hoc networks. In *Proc. IEEE PIMRC 2002,* Lisbon, Portugal, 2002.
7. M. Chiang. To Layer or not to layer: balancing transport and physical layers in wireless multihop networks. In *Proc. IEEE INFOCOM 2004,* Hong Kong, China, 2004.
8. U.C. Kozat, I. Koutsopoulus, and L. Tassiulas. A framework for cross-layer design of energy-efficient communication with QoS provisioning in multi-hop wireless netwroks. In *Proc. IEEE INFOCOM 2004,* Hong Kong, China, 2004.
9. V. Kawadia and P.R. Kumar. A Cautionary Perspective on Cross Layer Design. http://black.csl. uiuc.edu/~prkumar/ps_files/cross-layer-design.pdf, 2003.
10. M. Conti, S. Giordano, G. Maselli, and G. Turi. MobileMAN: mobile metropolitan ad hoc networks. In *Proc. 8th IFIP-TC6 Int. Conf. on Personal Wireless Communications,* pp. 169–174, Venice, Italy, 2003.
11. M. Conti, G. Maselli, G. Turi, and S. Giordano. Cross-layering in mobile ad hoc network design. *IEEE Computer, Special Issue on Ad Hoc Networks,* 37(2):48–51, 2004.
12. Sun Microsystems. The JAVA Reflection API. http://java.sun.com/j2se/1.4.2/docs/guide/reflection/index.html, 2002.
13. A.J. Goldsmith and S.B. Wicker. Design challenges for energy-constrained ad hoc wireless networks. *IEEE Wireless Communication,* 9(4):8–27, 2002.
14. E.M. Royer and C.-K. Toh. A review of current routing protocols for ad hoc mobile wireless networks. *IEEE Wireless Communications,* 6(2):46–55, 1999.
15. I. Chlamtac, M. Conti, and J.J.-N. Liu. Mobile ad hoc networking: imperatives and challenges. *Ad Hoc Networks Journal,* 1(1):13–64, 2003.
16. M. Conti, E. Gregori, and G. Turi. Towards scalable P2P computing for mobile ad hoc networks. In *Proc. First Int. Workshop on Mobile Peer-to-Peer Computing (MP2P'04), in conjunction with IEEE PerCom 2004,* Orlando, FL, 2004.
17. E. Guttman, C.E. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. IETF RFC 2608, June 1999.
18. R. Koodli and C.E. Perkins. Service Discovery in On-Demand Ad Hoc Networks. Internet Draft, October 2002.

19. S. Corson. A Triggered Interface. http://www.flarion.com/products/drafts/draft-corson-triggered-00.txt, May 2002.
20. M. Waldvogel and R. Rinaldi. Efficient topology-aware overlay network. *ACM Computer Commun. Rev.,* 33(1):101–106, 2003.
21. D.D. Clark and D.L. Tennenhouse. Architectural considerations for a new generation of protocols. In *Proc. ACM Symp. Communications Architectures and Protocols,* pp. 200–208. ACM Press, 1990.
22. G. Chen and D. Kotz. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.
23. C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing (a survey). In E. Gregori, G. Anastasi, and S. Basagni, Editors, *Neworking 2002 Tutorial Papers,* LNCS 2497, pp. 20–58, 2002.