Architectural Support for Internet Evolution and Innovation

Rudra Dutta[†], Ilia Baldine[‡], Anjing Wang[†], Mohan Iyer[†], George N. Rouskas[†]
[†]Computer Science Department, NCSU, and [‡]Renaissance Computing Institute, UNC-CH

***Abstract*—The architecture of the modern Internet encompasses a large number of principles, concepts and assumptions, that have evolved over several decades. In this paper, we argue that while the current architecture houses an effective design, it is not itself effective in enabling evolution. To achieve the latter goal, we introduce the SILO architecture, a *meta-design* framework within which the system design can change and evolve. We list some insights about architectural research that guided our work, and also state the goals we formulated for our architecture. We then describe that actual architecture itself, connecting it with relevant prior and current research work. We show how the promise of enabling change is validated by showing our recent work on supporting virtualization as well as cross-layer research in optics using SILO. We present an early case study on the usefulness of SILO in lowering the barrier to contribution and innovation in network protocols, and we conclude with a list of open research problems.**

I. Toward a new Internet Architecture

In 1972, Robert Metcalfe was famously able to capture the essence of networking with the phrase "Networking is inter-process communication." Nevertheless, describing the *architecture* that enables this communication to take place is by no means an easy task. The architecture of something as complex as the modern Internet encompasses a large number of principles, concepts and assumptions, which necessarily bear periodic re-visiting and re-evaluation in order to assess how well they have withstood the test of time. Such attempts have been made periodically in the past, but really started coming in force in the early 2000's, with programs like DARPA NewArch [27], NSF FIND [11], EU FIRE [13] and China's CNGI all addressing the question of the "new" Internet architecture.

The degree to which the Internet continues to permeate modern life with hundreds of new uses and applications, adapted to various networking technologies (from optical, to mobile wireless, to satellite), raises concerns with the longevity of the Internet architecture. The original simple file transfer protocols and UUNET gave way to e-mail and WWW, which by now are becoming eclipsed by streaming media, compute grids and clouds, instant messaging and peer-to-peer applications. Every step in this evolution raises the prospect of re-evaluation of the fundamental principles and assumptions underlying the Internet architecture. So far this architecture has managed to survive and adapt to the changing requirements and technologies while providing immense opportunities for innovation and growth. On the one hand, such adaptability seems to confirm that some of the original principles have truly been prescient to allow the architecture to survive for over 30 years. On the other, it begs the question if the survival of the architecture is in fact being ensured by the reluctance to question those principles, cemented by shoehorning novel applications and technologies into the existing architecture without giving thought to its suitability.

This work was supported in part by the NSF under grants CNS-0626103 and CNS-0732330.

Such contradiction will not be easily resolved, nor should it be. A dramatic shift to a new architecture should only be possible for the most compelling of reasons, and so, the existence of this contradiction creates the ultimate "tussle" [7] for the networking researcher community. This tussle pits the investment in time, technologies and capital made in the existing architecture against the possibilities which open up by adapting the new architecture in allowing for creation of novel and improved services over the Internet as well as opening new areas of research and discovery. It also allows us to continually refine the definition of the Internet architecture and separate and re-examine the various aspects of it. A sampling of the projects funded through the NSF FIND program, targeted at re-examining the architecture of the Internet, illustrates the point: there are projects concerned with naming [19], [16], routing [3], [17], protocol architectures [9], which examine these and other aspects from perspectives of security, management [24], environmental impact [1] and economics [16]. Another dimension is presented by the range of technologies allowing devices to communicate: wireless, cellular, optical [4] and adaptations of the Internet architecture to them.

This diversity of points of view makes it difficult to see clearly the fundamental elements of the architecture and their influence over each other. Most importantly for the researcher interested in architecture, this makes it nearly impossible to answer concisely the question of what the Internet architecture actually is, or even what concerns are encompassed by the term "Internet architecture". What things should be considered part of the architecture of a complex system, and what should be considered specific design decisions, comparatively more mutable? This further fuels the "to change or not to change" tussle we alluded to above.

