The RNA Metaprotocol

Joseph D. Touch, Senior Member IEEE, and Venkata K. Pingali

***Abstract*— The Recursive Network Architecture (RNA) explores the relationship of layering to protocol and network architecture. RNA examines the implications of using a single, tunable protocol, called a metaprotocol, for different layers of the protocol stack, reusing basic protocol operations across different protocol layers to avoid reimplementation. Its primary goal is to encourage cleaner cross-layer interaction, to support dynamic service composition, and to gain an understanding of how layering affects architecture. This paper provides a description of RNA and a recently completed initial prototype. The prototype extends the Click modular router with control capabilities including dynamic composition and discovery. These capabilities are used to demonstrate simple but flexible stacks of instances of a metaprotocol that are customizable at runtime.**

***Index Terms*—Algorithm/protocol design and analysis, Internet, Network Architecture and Design, Protocols.**

I. Introduction

THE Recursive Network Architecture (RNA) reuses a single, flexible protocol, called a metaprotocol, for different layers of the protocol stack (Figure 1). RNA allows basic protocol operations to be reused in different protocol layers, avoiding recapitulation of implementation as well as encouraging cleaner cross-layer interaction. It allows protocols and protocol stacks to adjust at runtime, which allows more dynamic composition of services, both within stacks and in the way networking combines the stacks of individual hops into an overall network architecture.

RNA helps explore the impact of layering on network architecture and avoid the redesign of basic protocol constructs used in a variety of layers. By providing a metaprotocol, a single protocol that is instantiated and customized at different layers of a stack, RNA facilitates the composition of as-needed stacks at runtime, based only on the capabilities required over the regions desired. This extends the notion of a single configurable protocol, as in XTP [2] and TP++[3], to retain the layering necessary to partition capabilities across network regions (*e.g.*, links, subnets, nets, ASes). The resulting architecture makes it easier to apply a wide range of capabilities throughout the stack, to combine these layers dynamically, and to integrate related capabilities like security and congestion control among different layers.

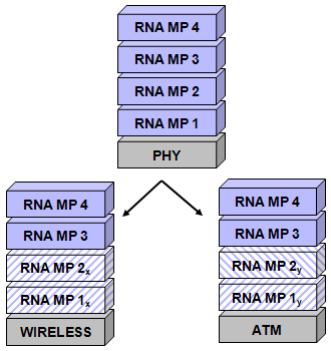

Figure 1 Runtime customization of RNA metaprotocol stack

A prototype metaprotocol is under active development, and an initial version has been recently completed. It extends the Click modular router with additional discovery and negotiation capabilities, and template, or composition rules. Together they enable simple stacks that are customizable at runtime.

II. Reasons for a New Architecture

Protocol stacks are currently treated as a static entity, but they are actually dynamic over certain timeframes in a number of ways. First, protocols change over time. At any point in time, multiple versions of a protocol are often deployed in a given network. Second, the set of protocols that constitute a protocol stack changes over time. Various protocols and capabilities recently added include shim layers such as SHIM6 and HIP [11], and security *via* IPsec [4], IKE [10], and TLS. Third, the nature of use of a given protocol also changes with time and place. The particular capabilities of a given protocol that make it interesting depend on its context and objectives.

This dynamism introduces several challenges. The lack of a way to dynamically relate a protocol to other protocols has resulted in many families of similar protocols, including tunnel protocols (MPLS, GRE) [12], key exchange and filtering protocols (GRE, IPsec/IKE), and transport protocols (SCTP, DCCP, TCP). There is little operational reuse of the functionality that already exists in the network. This lack of reuse increases complexity in the network.

