Computer Networks 55 (2011) 919–935

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

A Dynamic Recursive Unified Internet Design (DRUID)

Joe Touch [a,*], Ilia Baldine [b], Rudra Dutta [d], Gregory G. Finn [a], Bryan Ford [e], Scott Jordan [f], Dan Massey [g], Abraham Matta [c], Christos Papadopoulos [g], Peter Reiher [h], George Rouskas [d]

[a] USC/ISI, 4676 Admiralty Way, Marina del Rey, CA 90292-6695, United States
[b] Renaissance Computing Institute (RENCI), North Carolina State University's Centennial Campus, Partners I Building, Suite 1500 and Engineering Building II, Room 1235, Raleigh, NC 27606, United States
[c] Computer Science Department, Boston University, Boston, MA 02215, United States
[d] Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, United States
[e] Department of Computer Science, Yale University, 51 Prospect Street, New Haven, CT 06511, United States
[f] 3019 Bren Hall, Department of Computer Science, University of California, Irvine, Irvine, CA 92697-3435, United States
[g] Colorado State University, Department of Computer Science, 1873 Campus Delivery, Fort Collins, CO 80523, United States
[h] 3564 Boelter Hall, UCLA, 405 Hilgard Ave., Los Angeles, CA 90095, United States

ARTICLE INFO

Article history:
Available online 24 December 2010

Keywords:
Network architecture
Future internet
Recursive networks
Dynamic stacks

ABSTRACT

The Dynamic Recursive Unified Internet Design (DRUID) is a future Internet design that unifies overlay networks with conventional layered network architectures. DRUID is based on the fundamental concept of recursion, enabling a simple and direct network architecture that unifies the data, control, management, and security aspects of the current Internet, leading to a more trustworthy network. DRUID's architecture is based on a single *recursive block* that can adapt to support a variety of communication functions, including parameterized mechanisms for hard/soft state, flow and congestion control, sequence control, fragmentation and reassembly, compression, encryption, and error recovery. This recursion is guided by the structure of a graph of *translation tables* that help compartmentalize the scope of various functions and identifier spaces, while relating these spaces for resource discovery, resolution, and routing. The graph also organizes *persistent state* that coordinates behavior between individual data events (*e.g.*, coordinating packets as a connection), among different associations (*e.g.*, between connections), as well as helping optimize the recursive discovery process through caching, and supporting prefetching and distributed pre-coordination. This paper describes the DRUID architecture composed of these three parts (recursive block, translation tables, persistent state), and highlights its goals and benefits, including unifying the data, control, management, and security planes currently considered orthogonal aspects of network architecture.

1. Introduction

The Dynamic Recursive Unified Internet Design (DRUID) is a future Internet architecture based on the repeated use of a single, flexible functional unit for different capabilities over different scopes of a communication service. DRUID allows common protocol functions and capabilities to be reused from within a single block, avoiding the need for recapitulated implementation, and allowing these functions and scopes to be dynamically determined, enabling the service to adapt to changes in the local machine and

* Corresponding author.
E-mail addresses: touch@isi.edu (J. Touch), ibaldin@renci.org (I. Baldine), dutta@ncsu.edu (R. Dutta), finn@isi.edu (G.G. Finn), bryan.ford@yale.edu (B. Ford), sjordan@uci.edu (S. Jordan), massey@cs.colostate.edu (D. Massey), matta@bu.edu (A. Matta), christos@cs.colostate.edu (C. Papadopoulos), reiher@cs.ucla.edu (P. Reiher), rouskas@ncsu.edu (G. Rouskas).

1389-1286/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2010.12.016

network context. It also unifies many different aspects of networking, providing a single architecture to integrate the data, control, network management, and security planes in a single, coherent approach. DRUID allows tremendous flexibility and extensibility in network behavior and functionality, while simultaneously maintaining a simple unified architecture.

DRUID explores the impact of layering and scoping on network architecture. It is composed from a single *recursive block* together with a graph of *translation tables* representing relationships between different scopes in the network. The recursive block includes both code and data, and as it recurses, guided by this graph, it refers to and modifies *persistent state*, so these simple components can support a wide range of communication services. Paths through the graph of these tables – both at end systems and intermediate nodes, including routers, tunnel boxes, and NATs – can adjust, *e.g.*, in reaction to DoS attacks, when local or intermediate node resources change, or in reaction to network path properties.

DRUID applies the concept of recursion as a fundamental network primitive, unifying aspects of USC/ISI's RNA [56,58] and BU's RINA [44] projects, and augmenting them with more detailed description of the impact of the naming hierarchy on the recursive architecture, and discussing how this approach is related to the first principles of multiparty communication. DRUID also explores the relationship of resource discovery, routing, forwarding, and layering as aspects of a single, unified approach. This provides an opportunity to unify the data, control, management, and security planes, allowing one architecture to support coordinated transport state control, network monitoring and management, and reaction to attacks, and to integrate stateful associations at different scopes, *e.g.*, allowing end-to-end streams (*e.g.*, TCP) to easily map onto subpath streams (*e.g.*, wavelength lightpaths). DRUID further affords an opportunity to explore more dynamic service composition and service adaptation, allowing composed protocols to react to local resources, network resources, policy, economics, and threats.

DRUID's approach helps provide a basis for potentially new understanding of multiparty communication, and for integrating many extensions currently considered artificial or external. It provides a coherent, unifying view of networking, supporting the coordination of the data, control, management, and security planes rather than considering them independently. Whether successful as a replacement to the Internet or not, it represents a unique opportunity to impact the community's view of network architecture as more than mere archaeology (studying implementation artifacts), driven by a concept core to the basis of computer science – recursion.

The remainder of this paper presents the motivation for a recursive architecture in Section 2, the architecture itself in Section 3, including how it addresses trust, a key deficiency in the current Internet. Section 4 discusses issues and challenges in realizing the architecture, and Section 5 presents other discussion. Section 6 summarizes the current implementation status, which focuses on our individual current projects. Our future plans for the implementation are discussed in Section 7, and some related work on which DRUID is based is presented in Section 8.

2. Motivation

DRUID is partly motivated by some fundamental observations about multiparty communication [61]. Consider first the standard Shannon two-party communication channel model. In this model, the exchange of data between two endpoints is typically characterized in terms of the channel error and encoding overhead. The model and its descendents derive numerous properties about such a two-party channel, but this is of little direct relevance to network architecture because the two endpoints are considered known *a priori*. The most challenging part of network architecture – knowing who you want to talk to – is removed from the model by the initial conditions.

When going beyond two parties, multiparty communications are driven by three properties – the heterogeneity of the parties, the potential that any subset might want to communicate, and the dynamics of the ways in which communicating parties associate. Given a set of M heterogeneous endpoints, communication either requires $O(M^2)$ translators, or M translators that all support a common interchange format, as shown in Fig. 1.

In either case, at some layer the data is converted between a local representation and one used to reach the destination that requires a different format, so as data traverses between layers, it needs to be converted, not only in the common interchange format of the layers, but also between the names or identifiers available at a given layer (Fig. 2). Layering itself thus leads to the need for a resolution mechanism.

All parties might want to interact with each other, but just as it is not feasible to have M^2 translators, it is not always feasible to assume there are $O(M^2)$ direct links. Supporting arbitrary pairwise communication thus requires some sort of forwarding as well (Fig. 3). RNA demonstrated that forwarding is equivalent to tail recursion [56].

Finally, the desire to vary the association of groups dynamically itself leads to recursion, because each group can easily be considered a recursive component of the larger set, as a virtual subset (Fig. 4) [15,56]. Such recursive virtualization was explored in the X-Bone [53], and is currently a fundamental part of several emerging extensions to the Internet, including Rbridges (TRILL in the IETF [60]) and LISP (in the IETF [21]).

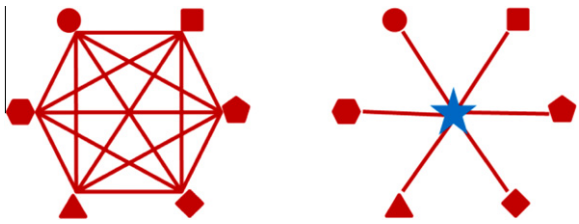


Fig. 1. Heterogeneity of parties leads to $O(M^2)$ translators (left) or a layer of M translators (right).

Fig. 2. Layering leads to the need for resolution (arcs) between layers (bars).

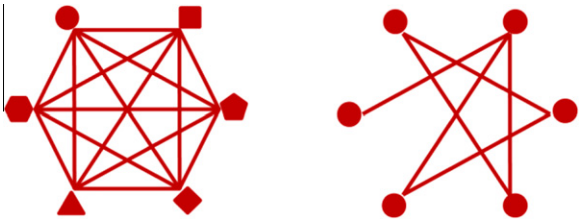


Fig. 3. Arbitrary communication requires $O(M^2)$ links (left) or forwarding (right).

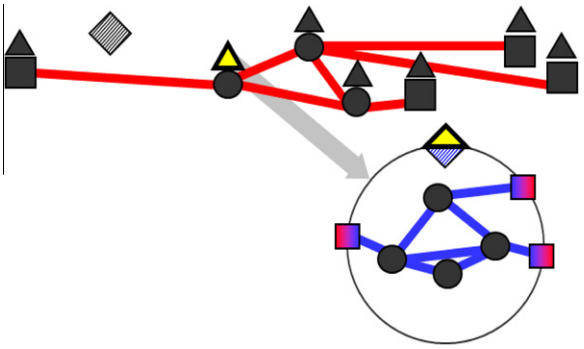


Fig. 4. Dynamic grouping can be easily supported by recursion (bubble to lower right).

Overall, this suggests the conclusion that recursion is a native property of multiparty communication, and that it can thus support the required layering, resolution, forwarding, and virtualization, and further does all these in a dynamic, flexible fashion. The remainder of this paper develops this concept further.

