From Protocol Stack to Protocol Heap

-- *Role-Based Architecture* (RBA)

Bob Braden, Ted Faber

USC Information Sciences Institute

Mark Handley

ICSI Center for Internet Research

ACM HotNets I
Princeton University
October 28, 2002

Outline

- Motivation
- Overview of Role-Based Architecture (RBA)
- Using RBA
- Related Work
- Conclusions

Motivation

- The IETF has become an *architectural pretzel factory*.
  - Layer violations
  - Sub-layer proliferation
    - E.g., MPLS at 2.5, IPsec at 3.5, and TLS at 4.5.
  - Feature interactions
    - Cross-product complexity
  - Erosion of E2E model -- middleboxes
    - Firewalls, NATs, proxies, caches, ...
- A paradise for lovers of complexity
- Can we somehow reduce the complexity and increase the architectural flexibility?

Motivation ...

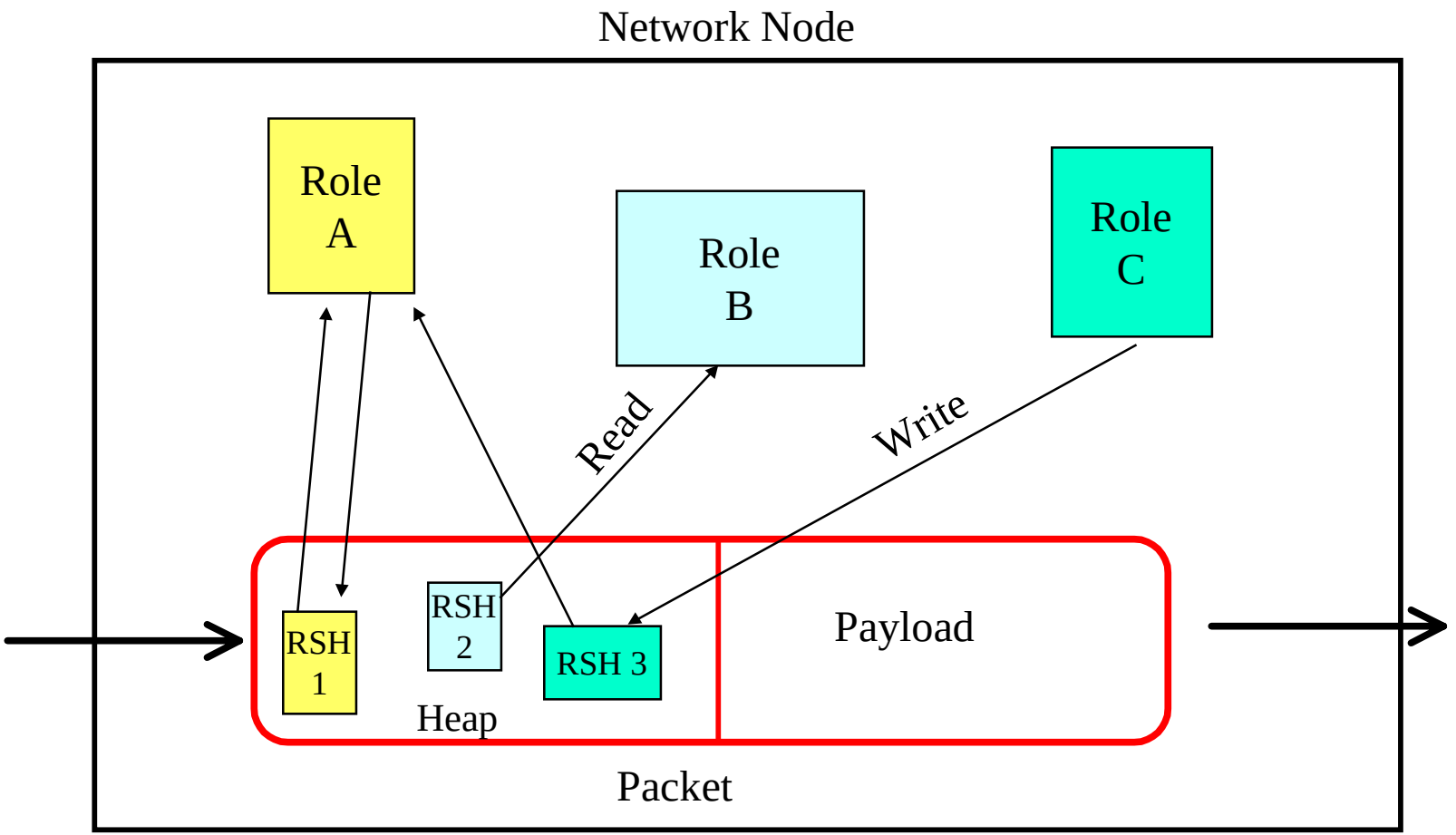
- Suggestion 1: Replace the traditional protocol layering paradigm with a more general model.
  - Many of these problems seem to be related to traditional layering.
- Suggestion 2: Provide a protocol mechanism to attach additional metadata to data packets -- “in-band signaling” -- for middleboxes.
  - Attach color-coded “stickies” to packets in the network.
- These suggestions led to the concepts of Role-Based Architecture (RBA)
- Giving up layering has profound consequences for how we think about protocols.