Further, the treatment of protocols is non-uniform; there is an artificial separation between protocols and their virtualized versions. Virtualization has been added at a variety of layers, including link, network, and application. Virtual layers, like shims noted earlier, add new layers to a formerly static protocol stack. These changes impact assumptions made by

Manuscript received February 13, 2008. This work was partly supported by the NSF (Grant No. CNS-0626788). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

J. D. Touch and V. K. Pingali are with the USC Information Sciences Institute, Marina del Rey, CA 90292, USA (phone: +1-310-448-9151 / +1-310-448-8222; e-mail: touch@isi.edu / pingali@isi.edu).

protocols, such as existence or non-existence of a choice at a given layer, *e.g.*, IPv6 where only IPv4 was available previously, the existence or non-existence of a control path, *e.g.*, PPP, and the scale of operation, *e.g.*, TRILL [13], and performance, *e.g.*, BTNS [9]. Choices introduced at a given layer, *e.g.*, IPv4 *vs.* IPv6, introduce a new decision at the next layer, (here, TCP). When the layer neither has the information nor the logic to make that decision, it is pushed up the stack, ultimately to the user.

Ultimately, RNA is motivated by the desire to integrate support for virtualization, as well as to accommodate other kinds of variety in protocol stacks. Although this can be done in typical "dynamic stack" architectures, RNA takes this one step further – recognizing that the layers aren't all that different from each other (in fact, their capabilities are converging), but also that layering itself is valuable. Instead of integrating the functions of many layers into a single, monolithic layer, RNA explores the nature of layering itself – in partitioning protocol functions across regions, and providing a structured framework in which to realize flexible protocol behavior.

III. RECURSIVE NETWORK ARCHITECTURE

RNA addresses the shortcomings of the current Internet protocol architecture by providing a flexible architecture based on the reuse of a protocol over different regions and within different layers in a protocol stack. RNA uses a single metaprotocol as a generic protocol layer (Figure 2). The metaprotocol includes a number of basic services, as well as hooks to configurable capabilities. These services include discovery, negotiation, template matching. This distinguishes RNA from configurable protocol systems such as Click [4] and Netgraph [6]; RNA emphasizes the common structure of protocols, rather providing a software system in which that structure could be built. Our focus is on network-level composition with other layers and nodes using capabilities that are extensions to a protocol composition language.

The metaprotocol provides the building block from which protocol layers are formed. RNA recognizes that protocol stacks have design gaps, both between the layers (vertically) and between stacks (horizontally, hop-by-hop). These gaps stem from the lack of understanding of how one protocol can link to or stack upon another (vertical), how the forwarding operation (horizontal) integrates with traversing layers in a stack (vertical), and how these relationships change over time. These interlayer gaps affect next-layer discovery, selection, and resolution between layers. The inter-stack gaps affect next-hop resolution, in which each stack is typically bound to a particular network forwarding mechanism. RNA highlights and addresses these gaps, to enable protocol layers to be more coordinated within a stack and between different stacks throughout network. The architecture is based on three overall observations about protocol design: that services are relative, that recapitulation of services is avoidable, and that coordination is fundamental to stacks.

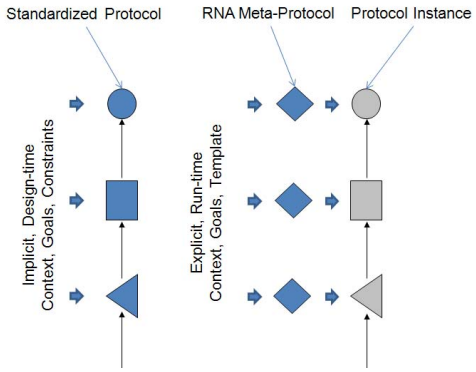

Figure 2 Existing and modified stack

A. *Services Are Relative*

End-to-end services are achieved through the composition of services at various layers and hops. A variety of services can be implemented at multiple layers in a conventional protocol stack including security, reliability, state management, policy (filtering), and congestion control. The particular set of services of a protocol is context-dependent, relative to the layers below and above, and the services are local to its endpoints. Ethernet delivers frames between Ethernet endpoints, and IP delivers packets to IP endpoints by assuming a link layer such as Ethernet coupled with a forwarding mechanism at intermediate locations. Layers are also specific to regions; IP encompasses the public Internet, but a VPN encompasses private regions as well. IP is local to a pair of end systems, whereas HTTP is local to a pair of end applications.