3. Architecture

The DRUID architecture is based on the fundamental principle of recursion, where a single recursive block is reused to create composed layers of capability with different scopes and between different named regions. The architecture is very simple and direct, yet powerful enough to express not only the existing Internet architecture but also the variety of advanced capabilities DRUID affords, including late binding, unification of network management, monitoring, provisioning, routing, forwarding, naming, and protection (including trust) in a single basic mechanism. DRUID utilizes three basic components: the *recursive block*, *persistent state*, and the *translation table*. The recursive block is where protocol functions are realized. The translation table is where name spaces are represented and where conversion between name spaces is indicated **From** a space **To** another space. Persistent state is accessed and maintained by the recursive block in the context of the translation table it previously traversed through and the translation table it will ultimately next traverse through. These components are described further as follows.

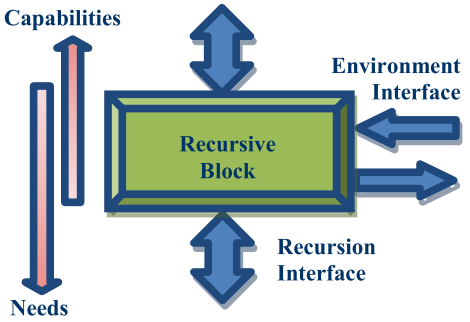
3.1. Recursive block

The DRUID architecture is based on the idea that all protocols can be expressed as varying recursive instances of a single, universal *recursive block* of code and data, a combined perspective of the metaprotocol (MP) of USC/ISI's Recursive Network Architecture (RNA) [56,58,65] and the distributed IPC facility (DIF) of Boston University's (BU) Recursive InterNet Architecture (RINA) [15,16,44] (Fig. 5).

Example code for RNA's version of this block is also shown in Fig. 5. The block is called with a message, from a source to a destination. Inside the block, the message is *processed*, which can include recoding, fragmentation/reassembly, or various other data manipulation functions. The location is examined, and if the data is at the destination, the block returns; if not, the block first determines the corresponding source/destination address of the next layer of recursion (the *resolve* function), and then calls itself (recursing). The process of traversing a network, either vertically through protocol layers or horizontally within a protocol layer through forwarding, is accomplished by this one basic process through a combination of resolution and recursion.

This block is initially invoked by an application, and ultimately invokes a physical interface to allow data to exit a network node. When invoked, the application indicates its communication requirements (*e.g.*, stream or message, reliable or not, ordered or not, *etc.*). As it recurses, these requirements (needs) are matched to the configuration of new recursive blocks (*e.g.*, by setting a flag indicating stream *vs.* message, *etc.*), or to the properties native to physical interfaces; this needs/capability matching operates as a search to provide the desired services from among the available blocks.

The recursive block includes a variety of mechanisms, all implemented in a single location, that support the variety of data operations typical in protocols, such as compression, fragmentation, error correction, reordering, flow control, congestion avoidance, cryptography, *etc.* This principle represents the observation that many of today's protocols already recapitulate similar mechanisms at various layers of a layered protocol stack, *e.g.*, including flow control at the transport, tunnel, and network layers, or adding repeated layers of encryption at different endpoints. DRUID retains the use of multiple layers, as noted in RNA, to support the instances of these same functions over




```
RECBLOCK(DATA, SRC, DST)
  Process DATA, SRC, DST into MSG
  WHILE (Here <> DST)
    IF (exists(lower layer))
      Select a lower layer
      Resolve SRC/DST to lower layer S',D'
      RECBLOCK(MSG, S', D')
    ELSE
      FAIL /* can't find destination */
    ENDIF
  ENDWHILE
  /* message arrives here */
```

Fig. 5. DRUID recursive block, with RNA's version of its internals (right).

different scopes, in order to support aggregation at various locations in a network, to support scale, and to recognize that different mechanisms and policies are appropriate and efficient over different timescales and numbers of participants.

The use of a single recursive block allows mechanisms to operate at different scopes, but also facilitates mechanism reuse, so that, *e.g.*, three-way handshake's complicated set of states and transitions need not be reimplemented to be useful for both TCP and VPN tunnel coordination. The set of such core mechanisms (*e.g.*, as would be included in the *process* step in the RNA example code in Fig. 5) includes the following:

- Data transfer issues.
  - Fragmentation/reassembly (data unit management).
  - Error detection and correction (FEC, ARQ, reordering, *etc.*).
  - Compression.
  - Privacy (encryption, traffic hiding, *etc.*).
- Meta-data (control) issues.
  - State management and parameter negotiation (hard state, soft state).
  - Flow and congestion control.
  - Tuning.
- Policy (management) issues.
  - Access (permissions, authorization).
  - Identity (credentials, anonymous identifiers).
  - Resources (BW, CPU, memory, payment).
- Ordering of the above functions (*e.g.*, compress before encrypting).

The block includes a *recursion* interface that describes how instances of the block interact with other instances, whether recursing down as data is emitted at a sender or recursing up ("popping") as data is handled at the receiver. The *environment* interface describes how the block interacts with persistent state, such as cryptographic keys or reliable data delivery information, and shared non-communication resources, such as memory and CPU.

3.2. Persistent state

During the recursion, the block also interacts with its environment, representing shared resources (*e.g.*, the OS interface, for CPU, memory, *etc.*), and persistent state (Fig. 6). This needs/capability coordination mechanism is similar to that explored in North Carolina State University (NCSU) and the Renaissance Computing Institute's (RENCI) SILO project [3,4,18,19,52,64].

This state can be transient, created anew when the recursive block arrives on an arc, and destroyed when it leaves (by using the next translation table), or the state can persist, matching the name of the current scope of the recursive block (thus enabling a recursive block to find the appropriate state among many). State is maintained by information added to messages within the *process* step inside the recursive block; state is maintained using conventional means, including hard state establishment (three-way handshake), state update (*e.g.*, updating parameters and acknowledging them), and timers (to evolve state in the absence of such information). We are specifically looking at variants of the Delta-*t* protocol to manage such state [24,66].

3.3. Translation tables

The block interacts with *translation tables* (Fig. 7). These tables represent the way in which names and identifiers at one domain are translated into their corresponding values in other domains. Each instance of a table represents a map between a **From** domain to a **To** domain, where the map represents the scope of the translation table. Such tables already exist in the Internet, implemented via a distributed protocol as with DNS, or by simple flooding mechanisms such as ARP. IP forwarding uses the same kind of tables, where the **From** and **To** domains are both, *e.g.*, IPv4, but

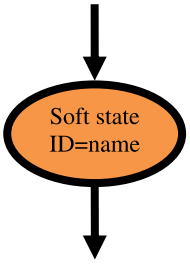


Fig. 6. DRUID persistent state.

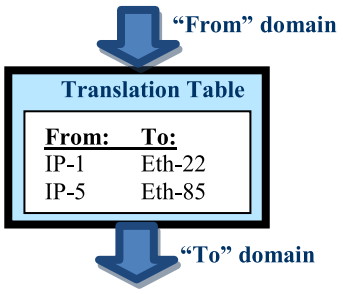


Fig. 7. DRUID translation table.

the metric for selecting a valid **To** value is nearness to the desired destination. DRUID's translation tables can be managed by such pull (ARP) or push (OSPF, BGP) protocols, or can be managed either explicitly (manually) or using metadata (automatic network management); by using a single translation table structure, any of these mechanisms can be used with any table, as desired.

These translation tables are related to each other such that a recursive block can walk a graph that represents all possible protocol stacks (*i.e.*, composed services). This graph connects translation tables by directed arcs, where the **To** domain of one table is connected by an arc to every table whose **From** domain has the same domain type (as illustrated in Fig. 8). At each arc – where the block's core functions operate – there is *persistent state* (Fig. 6). The structure of translation tables and persistent state is what the recursive block traverses until it reaches an exit interface; such interfaces are shown as diamonds in Fig. 8. Also in this figure, there are multiple such states shown for the WDM path on the right, so that the table above these (shown with a shadow) would indicate a specific state by the optical ID (O-ID) determined in the map of the table. Note that this table is similar to protocol graphs in the X-Kernel [26], Click [35], or Netgraph [39], except that in DRUID the graph components and recursive block are each just a single implementation with many instances.

Fig. 9 demonstrates the two modes of the recursive block: processing and recursive graph traversal. Processing involves interacting with local state, here soft state that matches the identifier in a given scope (here "JN3E"). Recursive graph traversal occurs when processing is complete, and the block determines what next scope is most appropriate. It resolves the local identifier ("JN3E") in the table, based on access controls, and uses the To identifier (here "223.45-7") in the scope of the recursive instantiation (shown small in yellow, expanded below to show detail). Note that here the service also has an identifier ("42"), which can help a node optimize searches for corresponding persistent state and paths through the translation table graph by caching or prefetching state using protocol negotiation (*e.g.*, "prefetching the means" [14]).

This graph structure exists at all nodes that participate in any protocol, both the communication endpoints and intermediate devices, such as routers, switches, home gateways, NATs, or tunnel managers. The graph is traversed from the top-down for outgoing data, and each recursive invocation (calling the block at each translation table boundary) accesses not only the available persistent state, but also the information in the data. This information can be augmented with additional metadata that helps drive the "popping" of the recursion at the receiver, directing the traversal up the graph there as well as providing context for updating the state as it goes. At intermediate nodes, the graph can be augmented (extended using additional protocols as transits, *e.g.*, via encapsulation using tunnels), or truncated and replaced (as in a router, when the later recursive steps are "popped" relative to the incoming context and replaced with a different set of recursive operations, based on the outgoing context).

3.4. Invariants and interactions

The basic components of this architecture are the *recursive block*, *persistent state*, and namespace *translation tables*. These components are the basis of the first set of invariants:

- There is exactly one recursive block; all protocol functions are contained therein.
- All services are created by the recursive use of the recursive block.
- All decisions are made inside the recursive block.
- All namespaces are encoded in at least one translation table.

