From Protocol Stack to Protocol Heap – Role-Based Architecture

Robert Braden and Ted Faber
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA
Braden@isi.edu, Faber@isi.edu

Mark Handley
International Computer Science Institute
1947 Center St, Suite 600
Berkeley, CA 94704
mjh@icir.org

Abstract

Questioning whether layering is still an adequate foundation for networking architectures, this paper investigates non-layered approaches to the design and implementation of network protocols. The goals are greater flexibility and control with fewer feature interation problems. The paper further proposes a specific non-layered paradigm called *role-based architecture*.

1 Introduction

Traditional packet-based network architecture assumes that communication functions are organized into nested levels of abstraction called *protocol layers* [6], and that the metadata that controls packet delivery is organized into *protocol headers*, one for each protocol layer [3].

Protocol layering has served well as an organizing principle, but it worked better for the more strict end-to-end model of the original Internet architecture than it does today. We see constant pressure for "layer violations" (which are often assumption violations)[1], and unexpected feature interactions emerge. In large part this is due to the rapid proliferation of "middle boxes" (firewalls, NAT boxes, proxies, explicit and implicit caches, etc.), but other multi-way interactions such as QoS, multicast, overlay routing, and tunneling also complicate the picture.

The complex interactions that result are difficult to describe using strict layering, and the implicit "last on, first off" assumption of layering often makes a new service fit poorly into the existing layer structure. The result is an inability to reason about feature interaction in the network. A reluctance to change working implementations and long-standing inter-layer interfaces often lead designers to insert new functionality between existing layers rather than modify existing layers.[2]

It is very hard to evolve network protocols, especially at the network level. Partly this is for performance reasons[3], but partly it is because layering tends to lead to a relatively large granularity of protocol functionality.

We are forced to conclude that layering is not a sufficiently flexible abstraction for network software modularity. This inflexibility might be considered desirable, as it forces compliance with existing standards, but in practice it often results in a tussle [1] between the needs of protocol designers, ending in short-sighted solutions that violate some of the assumptions made by other protocols. Suggesting that layering is inadequate calls upon us to propose an alternative organizational principle for protocol functionality. Our task is to allow more flexible relationships among communication abstractions, with the aim of providing greater clarity, generality, and extensibility than the traditional approach allows.

This paper proposes a new non-stack network architecture that we call *role-based architecture* or RBA. Instead of using protocol layers, RBA organizes communication using functional units that are called *roles*. Roles are not generally organized hierarchically, so they may be more richly interconnected than are traditional protocol layers. The inputs and outputs of a role are application data payloads and controlling metadata that is addressed to specific roles.

With a non-layered approach, layer violations should be replaced by explicit and architected role interactions. Of course, "role violations" will still be possible, but the generality of the mechanism should typically make them unnecessary, and suitable access controls over metadata can make them difficult.

The intent is that roles will be building blocks that are well-defined and perhaps well-known. To enable interoperability, a real network using RBA would need a rel-

[1]Current protocol engineering efforts are producing some remarkable examples of layer muddling.

[2]For example, MultiProtocol Label Switching was inserted at "layer 2.5", IPsec at "layer 3.5", and Transport-Layer Security at "layer 4.5".

[3]For example, IPv4 options are rarely used.

atively few (tens to hundreds) of *well-known* roles defined and standardized.[4] However, the number of special-purpose, experimental, or locally defined roles is likely to be much greater.

There has been much prior work on the modularization of protocol processing, generally with the primary objective of easing the development of new protocol stacks; see for example [2], [4], [5]. However, we believe that RBA is the first general proposal for achieving modularity through a non-layered protocol paradigm in both protocol headers and processing modules.

1.1 Role-based Architecture

A general idea behind role-based architecture is that of explicit signaling of functionality. The lack of architected signaling is one of the main reasons why middleboxes do not fit into the current layered architecture. For example, there is no defined way to signal to an end-system that a packet really did traverse the firewall protecting the site, or to signal that an end-system does not want its request redirected to a web cache.

Even in simple examples, the limits of layering are clear. How can a TCP port 80 SYN packet signal that it does not want to be redirected to an application-layer web cache? Network congestion is signaled by a router (network layer), but rate-control occurs at the flow level (transport layer), so signaling between the flow and the network is difficult. In both cases, layering is part of the problem, and explicit signaling is a possible solution. RBA is designed to perform such signaling in a robust and extensible manner.

