A Unified Software Architecture to Enable Cross-Layer Design in the Future Internet

Ilia Baldine
RENCI, Chapel Hill, NC
ibaldin@renci.org

Manoj Vellala, Anjing Wang, George Rouskas, Rudra Dutta
NCSU, Raleigh, NC
{mvellal, awang, rouskas, rdutta} @ncsu.edu

Daniel Stevenson
RTI, RTP, NC
dstevenson@rti.org

Abstract— While research on cross-layer network optimization has been progressing, useful implementations have been lagging because the current Internet architecture does not accommodate cross-layering gracefully. As part of our FIND project, we propose a software architecture for the future Internet that is designed to accommodate such interactions. We present a conceptual overview as well as high level software design and an early prototype implementation, and point out the strengths of our architecture.

I. INTRODUCTION

Over the course of the past several decades, networking technology has provided incalculable benefit for education, government, commerce. In the last few years, it has become increasingly clear that the next big wave that will change the world will be the incorporation of small information devices into every aspect of our lives, such as sensors and actuators, and the constant access to such devices provided by networking. We are already seeing the beginnings of this: in sensor/actuator networks, in the increasing functionality of mobile handheld devices, and in the migration of many services to network appliances and the network itself. The dominant vision of networking in the future, and computing in general, has been called *ubiquitous* or *pervasive* networking.

Even as computing technology reaches new heights of ubiquity, a crisis has been seen to be developing that can jeopardize this future vision. The pervasive network of the future is enabled by and must serve a new generation of communication endpoints that are very different from the personal computers and servers that form the bulk of the network endpoints in today's Internet. Communication devices are already appearing which are more integrated, more embedded, with sensors and other ubiquitous computing devices. Such devices often have unique characteristics, both advantages and limitations, which are not common in currently popular networking devices. For example, power consumed by network interfaces and the corresponding battery lifetime are often critical considerations for small sensor network nodes or mobile handheld devices; they are not generally considered critical for personal computers. Current internetworking protocols are flexible enough to handle them, but not gracefully. The overhead imposed in bending the capability of such devices to existing network architecture can make the use of such devices prohibitive and even useless.

A. "Clean-Slate" Internet

Currently, a consensus appears to be forming within the community regarding the need to think carefully about the requirements for the Internet in 15-20 years, to formulate a vision for the future, and to carry out a focused research agenda to realize this vision, possibly starting with a "clean slate". A new initiative of the National Science Foundation has targeted this issue [1]. This project team is currently working on a project that is part of this initiative [2]. A primary goal of our research project is to allow integration of cross-layer design and optimization solutions into the future Internet, because an important the inability to integrate cross-layer interactions is seen as one of the significant shortcomings of the current architecture.

Such interactions have become a common theme in handling new communication devices efficiently. In general, the term refers to the increasingly common tendency to leverage the capabilities offered by emerging network devices by taking them into account at all levels of operation of the network, even operations such as routing or transport, which are traditionally considered to be disjoint from the physical communication device. Emerging pervasive devices are likely to provide powerful capabilities, such as transmission power control or angle-of-arrival detection, which impact all levels of network operation. Similarly, the emerging class of ubiquitous applications pose unique new challenges, such as mobility or disconnection tolerance, which cannot be naturally mapped to be the responsibility of any single one of the traditional networking layers. However, the only way to currently implement cross-layer control and optimization is by custom implementation of the application and the entire protocol stack. Flexibility is attained at the cost of a unified architecture.

We propose a new network architecture that represents a departure from current philosophy and practice. We outline a framework consisting of (1) building blocks of fine-grain functionality, (2) explicit support for combining elemental blocks to accomplish highly configurable complex communication tasks, and (3) control elements to facilitate (what is currently referred to as) cross-layer interactions. We take a holistic view of network design, allowing applications to work synergistically with the network architecture and physical layers to select the most appropriate functional blocks and tune their behavior so as to meet the application's needs within resource availability constraints. We call our architecture the Services Integration,

controL, and Optimization (SILO) architecture.