The translation tables represent an implicit graph, where each translation **From** namespace A **To** namespace B is connected by directed arcs both to potential child tables (**From** namespace B **To** some other namespace, *e.g.*, C), and to potential parent tables (**From** some other namespace, *e.g.*, D, **To** namespace A), as shown in Fig. 10. Notice in this figure that arcs always connect compatible namespaces, but some **To** names in one translation table may not exist in all target translation table **From** entries; here IP-1 is in both tables, but IP-9 is in only the right optical (WDM) table. Also note that some arcs pass through persistent state, here indicating that IP-1 and IP-9 have soft-state associated with them, such as monitoring state at the IP scope associated with these IP addresses (an indicator that messages involving this path should be used to update the accounting information in the state).

This structure provides the basis of the remaining invariants:

- All physical network interfaces must have an entry in at least one **To** namespace of a translation table, or they cannot be used to compose services in DRUID.
- All user applications can refer to remote network endpoints only by names that have a **From** entry in at least one translation table, or are the name of a physical interface (in the latter case, the application cannot use those names in any DRUID composed services).
- All persistent state exists relative to two namespaces, so that the identifier of the persistent state exists in at least one **From** namespace and at least one **To** namespace.
- All services are composed of acyclic paths through the translation table graph.

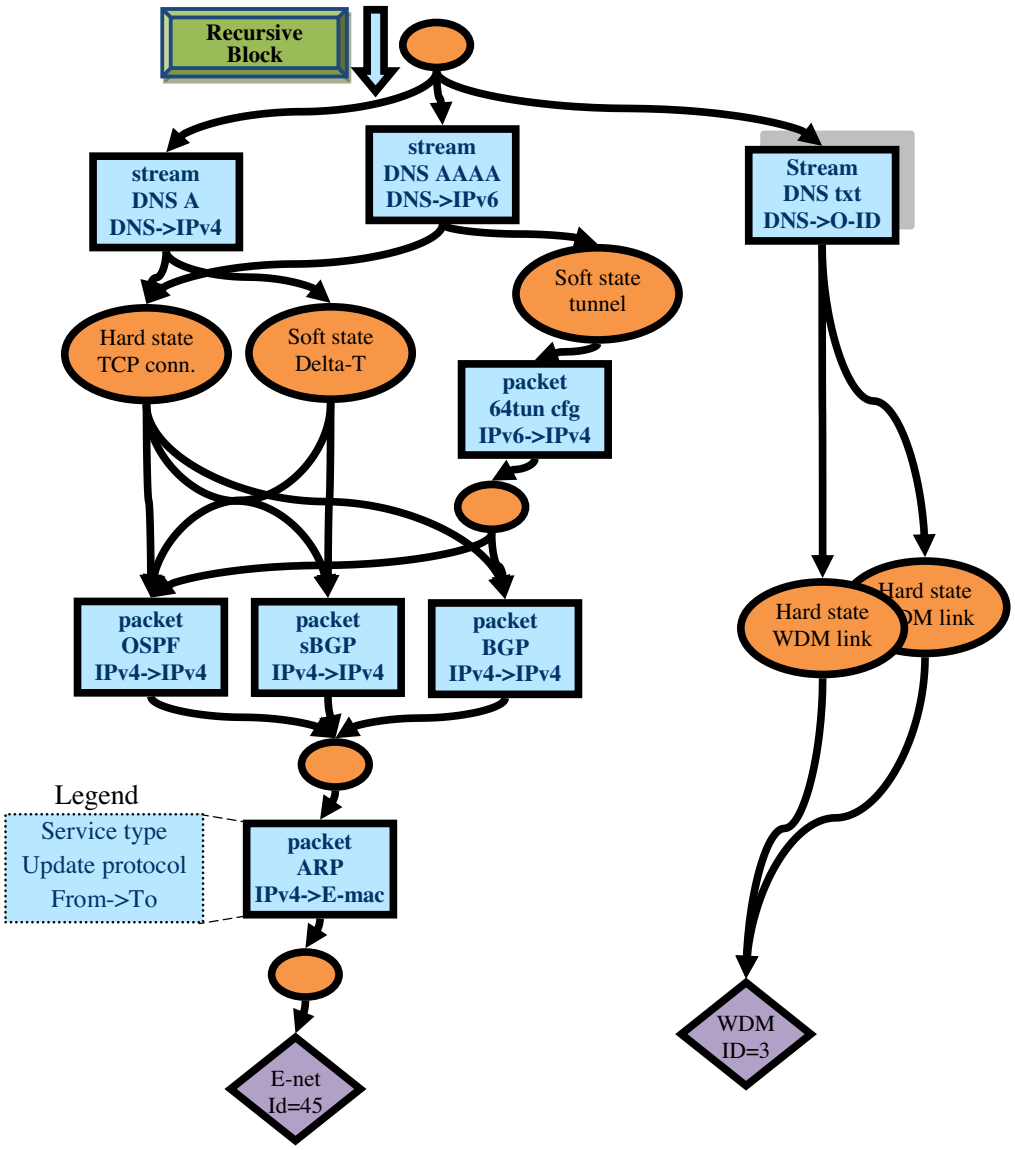


Fig. 8. Graph structure of translation tables and state showing some paths of recursive blocks.

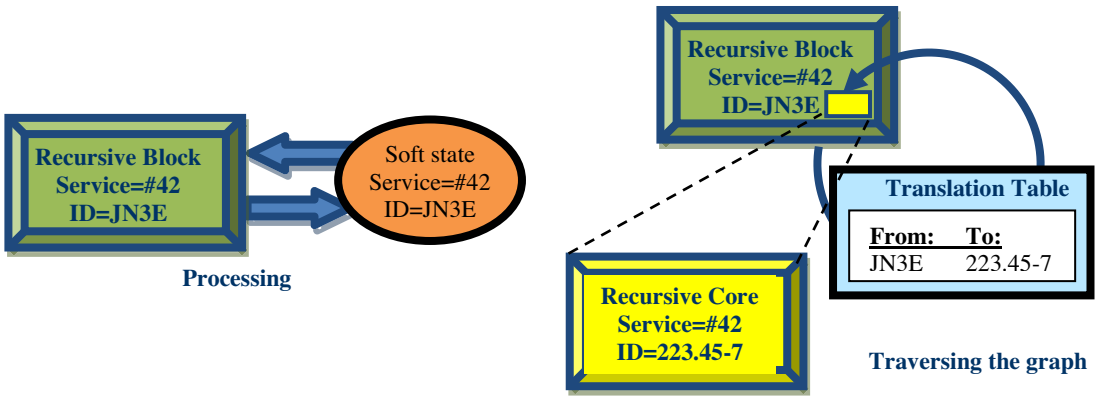


Fig. 9. The recursive block during the processing (left) and recursive graph traversal (right).

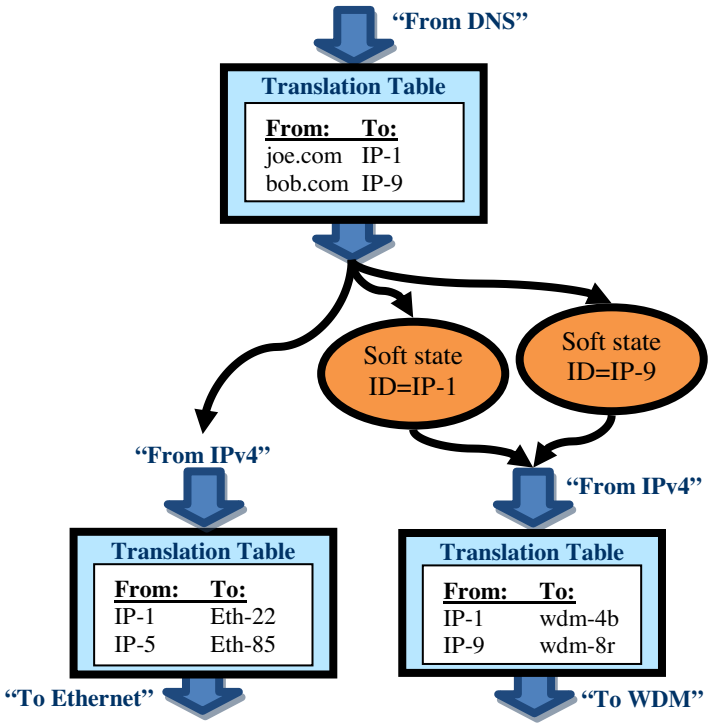


Fig. 10. DRUID translation table graph structure.

The description above indicates the interaction between the components, *i.e.*, that the recursive block, when recursing, considers its current namespace as **From** and examines the translation tables for matching **From** namespaces, with matching entries for the current name being used, and permitted policies, *etc.* The recursive step involves encapsulating information in the data path sufficient to coordinate state on the receiver's graph (where the recursion "pops"), using the given **To** name as indicated in the translation table. The recursive block restarts in the context of any state located that matches its current scope (between the **From** and **To** translation tables it is traversing) and current name. The recursive block may also obtain information about shared context that represents, *e.g.*, local memory, processing, or other resources; this state is no different in concept than that deposited and retained to encode a persistent connection, provisioned circuit, or other endpoint association.

The tables are maintained themselves by applications that use the protocol stack, just like any other applications. This is how current Internet routing protocols operate, *e.g.*, where OSPF uses IP to update intradomain routing tables, or BGP uses TCP to manage interdomain routing tables. In DRUID, each table can be managed by any mechanism desired, with the advantage that once a mechanism is developed it can be deployed to manage tables at any desired scope. A mechanism that maintains these tables as a cache with broadcast queries can be used for the link-to-IP tables, implementing ARP; that same mechanism can relay queries based on patterns in the namespace, implementing the DNS. Common mechanisms, such as cache maintenance, can be used in a variety of namespaces (ARP tables, DNS resolvers) without requiring repeated custom reimplementation.