Protocols at the link, network, transport, and session layers all require shared state to manage authentication and its associated filtering, but the distinctions between WEP, IPsec/IKE, TCP MD5, and TLS are often less significant than the assumptions each makes about the security (or lack thereof) at lower layers. There are places where a particular protocol mechanism is better suited to its context, *i.e.*, where stateless or soft-state coordination is better than hard state, but that is based on context more than layer *per se*. As a result of the current fixed layer architecture, new services are often added either by wedging new layers between existing ones or by adding layers of virtualization. Neither fits well within the current, static notion of a stack, and each begs the question of what services need to be added to existing protocol layers, and whether a new protocol is required to do so.

B. *Recapitulation is Avoidable*

Some common services include handshake/state management, security policy (admission control, filtering), multiplexing and demultiplexing, retransmission, reordering, pacing/congestion control, and switching/forwarding. Recapitulating services can result in inefficiencies and errors. There are many such examples, *e.g.*, security being re-implemented at the link, network, transport, session, and application layers, sometimes based on the same algorithms (*e.g.*, DES or MD5), and without specific benefit. It is not

always useful to implement services at every layer; repeated reordering is inefficient and unnecessary. It may be useful to reorder packets before transport protocol processing, *e.g.*, where processing predicts the structure of headers in sequence. Such prediction has been used to streamline TCP, and can also be useful when the security algorithm involves chaining. Reordering when such sequencing is not required, however, serves only to delay packet delivery unnecessarily.

There is also fundamentally little difference between the layers of an overlay *vs*. the base layers of the protocol that the stack natively supports. Virtualization layers need state coordination to establish and maintain tunnels; they also require a switching/forwarding mechanism so that more than two tunnels can be joined at a node. A virtual protocol further requires a way to distinguish messages of one overlay from those of another; this involves labeling and a way to associate labels with context. These mechanisms are already available at the link or network layer, and are reimplemented for tunnels.

RNA observes that these capabilities may all be part of any protocol at any layer, which is why it represents them as a single metaprotocol. Additional layers of the metaprotocol can be instantiated where desired, *e.g.*, to enable particular services over sub-regions of a network, or to add services which have not been enabled by others at lower layers. This allows RNA stacks to accommodate both shim layers and virtualization layers without additional, separate mechanisms.

C. Composition Requires Coordination

Service composition requires that service instances interact across protocol and host boundaries. There are several issues thus raised including discovery of the instances, the communication interface between the instances, the semantics of identities exchanged/used, and the flow of information and control – all of which require coordination. Coordination may be provided at various times including design, deployment and execution time. The key issue is that this coordination is missing, implicit, or *ad hoc* in many circumstances.

Connection binding, for example, allows authentication from a higher layer to be used at a lower layer. This permits upper-layer identities, as might be derived from credit card information on web transactions, to be bound to network-level security associations. Connection latching [9] is closely related to connection binding; latching ensures that data at one layer of a protocol is exchanged over a connection at that layer only when another layer is also connected. This ensures that traffic between a customer and website can proceed through all layers of the stack only when the appropriate layer indicates permission, *e.g.*, when the user's credit card has been authenticated. It is currently very difficult to combine connection binding and connection latching with existing protocol stacks. TCP has one model of connection establishment, IPsec another (via IKE), and TLS yet another (at the application layer). Because these protocols use different mechanisms for state establishment and filtering, it is not always possible to merge their capabilities in a meaningful way without awkward external mechanisms. RNA adds that coordination *via* a common metaprotocol module interface, and shared services to support that interface.

IV. Prototype

A simple RNA prototype has been built to explore the structure of the generic metaprotocol and its reuse at different layers to describe a protocol stack. The prototype is based on Click modular router [5]. Click provides a powerful language to compose protocols from building blocks, and a library of building blocks, called elements, that include queues, packets, switches, and header processing modules. Click 'programs' specify both the elements that must be composed and the rules of the composition in the form of interconnections between these elements. The composed protocols are assumed to be static and known *a priori*. RNA extensions enable Click compositions to be module-driven and synchronized with peers and layers above/below. RNA focuses on runtime network-level protocol composition and capabilities that enable them.

