From Protocol Stack to Protocol Heap – Role-Based Architecture

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

Ted Faber
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA
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 interaction problems. The paper further proposes a specific non-layered paradigm called *role-based architecture*.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Architecture*

Keywords

Non-layered architecture, role-based, modularity, metadata, signaling, processing rules

1. INTRODUCTION

Traditional packet-based network architecture assumes that communication functions are organized into nested levels of abstraction called *protocol layers* [7], and that the metadata that controls packet delivery is organized into *protocol headers*, one for each protocol layer [4].

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), and unexpected feature interactions emerge. In 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 are also guilty.

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.[1] In addition, it is very hard to evolve network protocols, especially at the network level. Partly this is for performance reasons[2], but partly it is because layering tends to lead to a relatively coarse granularity of protocol functionality.

The limits of layering are clear from even simple examples. There is currently no way for a TCP SYN packet to port 80 (HTTP) to signal that it does not want to be redirected to an application-layer web cache. Also, 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 an important part of the problem.

These considerations suggest that layering may not be 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 short-sighted solutions that may violate assumptions made by other protocols.

If protocol layering is inadequate as an abstraction, we need 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 non-stack approach to network architecture that we call *role-based architecture* or RBA.

1.1 Role-based Architecture

Instead of using protocol layers, an RBA organizes communication using functional units called *roles*. Since roles are not generally organized hierarchically, 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.

HotNets I, Princeton, NJ, USA, October 2002. This research was sponsored in part by DARPA contract F30602-00-1-0540 and by AT&T Research.

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

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

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 relatively few (tens to hundreds) of *well-known* roles defined and standardized.[3] However, the number of special-purpose, experimental, or locally defined roles is likely to be much greater.

An important attribute of role-based architecture is that it can provide 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. RBA is designed to perform such signaling in a robust and extensible manner.

Role-based architecture allows *all* the components comprising a network to be explicitly identified, addressed, and communicated with. RBA could allow 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 that could be performed by a specific role in a role-based architecture.

The purpose of this paper is to examine the general properties of any role-based architecture rather than to describe in detail any particular instance of an RBA. It is intended to suggest a fruitful research direction, which holds some promise for improving the clarity and generality of network protocol design. Since moving to a non-layered architecture requires a distinct shift in thinking, the next section examines the implications of removing layering constraints. Section 2 then outlines what role-based architecture might actually look like in principle and gives some simple examples of the use of RBA. Section 3 describes a range of approaches to applying the RBA ideas in the real world of networking and discusses implementation issues.

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: Without layering, the structure of metadata carried in a packet header no longer logically forms a "stack", it forms a logical "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 to control access to metadata, to replace the rules implicit in layering.

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, a different organizational principle is needed for the data and metadata in a packet. Encapsulation does not disappear, but its role is much reduced. Encapsulation is no longer the main enforcer of processing order. It is reserved for cases where the functionality is that of a container or adaptor, such as the encapsulation of a reliable byte stream within a flow of packets. Even in such cases, metadata about the data being encapsulated need not be itself encapsulated, as it would be in a layered architecture.

1.3 Prior Work

There have been very many papers about different ways to generalize protocol design and processing, to ease the limitations of strictly-layered stacks of complex protocols and of monolithic implementations.[4]

Early work [3, 9] emphasized the modular construction of protocol processing stacks for flexibility and extensibility. Many later papers have discussed the decomposition of complex protocols into *micro-protocols*, either for reusability, customization, and ease of programming [1, 5, 8, 6], or to improve protocol processing performance using parallelism [2, 10]. Here micro-protocols roughly correspond to our roles (as abstractions) or to our actors (as a protocol processing modules); see [1] for example.

Some of these papers suggest generalizations from strict layering, but their primary emphasis is on protocol implementations rather than on the protocols themselves. This paper focuses on the protocols themselves (the distributed algorithms and the "bits on the wire") and on what it means to completely give up layering. Our work has also been aimed at the new architectural issues raised by middleboxes. RBA is a general proposal for achieving modularity through a non-layered protocol paradigm, non-layered in both protocol headers and processing modules.