One way to make progress in the tussle appears to be in creating modifications in the current architecture, which enable new functionality or services not possible today, while limiting the impact on the rest of the architecture, in essence evolving the architecture while preserving backward compatibility [8]. This approach has the additional merit of paying heed to the concerns expressed by some in the research community regarding the potential of clean-slate approaches to be far divorced from reality, with no reasonable chance of translating to deployment [8]. Such concerns have been epitomized by the phrase “Clean-slate is not blue-sky.”

In our project named SILO (**S**ervices **I**ntegration, contro**L** and **O**ptimization) we started, in a way, by following this approach. We did not attempt to rethink the Internet as a whole. Instead, we identified one particular aspect of the Internet architecture that, in our opinion, created a significant barrier to its future development. We proposed a way to modify this aspect in a way that is least impactful on the rest of the architecture and demonstrated the use of this new architecture via a prototype implementation and case studies.

Somewhat to our surprise, however, what emerged from our research was a new understanding regarding the problem at hand. Specifically, we came to recognize that the important problem is *not* to obtain a particular design or arrangement of specific features, but rather, to obtain a *meta-design* that explicitly allows for future change. With a system like the Internet, the goal is not to design the “next” system, or even the “best next” system, but rather a system that can sustain continuing change and innovation.

This principle, which we call *designing for change*, became fundamental to our project. In the process, we have come to develop our own answer to the question of what architecture actually is: *it is precisely the characteristics of the system that does not change itself, but provides a framework within which the system design can change and evolve.* The current architecture houses an effective design, but is not itself effective in enabling evolution. Our challenge has been to articulate the necessary minimum characteristics of an architecture that will be successful in doing so.

This paper is organized as follows. In Section II we discuss some pertinent problems with the current Internet architecture, motivating the SILO architecture we describe in Section III. We describe the SILO software prototype and an early case study in Section IV, and in Section V we report our ongoing research. We discuss prior related research projects in Section VI, and we conclude with a list of open problems in Section VII.

II. The Problems with the Current Architecture

As witnessed by the breadth of scope of the various FIND-related projects, ideas on how to improve the current Internet cover a wide range of approaches. These ideas are frequently driven by the difficulties that arise in attempting to integrate some new functionality into the Internet architecture. In the SILO project we began with a single basic observation: that protocol research has stagnated despite the clear need to improve data transfers over the new high-speed optical and wireless technologies and has been reduced to designing variants of TCP. This stagnation points to a weak point in the original Internet architecture, that somehow has disallowed the evolution and development of this aspect of the architecture. The cause of this stagnation, in our opinion, lies in:

1) the difficult barrier to entry in implementing, deploying and experimenting with new data transfer protocols in the TCP/IP stack, except for user-space;
2) perhaps more importantly, the lack of clear separation between policies and mechanisms in TCP/IP design (e.g., window-based flow control vs. the various ways in which the window size can respond to changes in the network environment) preventing the reuse of various components; and
3) the lack of a pre-defined agreed-upon way for protocols at different layers to share information with each other for the purpose of optimizing their behavior for different optimization criteria (of the user, the system, the network or a combination thereof).

This lack of flexibility, for example, prevents applications that would ideally prefer to use some parts of the functionality of the TCP/IP stack, but not others, in transmitting data. For instance, being able to request a specific mode of flow control (or totally remove it), while still retaining in-order delivery of TCP may be desirable. However the current implementations make no allowance for such flexibility.

The lack of explicit and well-defined cross-layer interaction mechanisms have resulted in more subtle problems: these interactions are implemented anyway, but in an *ad-hoc* fashion, resulting in a monolithic implementation where TCP and IP codes are intermingled to achieve higher efficiencies. As a result, clarity and reusability are sacrificed, with the unintended consequence of making each further unit of development and research more difficult. In a way, this is a self-reinforcing process, with each modification making further modifications of the whole structure more difficult, ensuring that in the long run TCP and its modifications remain the dominant mode of data transport. When it comes to adding new cross-layer interactions, particularly with the physical layer, the problem is even more pronounced, as is indicated by the fact that no standard cross layer solution has been widely adopted, for example, to assist TCP over wireless by taking advantage of physical layer conditions, despite a clear need and more than a decade of extensive research results in this area.