These elements compose the basic core of the architecture of DRUID. The remainder of the architecture is represented in the interfaces of the recursive block, the translation table entries and their associated context (representing distance, cost, capacity, permissions, trust, policy, *etc.*), and the way in which persistent state is managed and aggregated.

4. Issues and challenges

DRUID is a single integrated system with a simple core architecture, but it also affords an opportunity to investigate a number of different network architecture issues, including trustworthiness and security, protocol discovery and composition, state management and coordination, resource management, network management and monitoring, and routing and naming. There are also a variety of implications of DRUID's recursive architecture that are interesting to explore, as we discuss at the end of this section.

Trustworthiness and security are of key importance to DRUID, not only because they have often been overlooked in the current Internet, and they thus represent a critical issue to address in future Internet architectures, but because DRUID presents interesting opportunities and challenges. DRUID's flexibility and dynamic composition present new potential vulnerabilities, where one user can masquerade as another and acquire their resources. For example, the

stack used for a particular user and transmission through a DRUID network may be something that requires access control. Perhaps another user should not be allowed to use the same composition of blocks, particularly if all instances of a particular block at one router share some resource. If sharing of such a block is not controlled, there may be opportunities for information to leak between users or for denial of service by overloading the shared resource. If blocks can be guaranteed to avoid such sharing and the resulting possibilities for unintended interactions, these security issues can be avoided, but it is not clear that we can design DRUID's interfaces and supporting systems to guarantee such non-interference. If we cannot, identifying management and trust accumulation will be of particular interest in DRUID.

However, DRUID also provides an opportunity to manage these identities, resources (including economics), and policy through a unified mechanism in the recursive block and a unified mechanism in the translation tables in the graph. The dynamic decisions of the recursive traversal of the translation table graph, as well as the persistent state, help DRUID to be more adaptive and context dependent, so that various protections and compensation can be added as needed, delaying the cost of expensive mechanisms until they are of real benefit. For example, certain types of DDoS protection [37,41] require adding marks to packets and checking for those marks at various points in the network. DRUID's ability to add and remove blocks dynamically would permit inserting and removing the necessary blocks at the appropriate locations only when DDoS defense was actually required, rather than at all times. Although other networking approaches could be adapted to achieve similar goals, the inherent dynamism of DRUID stack composition would make the process simple and less prone to errors, as well as making it possible for a wide range of options to be deployed, instead of just one previously chosen defense. DRUID's modularity further makes it easier to integrate security across different scopes, as well as to add or augment capabilities throughout the system, *e.g.*, to replace flawed component mechanisms.

The recursive block operates and recurses, guided by the graph of translation tables, making decisions at each step to result in a graph path that effectively composes protocol functions active at various scopes. The recursive block thus needs a reentrant interface, together with a language to specify the needs (from the user level down) and capabilities (from the physical layer up) of each instance of recursion. DRUID needs to manage the stability of the protocol composition of the recursive layers, *e.g.*, so that congestion control at one layer does not interfere with congestion or flow control at another. For some cases, this can be managed through the reentrant recursive block interface, *e.g.*, by passing the timescale of the feedback mechanism (so that lower layers operate over longer, more stable timescales). For other functions, a simple flag suffices – *e.g.*, once the data is compressed at one layer, it is unlikely that further compression will help. Many of these issues have been explored in SILO [18,19,52,64] and will be incorporated into DRUID as a result, and others are being developed in RINA and will be adapted here as well.

DRUID relies heavily on a coordinated state management capability, both to establish state at various scopes (*i.e.*, for a single connection), as well as to coordinate state across instances, *i.e.*, to manage the persistent state located throughout the translation table graph. This state also includes protocol capability discovery [23], so that one endpoint can determine how best to communicate with another – rather than determining its local graph path solely on local information. This protocol coordination can help optimize communication between two endpoints, notably by managing persistent state between the endpoints in advance of other data connections (*i.e.*, persistent state pre-placement). State management also helps determine what metadata is needed to manage and coordinate persistent state between the endpoints of a connection, as well as enabling that information to be used for network monitoring and management, in some cases also driving provisioning.

The DRUID architecture manages resources as part of how the recursive block interacts with the environment, including how it interacts with persistent state and how it recurses. This management can be used to address QoS by reserving or coordinating use of network capacity [32], and the same mechanism and interface can help limit use of critical shared endpoint resources such as memory and CPU, limiting DoS attacks. This resource information can also be used to help manage automatic provisioning, where this information can be left behind as part of the graph path search process, *e.g.*, by leaving information behind about previous requests that failed – such as when a user wants a stream, but cannot alone afford (or fully utilize) an optical WDM circuit. Resources also help determine authorization and payment to use these resources [28], and are guided by the trust and security issues previously discussed.

As noted earlier, DRUID incorporates network management and monitoring as an integral aspect of the recursive mechanism and persistent state. The metadata – information about the data, such as delay, loss, reordering – can be used not only to manage the transfer (for retransmission, pacing, *etc.*), but also can be extracted for use in network monitoring and management. The same information can be collected and used at any level of the system, because it can be relied upon as it is implemented in the recursive block, and maintained in the persistent state. Of particular interest is the notion that such statistics are always being collected at various granularities and aggregated and stored in soft-state, where user network management applications can send messages to collect the information before it expires if desired. Similar soft-state information is kept about the recursive graph path traversal process itself, so that an attempt to use a translation table that fails (for policy, economics, or other reasons) can later be reconsidered in the context of other requests some short time in the future, and can potentially assist in automatic network reprovisioning, or used to share context across connections and state at a given scope.

Routing and naming are integral to the DRUID architecture, represented in the translation tables and their maintenance. Namespaces are global within a particular layer, and translation between namespaces happens only when the

recursive block indicates. New namespaces can be added dynamically by creating a new table that maps between values of existing namespaces and inserting it into the appropriate location in the namespace traversal graph (Fig. 10). DRUID emphasizes that namespaces do not exist solely at a single layer, but rather rely on translation tables both **To** and **From** that namespace, so that the namespace exists only when it is defined in terms of translation to other, existing namespaces. The only exception is the physical namespace, which is defined by the physical interfaces; all other names must map to those names, ultimately.

Routing is the distributed maintenance of the shared entries of the translation table across different nodes, and these tables incorporate access control and resource protection to further integrate trustworthiness and security. DRUID also enables concurrent use of overlapping namespaces, so that, *e.g.*, both BGP and secure BGP (sBGP) can have actively maintained translation tables, and a service can choose based on policy (use only secure routing), permissions, or availability (prefer secure routing, use nonsecure if needed).

Finally, there are various issues of interest based on the novelty of the DRUID architecture itself, as an example of a future Internet architecture. DRUID was developed from first principles of multiparty communication, and thus has potential impact on the science of networking, to explore ways in which our understanding of networking can be driven by fundamental concepts rather than just the examination of artifacts (so-called "network archaeology"). The architecture further affords interesting opportunities to leverage the existing Internet as a constrained case, where the tables and service paths are pre-selected (except at the last, link access layer), to support incremental deployment and backward compatibility. DRUID also creates services on demand, and does not distinguish between the kind of services offered by an ISP and that offered by a single user node; tunnel creation, multiplexing, and other coordinated transit capabilities could exist at a continuum of capabilities, not as a binary decision as in today's Internet (one is either an ISP or one is not). As a result, DRUID has implications on the legal regulation of Internet access [29,30] (*e.g.* net neutrality), and on the economic motivations of ISPs and application providers [31] (*e.g.*, vertical integration and exclusive service offerings).

5. Discussion

DRUID's high-level objective is to explore the unique capabilities of this approach for integrating a variety of network concepts, many as graceful continua rather than orthogonal aspects, including the following:

- Resource discovery, address resolution, scoped naming, routing, and forwarding.
- Recursive functions, virtualization, and protocol layering.
- Data, control, and management planes.
- Hard state, soft state, and network provisioning.

The DRUID approach supports the dynamic composition of services and protocols, and unifying a number of networking aspects currently considered outside the conventional stack architecture. DRUID's architecture encourages a modular design, in which different protocol functions are more easily integrated. This modularity and the repeating nature of the components of the system supports compartmentalization; compartments, together with dynamic capabilities, make it easier to integrate security and trustworthiness into the architecture at the outset and to augment it as needed. By ensuring that modules adhere to a well-understood interface and are constrained by strong compartmentalization, we can better analyze and understand the security implications of transitions between modules. We can also enforce security assertions during dynamic composition. In contrast to less structured methods of extending the functionality of a network, the security issues involved in the addition of a new module can be more easily understood and guaranteed. In terms of trustworthiness, since each module is intended to provide particular well-defined functionality, we can either statically analyze the module to determine if it does indeed do what it says, or dynamically watch its operations to detect deviations from its stated purpose. In an approach where there is no standard interface and no common definition of functionality, determining the trustworthiness of extensions to the system is much harder. Compartmentalization also allows the isolation of modules that are deliberately or inadvertently attacked, perhaps temporarily replacing them with restrictive but more resilient modules to operate that network layer in a "safe" mode.

The use of the recursive block allows each layer to dynamically determine the next layer, where that choice is first limited by the domain of the current namespace, but also by the context of the current block – its identity, privileges, and resources – and the corresponding access controls on the translation tables available (including variations on a per-entry basis). This dynamic nature makes DRUID very flexible and adaptive because recursive instances make decisions in different contexts. Some decisions can be shifted from very high in the stack to later, a kind of late-binding, to allow the composed service (the result of the entire path chosen) to more rapidly react. Other decisions can allow overlapping choices, so a single address might have appropriate translations in a number of tables, where the recursive block selects the desired one based on context. However, such decisions can also be inefficient if made repeatedly and often, which is where DRUID's persistent state can be used to dampen the decision process as desired, caching previous results, whether from previous connections (*e.g.*, support for TCP Control Block Sharing [6,62]) or to support multiplexing associations over a single connection.