RNA extends Click in three ways (Figure 3). First, the protocol configuration files are used only to specify the modules that are available for composition, not the composition order (as in conventional Click). These modules may or may not be used depending on the RNA metaprotocol instantiation process, discussed below. Second, some RNA modules actively drive the composition of other modules, whereas all existing Click modules are passive. A small number of Click internal functions that execute the composition were exposed to enable the composition to be driven by the modules. A new control interface was added to enable cross-module and cross-protocol discovery and control.

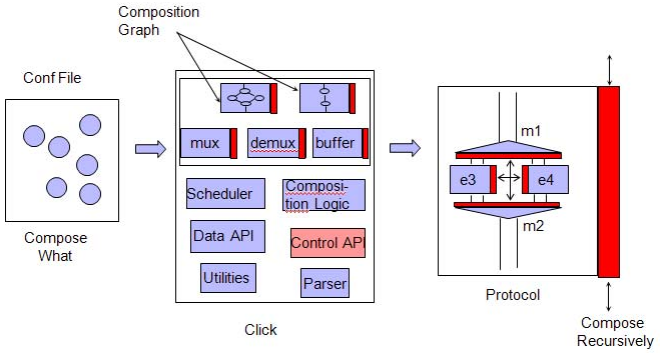

Figure 3 Modified Click

To understand metaprotocol operation, consider the stack shown in Figure 4(c). The stack is composed to two trivial protocols, both of which are instantiations of the same metaprotocol. There is a similar stack at the remote peer. The stack is constructed bottom up with the *instance2* following *instance1,* as shown in Figure 4(a) and 4(b). In both cases the metaprotocol is instantiated with three basic capabilities – discovery, peer negotiation and composition – but no other functionality. The protocol discovers the context of operation and feasible compositions, iteratively negotiates with peer to

select one of them, and executes the selected composition.

This minimal protocol discovers its context, *i.e.*, goals, composition rules, available composable modules, downstream protocols and peers within the protocol. The method used to discover each is different. The composition rules, including preferences and constraints, are available as a template module that may be queried at runtime. The template is discussed in more detail in Sec. IV.D. The goal of the composition may be specified in the template, as a parameter to the metaprotocol, or through the control interface from another element coordinating the entire stack. The list of modules available for composition is discovered using a combination of Click and the RNA control interface. The protocol endpoints may be implicit, such as when the lower layer connects to only one other endpoint, or explicitly specified. When endpoints are specified, the information may be exact or in terms of parameters for the discovery function at next lower layer.