Next we briefly review some relevant prior work. In Section II we present the conceptual framework of the SILO, and follow up in Section III with the software architecture to implement it. Section IV concludes the paper.

B. Prior Work

Recently, the concept of cross-layering has gained attention. This increasingly common concept refers to control and performance optimization actions taken by the network control algorithm that *cannot be localized in any one of the layers* of a layered architecture such as the OSI reference model. It has become clear that cross-layer control is indispensable in many contexts. The development of Software Defined Radios (SDR) and of dynamic approaches to spectrum efficiency such as that advanced by the FCC based on "interference temperature" [3] shows that such cross-layer interaction will span the entire gamut of layered architecture. Current literature contains numerous examples of cross-layer design in many contexts, [4]–[9] provides a representative sample. In fact, the lack of mechanisms for cross-layer interactions (e.g., for performance tuning) has led to frequent layer violations and the proliferation of $\frac{1}{2}$ layer solutions (e.g., IPSec and MPLS) even in the Internet. As we mentioned before, implementations for such strategies have been largely treated as an ad-hoc and custom projects.

On the architectural front, there have been work proposing new models, though none that specifically targets cross-layering goals. Among recent research, the work most closely related to ours is that on role-based architecture (RBA) [10], [11]. RBA represents a non-layered approach to the design of network protocols, and organizes communication in functional units referred to as "roles." Roles are not hierarchically organized; as a result, the metadata in the packet header corresponding to different roles form a "heap," not a "stack", and may be accessed and modified in any order. The main motivation for RBA was to address the frequent layer violations that occur in the current Internet architecture, the unexpected feature interactions that emerge as a result [10], and to accommodate "middle boxes." We also advocate an architecture based on collections of services assembled on demand and specific to an application and network environment (refer to Section II) as well as a more flexible header structure (as in [12]). However, our goal is on facilitating what in today's layered architecture is referred to as cross-layer interactions, in a manner that meets the exact user requirements and optimizes performance; we also leverage the beneficial aspects of layering, unlike RBA.

Some earlier work also investigated more flexible frameworks for realizing protocols. The use of micro-protocol objects, each encapsulating a single function, to facilitate the development of protocol stacks was considered in [13], [14]. Both approaches are x-kernel specific, as they rely on the x-kernel [15] environment and its mechanisms for communication between micro-protocols. We are not tied to any special purpose environment, rather, we target our approach to a more common POSIX-like OS.

II. SILO ARCHITECTURE

We briefly describe the key concepts of our architecture in this section. For a full description, please see [16] or [2].

A. Services

The fundamental building blocks in the SILO architecture are *services*. A service is a well-defined and self-contained function performed on application data, and which is relevant to a specific communication task. "In-order packet delivery," "end-to-end flow control," "packet fragmentation", "compression," "encryption," and "multi-rate RF PHY" are all examples of services in this context. Each service addresses a separate, atomic function, hence the architecture provides more flexibility and a much finer granularity than current protocols which typically embed complex functionality.

At the core of the architecture is the mechanism through which services interact in order to accomplish complex communication tasks. Our approach represents a middle ground between the strict protocol stack imposed by current architectures and the "heap" approach advocated by the RBA [10]. Specifically, we allow any set of services to be selected dynamically for a particular task, but the order in which these services are applied is not tied to the "layer" in which the service belongs, but rather to a set of well-defined precedence constraints; for instance, when the application requires both a "compression" and an "encryption" service, the only meaningful interaction is when compression is applied before encryption. In general, the precedence constraints impose a partial ordering among services. Once selected, however, the subset of services is arranged in a specific order, derived from the partial ordering and other rules, and this binding remains in effect for the duration of the associated communication task (typically, the lifetime of a connection).