DRUID unifies a number of different aspects of networking, as noted earlier. Its translation tables support name resolution and resource discovery [5]. As RNA demonstrated, the same kind of recursive mechanism can support forwarding, using tail recursion. DRUID carries that one step further, recognizing that routing protocols are just different ways of managing a distributed set of such tables, where different structure of the namespace can require broadcast (unstructured names) or can be used to direct updates only where relevant (for structured names, *e.g.*,

DNS, BGP updates to IP, *etc.*). DRUID's use of a continuum of state management mechanisms, together with a needs/capabilities matching system, allows a single architecture to support packets and circuits, and everything in-between. It also allows, for instance, a user stream service to use TCP to support that stream over unreliable, packet delivery mechanisms or to use a native optical circuit (*e.g.*, WDM) for that same user if available (and the user can afford it). This unification also coordinates the data, control, and network management 'planes', currently considered orthogonal aspects of networking, into a single, coherent system. In DRUID, a layer can collect statistics, such as round trip time (RTT), packet loss, and reordering statistics, and that information can be used not only to manage the data at that layer (*e.g.*, a data transport protocol, as with TCP, SCTP, *etc.*), but also to simultaneously and seamlessly support network monitoring and management. In DRUID, because all such information is collected by a single recursive module, every instance can (given appropriate permissions) collect or react to that information, all using the same mechanism.

Security is supported more easily in DRUID because of the recursive block and translation table's natural compartmentalization and reuse of mechanism. The architecture thus facilitates linking security at various layers, which is currently understood but cumbersome (*e.g.*, connection latching [68] and channel binding [67]). DRUID's use of repeated dynamic decisions allows policy and authorization to be considered at all layers of the system once implemented at any layer. This also makes it easier to support reactive security, where costly (*e.g.*, computationally intensive) mechanisms can be activated on-demand at the strength needed [57], and where faulty mechanisms can be replaced at many layers in one operation.

5.1. Underlying principles

The two main underlying principles are *recursion*, as the unifying operation, and *resolution* (name translation), which provides constraints on the recursion and drives the structure of the resulting composed services. This approach supports dynamic composition, as well as late-binding, where decisions can be made later and thus more locally where possible, enabling a reactive, unified architecture. An optional, but also somewhat desirable principle for DRUID is to support the existing Internet architecture as a special case, so as to provide a potential validation and to determine ways to support graceful transition. The DRUID approach based on these principles unifies a number of currently disparate aspects of existing networks, including translation/forwarding/routing, softstate/hardstate/network provisioning, and virtualization/real networking.

There are a number of requirements that the DRUID architecture is designed to satisfy. DRUID is dynamic and adaptive, allowing context-dependent decisions at all scopes. Its unified recursive module makes it easier to integrate new capabilities and create new services, rather than including them as awkward exceptions or 'shims'. Network provisioning, the process of allocating network capacity along various paths, can be integrated as just a variant of soft/hard state management, *i.e.*, there is no difference between creating a new TCP connection and a new WDM circuit. It natively incorporates virtualization, *i.e.*, there is no difference between an interface to a newly created tunnel and a physical interface [55]. Finally, it is easier to incorporate trust, security, policy, and economics as coordinated constraints on the overall system due to the uniformity of the architecture.

5.2. Trustworthiness

DRUID addresses trustworthiness across several dimensions, including resource protection, policy management, identification and authentication, and data privacy and integrity. The architecture supports trust as an aspect of a service that provides assurances as to the availability and correctness of the asserted capabilities of a service. The reuse of a common recursive block and translation table structure makes it easier to compartmentalize resource protection and privacy, as well as to leverage known techniques at various scopes in various naming contexts. DRUID will provide authentication when it is needed, without requiring expensive cryptographic authentication operations when it is not. Further, DRUID enables monitoring mechanisms normally embedded in each protocol layer to interact and gather accumulated information, which can be used to check the asserted capabilities of a service, helping provide an endorsement approach based on explicit validation. This approach may be useful in verifying that other layers are honoring agreements to provide particular qualities of service, for example.

By regularizing how services are incorporated into recursive stacks, there is less *ad hoc* retrofitting and squeezing of shim layers into awkward places. Regular, well-defined interfaces are far less likely to lead to unconsidered security and stability problems than jerrybuilt collections of irregular bits and pieces of code. To specifically address security issues, the recursive interface of the recursive block includes security properties, so that a given user or application can obtain precisely the services permitted and required. Trust is built using this same interface, where even anonymous identifiers passed across the recursive interface can be presented to shared resource management via the environment interface, or be used in different scopes to determine access to translation tables and their entries. Different forms of identifiers and mechanisms for authenticating them can be easily incorporated. Such information can also govern access to persistent state. The dynamic nature of DRUID allows addition of new security services in the middle of a transmission, when necessary. Overall, this basic architecture supports a uniform interface for identification and associated authorizations, accounting, and trust management. Resources – presented either in shared state managed by the recursive block, or by the operating system for environment properties (*e.g.*, CPU, memory, interrupt frequency) can be managed in a consistent manner using a single interface. The flexibility of the DRUID architecture permits addition of new security services in a regular, predictable manner, overcoming the existing problem of deployment of such services.

DRUID also offers advantages for other aspects of trustworthiness, beyond security. By having a single recursive interface, it is easier to write correct network modules, and to debug their operation when the network behaves in unexpected ways. Standardized interfaces to environmental information and controls also offer advantages for stability. Generally, the regularization provided by the DRUID recursive architecture will lead to more comprehensible network code, which in turn will lead to greater stability and better understanding of network behavior under varying conditions.

6. Current status

DRUID describes the common aspects and approaches of the USC/ISI RNA and BU RINA NSF projects, which are continuing independently in collaboration. As noted in the Introduction, DRUID unifies these architectures as a common description, and adds more detail on particular aspects, such as the interaction with the structure of the namespace hierarchy, as well as the relationship of recursion to first principles of multiparty communication.

RNA began in August 2006, and a preliminary implementation available as patches to the Click modular router software system has been available since 2008 [35]. This code extends Click with a dynamic multiplexer, demultiplexer, buffer, and graph composition functions, as well as adding a control API supporting on-line modification and monitoring of the modules (Fig. 11). A configuration file indicates the desired capabilities (what to compose), which is translated into a set of composed recursive blocks with particular parameters, which are dynamically configured and assembled into a protocol stack (Fig. 12).

A more detailed description of the current RNA implementation is provided in Table 1, and the software is available at the RNA website [45]. In addition, the RNA approach has been applied to a number of different proposed future network architectures, most recently to the design of quantum networks [63].

RINA only recently commenced (May 2010), and is currently focused on developing specifications. This includes its recursive block, called the Distributed IPC Facility (DIF) [16], the IPC mechanism and associated IPC management, as well as the Data Transfer Protocol (DTP) with tightly bound mechanisms, and the loosely-bound Data Transfer Control Protocol (DTCP). DTP/DTCP are modeled after the soft-state Delta-*t* transmission protocol [24,66].

The project is also specifying an object-based stateless Common Distributed Application Protocol (CDAP) to be used by any RINA application, including management applications, as well as an Inter-DIF Directory (IDD) that supports the dynamic construction of DIFs. The recursive routing process of RINA has been described and compared against other approaches, including LISP [27]. A prototype implementation is expected this Spring.

7. Future plans

DRUID is currently in the preliminary design phase, where the overall recursive architecture is being augmented with dynamic layered protocol systems, trusted routing and naming systems, resource protection and trust management systems, protocol coordination mechanisms, and legal and economic policy frameworks.

The DRUID architecture and approach provides a convenient catalyst for collaborative research. Its use of a single, common recursive block encourages integrated design, and makes each new function responsible for importing and exporting its own signals. As such, it lends itself to incremental development and evolution, not only over the

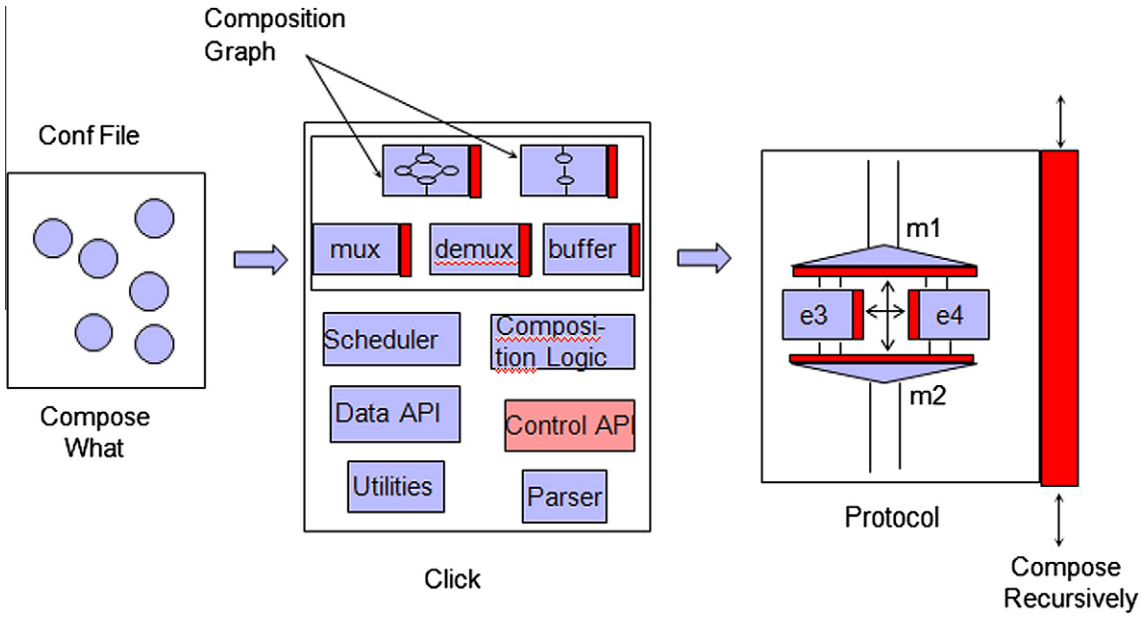


Fig. 11. RNA implementation detail.