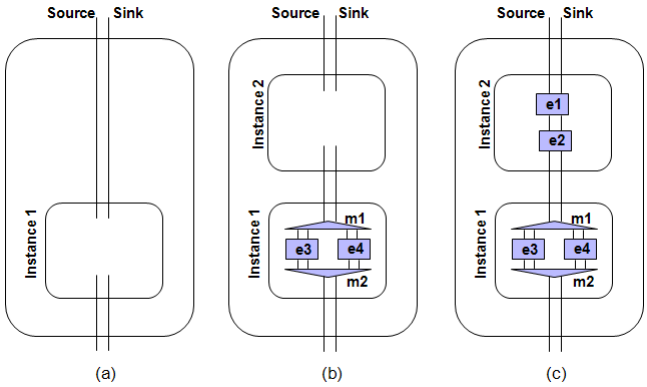

Figure 4 Construction of an RNA metaprotocol stack

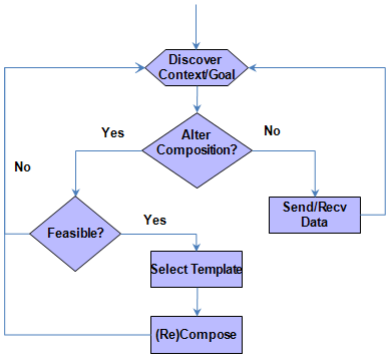

Figure 5 Composition process

Endpoints with the minimal protocol instantiated and the communication path established go through a negotiation process to identify functionality that is acceptable to both ends (Figure 5). The negotiation process is iterative and uses a shared language to describe the functionality. The language is also used for cross-layer control operations, and is based on the notion of a template and patterns that is discussed later in this section. Once an agreement is reached, the composition is executed and the protocol is functional. Once *instance1* is completed, the metaprotocol is instantiated a second time, as *instance2*. This minimal protocol goes through similar negotiation process as *instance1* and takes into account the functionality already provided by *instance1*.

Protocol construction is an open-ended process. Unlike existing protocols that are instantiated once and remain fixed during a system lifetime, an RNA protocol instance may change over time. Initial composition is just the beginning of a cycle of ongoing reevaluation, driven by changes in goals, context, and the nature of communication. The prototype supports a simple recomposition process that is triggered using *ad hoc* methods such as a timer callback and a management message from adjacent layers indicating changes therein.

We discuss the various components of the prototype below.

A. Cross-Host Namespace

The global namespace provided by Click is not visible outside the Click process and host. We add a cross-host namespace for components and composed functionality for use during control communication between peers and layers. Each type name, such as *buffer* or *ordered delivery,* carries its own syntax and semantics. In case of cross-layer/cross-module communication, it also specifies a control interface understood by other modules when they use the name. Although this namespace is shared only between communicating endpoints, they are currently expected to be global, due to cost of maintaining a new namespace among each pair of endpoints. There are significant challenges associated with this namespace, as with any other ontology. This additional cost may be offset by the gains from reduced protocol complexity and the improved ability to evolve over time.

B. RNA Elements

Three kinds of RNA elements are currently supported - Data, Control, and Template. Data RNA elements are the simplest modules performing operations on the data path, *e.g.*, a buffer. One or more Data RNA elements are composed into a protocol by the Control RNA elements. The latter uses the Template RNA elements to determine the feasibility of compositions and preferences, and drive the process. In effect, the Template RNA elements act as knowledge base for the active Control RNA elements. The Control and Template elements together constitute the metaprotocol. Instances of the RNA protocol that may use the same or different templates can be composed into a single stack at runtime. The composition of two elements involves creating unidirectional links between those elements and using the control interface to provide additional information about the context and parameters. The process is recursive, starting from the stack down to the last data element, with Control RNA elements at every level providing the necessary coordination.

C. RNA Control Interface

RNA Elements support an explicit control interface to communicate with each other. Click provides the basic capability to list all the elements, elements at the remote end of any incoming/outgoing unidirectional link/connector, and read/write to element-specific variables. RNA elements extend

the control interface to discover RNA-specific capabilities and state, annotate links between elements, and alter link or element state. This control interface is used by the Control RNA elements during the bootstrap phase to add and delete links between composed RNA elements and informs them of the existence, direction and nature (intra/inter-protocol) of those links. External information required by the elements, such as parameters (buffer length, *et al.*), and binding information, (IP addresses, *etc.*), can also be provided. Binding between modules and layers today is currently arbitrary, with many and varying interfaces; RNA elements provide a generic explicit control interface that simplifies the binding process. In Figure 4, for example, the elements *e3*and *e4* explicitly register the data pattern that should be used to dmultiplex the incoming packets from the downstream nodes.

D. RNA Template

The RNA template, or rule of composition, is used to determine the feasibility, preference and timing of composition at runtime. The rules are grouped in terms of patterns, which specify four kinds of instructions: (1) a starting pattern, *i.e.*, initial composition, (2) a set of prerequisites that must be satisfied before executing a pattern (given in terms of existence or number of instances of given type of element, and earlier patterns that must be executed), (3) the parameters for the element when it is composed, and (4) the channels, number and direction between composed modules that must be added or deleted. This language is still under refinement.