Role-based architecture also allows *all* the components comprising a network to be explicitly identified, addressed, and communicated with. RBA would allow re-modularization of current "large" protocols such as IP, TCP, and HTTP into somewhat smaller units that are addressed to specific tasks. Examples of such tasks might be "packet forwarding", "fragmentation", "flow rate control", "byte-stream packetization", "request web page", or "suppress caching". Each of these comprise separable functionality and could be performed by a specific role in a role-based architecture.

Moving to a non-layered architecture requires a distinct shift in thinking, so the next section examines the implications of removing layering constraints. Section 2 examines what role-based architecture might actually look like in principle, while Section 3 describes a practical adaptation of RBA to real-world networking.

[4]Note that role-based architecture will not remove the need for standardization.

1.2 Non-Layered Architecture Implications

The concept of a non-layered protocol architecture has immediate implications. Layering provides modularity, a structure and ordering for the processing of metadata, and encapsulation. Modularity, with its opportunity for information hiding and independence, is an indispensable tool for system design. Any alternative proposal must provide modularity, but also adequately address the other aspects of layering:

Metadata Structure: The structure of metadata carried in a packet header no longer forms a "stack", it forms a "heap" of protocol headers. That is, the packet header is replaced by a container that can hold variable-sized blocks of metadata, and these blocks may be inserted, accessed, modified, and removed in any order by the modular protocol units.

Processing Rules: A non-layered architecture requires new rules to control processing order and control access to metadata to replace the corresponding rules of a layered architecture.

Consider ordering first. In the simplest case when roles are completely independent, a non-layered architecture specifies no processing order; protocol modules may operate in any order, or even simultaneously, on various subsets of the metadata. More commonly, however, an appropriate partial ordering is required among specific roles.

Other rules must specify how access to metadata is to be controlled. By controlling the association between program and (meta)data, the architecture can explicitly control interactions among the different protocol modules, enhancing security as well as extensibility.

Encapsulation: In a layered architecture, each layer is encapsulated in the layer below. In non-layered architectures, there needs to be a different organizational principle for the data and metadata in a packet. Encapsulation does not disappear, but it is reserved for cases where the functionality is that of a container. For example, a protocol providing a reliable byte stream over a packet network encapsulates that byte-stream, but it does not encapsulate all the metadata associated with the byte stream if that metadata affects packet processing.

2 The Idealized RBA

In light of the discussion above, we propose *role-based architecture* (RBA) as a particular non-layered architecture in which the modular protocol unit is called a *role*. A role is a functional description of a communication building block that performs some specific function relevant to forwarding and/or processing packets. Roles are abstract

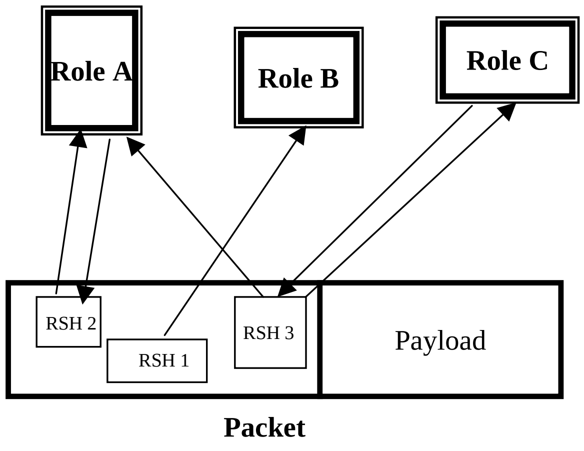

Figure 1: Roles and Role-Specific Headers

entities, and it should be possible to reason about them somewhat formally.

Roles are instantiated in nodes by code called *actors*. For many purposes, the distinction between a role and its actors will not matter, so we can often simply speak of roles as having particular properties. A role may logically span multiple nodes, so it may be distributed in an abstract sense, while each actor executes in a particular node.

The metadata in a packet, called *role data*, is divided into chunks called *role-specific headers* (RSHs). This is illustrated in Figure 1, which shows a packet containing three RSHs in its heap and three roles that read and write these RSHs, as shown by the arrows.

An end system does not necessarily know that a packet it sends will encounter a node that cares about a particular role. For example, there may be no web-cache-redirector role on a particular path, but if there is, including a signaling RSH addressed to this role will ensure that the cache receives the metadata. Any node along the path can add an RSH to a passing packet. For example, suppose that a firewall has some lightweight way of signing that a packet has been examined and found to conform to the site's security policy; it can include this signature in an RSH attached to the data packet and addressed to the host firewall role.

2.1 RBA Objectives

RBA is designed to achieve the following objectives.