B. Knobs

A service is fully defined by describing: (1) the function it performs, (2) the interfaces it presents to other services, (3) any properties of the service that affect its relation with other services (e.g., as required to establish a partial ordering), and (4) its control parameters, which we also refer to as *knobs*, and their actions and constraints. The knobs are adjustable parameters specific to the function performed by the service, with a specified range of values and a well-defined relationship between these values and the perceived performance of the service. For instance, "compression factor" is a knob for the "compression" service. The knobs are manipulated by the control agent (defined below) so as to optimize the performance of the subset of services selected for the specific task.

C. Methods

We distinguish between a service and its implementation. A *method* is an implementation of a service that uses a specific mechanism to carry out the functionality associated with the service. For instance, "re-sequencing" is one method for implementing the "in-order packet delivery" service, "window-based flow control" is a method for the "end-to-end flow

control" service, and "802.11a OFDM PHY" is one method for the "multi-rate RF PHY" service. A method implementing a service must implement the service-specified interfaces, as well as any service-specific knobs; in other words, service-specific interfaces and knobs are polymorphic to all methods implementing a given service. A method may also implement method-specific knobs, i.e., control parameters unique to this implementation of a service; for instance, "length of Reed-Solomon FEC" is a knob specific to the "Reed-Solomon FEC" method implementing the "error-free delivery" service. These knobs are adjusted by the control agent to refine the method behavior and optimize it for a specific environment. A method is fully defined by describing (1) the service it implements, (2) the specific algorithm/mechanism it uses to implement the service, and (3) optional method-specific control parameters, and their actions and constraints. We emphasize that the architecture defines services and their interfaces, but it does not define the methods that implement them; therefore, it is possible that several alternative methods for a given service co-exist within the network.

We refer to an ordered subset of methods, each method implementing a different service, as a *silo*. One can think of a silo as a vertical stack of methods; conceptually, applications reside at the top of the stack, and network interfaces reside at the bottom. A silo performs a set of transformations on data from the application to the network or vice versa, so that the delivery of data from an application to its peer is consistent with the application's requirements. Each data transformation corresponds to a method in the silo, and may include the generation (or processing) of metadata to be included (respectively, present) in the packet. A silo possesses a state that is a union of all constituent method states as well as any shared state resulting from cross-method interactions. A silo structure and all related state information are associated with a specific traffic stream (equivalently, a connection or flow) and persist for the duration of the connection. One important aspect of silos is that they can be optimized for each traffic stream, as we explain next.

D. Control Agent

A *control agent* is an entity residing inside a node, which is responsible for (1) composing a silo for an application stream (or selecting an appropriate commonly-used silo, as we discuss shortly), and (2) appropriately adjusting all the service- and method-specific knobs and facilitating cross-service interactions. Composing a silo refers to determining the subset of services it contains, their order in the stack, and the method implementing each service. The objective is to dynamically custom-build a silo for each new connection. To this end, the control agent takes into account the application's QoS requirements, current network resource availability and other conditions, the precedence constraints among services, and any policy in effect at the time. The current policy is derived from a combination of local node policies (e.g., battery-saving mode) as well as, possibly, one or more network-wide policies of varying scopes. An example of control agent behavior is
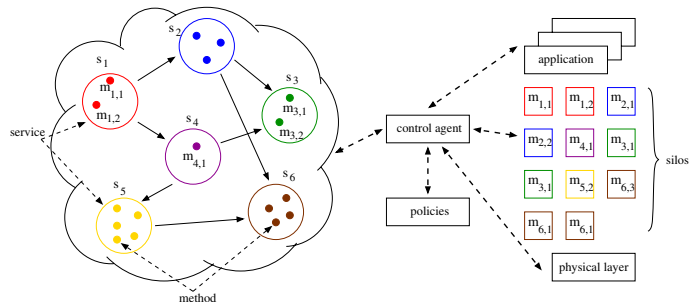

Fig. 1.   Elements of the SILO architecture

tuning the length of the FEC in order to enhance the "error-free delivery" service in response to increased radio interference reported by the "PHY" service. This example clearly illustrates an intentional design feature of the silo architecture, namely, the explicit ability to perform cross-service optimization.