A simple example template is shown in Figure 6. The protocol using this template starts in the MIN state. The MIN state composes an instance of BUFFER element. The SELF refers to the Control RNA element that is executing the composition. The pattern specifies that the logical channels that must be created to send and receive messages from the BUFFER. The identifier associated with BUFFER indicates the instance of the BUFFER element that must be incorporated or removed. Upon completion, the template provides possible options for expanding the composition further to the ORDERED_DELIVERY and other patterns.

The Control RNA element first checks the feasibility of the patterns locally and negotiates with the peer to select zero or more patterns to execute. Once the negotiation is completed and a preference identified, the Control RNA element proceeds to execute the selected pattern. During the lifetime of the communication the composition may be modified, and there may be additional rules for determining the timing and process of re-composition.

```
START PATTERN MIN

# This simply specifies a buffer. no reodering etc.
PATTERN MIN
   REQ MUST BUFFER 1
   ARG BUFFER 1 VAR size 1000
   LINK ADD SELF 0 BUFFER 1
   ...
# Next use this pattern if MIN is successful
PATTERN ORDERED_DELIVERY
   FOLLOWS MIN
   REQ MUST REORDERING 1
   LINK DEL ….
   LINK ADD ….
…
# If reordering successful, try more stuff…
PATTERN ENCRYPTED_ORDERED_DELIVERY
   FOLLOWS ORDERED_DELIVERY
   REQ MUST ENCRYPTION 1
   ARG ENCRYPTION 1 VAR algo des
   ARG ENCRYPTION 1 VAR keysize 512
   ....
```

Figure 6 Example RNA template

E. RNA Data Path

Each module processes messages differently based on the nature of the module, number of links, type (intra or inter-layer) and direction (upstream/downstream). This is similar to, and an extension of, the MDCM data path template [8]. Data path operations may trigger control path operations. For example, messages flowing up and down the RNA stack are annotated with connection state. The connection-orientedness of the downstream RNA elements is explicitly discovered using the control interface. An explicit connection-open is performed and a handle for connection-specific state obtained, if necessary. The call may result in a new state being allocated or the old state being returned, and could further result in a nested call to the layer below. This is also the mechanism used to achieve connection latching.

V. RELATED WORK

There is significant amount of work that deals with intra-protocol structure. We review only a small but representative subset of the related work here; other examples have similar relationships to our work. Two features that distinguish the RNA examples from most of the previous work are the reuse of services across multiple layers and the coordination with other protocol endpoints within and across hosts. RNA further ultimately intends to explore the relationship of layering and scope in protocols and protocol stacks.

Modular protocol systems such as Click and Netgraph provide a programming framework for flexible construction of protocols at runtime. They support a composition language in which the protocol can be expressed as a program. RNA's emphasis is on the structure of protocols rather than the programming environment itself; RNA provides a highly structured protocol template in these general-purpose programming environments. RNA metaprotocol-specific extensions may be required that give greater control to

modules in the composition process, as was the case for our Click implementation of RNA.

RNA builds upon universal configurable protocols such as the eXpress Transfer Protocol (XTP) [2] and TP++ [3] that allow for selective enabling of services at runtime, but focuses on the sequence of basic operations than a particular template instantiation that covers all cases. XTP and TP++ were intended to be a single protocol layer that captures an entire stack, whereas RNA recognizes that layering of copies of the template are a key aspect of RNA protocol stack design. RNA's layering represents the localization of function in vertical layers, and the localization of resolution information in the horizontal (spatial) scope of a single layer, even when the same template is replicated at all layers.

The RNA metaprotocol is based on a structured abstract template developed for the Virtual Internet Architecture [7] called the Multi-Domain Communications Model (MDCM), referred to above [8]. RNA extends this template by incorporating the control path, as well as the resolution path MDCM was created to describe. In addition, RNA's template for the control path is discovered and negotiated as the protocol adapts to the context.

A number of mechanisms have been developed to allow for adaptation to context, primarily aimed at improving performance. Examples include limited retransmission [1], or performance enhancing proxies, which deploy tunnels with corrective capabilities (such as FEC). RNA enables protocol adaptation through its cross-layer interface that enables context discovery and coordination, rather than requiring static deployment of tunnel-based patches.