Extensibility: RBA is inherently extensible, both mechanically and conceptually. Mechanically, RBA uses a *(type, length, value)* mechanism to encode all role data. Conceptually, the general modularity supported by RBA should enhance extensibility.

Portability: The role abstraction is designed to be independent of the particular choice of nodes in which a role will be performed. This *portability* of roles enables flexible network engineering, since functions can be grouped into boxes as most appropriate. Role portability may, but won't generally, imply role *mobility*, in which a role could migrate dynamically to a different node.

Explicit Architectural Basis for Middle Boxes: RBA is intended to allow endpoints to communicate explicitly with middle boxes and middle boxes to communicate with each other.

Controlled Access to Metadata: RBA includes a general scheme for controlling access to metadata, allowing control over which nodes can read and modify specific subsets of the metadata as well as the application data. This access control implies control of the services that can be requested and control of the operations that can be performed on a packet.

Auditability: An endpoint may wish to *audit* received data packets to ascertain that they were subjected to requested processing, for example through a validator or an encrypted firewall. Auditability is not generally feasible with the current layered architecture because the relevant metadata will have been processed and removed before the data reaches the receiver. RBA role data can be used to indicate that the requested service was provided.

2.2 General Properties of Roles

A role has *well-defined inputs and outputs*, in the form of RSHs whose syntax and semantics can be tightly specified, and/or in the form of APIs to other software components in the local node.

A role is identified by a *unique name* called a **RoleID**. A RoleID reflects the functionality provided. A full RoleID may have a multicomponent structure like a file name; the hierarchy would reflect the derivation of a specific role from a more generic one (Section 2.4). For efficient transport and matching, a corresponding short-form fixed-length integer RoleID will normally be used.

The RoleID only addresses meta-data to the provider of a type of functionality; it does not indicate which node will perform that functionality. RSHs in packets can also be addressed to the specific actor that instantiates a role[5]. RBA requires that node interfaces have unique addresses called NodeIDs, which would correspond to "network addresses" in the traditional layered architecture. Symbolically, we denote a **role address** in the form RoleID@NodeID, or RoleID@$*$ if the NodeID is to be left unspecified.

[5]We speak of the *address* of a role, meaning the address of one of its actors.

To perform their tasks, role actors may contain internal **role state**. Establishing, modifying, and deleting role state generally requires signaling, which is done through the exchange of RSHs.

Often roles are defined in complementary pairs; these are called **reflective roles**. Simple examples of such reflective roles might be: (*Fragment, Reassemble*), (*Compress, Expand*) or (*Encrypt, Decrypt*).

Other special role categories may emerge as the role-based model is developed further. These sort of categories are useful in bounding the flexibility that RBA provides, so that we can reason about the interaction between roles.

There are **families** of related roles that differ in detail but perform the same generic function. This generic function may be abstractly represented by a *generic role*. Specific roles may be derived from the generic role through one or more stages of specification (see Section 2.4). For example, corresponding to the generic role *ReliableDataDelivery* there might be specific roles for reliable ordered byte streams and for reliable datagrams.

For designing and analyzing roles, it will be useful to define the **composition** of a role from sub-roles. This enables a building block approach to creating relatively complex roles, and it allows the successive decomposition of abstract roles into sub-roles.

2.3 Role Data

Under the idealized RBA model, all data in a packet, including the payload, is role data that is divided into RSHs. The set of RSHs in a particular header may vary widely depending on the services requested by the client and can vary dynamically as a packet transits the network. The relationship between roles and RSHs is generally many-to-many – a particular RSH may be addressed to multiple roles, and a single role may receive and send multiple RSHs.

Just as roles modularize the communication algorithms and state in nodes, so RSHs modularize the metadata carried in packets. RSHs divide the metadata along role boundaries, so an RSH forms a natural unit for ordering and access control on the metadata; it can be encrypted or authenticated as a unit.

The granularity of RSHs is a significant design parameter, since role data cannot be shared among roles at a smaller granularity than complete RSHs. At the finest granularity, each header field could be a distinct RSH; this would avoid any replication of data elements required by multiple roles. However, overhead in both packet space and processing requirements increases with the number of RSHs in a packet, so such fine-granularity RSHs are not generally feasible. As in all modularity issues, the optimal division into RSHs will be an engineering trade-off.

A role might modify its activity depending upon the particular set of RSHs in the packet. Furthermore, the presence of a particular RSH may constitute the request for a service from subsequent nodes. Thus, the RSHs may be considered to provide a form of signaling that may piggy-back on any data packet.