Finally, the proliferation of half-layer solutions, including MPLS and IPSec, point at another aspect of this problem: that the protocol layers as we know them (TCP/IP or OSI stacks) may no longer be relevant but are merely markers for some vague functional boundaries within the architecture. These half-layer solutions clearly address important needs, yet the original Internet architecture had no way of describing their place within a data flow. Furthermore, the “layering as optimization decomposition” framework [6], [18], a systematic and formal approach in defining the network protocols within each layer, highlights another shortcoming of fixed protocol stacks, namely, that the optimal layering structure depends strongly on both application requirements and the underlying network environment, and can be quite different than the existing TCP/IP stack.

In essence, the TCP/IP stack has become ossified, preventing further development and evolution of protocols within its framework [2]. Even when new transport service abstractions have been developed to address emerging application requirements (e.g., reliable datagrams [21] or structured streams [12]) they cannot be accommodated by, or co-exist within, the current architecture. Applications written today that require data services not accommodated by the TCP/UDP dichotomy are left to take one of several paths:

- implement their own UDP-based data transfer mechanisms without the ability to reuse the elements of the existing architecture or to take advantage of kernel-space optimizations in buffer management;
- adapt an existing TCP implementation to new situations, e.g., new media like wireless, or large bandwidth-delay product optical networks; or
- abandon the old and "roll their own" implementation, as has occurred in sensor networks, where TCP/IP has been supplanted by a simpler implementation suitable for the low-cost/low-power sensor hardware.

These approaches point to a significant risk of fracturing the future protocol development into their applicable domains (wireless, optical, sensor, mobile). In turn, these networks are then forced to communicate with each other or "the (canonical) Internet" via proxies or gateways. In a way, this is a "balkanization" of the network as apprehended in [20]. From our perspective, such an outcome is undesirable and presents the fundamental challenge to the concept of IP as a simple convergence point (often referred to as IP being the "narrow waist" of the "hourglass" protocol stack), which stands as one of the fundamental assumptions of the current Internet architecture.

Based on the identified shortcomings of the current Internet architecture, it became clear that what is needed, is a new architectural framework that will address these deficiencies and allow for a continuing evolution of protocols and their adaptation to new uses and media types.

III. SILO Architecture: (Meta-)Design For Change

A. Design Principles

As a starting point, we adopted a view that layering of protocol modules within a data flow was a desirable feature that has withstood the test of time, as it made data encapsulation easy, and simplified buffer management. The layer *boundaries*, on the other hand, do not have to be in specific places; to our minds, this caused entrenchment of existing protocols, and is one of the causes of the identified ossification of the Internet architecture. Based on this initial assumption, the desirable characteristics of a new architecture to *generalize protocol layering* started to emerge: that

1) each data flow should have its own arrangement of layered modules, such that the application or the system could create such arrangements based on application needs and underlying physical layer attributes;
2) the constituent modules should be small and reusable to assist in the evolution by providing ready-made partial solutions; and
3) the modules should be able to communicate with each other over a well defined set of interfaces.

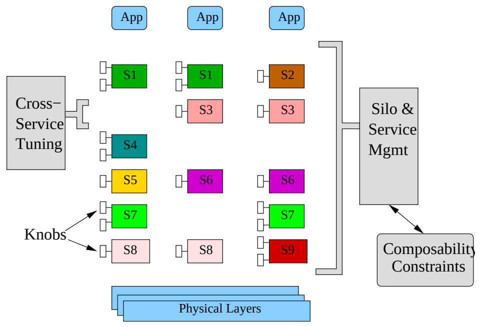

Fig. 1. Generalization of layering in SILO

These three principles became the basis of the SILO architecture. We refer to each individual layered arrangement serving a single data flow as a *silo* and we refer to individual layers within a silo as *services* and *methods* (more on this later). Figure 1 depicts three application-specific silos, each consisting of a vertical arrangement of services (each service is represented as a rectangle labeled S1,S2, . . .); the other elements shown in the figure are explained shortly.