A control agent may optionally be able to communicate with control agents at other nodes in the network (e.g., neighboring nodes, nodes on the connection path, or the connection peer node) in order to optimize the behavior of a silo further; this communication may take place either in- or out-of-band. The control entities should be able to function without the ability to communicate (e.g., due to network bandwidth constraints), but should it be available, they should be able to utilize it.

We expect that in a network following the SILO architecture a number of services will be defined and standardized; the architecture, however, does not impose any limit on the supported services, and is designed to facilitate the addition of new services. Specifically, it should be possible to construct abstract representations of services so as to reason formally about their properties and interactions. Therefore, we expect a large number of experimental and special purpose services to emerge, the most successful of which (e.g., in terms of adoption) may eventually become standardized. Similarly, for common and/or straightforward communication tasks, we expect that a set of pre-constructed silos will be defined. At the same time, we envision many scenarios in which the silo will need to be constructed on-demand, by selecting and vertically arranging a needed set of services, further specialized into methods, in order to tailor its behavior to the application requirements and the network environment.

Figure 1 illustrates the elements of the architecture. The cloud represents the universe of services, each service represented as a circle, with dots within a circle denoting the various methods implementing a particular service. The solid arrows represent precedence constraints among services. The control agent interacts with all elements of the architecture and is responsible for constructing silos of methods consistent with the constraints.

E. Edge and Core

We note that the silo concept blurs the distinction between "core nodes" and "edge devices," in the sense that the role of a node is not tied to specific layers of the protocol stack. Instead,

each network node/device is free to implement any service, and its actual role is determined by the services included and the selective construction of silos out of the existing services to accommodate the communication tasks at hand; for instance, a sensor node may not include a "reliable delivery" service, whereas servers in a Grid environment may include implementations of a "congestion control" service customized for links with high bandwidth-delay products. As a consequence, a node may freely transition from one role to another, e.g., as in a wireless device which may act as either an edge system or a router, depending on the type of network to which it is attached, while remaining consistent with the SILO framework. Furthermore, the SILO architecture removes the necessity of having different control and management paradigms or interfaces for routers as opposed to endpoint devices. In this context, for instance, the collection of data for monitoring resource usage and performance can be thought of as a service that is specific to switches/routers; similarly, the collection of usage metrics related to billing is a category of services specific to access and border switches.

F. Cross-Layering with SILO

We also point out the inherent strength of the SILO architecture with respect to cross-layering. As we remarked before, cross-layer control is indispensable in many emerging environments, such as wireless multihop networks. However, layering has been a tremendously useful networking paradigm *precisely* because it limits the interaction with the internals of the protocol at one layer with that in another. As a consequence, protocols can be designed and implementations can be written comparatively in isolation, more manageably and leading to more maintainable software. Different protocols for the same layer and different implementations of the same protocol can be "plugged in and out" without affecting the correctness or functionality of protocols at other layers. A cross-layer control algorithm by its nature destroys this useful characteristic, because each layer must make some of its internals visible and accessible to other layers. Further, it is rather brittle, because changing the protocol at a given layer or even just the implementation of the same protocol may break cross-layer interactions. Thus the proliferation of cross-layer methods have raised the fear of a regress to monolithic software, unmanageable and unmaintainable. Alternatively, cross-layer approaches would remain curiosities and special cases, never to enter significantly the mainstream architecture.

The concept of knobs for services and methods side-steps these problems. The SILO approach can be viewed as "operate in layers, control across layers." As today, services and methods are required only to provide a minimal interface, hiding internal details. However, traditional protocols are only required to provide invocation methods (APIs), whereas in the SILO architecture we require them to provide a minimal control interface as well. Beyond this, the methods can be designed and implemented in isolation as before. However, the control agent has a unique view into the knobs of every method in the silo, and can embody all the integrated control concerns. In

this way, "cross-layer" (or, more appropriately, "cross-service") can become part of the mainstream architecture. Note also that the fact that the transport occurs strictly in layers is one of the strengths of our approach as opposed to approaches like RBA. Once a silo is formed, the order in which services process a data unit is fixed for the lifetime of that silo; thus header processing can be efficient since the number and order of service headers is always the same for all data being transported through a give silo.