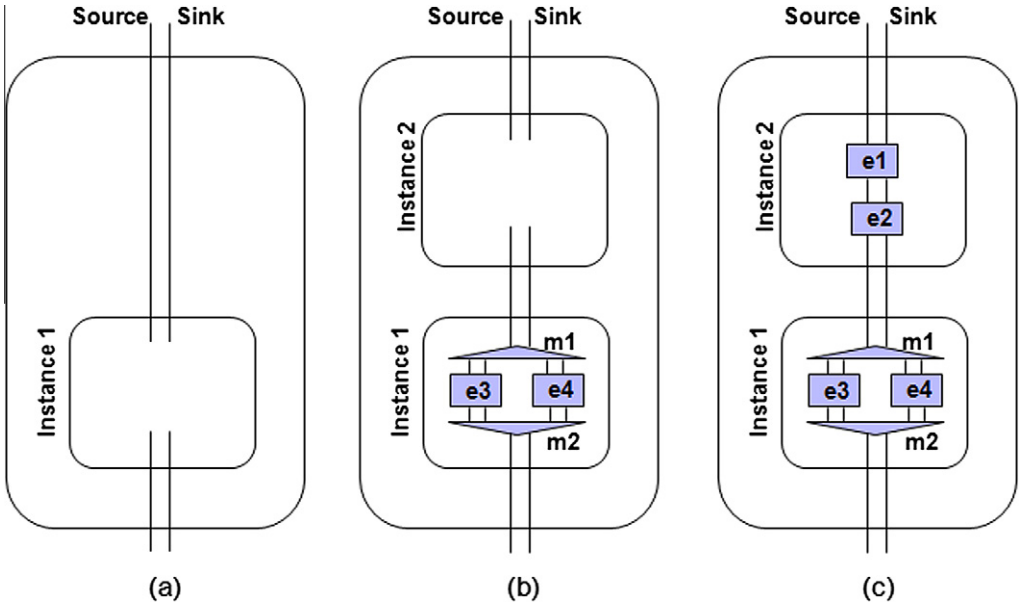


Fig. 12. The process of creating an RNA protocol stack.

Table 1
RNA implementation status.

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

lifetime of this project, but as an extensible architecture for others to later augment. DRUID utilizes the basic approach of the incremental evolution of a native implementation, using real user data and real physical networks, and evaluating the results empirically.

The basic recursive architecture includes the detailed development of the recursive block and translation tables, and mechanisms that manage the structure of the table graph and the way in which state is established, maintained, and/or pre-placed by various recursive block actions. This core architecture is augmented with semantic resource representations to express needs and capabilities as part of DRUID's recursive block interface. We are also developing component selection mechanisms for DRUID's next-table selection process.

Within the recursive block, we are developing a unified transport protocol based on soft-state based on Delta-*t* [66], to be used not only for transport-like management (reliability, reordering), but also to help manage state that guides provisioning, monitoring, and security. This will help in exploring the implications of composition on protocol stability, *e.g.*, recursive flow and congestion control, and optimizations not only of these compositions, but of next-table/layer selection.

We are also exploring ways in which the composition of services leads not only to integration of existing resources, but also requires that the resulting service be offered as a resource, *i.e.*, that consumers of resources need to also become providers in order for recursion to succeed. Our exploration includes this aggregation-as-server aspect as part of the basic recursive mechanism, extensions to address the impact of prices on composition choices and the impact of economics and policy, and investigating the related trust management and resource protection issues, *e.g.*, to mitigate the impact of DoS attacks, IP spoofing, and resource reservation attacks. We are also exploring ways in which trust, protection, and policy can support recursion, and/or are impacted by recursion, and the dynamic deployment of protection mechanisms in DRUID's recursive architecture. DRUID includes state coordination between graphs on different hosts, to optimize future interactions based on aggregation of state, pre-placement, and caching of previous state. Such previous state can also include potential request failures, which can be used to manage provisioning and reactions to routing, security, and trust anomalies.

Other more specific issues being investigated include how to incorporate storage and computation as services

in the architecture, and how to address temporal aspects of namespaces [43] (in support, *e.g.*, of renaming and/or mobility). There are security issues raised by the use of unified translation tables, and the applicability of existing naming protections (*e.g.*, DNSSEC [2]) to broader use at other levels, as well as the interaction between table maintenance and structured naming. We are also exploring the difference between the recursive, directed-acyclic graph of paths in the DRUID graph and the linear stack model in SILO, as well as ways in which reentrancy (as required by recursion) extends their interface model. There are numerous challenges in coordinating highly heterogeneous hosts and ensuring their long-term mutual compatibility, as well as the stability, security, and reliability of the resulting systems, when different hosts and interconnected networks use varying implementations of the DRUID architecture that may have evolved independently in different directions from the DRUID team's "baseline" implementation.

In addition to the technical aspects of our architecture, we are also considering how it interacts with legal and economic issues, especially because DRUID's continuum of services and capabilities challenges many current fixed-tier legal and public policy approaches. Our investigation includes how the equivalence of all layers and implicit convergence of services challenge the current partitioned approach, such as deep packet inspection issues [31], interactions between ISPs and service providers [29], and joint traffic management and network neutrality [30]. The dynamic, adaptive nature of DRUID may expose new potential vulnerabilities – such as 'freeloader' attacks where others use a service they didn't create, as well as new opportunities, *e.g.*, to dynamically adjust the protections provided in a service in reaction to detected threats. For example, DDoS protection services like DefCOM [41] can be dynamically deployed when attacks are present [25], without incurring overhead when no attack is taking place. DRUID also presents an opportunity to balance the dynamic possibilities of sender choice with receiver responsibility, *i.e.*, in conjunction with protocol coordination. DRUID requires resource protection in ways somewhat beyond current architectures, which may require novel approaches, *e.g.*, ways to support resource protection in the absence of a key management infrastructure, using ISI's "anonymous security" approach [59], including trust accumulation techniques. There are also issues of privacy in the DRUID architecture, and the extent to which application interactions with the network system can/should be private, and whether this has other architectural impacts.

We are developing a complete implementation of DRUID, including all elements required to demonstrate the value and potential of a recursive approach to networking. This implementation will include functionality to support many of the examples discussed earlier in the paper, representing a wide range of protocol functions, different types of translation tables, and strategies for recursively creating network services. The project includes a comprehensive strategy to test and measure the system in real working conditions, to ensure that DRUID is not merely intellectually interesting, but a practical approach to improving the Internet. Our evaluation will not be limited to classic network performance testing, but will incorporate thorough security evaluations, stability testing, and economic and legal analysis of the implications of the DRUID approach.

8. Related work

The current Internet architecture has been accused of ossification [42], but has supported numerous extensions, including shim layers (SHIM6 [40], HIP [38], MPLS [47], security with IPsec and IKE [33,34], and TLS [17]), as well as new transport protocols (DCCP [36], RTP [49], SCTP [51]), and other services (P2P nets, performance enhancing proxies [9], BEEP [46]). Many of these extensions challenge the current use of largely static protocol stacks, making it difficult to support virtualization (*i.e.*, VPNs, virtualization as in the X-Bone [53], partial overlays for routing as with RONs [1] and Detour [48], and recursive overlays [54]) and to support on-the-fly addition of capabilities (such as to react to attacks) or changes (from IPv4 to IPv6). DRUID overcomes these challenges by providing a simple, flexible, architecture that unifies many different aspects of networking, including forwarding, scoped naming, routing, and virtualization; connection associations, provisioning, and monitoring; security, policy, and economics, among others. This approach differs from the Autonomic Network Architecture (ANA) project [10] which seeks self-managing protocols and algorithms to sustain network evolution. The ANA framework provides abstractions, communication primitives, and a functional management system [50] that composes the available (data, control, or management) functional blocks in a customized arrangement in order to provide a service. DRUID is developing a single block to accomplish the same goal, but is not focused on self-management.

DRUID unifies aspects of both USC/ISI's RNA and BU's RINA approach to network architectures, in which a single protocol operations block is reused recursively to achieve layered, scoped services. Many protocol functions were originally considered specific to one layer in the conventional ISO seven-layer model. For example, typically, multiplexing of data units from a host happens at the network layer, and stateful connections and congestion control happen at the transport layer. Recently many of these functions are being repeated at many layers in the stack, where stateful connections can occur at the link (circuit), network (tunnel, *e.g.*, for GRE [20], MPLS [47], or even to provide enhancements, *e.g.*, using PEPs [9]), transport (connection), and even higher layers (in multiplexing layers such as BEEP [46]).

The historically *ad hoc* manner of providing additional layer functionality has led to concerns both of design manageability and how to provide user choice. Interaction complexity has produced service incompatibilities that point out a fundamental weakness in our current layered-design methodology [7,8]. Increased choice has led to a proposal for end-to-end cross-layer negotiation [23]. Unfortunately, layering as a method of service decomposition is not now well understood. Attempts are being made to provide a theoretical framework for it [11,12]. DRUID is

designed to provide the flexibility needed to investigate and address these concerns.

This replication of function suggests that the seven-layer model could be replaced with a single, unifying protocol, an "über-protocol". Such monolithic approaches include the eXpress Transfer Protocol (XTP) [13] and TP++ [22]. However, both RNA and RINA consider that although there is merit to incorporating any protocol function at any layer, there remains substantial merit to retaining the layering structure itself. Layering is often considered the result of good software engineering principles, an artifact of judicious implementation strategies. RNA and RINA consider that layering itself has other merits: it helps isolate naming domains which are viable over different timescales and numbers of participants, and it helps compartmentalize functionality within these scopes. The layering of these scopes, while reusing a single common recursive block, is thus the basis of DRUID.

9. Conclusions

Through these various explorations, DRUID provides a coherent, focused approach to a comprehensive network architecture. The DRUID architecture provides a simple and straightforward basis for this integration. Using only a few simple components (the recursive block, the translation table, and persistent state), and their equally simple relationships (as described by the invariants in Section 3.1), DRUID affords the flexibility to incorporate these different dimensions of network architecture in a single, unifying mechanism.