The format of an RSH is role-specific. It might have the fixed-field format of a conventional protocol header or it might have a (type, length, value) format, for example. An RSH contains a list of role addresses to which this RSH is directed and a body containing role data items. We denote this symbolically as:

RSH(<*RoleAddressList*> ; <*RoleBody*>)

For example, RSH(Expand@*N3*, Decrypt@$*$; ...) represents an RSH addressed to the role named *Expand* at node *N3* and to the role named *Decrypt* at any node.

RBA provides a model for packet header processing, not a mechanism for routing packets. Rather, RBA incorporates whatever forwarding mechanism is in use through a generic *Forward* role, which may depend upon global or local state in each node. A mechanism to create that state, e.g., a distributed SPF routing calculation, is simply another application from the viewpoint of RBA. Once the forwarding rules determine the actual route taken by a packet, the RBA sequence and scheduling rules come into play to determine the sequence of operations.

2.4 Technical Issues

Further definition of RBA requires specific solutions to a number of technical problems.

- Role Matching

  Rules must be specified for matching role addresses in RSHs with actors, taking into account the access control rules. An actor may have access to an RSH either because the RSH was explicitly addressed to that actor or because the actor was promiscuously "listening" for particular RSHs (again subject to access control rules.) An actor may read or write (add, modify or delete) an RSH (see the arrows in Figure 1).

- Actor Execution Scheduling

  Once the matching actors are selected, the node must determine in what order they should be executed. This scheduling problem must consider ordering requirements imposed by roles; these requirements are called *sequencing rules*. For example, such rules might prevent undesirable sequences like *Encrypt, Compress* (compression is not useful after encryption) or *Expand, Compress* (wrong order), or *Compress, Encrypt, Expand, Decrypt* (reflective pairs are improperly nested). These rules must consider dynamic precedence information carried in packets as well as static precedence associated with the actors in the nodes.

- RSH Access Control

  By controlling access to RSHs, RBA allows nodes, including end systems, to control what network services can be applied to specific packets. RBA provides two levels of access control, de jure and absolute. De jure access control is provided by bits in each RSH that grant specific roles read and/or write permission for the RSH. Write access would provide the ability to modify or delete the RSH from the packet.

  De jure access control is sufficient as long as nodes follow the RBA rules. Otherwise, nodes can absolutely control access to RSHs by encrypting these RSHs; of course, this greater certainty has greater cost.

- Role Definition

  To fully define a specific role, it is necessary to define its internal state, its algorithms, and the RSHs to be sent and received. In addition, some roles have non-network interfaces that must be defined.

  It remains to be seen whether RBA is amenable to the use of formal protocol specification techniques. One possible direction is to exploit the analogy between object-oriented programming and the derivation of specific roles from generic roles. If roles correspond to classes, then actors are instantiations of these classes, and RBA communication can be modeled by actors communicating via message passing.

- Role Composition

  Two roles R_a and R_b that communicate directly with each other using RSHs (which may originate and terminate in the two roles, or may be passing through one or both) should be composable into a larger role R_c. This binds R_a and R_b into the same node, and allows inter-role communication to be replaced by internal communication, e.g., shared data.

  Conversely, a complex role may be decomposed into component roles, replacing shared data by explicit role data communication using RSHs.

2.5 RBA Examples

2.5.1 Simple Datagram Delivery

As a very simple RBA example, the RBA equivalent to a simple IP datagram might be a packet containing the four RSHs:

{ RSH(LinkLayer@*NextHopAddr*;),
RSH(HbHForward@$*$; *destNodeID*),
RSH(HbHSource@$*$; *sourceNodeID*),
RSH(DestApp@*destNodeID*; *AppID*, *payload*) }

Here the *LinkLayer* role presents the link layer protocol, and its RSH is addressed to the next hop node. The *DestApp* role is the generic destination application role that delivers the payload in the role data to the application-level protocol specified by $AppID$. The *HbHForward* role represents a hop-by-hop forwarding function, invoked in every node along the path, with the destination address as its role data. It is one specific rule derived from the generic *Forward* role, which is the fundamental action of a router and of most middle boxes. It uses role data to determine one or more outgoing interfaces or next hops. *HBHSource* indicates the node ID that can be used to return a response by hop-by-hop forwarding.

2.5.2 Network Address Translators

Regardless of their architectural merit, network address translators (NATs) make a good RBA example since they do not fit well into a layered architecture. A NAT is essentially a packet relay that separates two different addressing realms. Complication is added by application-level protocols that are unaware of the NAT's existence but need to communicate addresses or ports end-to-end.