B. Support for Service Deployment and Composition

Several other architectural elements developed from these basic principles. As the system evolves, new reusable modules (services and methods) may be implemented and deployed to fulfill the changing requirements of various applications, while allowing the reuse of existing ones. One may think of a downloadable driver/plug-in model as being appropriate in this context – new services and methods may be added to the system via one or more trusted remote repositories.

Since not all modules can be assumed to be able to co-exist with each other in the same silo, it is necessary to keep track of module compatibility. We refer to these as *composability constraints*. These constraints may be specified by the creators of the modules when the modules are made available, or they may be automatically deduced based on the description of module functionality. We envision that knowledge distilled from deployment experience of network operators, collectively, can also be stored here. The number of such constraints can be expected to be large and grow with time. This pointed out to us the need for automated silo composition, which can be accomplished by one or more algorithms based on the application specification. This automated construction of silos became a crucial part of the architecture.

C. Support for Cross-Layer Interactions and Control

From the perspective of cross-layer interactions, it also became desirable to not simply allow modules to communicate with each other outside the data flow, but to allow for an external entity to access module states for purposes of optimizing the behavior of individual silos and/or the system as a whole. We refer to this function as *cross-service tuning*, and it is accomplished by querying individual modules via *gauges* and

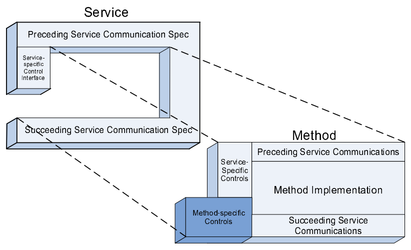
Service
Preceding Service Communication Spec
Service-specific Control Interface
Succeeding Service Communication Spec
Method
Service-Specific Controls
Preceding Service Communications
Method Implementation
Method-specific Controls
Succeeding Service Communications

Fig. 2. Services vs. methods

modifying their state via *knobs*. Both gauges and knobs must be well-defined and exposed as part of the module interface. The important aspect of this approach is that the optimization algorithm can be pluggable, just like the modules within a silo, allowing for easy re-targeting of optimization objectives by a substitution of the optimization algorithm. This addresses the previously identified deficiency of the current architecture, where policies and methods in protocol implementations are frequently mixed together, not allowing for evolution of one without affecting the other.

D. Services vs. Methods

The service/method dichotomy introduced earlier becomes important from the point of view of system scalability. Borrowing from object-oriented programming concepts, what we call services are generic functions (such as *encryption*, *header checksum*, or *flow control*), while methods are specific implementations of services. Thus, in some sense, methods are polymorphic on services. This relationship allows for aggregation of some composability constraints based on generic service definitions, which necessarily propagate to the methods implementing this service, thus making the job of the developer, as well as of the composition algorithm, substantially easier.

Each service is described from the point of view of its functionality, its generic interfaces (to the services immediately above and below it in a silo), as well as the knobs and gauges it exposes. These, as well as composability constraints are inherited by methods implementing this service. The methods implementing services must conform to this interface definition, however they may be allowed to expose method-specific knobs and gauges, as seen in Figure 2.

E. SILO Functional Blocks

Another way to look at the SILO architecture is from the point of view of functional blocks. This architectural view also served as the basis of the prototype implementation described in the next section. At the heart of the system is the *Silo Management Agent* (SMA), which is responsible for maintaining the state of individual data flows and associated silos. The application communicates with this entity via a standard API passing data, as well as silo meta-information, including the description of desired services. The SMA is assisted by a *Silo Composition Agent* (SCA) which contains algorithms responsible for assembling silos based on application requests and known composability constraints between services and

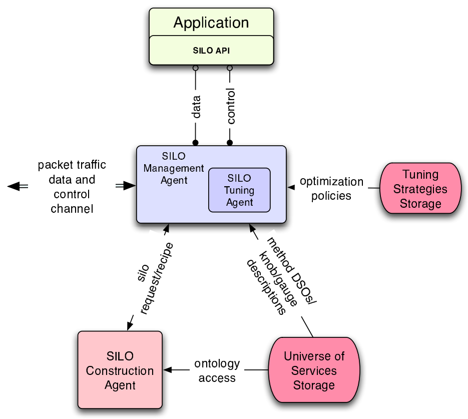

Fig. 3. SILO functional architecture.