Acknowledgements

This work was partly supported by the NSF Grant Nos. CNS-0626788 and CNS-0963974. 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.

References

[1] D. Andersen, H. Balakrishnan, M. Kaashoek, R. Morris, Resilient overlay networks, in: Proceedings of the 18th ACM SOSP, Banff, Canada, October 2001.
[2] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, Protocol modifications for the DNS security extensions, RFC 4035, March 2005.
[3] I. Baldine, M. Vellala, A. Wang, G. Rouskas, R. Dutta, D. Stevenson, A unified software architecture to enable cross-layer design in the future Internet, in: Proceedings of the ICCCN, Turtle Bay, Hawaii, August 2007, pp. 26–32.
[4] I. Baldine, J. Chase, G. Rouskas, R. Dutta, At-scale experimentation with resource virtualization in a metro optical testbed, in: Proceedings of the ICVCI, May 2008.
[5] G. Bartlett, J. Heidemann, C. Papadopoulos, Understanding passive and active service discovery, in: Proceedings of the Sixth ACM SIGCOMM Conference on Internet Measurement Conference (IMC), San Diego, CA, October 2007.
[6] J. Bannister, W. Shen, J. Touch, F. Hou, V. Pingali, Applied learning networks, ISI Technical Report ISI-TR-2007-637, April 2007.
[7] A. Bestavros, A. Bradley, A. Kfoury, I. Matta, Safe compositional specification of networking systems, ACM SIGCOMM Computer Communication Review 34 (3) (2004) 21–33.
[8] A. Bestavros, A. Bradley, A. Kfoury, I. Matta, Typed abstraction of complex network compositions, in: Proceedings of the IEEE ICNP, November 2005.
[9] I. Border, M. Kojo , J. Griner, G. Montenegro, Z. Shelby, Performance enhancing proxies intended to mitigate link-related degradations, RFC 3135, June 2001.
[10] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, M. May, The autonomic network architecture (ANA), IEEE Journal on Selected Areas in Communications 28 (1) (2010).
[11] M. Chiang, S.H. Low, A.R. Calderbank, J.C. Doyle, Layering as optimization decomposition: current status and open issues, in: 40th Annual Conference on Information Sciences and Systems, March 2006, pp. 355–362.
[12] M. Chiang, S.H. Low, A.R. Calderbank, J.C. Doyle, Layering as optimization decomposition: a mathematical theory of network architectures, in: Proceedings of the IEEE, vol. 95, 2007, pp. 255–312.
[13] G. Chesson, B. Eich, V. Schryver, A. Cherenson, A. Whaley, XTP protocol definition revision 3.0, Tech. Rept. Protocol Engines, Inc., Santa Barbara, CA 93101, January 1988.
[14] E. Cohen, H. Kaplan, Prefetching the means for document transfer: a new approach for reducing web latency, in: Proceedings of the IEEE Infocom, 2000, pp. 854–863.
[15] J. Day, Patterns in Network Architecture: A Return to Fundamentals, Prentice Hall, 2008.
[16] J. Day, I. Matta, K. Mattar, Networking is IPC: a guiding principle to a better Internet, in: Proceedings of the ACM SIGCOMM CoNext ReArch'08 Workshop, December 2008.
[17] T. Dierks, C. Allen, The TLS protocol version 1.0, RFC 2246, January 1999.
[18] R. Dutta, G.N. Rouskas, I. Baldine, A. Bragg, D. Stevenson, The SILO architecture for services integration, control, and optimization for the future Internet, in: Proceedings of the IEEE ICC, Glasgow, Scotland, June 2007.
[19] R. Dutta, G. Rouskas, I. Baldine, D. Stevenson, The SILO NSF FIND Project Website, 2008. <http://www.net-silos.net/>.
[20] D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, Generic routing encapsulation (GRE), RFC 2784, March 2000.
[21] D. Faranacci, V. Fuller, D. Meyer, D. Lewis, Locator/ID separation protocol (LISP), Internet Draft, March 2009 (work in progress).
[22] D.C. Feldmeier, An overview of the TP++ transport protocol, in: A.N. Tantawy (Ed.), High Performance Communication, Kluwer Academic Publishers, 1994.
[23] B. Ford, J. Iyengar, Efficient cross-layer negotiation, in: Proceedings of the ACM HotNets-VIII, October 2009.
[24] G. Gursun, I. Matta, K. Mattar, Revisiting a soft-state approach to managing reliable transport connections, in: Proceedings of the Eighth International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT), November 2010.
[25] A. Hussain, J. Heidemann, C. Papadopoulos, Identification of repeated denial of service attacks, in: Proceedings of the IEEE Infocom, Barcelona, Spain, IEEE, April 2006.
[26] N. Hutchinson, L. Peterson, The X-Kernel: an architecture for implementing network protocols, IEEE Transactions on Software Engineering 17 (1991) 64–76.
[27] V. Ishakian, I. Matta, J. Akinwumi, On the cost of supporting mobility and multihoming, in: Proceedings of the Third International Workshop on the Network of the Future (FutureNet III), December 2010.
[28] H. Jiang, S. Jordan, The role of price in the connection establishment process, European Transactions on Telecommunications 6 (4) (1995) 421–429.
[29] S. Jordan, A layered network approach to net neutrality, International Journal of Communication 1 (2007) 427–460.
[30] S. Jordan, Implications of Internet architecture upon net neutrality, ACM Transactions on Internet Technology 9 (2) (2009) 5:1–5:28.
[31] S. Jordan, Some traffic management practices are unreasonable, in: Proceedings of the IEEE International Conference on Computer Communications and Networks (ICCCN), August 2009, pp. 137–140.
[32] S. Jordan, H. Jiang, Connection establishment in high speed networks, IEEE Journal on Selected Areas in Communications 13 (7) (1995) 1150–1161.
[33] C. Kaufman (Ed.), Internet key exchange (IKEv2) protocol, RFC 4306, December 2005.
[34] S. Kent, K. Seo, Security architecture for the Internet Protocol, RFC 4301, December 2005.
[35] E. Kohler, R. Morris, B. Chen, J. Jannotti, M. Kaashoek, The Click modular router, ACM Transactions on Computer Systems 18 (3) (2000) 263–297.
[36] E. Kohler, M. Handley, S. Floyd, J. Padhye, Datagram congestion control protocol (DCCP), Internet Draft May 2003 (work in progress).

[37] R. Mahajan, S. Bellovin, S. Floyd, Y. Ionnidis, V. Paxson, S. Shenker, Controlling high bandwidth aggregates in the network, ACM SIGCOMM Computer Communications Review 32 (3) (2002).
[38] R. Moskowitz, P. Nikander, Host identity protocol architecture, RFC 4423, May 2006.
[39] Netgraph: Graph-Based Kernel Network Subsystem, FreeBSD Manual Page. <http://www.freebsd.org>.
[40] E. Nordmark, M. Bagnulo, Shim6: Level 3 multihoming shim protocol for IPv6, RFC 5533, June 2009.
[41] G. Oikonomu, J. Mirkovic, P. Reiher, M. Robinson, A framework for collaborative DDoS defense, in: Proceedings of the Annual Computer Security Applications Conference, December 2006.
[42] L. Peterson, S. Shenker, J. Turner, Overcoming the Internet impasse through virtualization, in: Proceedings of the Hotnets-III, November 2004.
[43] V. Pingali, J. Touch, Recursive temporal namespaces, in: Proceedings of the ACM SIGCOMM CoNext ReArch Workshop, December 2008.
[44] Recursive INternet Architecture (RINA) Project Website. <http://csr.bu.edu/rina/>.
[45] Recursive Network Architecture (RNA) Project Website. <http://www.isi.edu/rna/>.
[46] M. Rose, The blocks extensible exchange protocol core, RFC 3080, March 2001.
[47] E. Rosen, A. Viswanathan, R. Callon, Multiprotocol label switching architecture, RFC 3031, January 2001.
[48] S. Savage, T. Anderson, et al., Detour: a case for informed Internet routing and transport, IEEE Micro 19 (1) (1999) 50–59.
[49] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, RTP: a transport protocol for real-time applications, RFC 3550/ STD 64, July 2003.
[50] M. Sifalakis, A. Louca, L. Peluso, A. Mauthe, T. Zseby, A functional composition framework for autonomic network architectures, in: Proceedings of the Second IEEE International Workshop on Autonomic Communications and Network Management (IEEE NOMS/ACNM '08), Salvador, Bahia, Brazil, April 7–11, 2008.
[51] R. Stewart, Q. Xie, Stream Control Transmission Protocol (SCTP): A Reference Guide, Addison Wesley, New York, NY, 2001.
[52] D. Stevenson, R. Dutta, G.N. Rouskas, D. Reeves, I. Baldine, On the suitability of composable protocols for the assurable future Internet, in: Proceedings of the MILCOM, Orlando, Florida, October 2007.
[53] J. Touch, Dynamic Internet overlay deployment and management using the X-Bone, Computer Networks (2001) 117–135 (A previous version appeared in Proceedings of the ICNP 2000, pp. 59–68).
[54] J. Touch, G. Finn, Y. Wang, L. Eggert, DynaBone: dynamic defense using multi-layer Internet overlays, in: Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX-III), Washington, DC, USA, vol. 2, April 22–24, 2003, pp. 271–276.
[55] J. Touch, Y. Wang, L. Eggert, G. Finn, Virtual Internet architecture, Presented at Future Developments of Network Architectures (FDNA) at SIGCOMM, August 2003, Available as ISI-TR-2003-570.
[56] J. Touch, Y. Wang, V. Pingali, A recursive network architecture, ISI Technical Report ISI-TR-2006-626, October 2006, Presented at the IEEE Workshop on Computer Communications (CCW), Pittsburgh PA, February 2007.
[57] J. Touch, Y. Yang, Reducing the impact of DoS attacks on endpoint IP security, in: Proceedings of the NPSec 2006, in Conjunction with ICNP 2006, November 2006.
[58] J. Touch, V. Pingali, The RNA metaprotocol, in: Proceedings of the IEEE International Conference on Computer Communications (ICCCN), August 2008.
[59] J. Touch, D. Black, Y. Wang, Problem and applicability statement for better than nothing security (BTNS), RFC 5387, November 2008.
[60] J. Touch, R. Perlman, Transparently interconnecting lots of links (TRILL): problem and applicability statement, RFC 5556, May 2009.
[61] J. Touch, From Shannon to recursive nets: multihop/multiparty influences on network architecture, in: IEEE Computer Communications Workshop (CCW), Lenox, MA, October 2009.
[62] J. Touch, TCP Control Block interdependence, RFC 2140, April 1997.
[63] R. Van Meter, J. Touch, C. Horsman, Recursive quantum repeater networks, Progress in Informatics N8 (special issue on Quantum Information Technology), March 2011.
[64] M. Vellala, A. Wang, G. Rouskas, R. Dutta, I. Baldine, D. Stevenson, A composition algorithm for the SILO cross-layer optimization service architecture, in: Proceedings of the First International Conference on Advanced Network and Telecommunications Systems (ANTS), December 2007.
[65] 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.
[66] R. Watson, Delta-*t* protocol specification, Tech. Report UCID-19293, Lawrence Livermore National Laboratory, December 1981.
[67] N. Williams, On the use of channel bindings to secure channels, RFC 5056, November 2007.
[68] N. Williams, IPsec channels: connection latching, RFC 5660, October 2009.