III. IMPLEMENTATION FRAMEWORK

In this section, we describe the architecture of the prototype software demonstrating the capabilities of the SILO concept. The prototype is built in accordance with the Pilot System principle [17], which implies a limited lifespan of the first prototype system. It is intended to be a tool to help us learn about how to properly implement a fully functional SILO framework as well as how not to implement such a system. The anticipated lifespan of this prototype is 2 years.

A. Overview

For ease of discussion, we use the following terminology (note the contrast in the first two items):

- SILO : A framework for creating flexible networking applications
- silo : State storage, associated executable code and execution contexts necessary to perform communication functions on behalf of an application. A silo represents a collection of services and methods operating on a data flow.
- silo state : A storage abstract which maintains information necessary for silo operation (example - congestion window size, number of packets/bytes transmitted etc)
- silo handle : Unique identifier of silo state used between the application and the SILO framework
- Service request : Description of desired services communicated by the application to the SILO framework.
- silo recipe : An XML-based description of the composition and state necessary to create a silo. Contains pointers to dynamically linkable code to methods constituting a silo.
- Control interface : An abstract describing control options of a specific method within a silo. Control interface is composed of method-specific and service-specific control knobs. Service-specific knobs are inherited based on polymorphism of services and methods.
- Control strategy : An algorithm used to manipulate silo control interfaces in concert in order to achieve a specific optimization goal.

The prototype is implemented as a series of user-space components implemented in C++ interconnected using traditional UNIX IPC mechanisms (e.g. UNIX sockets, shared memory, message queues etc). Individual components may incorporate multiple threads depending on the needs of the components. Whenever appropriate, thread-based concurrency will be replaced with event queues to simplify locking and debugging.

Because of the need to develop this prototype rapidly, user-space approach was chosen over a kernel-based implementation. High performance, generally ascribed to kernel-based implementations is not of high priority in this case. User-space approach will allow us to incorporate the code and components from other Open Source Software (OSS) projects without regard to their implementation details. It allows us to mix and match implementation frameworks and languages to achieve the fastest result. An example includes using some OSS Java components alongside the C/C++ implementation of the SILO framework.

B. High-level prototype architecture

The architecture will consist of the following major components:

- The SILO framework
- The SILO ontology
- The SILO-Enabled Application (APP)
- The SILO Construction Agent (SCA)
- The SILO Management Agent (SMA)
- Universe of Services Storage (USS)
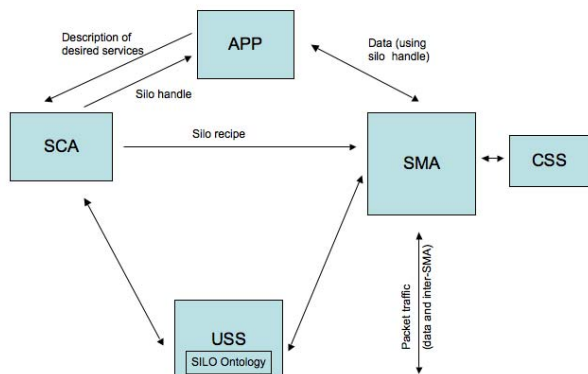- Control Strategies Storage (CSS)

Fig. 2.   High-level prototype architecture

The application creates a service request, which describes its communications requirements. Based on the requirements the SCA constructs a silo recipe, which it then passes to the SMA. The SMA dynamically links in necessary code and instantiates the state for the new silo using the silo recipe. The application and the SMA communicate by referencing a silo handle. The SMA maintains the silo state and when necessary, manipulates the control interfaces in order to optimize performance. A control strategy is used to govern the manipulation of the control interfaces. The SMA selects appropriate control strategies from the Control Strategies Storage based on the desired optimization goals. The high-level behavior of the framework is affected by the policies that are currently in effect (as selected by the user and/or system and/or network administrator). The described

interactions are shown in Figure 2. Below, we describe the individual components in greater detail.