What Does Non-Layered *Mean*?

- Traditional layered architecture
  - Modularity
    - Functional unit for each protocol layer.
  - Packet header format:
    - Sub-header for each layer, forming a logical stack.
  - Header processing rules:
    - Order: Headers processed in order by layer (LOFO)
    - Access: A functional module can read/write only its own sub-header

- Non-Layered architecture
  - Modularity:
    - Role: Functional spec of a communication building block.
  - Packet header format:
    - An arbitrary collection of sub-headers: “role data”.
    - These are Role-Specific Headers (RSHs).
    - RSHs are addressed to roles.
    - Header data structure is now a logical *heap* of RSHs.
  - Processing rules: need new rules for order, access.

RSH Processing in a Node



Objectives of RBA (1)

- Clarity:
  - Replace “layer violations” with architected role interactions
- Flexibility
  - Roles have more flexible relationships than layers
- Extensibility
  - Roles are modular and hopefully orthogonal. No layer restrictions.
- Inband Signaling
  - RSHs can act as “stickies”, e.g., to control middle boxes.
- Auditability
  - Can leave RSHs after they have been “consumed”, to signal to downstream nodes that a function has been performed.

Objectives of RBA (2)

- Portability
  - Allow roles to be sited arbitrarily on nodes.
    - *For extra credit*: *mobile* roles that migrate among nodes
- Re-Modularization
  - Current monolithic protocol layers are large and complex; can re-modularize into smaller units.
    - This is not a new idea
    - It is unclear how far one should go towards micro-roles
    - But RBA gives us freedom of choice on functional granularity
- Security
  - Hide particular role data *(Don't muck with my meta-data!)*
  - RSH might be unit for encryption of role data

Brief Overview of RBA

- Outline
  - Role Data
  - Role Definition
  - Naming and Addressing
  - Processing Rules
  - Trivial Example
  - Implementation: Packet Layout

More About Role Data

- RSHs can be added, modified, or deleted as a packet is forwarded.
- RSHs subdivide the header information (meta-data) along role boundaries.
  - Granularity of RSHs is an important design parameter
  - Trade off processing overhead against reusability
- RSHs generally carry metadata, but some may not, only modifying processing by their presence.

Defining Roles

- Roles communicate with each other only via RSHs
  - (for role mobility)
- Roles may have local APIs to node software.
- A fully-specified role will be specified by:
  - Its internal state, its algorithms, its APIs, and the RSHs it will send and receive.
- Generic roles
  - Want to be able to derive a full role specification from a generic functional definition by stepwise refinement.
  - Aid reasoning about protocols and for developing new roles.