Joe Touch is the Postel Center Director in the University of Southern California's Information Sciences Institute (ISI) and a Research Associate Professor in USC's Computer Science and EE/Systems Departments. He received a B.S. with Honors in biophysics and computer science from the Univ. of Scranton in 1985, an M.S. in CS from Cornell Univ. in 1987, and a Ph.D. in CS from the Univ. of Pennsylvania in 1992. He joined ISI in 1992, and his current projects include virtual networks, optical Internets, automatic networks, and high-performance zero-configuration network security. His interests include Internet protocols, network architecture, high-speed and low-latency nets, network device design, and experimental network analysis. He is co-author of a high-speed networks book and chapters on Internet architecture and overlay networks, and has 4 US patents and over 75 conference and journal publications. Joe is a member of Sigma Xi, a distinguished scientist of the ACM, a senior member of the IEEE, and currently serves as IEEE TCCC Chair, ACM SIGCOMM's Conference Coordinator Emeritus, as a member of numerous conference steering and program committees, and is active in the IETF. He also serves on the editorial board of IEEE Network and Elsevier's Journal of Computer and Systems Sciences.

Ilia Baldine leads RENCI (Renaissance Computing Institute, The University of North Carolina at Chapel Hill) network research and infrastructure programs. He is a networking researcher with a wide range of interests, including high-speed optical network architectures, cross-layer protocol interactions, novel signaling schemes and network security. He received B.S. in Computer Science and Mathematics from the Illinois Institute of Technology in 1993, M.S. and Ph.D in CS from North Carolina State University in 1995 and 1998, respectively. Before coming to RENCI, Baldine was the principal scientist at the Center for Advanced Network Research at the Research Triangle Institute, and a network research engineer at the Advanced Network Research group at MCNC, where was a team member and a leader of a number of federally funded research efforts. He is a member of IEEE.

Rudra Dutta received a B.E. in Electricial Engineering from Jadavpur University, Kolkata, India, in 1991, a M.E. in Systems Science and Automation from Indian Institute of Science, Bangalore, India in 1993, and a Ph.D. in Computer Science from North Carolina State University, Raleigh, USA, in 2001. From 1993 to 1997 he worked for IBM as a software developer and programmer in various networking projects. He has been employed from 2001 to 2007 as Assistant Professor, and since 2007 as an Associate Professor, in the department of Computer Science at the North Carolina State University, Raleigh. His current research interests focus on design and performance optimization of large networking systems. His research is supported currently by grants from the National Science Foundation and industry. His publications include more than 60 papers in many premium journals and conferences, book chapters, and one co-edited book. He has served as a reviewer for many premium journals, on NSF and DoE review panels, as part of the organizing committee of many premium conferences, including General Co-chair of IEEE ANTS 2010, and on the editorial board of the Elsevier Journal of Optical Switching and Networking.

Gregory G. Finn received the B.S. degree in Physics in 1973 from Brandeis University and the M.S. in Computer Science in 1978 from the University of Southern California. Greg joined ISI full-time in 1979 where he developed the first IP protocol suite for Xerox workstations and the first multimedia email system. He has developed a novel gigabit LAN and led the Netstation project to replace a workstation backplane with a high-speed LAN, demonstrating that IP protocols could be used for network attached storage, leading to SCSI over IP. He also developed the command language for virtual network deployment in the X-Bone project, used virtual nets to support geographic addressing and regional broadcast and developed wearable wireless Personal Node computers. His current projects include smart grid security and application anomaly detection. He holds a US patent, is a member of the ACM and has published over 25 papers.

Bryan Ford earned his B.S. at the University of Utah and his Ph.D. at MIT, and is now Assistant Professor at Yale University. He has authored more than 35 peer-reviewed publications, centering on operating systems but covering a range of topics from networking and security to programming languages. His past research projects include exploring novel microkernel architectures, and decentralized naming, routing, and transport protocols for mobile devices. He now leads the Decentralized/Distributed Systems (DeDiS) group at Yale, where he is exploring deterministic parallelism, certifiably secure OS architectures, and disruption-proof network communication.

Scott Jordan is a Professor of Computer Science at the University of California, Irvine. He received the B.S./A.B., the M.S., and Ph.D. degrees from the University of California, Berkeley, in 1985, 1987, and 1990, respectively. From 1990 until 1999, he served as a faculty member at Northwestern University. Since 1999, he has served as a faculty member at the University of California, Irvine. During 2006, he served as an IEEE Congressional Fellow, working in the United States Senate on Internet and telecommunications policy issues. His research interests currently include pricing and differentiated services in the Internet, resource allocation in wireless multimedia networks, and telecommunications policy.

Dan Massey is an associate professor at Computer Science Department of Colorado State University. Dr. Massey received his doctorate from UCLA and is a senior member of the IEEE, IEEE Communications Society, and IEEE Computer Society. His research interests include protocol design and security for large scale network infrastructures. and he is currently the principal investigator on research projects investigating techniques for improving the Internet's naming and routing infrastructures. He is a co-editor of the DNSSEC standard (RFC 4033, 40334, and 4035).

Abraham Matta received his Ph.D. in computer science from the University of Maryland at College Park in 1995. He is an associate professor of computer science at Boston University. His research involves transport and routing protocols for the Internet and wireless networks; feedback-based control design and analysis; architectures for protocol design and large-scale traffic management; modeling and performance evaluation. He published over 100 refereed technical papers, and was guest co-editor of three special journal issues. He received the National Science Foundation CAREER award in 1997. His current projects include recursive network architectures and formal methods for safe compositions of network services. He is the Technical Program Chair of the IEEE ICCCN 2011 Track on Network Algorithms and Performance Evaluation, and Co-chair of the NSF/PNNL CyberCARD 2011 Track on Trustworthy Cyber-Physical Systems and Infrastructures. He served on many program committees. He was on the Editorial Board of the Computer Networks Journal and was General Chair of WiOpt'06, Technical Program Co-chair of ICNP'05, Internet Co-chair of INFOCOM'05, Publication Chair of INFOCOM'03, co-organizer and Technical Program Co-chair of the EU-US NeXtworking'03. He is a senior member of the ACM and IEEE.

Christos Papadopoulos is currently an associate professor at Colorado State University. He received his Ph.D. in Computer Science in 1999 from Washington University in St. Louis, MO. His interests include network security, router services, multimedia protocols and reliable multicast. In 2002, he received an NSF CAREER award to explore router services as a component of the next generation Internet architecture. His current interests include network security and measurements. Current projects include the DHS-funded PREDICT project, making security data available to researchers and Named Data Networks, an NSF funded project looking at future Internet architectures. He is a senior IEEE member and has served on numerous ACM and IEEE conference program committees.

Peter Reiher received his B.S. from the University of Notre Dame in 1979, his M.S. from UCLA in 1984, and his Ph.D. in computer science from UCLA in 1987. He worked on the Time Warp Operating System at JPL until 1992, after which he returned to UCLA, where he is an adjunct professor in the Computer Science Department. Dr. Reiher has performed research on many topics in networks and systems, including network and system security (particularly distributed denial of service attacks), file systems, operating systems, wireless networks, mobility, active networks, and ubiquitous computing. He has co-authored books and published over 100 papers on these subjects in journals and conferences. He is a member of the ACM, IEEE, and the Usenix Association, served as Vice-Chair of IEEE Computer Society CSTS from 2005 to 2007, and is an associate editor for ACM Transactions on Internet Technologies.

George Rouskas is a Professor of Computer Science at North Carolina State University. He received the Diploma in Computer Engineering from the National Technical University of Athens (NTUA), Athens, Greece, in 1989, and the M.S. and Ph.D. degrees in Computer Science from the College of Computing, Georgia Institute of Technology, Atlanta, GA, in 1991 and 1994, respectively. His research interests include network architectures and protocols, optical networks, network design and optimization, and performance evaluation. He is coeditor of the book "Next-Generation Internet Architecture and Protocols" (Cambridge University Press, 2011), author of the book "Internet Tiered Services" (Springer, 2009), and coeditor of the book "Traffic Grooming for Optical Networks" (Springer 2008). He is a recipient of a 1997 NSF Faculty Early Career Development (CAREER) Award. Dr. Rouskas is a Senior member of the IEEE and an IEEE Distinguished Lecturer in 2010-2011.