1) Component Description:

a) The SILO framework: The SILO framework consists of C++ header files and library code that implements the necessary API. Two types of API and libraries are implemented:

- Application API - for creating SILO-enabled applications
- Internal API - library code common to the individual SILO components (e.g. to facilitate inter-component communications)

b) The SILO Ontology: The SILO ontology is an XML-based (RDF) description of the relationships between SILO services and methods used to create and operate SILOs. It describes interfaces between services as well as service and method control interfaces.

The SILO Ontology is stored by the Universe of SILO Services component.

c) SILO-Enabled Application : The SILO-enabled Application (APP) is any application that includes a networking communications component implemented using the SILO framework. This means the APP is linked against the SILO library and communicates with the SILO components. The APP could be an existing networked application (e.g. a web-browser) whose socket-based TCP/IP interface has been replaced with SILO framework API calls or, alternatively, a purpose-built application utilizing the SILO framework.

d) SILO Construction Agent: SILO Construction Agent (SCA) is a major component of the architecture whose responsibility it is to assemble a silo based on application service request. It utilizes the SILO Ontology, an inference Engine, and a collection of custom algorithms in order to turn the application request into a silo recipe. The silo recipe is used by the SMA to construct the custom silo for the application.

e) SILO Management Agent: SILO Management Agent (SMA) is responsible for (a) Constructing a silo for a specific application based on a recipe created by the SCA, (b) Maintaining the silo state during the communications session, (c) Manipulating the control interfaces within individual silos in order to optimize its behavior according to a specific optimization goal.

The construction of a silo involves instantiating the silo state, linking in the necessary method code from the Universe of Services Storage and starting any required execution context.

Control interface manipulation is performed in order to optimize either individual or collective behavior of silos within a single node or among many nodes. The selection of appropriate control strategy is governed by policies that are stored within Universe of Services Storage component.

f) Universe of Services Storage: The USS serves as the main repository of information about the SILO framework. It contains (a) The ontology that describes relationships between silo services and service interfaces, (b) A database of method implementations which helps the SMA locate the executable code necessary to construct a given silo, (c) Current policy setting which affect the operation of the SILO framework. These

can be application-, node- and network-specific. The USS has a query-based interface, which allows other components of the SILO framework to utilize its functionality.

g) Control Strategies Storage: The CSS serves as the repository of control strategies for the SMA. Initial functionality of the CSS will be subsumed within the SMA R1. Further SMA releases will rely on a standalone CSS equipped with a query interface to help select and retrieve an appropriate control strategy based on application requirements and policies currently in effect.

2) Component Interactions: Interfaces between different components are expected to be reliable. Blocking is performed where appropriate. The interfaces are:

- Between the Application and SCA. Application sends a service request for a silo, specifying the types of services it needs. SCA replies with a silo handle.
- Between the Application SMA. Application passes data to SMA and receives user data from SMA using the silo handle. Data can take the form of (a) Stream, represented either as non-delineated buffers in some traditional stream-oriented transport, or file descriptors; or (b) Sequence of records, or delineated buffers, which is a record-oriented transport that preserves record boundaries. Only suitable for silos that have been defined for purposes of record-oriented transport.
- Between SCA and SMA. SCA communicates to SMA the silo handle and the silo recipe.
- USS to SCA and SMA. USS presents a unified interface to the rest of the SILO framework. This interface allows other SILO components to query USS about its contents.
- Between CSS and SMA. CSS interface serves to enable search and selection of the best control strategy based on application requirements and active policies.