methods. All service descriptions, method implementations, constraints and interface definitions are stored in the *Universe of Services Storage* (USS) module. Both the SMA and SCA consult this module in the course of their operations. Finally there is a separate *Tuning Strategies Storage* module which houses various algorithms capable of optimizing the behavior of individual silos (or their collections) for specific objectives. This optimization is achieved by monitoring gauges and manipulating knobs that methods within instantiated silos expose. This functional view of architecture is shown in Figure 3.

A typical sequence of operations within the SILO architecture consists of the following: (**a**) an application requests a new silo from the SMA by specifying, possibly in some vague form, its communications preferences; (**b**) the SMA passes the request to the SCA, which invokes one of the composition algorithms and, when successful, passes back to the SMA a silo *recipe*, which explicitly describes the ordered list of services that will make up the new silo; (**c**) the SMA instantiates a new silo by loading the methods described in the recipe and instantiating a state for the new data flow, and it passes a silo handle back to the application; (**d**) the application begins communicating while an appropriate optimization algorithm is applied to the silo via the tuning agent.

It is quite clear that, while this architecture offers a great deal of flexibility in arranging communication services, this flexibility comes at some cost. One important problem that needs to be addressed is that of an agreement on the silo structure between two communicating systems, noting that the silos need not be identical to accomplish communications tasks (monitoring or accounting services are a trivial example of services that require no strict counterpart in the far-end silo). The solution to this problem may come in one of several flavors. One approach may be an out-of-band channel, which allows two SMAs to communicate and create an agreement between them prior to applications commencing their communications. This approach may be suitable for a peer-to-peer model of communication. Another method, more suitable for a client-server model, allows for a just-in-time

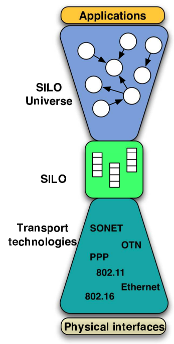

Fig. 4. The SILO hourglass.

analysis of compatibility between two silos by embedding a fingerprint of the client silo structure in the first packet that is sent out. Based on the information in that packet, the SMA can determine if the communication between a client and an already instantiated server is possible. In our work, this remains a problem still open for further investigation.

F. Support for Evolution and Innovation

Let us now address the issue of why this architecture is better suited for evolution than the current one. As we mentioned in the previous section, our mantra for this project has been "design for change", and we believe we have succeeded in accomplishing this goal. The architecture we have described does not mandate that any specific services be defined or methods implemented. It does not dictate that the services be arranged in a specific fashion, and leaves a great deal of freedom to the implementors of services and methods. What it does define is a generic structure in which these services can coexist to help applications fulfill their communications needs, which can vary depending on the type of application, the system it is running on, and the underlying networking technologies available to it. Thus, as the application needs evolve along with the networking technologies, new communications paradigms can be implemented by adding new modules into the system. At the same time, all previously developed modules remain available, ensuring a smooth evolution.

The described architecture is a *meta-design* which allows its elements (the services and methods, the composition and tuning algorithms) to evolve independently, as application needs change and networking technologies evolve. Where, in the current architecture, the IP protocol forms the narrow waist of the hourglass (i.e., the fundamental invariant), in the SILO architecture the convergence point is the conformance to the meta-design, *not* a protocol (which is part of the design itself). Rather than a protocol which all else must be built on and under, SILO offers the silo abstraction as an invariant, the narrow waist in the hourglass of this meta-design (Figure 4).

IV. Prototype and Case Study

We have developed a working prototype implementation which serves as proof-of-concept demonstration of the feasibility of the SILO framework. This prototype, which is publicly available from the project website [26], is implemented in portable C++ and Python as a collection of user-space processes running on a recent version of Linux (although the prototype carries no explicit dependencies on the Linux kernel). Individual services as well as tuning algorithms are implemented as dynamically loadable libraries (DLLs or DSOs). The general structure of the prototype follows the functional architecture in Figure 3.