More about Roles

- A role instantiation called an *actor*.
  - (MJH doesn't like the Hollywoodiness of this term)
- Roles are often coupled in conjugate pairs
  - E.g., {Encrypt, Decrypt} {Compress, Expand} {Fragment, Reassemble}
    - (Undecided: Is a conjugate pair one distributed role with two actors, or two interrelated roles?)

Naming and Addressing in RBA

- Role type is identified by unique name: RoleID
  - "Color-coded"
- RSHs are addressed to role(s)
  - Assume an address space for nodes {NodeID} [~IP addr]
  - *<RoleAddr> ::= <RoleID> @ <NodeID> | <RoleID> @ **

Wildcard NodeID: RSH will be processed by any instance of the RoleID that it encounters along the path.

- Symbolically, an RSH is:
  RSH(<RoleAddr>, ... ; <RSHbody>)

(More accurately: *RSH(<RoleAddr>:<access bits>, ...)*)

Processing Rules

- A Role R on node X may *access* an RSH if:

  (1) The RSH is explicitly addressed to R

  RoleAddr = R@X or R@*,

  (2) or R is *promiscously* listening for RoleID R' that *is* addressed by RSH

  Either may be restricted by access control bits.

- Enforce *Sequencing rules*
  - Legal ordering of conjugate roles
    - *compress -> expand,* or *encrypt -> decrypt*
  - Proper nesting: *compress -> encrypt -> decrypt -> expand*
  - Use presence/absence of RSHs (between nodes) plus precedence rules for roles (within the same node).

Simple Example Using RBA

{ RSH(HBHforward@* ; *dest-NodeID*, *src-NodeID*),
/* -> Forwarding role instance in every router */

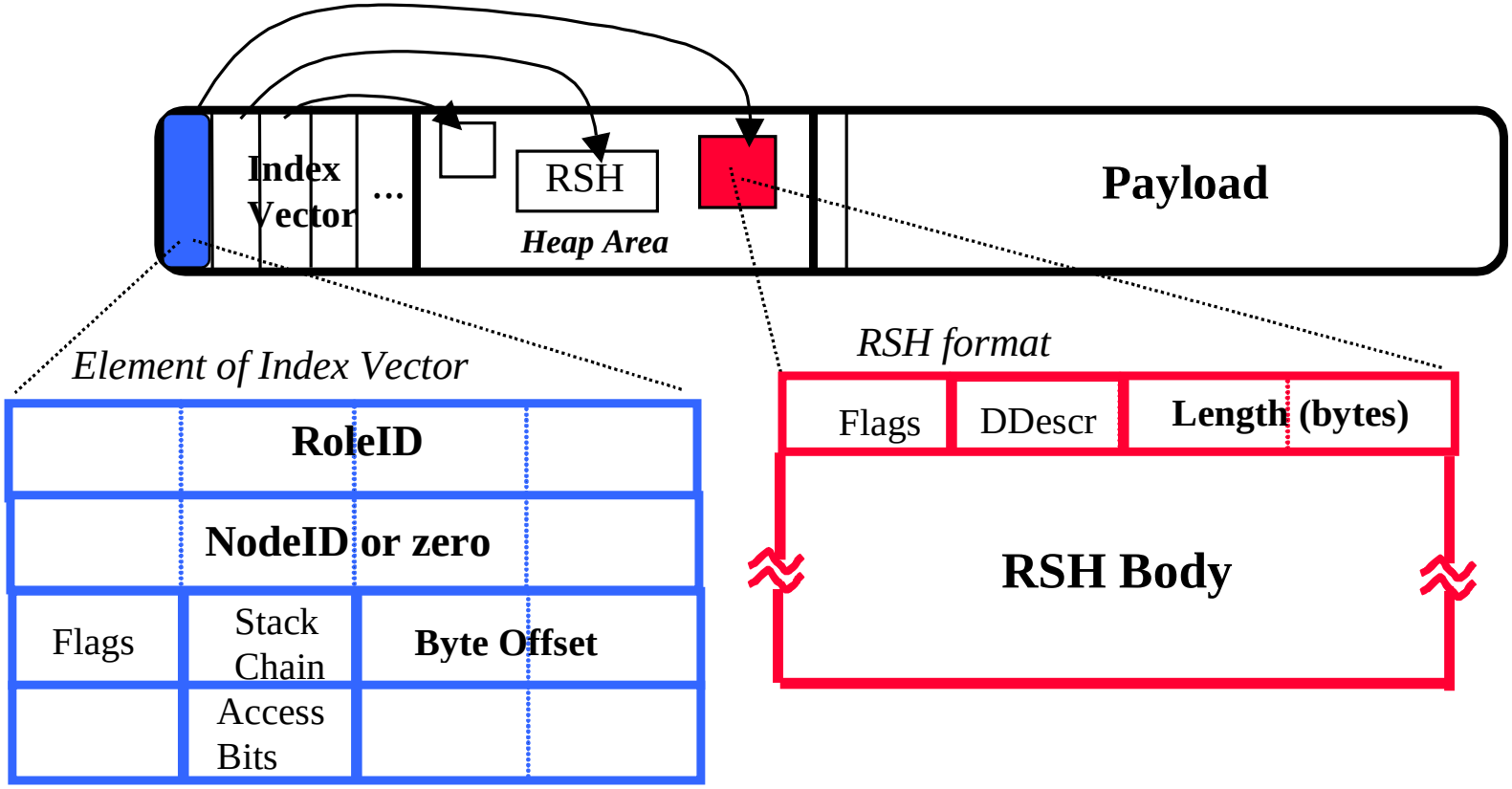
RSH(Deliver@*dest-NodeID* ; *serviceID, src-processID, payload*),
/* Deliver payload to specific service at dest node */