C. Ongoing Work

We are continuing to develop this framework while implementing the first prototype. We envisage that individual component releases will be bundled together into SILO releases. The first one, planned for September 2007, does not include the CSS. Also, the SCA functionality supported is as follows. Application service request consists of a proposed nearly complete set of services. SCA verifies precedence constraints and interface compatibilities, and puts the services in the correct order. It may perform trivial fill-ins of missing methods (e.g. shim methods that help resolve inter-service interface incompatibilities). The SMA is able to dynamically construct and instantiate a silo based on a recipe from SCA. The USS is able to integrate the initial ontology storage (likely a collection of XML files) with method implementation database (using BerkeleyDB format).

We have begun defining the ontology of the SILO framework, which describes individual services, methods and ordering constraints between them. The constraint specification mechanism is flexible so that a variety of rules can be easily specified. Constraints are represented by a class structure (Figure 3), with `PrimitiveConstraint` and `FunctionalConstraint`

as first-level children of the abstract `Constraint` class. `PrimitiveConstraint` is further subdivided into simple and compound constraints. A compound constraint allows the use of logical connectives in constraint specification. Simple constraints can be either a

- `FunctionalDependency` - a SILO service or method may depend on another specific function supplied by another service or method.
- `PrimitiveDependency` - a SILO primitive may need another specific primitive to be present anywhere in a silo.
- `OrderingRestriction` - an ordering restriction such as requiring that if another primitive is present in a silo, it can only be located above or below this primitive.

A `FunctionalConstraint` is currently a placeholder for future extensions.

Subsequent releases will include enhancements to the SCA so that application requests can be defined in more generic terms without spelling out the exact set of services. The SCA will be capable of constructing a silo recipe based on the request or, alternatively, notifying the application that its request cannot be satisfied. Additionally, it will allow for policy integration. Similarly, future releases of SMA will have the additional capabilities of selecting one control strategy from a set of hard-coded strategies to satisfy a simple optimization objective. Eventually, the SMA will be able to select and utilize a flexible closed-loop control strategy selected from CSS, in order to optimize the behavior of the silo based on the current policy. The USS will support policy storage and query in a future release. Finally, after the planned updates to the SCA and SMA, the CSS will be included in the release.

IV. CONCLUSION

We have advanced a new architecture for software implementation of networking protocols. This architecutre, called SILO, is based on fine-grain composable services, dynamic composition of network stacks with per-flow state, and is specifically designed to allow easy representation, implementation and optimization of cross-layer interactions. We hope to make example implementations showcasing the strength of the SILO architecture generally available in the near future. We invite the cross-layer design community to utilize the SILO framework to implement specific control strategies, and look forward to reporting further enhancements to the architecture based on their feedback.