We observe that a metaprotocol can support this coordination more easily, because it uses a generic API and explicitly supports negotiation between endpoints.

VI. Status and Conclusion

RNA extensions were written in C++, the same language used by Click (~8K lines of code; available at [14]). The status of the prototype is summarized in Table I. Our current focus is on the development of the template language.

The RNA metaprotocol is a vehicle to explore implicit and explicit information exchange, discovery and negotiation between protocols. Information contained in protocol data structures such as multiplexing/demultiplexing tables and cross-layer path preferences has global significance within and across hosts, and requires coordination. Protocol functionality is dependent of the goal and the context of operation, and requires coordination with other layers and endpoints as well. RNA enables explicit intra-host information discovery and coordination through control interfaces and coordination entities within stack. Coordination between endpoints of the same protocol is enabled through built-in support for negotiation and selective composition of functionality. Together they enable flexible protocols and stacks that are context-specific, simpler, and reusable across layers.

Further work in RNA will explore the relationship between the current prototype and the overall goal of a single, generic metaprotocol that reduces to existing protocols in various scopes, and in the environment of various lower and upper layers inside a stack. We hope this will help ultimately resolve how shim and virtual layers interact with existing stacks and further expose gaps in our understanding of protocol stacks.

TABLE I. Status of the Prototype

Aspect	Status
Simple Stack	Simple end-to-end communication with arbitrary number of MP Instances shown; Capabilities already in Click
Modules	Small number, basic; Adds a new namespace; Supported: Buffering, Reordering, Mux/Demux, Encrypt/decrypt, Options
Control Interface	Simple; Allows discovery and binding; Discovery:Type of module/protocol, connection-orientedness, connection state, channel properties; Binding: Mux patterns, connection state
Context Discovery	Simple, single layer within stack, simple negotiation b/w peers; Future work: Multilayer, service interface *etc.*
Template	Preliminary; a pattern language to express composition being developed
Performance	Future work

References

[1] I. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby, "Performance enhancing proxies intended to mitigate link-related degradations," IETF RFC 3135, June 2001.

[2] G. Chesson, B. Eich, V. Schryver, A. Cherenson, and A. Whaley, "XTP protocol definition revision 3.0," Tech. Rept. Protocol Engines, Inc., Santa Barbara, CA 93101, Jan. 1988.

[3] D. C. Feldmeier, "An overview of the TP++ transport protocol," In Tantawy A.N. (ed.), High Performance Communication, Kluwer Academic Publishers, 1994.

[4] S. Kent, and K. Seo, "Security architecture for the internet protocol," IETF RFC 4301, December 2005.

[5] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," ACM Transactions on Computer Systems, 18(3), pp. 263-297, 2000.

[6] "Netgraph: graph-based kernel network subsystem", FreeBSD manual page, http://www.freebsd.org.

[7] J. Touch, Y. Wang, L. Eggert, and G. Finn, "Virtual internet architecture," Future Developments of Network Architectures (FDNA) Workshop, SIGCOMM, August 2003.

[8] Y. Wang, J. Touch, J. Silvester, "A unified model for end point resolution and domain conversion for multi-hop, multi-layer communication," ISI Tech. Report ISI-TR-2004-590, June 2004.

[9] N. Williams, "IPsec channels: connection latching," IETF Internet Draft (Work in Progress), Jan 2008.

[10] C. Kaufman (ed.), "Internet key exchange (IKEv2) protocol," IETF RFC 4306, December 2005.

[11] R. Moskowitz, and P. Nikander, "Host identity protocol architecture," IETF RFC 4423, May 2006.

[12] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic routing encapsulation (GRE)," IETF RFC 2784, March 2000.

[13] E. Gray (ed.), "The architecture of an RBridge solution to TRILL," IETF Internet Draft (Work in Progress), Nov 2007.

[14] RNA Project Website, http://www.isi.edu/rna