However, in order to realize an RBA it will be necessary to specify ways to define and structure roles and to design machinery for instantiating and executing roles. Here the prior research will become highly relevant and may provide important answers. We therefore believe that this paper is complementary to much of the earlier work on generalized protocol 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

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

[4]This paper can list only some of the papers that are particularly relevant to RBA.

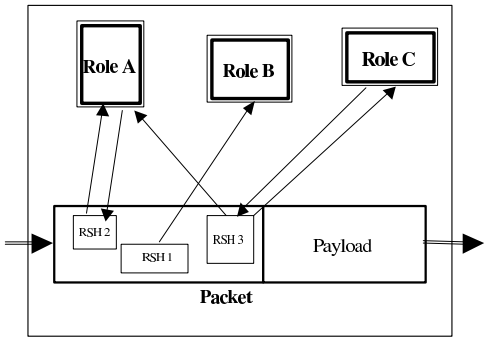

Figure 1: Role-Specific Header Processing within a Node

that performs some specific function relevant to forwarding or processing packets. Roles are abstract 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 node containing three roles that read and perhaps write specific RSHs in a packet header.

Roles are the principal form of addressing in a role-based architecture, in that RSHs are *addressed to roles*. Whether the RSH also specifies which actor should instantiate the role is optional. Typically a role will name some modular functionality or mechanism. As network functionality is often distributed with the actors forming a closely coupled system, sometimes a role will name a distributed mechanism, and sometimes it will name the components of such a mechanism. Only deployment experience can determine what the exact functional breakdown into roles should be.

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 flexible modularity of RBA should enhance extensibility of networked systems.

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*, where a role can migrate between nodes.

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. It may also have specified 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.

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.

Some roles operate in a pair of nodes to enforce some condition in the intervening data path; simple examples are the pairs: (*Fragment, Reassemble*), (*Compress, Expand*) or (*Encrypt, Decrypt*). We call these **reflective roles**. It is possible to consider a reflective role to be either a pair of distinct sub-roles or to be a single distributed role.

Other special role categories may emerge as the role-based model is developed further. These sort of categories are

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

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.

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 provide a form of signaling that may piggy-back on any 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*> ; <*RSHBody*>)

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 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 of TCP or UDP port fields.

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

The RBA equivalent of a NAPT is a little more complex. A NAPT can behave exactly like a pure-NAT in its insertion of the *nat-receiver* RSH, but it also needs some way to demultiplex incoming packets to the correct internal address. One way to do this might use a general-purpose *echo* role. On outgoing packets, the NAPT inserts an RSH addressed to the *echo* role, giving a token that is unique to the internal address. All RBA systems should be aware of the *echo* role. If any RBA node generates a response to a packet containing a RSH addressed to the *echo* role, it should echo the token by including an RSH in the response packet addressed to the *echo-sender* role. This token mechanism is not NAT-specific, and it can form a useful building block for many new mechanisms.

This NAT example raises an interesting issue with regard to role naming. If, in the traditional manner, we named the RSHs rather than the roles we would have called the RSHs *token* and *token-echo*. The RBA philosophy is that it is the role played by the recipient that is named, and not the RSH. The distinction is an important and subtle one, as it significantly affects how future extensions may be deployed.

3. REALIZATION OF RBA

The role-based architecture approach described in earlier sections may be applied to network protocol design in a variety of ways. Further research will be required to determine which of these directions will be most fruitful.

In the extreme, one could build an architecture that is entirely role-based, i.e., all protocol functions from the present link layer to the present application layer as well as all middlebox functions would be replaced by roles or sets of roles. This would yield a completely layer-free, remodularized architecture.

There are two possible directions for less extreme ways to use the RBA approach. First, one can apply RBA only above a particular layer of the stack, retaining layering below that point. These partial stack implementations of RBA trade off generality and flexibility for efficiency and reality. For example, we may consider link-layer protocols to be immutable, since they are designed by industry groups to match particular technological constraints. A practical RBA subset might therefore retain the link layer as a distinct layer "below" RBA. Furthermore, retaining the IP layer as the highly-optimized common end-to-end packet transport service could significantly help to solve the efficiency issues with RBA; RBA processing would be needed only in end systems and middleboxes. A less strong argument could be made to retain the transport layer and apply RBA only as an application-layer architecture (note that this could still help immensely with the middlebox problem.)