REFERENCES

[1] National Science Foundation, "Future internet design website," http://www.nets-find.net/.
[2] "Services integration, control, and optimization," http://www.net-silos.net/.
[3] F. C. Commission, "Notice of inquiry and notice of proposed rulemaking," Public Notice FCC 03-289, November 28, 2003, seeks comments on "Interference Temperature" Model.
[4] V. Srivastava and M. Motani, "Cross-layer design: A survey and the road ahead," *IEEE Communications Magazine*, vol. 43, no. 12, pp. 112–119, December 2005.
[5] J. Wang, L. Li, S. H. Low, and J. C. Doyle, "Cross-layer optimization in TCP/IP networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 3, pp. 582–595, June 2005.
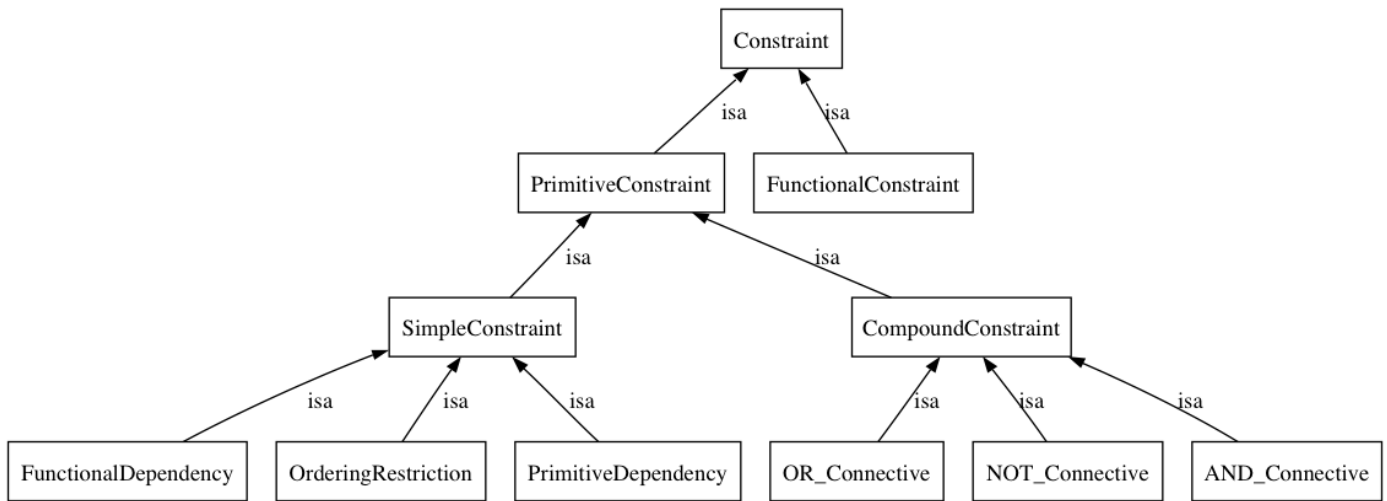
Fig. 3.   Representation of constraints in the SILO Ontology

[6] R. Winter, J. H. Schiller, N. Nikaein, and C. Bonnet, "CrossTalk: Cross-layer decision support based on global knowledge," *IEEE Communications Magazine*, vol. 44, no. 1, pp. 93–99, January 2006.

[7] V. T. Raisinghani and S. Iyer, "Cross-layer feedback architecture for mobile device protocol stacks," *IEEE Communications Magazine*, vol. 44, no. 1, pp. 85–92, January 2006.

[8] R. Madan, S. Cui, S. Lall, and A. Goldsmith, "Cross-layer design for lifetime maximization in interference-limited wireless sensor networks," in *Proceedings of the 2004 IEEE INFOCOM Conference*, Mar 2005.

[9] Y. Wu, P. A. Chou, Q. Zhang, K. Jain, W. Zhu, and S.-Y. Kung, "Network planning in wireless ad hoc networks: A cross-layer approach," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 1, pp. 136–150, January 2005.

[10] R. Braden, T. Faber, and M. Handley, "From protocol stack to protocol heap – role-based architecture," *ACM Computer Communication Review*, vol. 33, no. 1, pp. 17–22, January 2003.

[11] D. D. Clark *et al.*, "Newarch project: Future-generation internet architecture," http://www.isi.edu/newarch/.

[12] I. Baldine, G. N. Rouskas, H. G. Perros, and D. Stevenson, "JumpStart: A just-in-time signaling architecture for WDM burst-switched networks," *IEEE Communications Magazine*, vol. 40, no. 2, pp. 82–89, February 2002.

[13] S. O'Malley and L. Peterson, "A dynamic network architecture," *ACM Transactions on Computer Systems*, vol. 10, no. 2, pp. 110–143, May 1992.

[14] N. T. Bhatti and R. D. Schlichting, "A system for constructing configurable high-level protocols," in *Proceedings of the 1995 ACM SIGCOMM Conference*, Cambridge, MA, August 1995, pp. 138–150.

[15] N. Hutchinson and L. Peterson, "The x-kernel: An architecture forimplementing network protoccols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64–76, 1991.

[16] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The silo architecture for services integration, control, and optimization for the future internet," in *Proceedings of IEEE ICC, Glasgow, Scotland*, 2007, (to appear).

[17] F. P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975, 1995.