RSH(Reassemble@*dest-NodeID* ; *offset, MFflag*},

RSH(TrustScope@* ; <local scope>)

}

Possble RBA Packet Layout



Using RBA -- Possibilities

- Pure RBA architecture
  - All functions, from current link layer to applications, using roles.
- RBA only above the Link Layer
  - Probably want to treat the link layer as god-given.
- RBA only above IP layer
  - Retain forwarding efficiency of IP in routers.
  - RBA overhead then only in end systems and middleboxes
- RBA only in app layer
  - We need an application layer architecture; RBA could be a nifty framework for it. Would still help immensely with middleboxes.
- RBA only as abstraction for reasoning about protocols.

Related Work

- Hasn't this all been done before? Not really...
- Modular construction of protocol stacks
  - Peterson et. al. 1991 (X-kernel), Tschudin 1991.
- Protocol decomposition into micro-protocols
  - For re-usability & customization --
    O'Malley & Peterson 1992, Bhatti&Schlichting 1995,
    Kohler et al 2000 (Click), Kohler et al 1999 (Prolac).
  - For paralleism -- Haas 1991, Zitterbart et al 1993.
- These all focused on protocol implementations, not on the protocols themselves.
- RBA is orthogonal concept; in fact, the earlier work may provide a basis for realizing RBA.

Conclusions ...

- This is a position paper.
  - We have not yet built an RBA prototype, although a USC grad student is working on it.
  - We have worked through some simple examples.
  - Some of the basic definitions are still subject to debate.
- I hope I have convinced you that a non-layered approach to protocols might not be totally crazy.
  - But we are so used to thinking in a layerist manner that using RBA does twist the head a bit.

Conclusions

- Advantages of RBA
  - Modularizes functionality better then layering does.
  - Provides an explicit place for middlebox metadata
  - Should create fewer unexpected feature interactions
- Disadvantages of RBA
  - Replacement of deployed protocols
  - Less efficient (header space, processing).
  - Greater flexibility may itself increase complexity and confusion.

Conclusions ...

- RBA might be:
  - The Next Great Thing in networking, or
  - only useful for re-organizing particular protocol layers, e.g., the application layer, or
  - only an abstraction for reasoning about protocols.
- RBA appears to have considerable richness and scope for further research.