The other possible direction is to use RBA to provide an unlayered network control (signaling) mechanism for essentially the current general modularity. From this viewpoint the network functionality would be divided into major *protocol entities* that might (or might not) assume particular roles. This viewpoint emphasizes the addressability of a role; RSHs would generally be created by protocol entities but addressed to, and received by, roles assumed by other entities.

Finally, the idealized RBA may be useful simply as an abstraction for reasoning about protocols, their functions and interactions.

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 network like the Internet, there

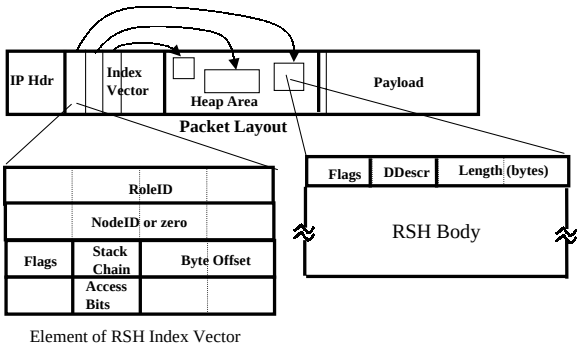

Figure 2: Possible RBA Packet Layout

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.

3.1 Packet Structure

Figure 2 suggests a possible packet format for a practical RBA. This figure shows an encapsulating IP header, assuming that RBA is going to be applied only above the IP layer; this assumption is not necessary. If the network layer were brought under RBA, the IP header would be replaced by the actual *Forward.HbH* RSH, placed at a fixed location at the beginning of the packet to allow very efficient hardware-based forwarding through the network core.

The role address lists (see Section 2.3) of all RSHs in the packet are gathered together into a *index vector* of fixed-length entries. This should allow efficient processing of packets under RBA. Each entry includes a pointer (byte offset) to the corresponding RSH body in the heap area.

The RoleID is a globally-unique 32-bit short name for a role to which the RSH specified by this index element is addressed. As suggested earlier, it can be generated as a hash of the long name. This is shown as a 32-bit IPv4 address of the node to which this index element is addressed (RoleID@NodeID), or zero to indicate a wildcard (RoleID@∗)

As a packet traverses the network, RSHs may be added and deleted from its header. There are many engineering strategies that can help to keep this reasonably simple and efficient. There is no specified maximum size for the RSH space – the index vector or the heap area – but generally a source will have a good guess on how much space to reserve between IP header and payload for RSHs. The boundary between index vector and heap can be flexible, and these two segments can grow towards each other. A series of deletions and additions of RSHs could force garbage collection of the heap or movement of the payload to expand the heap size in an intermediate node. This is relatively complex and expensive, but it should seldom be necessary. In case a node is unable to add a new RSH to a packet, it can send a "RSH Overflow" RBA control message back to the sender node, requesting a larger RSH space in succeeding packets.

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 thank Dave Clark, John Wroclawski, Karen Sollins, Noel Chiappa, and Aaron Falk.

6. REFERENCES

[1] N. Bhatti and R. Schlichting. A System for Constructing Configurable High-Level Protocols. *Proc. ACM SIGCOMM '95*, 138-150, 1995.

[2] Z. Haas. A Protocol Structure for High-Speed Communication over Broadband ISDN. *IEEE Network*, 5(1):66-70, January 1991.

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

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

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

[6] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek. The Click modular router. *ACM Trans on Computer Sys.*, 18(3):263-297, 2000.

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

[8] S. O'Malley and L. Peterson. A Dynamic Network Architecture. *ACM Trans. Comp. Sys.*, 10(2):11–143, May 1992.

[9] C. Tschudin. Flexible Protocol Stacks *Proc. ACM SIGCOMM '91*, 197-204, 1991.

[10] M. Zitterbart, B. Stiller, A. Tantawy. A Model for Flexible High-Performance Communication Subsystems *IEEE JSAC*, 11(4):507-518, May 1993.