There are essentially two types of NAT. *Pure NATs* perform a dynamic but one-to-one mapping between a small pool of external addresses and a larger number of internal addresses. *NAPTs* perform a one-to-many mapping between a single external address and many internal addresses, by overloading the TCP or UDP port fields.

Pure NATs are simple to accommodate using RBA. The NAT simply inserts a RSH for the role called *nat-receiver* giving the original address, and all software on any downstream nodes can see that the translation has occurred, and act accordingly.

Network address and port translators (NAPTs) are a little more complex. We wish to avoid overloading any fields, and we want NAPT to function for any application-level protocol. A NAPT can behave exactly like a pure-NAT in its insertion of the *nat-receiver* RSH, but it also needs some way to demux incoming packets to the correct internal address. This is done using a general-purpose *cookie* role. On outgoing packets, the NAPT inserts an RSH for the *cookie* role, giving a cookie that is unique to the internal address. All systems should be aware of the *cookie* role, and should echo the cookie in a *cookie_echo* RSH included in any response. This cookie mechanism is not NAT-specific, and it can form a useful building block for many new mechanisms.

3 Practical Realization of RBA

The idealized RBA described in the previous sections is useful as a model for network protocols. We furthermore believe that the RBA model can be adapted for practical implementation in a large-scale Internet world.

The idealized RBA replaces all layers of the traditional layered model, including the link layer. However, link layer protocols are given, at least in the short term; they are designed by industry groups to match particular technological constraints. A practical RBA subset would therefore retain the link layer as a distinct layer "below" RBA.

A critical design decision when instantiating a role-based architecture is designing the packet format. There is a clear tradeoff between making the RSH header format quite powerful and general, versus wasting too many bytes on role addressing relative to the size of the information carried in each RSH. In the earliest sketch of RBA, we imagined a small number of well-defined roles and a field as small as 6 bits to address each RSH. Later we realized that RBA would be much more powerful if we could address RSHs more generally, and so the addressing information grew to include NodeIDs and larger RoleIDs. This has a direct effect - it is probably not cost-effective to split very simple low-level functionality into separate roles. The advantage is that at higher levels we have a more powerful mechanism for expressing complex interactions.

Furthermore, forwarding performance is an important real-world issue. In a very large-scale network like the Internet, there are strong reasons to keep the basic packet forwarding machinery as simple and efficient as possible. A practical RBA would therefore retain the IP layer of the Internet, with its high-speed forwarding machinery and efficient packet header. RBA would then be applied above the Internet layer, i.e., it would replace only the transport and application layers. As a result, RBA would be implemented only in end systems and middle boxes, where performance requirements are much less severe than within the core network.

These assumptions could be incorporated into RBA by declaring that the generic *Forwarding* role and the reflective pair (*Fragment*, *Reassemble*) are "built in". These simplifications should not interfere with a major rationale for RBA, providing a consistent architectural basis for middle boxes, but they should make a RBA approach a realistic proposition for real networks.

4 Conclusions

This document has proposed role-based architecture to simplify the design and deployment of communication protocols in today's world, where the complex interactions among networking elements often do not follow a strict layering model. RBA provides a uniform way to structure protocols and protocol processing without the confines of strict layering.

The generality of RBA does not come without cost, of course. The layered-network model has been a very powerful tool for conceptualizing and designing protocols. We need to satisfy ourselves that roles will provide a tool that is at least as good as, if not better than, layers for developing protocols. Furthermore, RBA requires a more general data structuring in packet headers, which has a cost in implementation, packet size, and execution performance. We must show that these costs are containable.

5 Acknowledgments

We are grateful to the other members of the NewArch project, who have given much encouragement on developing the RBA concepts. We especially want to thank Dave Clark, John Wroclawski, Karen Sollins, Noel Chiappa, and Aaron Falk.

Development of RBA was funded in part by DARPA contract F30602-00-1-0540.

References

[1] D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in Cyperspace – Defining Tomorrow's Internet. *Proceedings of SIGCOMM 2002*, 2002. Position Paper.

[2] N. Hutchinson and L. Peterson. The x-Kernel: An Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, 1991.

[3] ISO. Information Processing Systems - Open Systems Interconnection - Basic Reference Model. ISO 7498, 1984.

[4] E. Kohler, M. Kaashoek and D. Montgomery". A Readable TCP in the Prolac Protocol Language. Proc SIGCOMM 99, 1999.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[6] E. Meyer. ARPA Network Protocol Notes. RFC 46, Network Working Group, April 1970.