One of the important challenges we encountered when addressing the problem of dynamically composable silos was related to the problem of representing the relationships (composability constraints) between the various services and modules. Essentially, this is a problem of knowledge representation. These composability constraints take the form of statements similar to "Service A requires Service B" or "Service A cannot coexist with Service B", which can be modulated by additional specifications such as 'above' or 'below' or 'immediately above' or 'immediately below'. Additionally, we also needed to deal with the problem of specifying application preferences or requests for silos, which can be described as *application-specific* composability constraints. To address this problem we turned to *ontologies*, specifically, ontology tools developed by the semantic web community.

We adopted RDF (Resource Description Framework) as the basis for ontology representation in the SILO framework. Relying on RDF-XML syntax we were able to create a schema defining various possible relationships between the elements of the SILO architecture, namely, services and methods. These relationships include the aforementioned composability constraints, which can be combined into complex expressions using conjunction, disjunction and negation. Using this schema, we have defined a sample ontology for the services we implemented in the current prototype. The application constraints/requests are expressed using the same schema. This uniform approach to describing both application requests as well as the SILO ontology is advantageous in that a request, issued by the application and expressed in RDF-XML, can be merged into the SILO ontology to create a new ontology with two sets of constraints – the original SILO constraints and those expressed by the application, on which the composition algorithm then operates. Using existing semantic web tools, we have implemented several composition algorithms [30] that operate on these ontologies and create silo recipes, from which silos can be instantiated.

Our RDF schema also allows us to express other knowledge, such as the functions of services (an example of a service function could be "congestion control" or "error correction" or "reliable delivery"), as well as their data effects (examples include cloning of a buffer, splitting or combining of buffers, transformation, and finally null, which implies no data effect). These are intended to aid composition algorithms in deciding the set of services to be included in a silo, when an application is unable to provide precise specifications in the request. Using this additional information in the composition algorithm is an active area of our research.

A. Case study

Does SILO work? Is there any evidence to show that it lowers the barrier to continuing innovation, its stated goal? Of course, the answer to such a question would take a long and diverse experimental effort, and to be convincing, would have to come at least partly from actual developer communities after at least partial deployment.

However, we have been able to conduct a small case study which has yielded encouraging results. In the Fall of 2008, we made a simplified version of the SILO codebase available to graduate students taking the introductory computer networks course at North Carolina State University. Students are encouraged to take this course as a prerequisite to advanced graduate courses on networking topics, and most students taking the course have no prior networking courses, or even a single undergraduate course on general networking topics. Students are required to undertake a small individual project as one of the deliverables, which typically involves conducting a literature research on a focused topic and synthesizing the results in a report. In this instance, students were offered the option to try their hand at programming a small networking protocol as a SILO service as an alternative project. Nine out of the approximately fifty students in the class chose to do so. All but one of these students had not coded any networking software previously.

To our satisfaction, all nine produced code to perform non-trivial services, and the code not only worked, but it was possible to compose the services into a stack and interoperate them, although there was no communication or effort among the students to preserve interoperability during the semester. In one case, the code required reworking by the teaching assistant, because the student concerned had (against instructions) modified the SILO codebase distribution. The services coded by the students were implementations of ARQ, error control, adaptive compression, rate control, and bit stuffing. Testing services such as bit error simulators were also coded, and two students attempted to investigate source routing and label switching, going into the territory of SILO services over multiple hops, which are as yet comparatively unformed and malleable in our architectural vision.

While this is only the veriest beginnings of trying to validate SILO, we feel that this case study at least demonstrates that the barrier to entry into programming networking services has been lowered, in that the path from conceptual understanding of a networking protocol function to attaining the ability to produce useful code for the same is dramatically shorter. In future similar case studies, we hope to study the reaction to such beginning programmers to the tuning agent and ontology capabilities. And as always, we continue to invite the community to download the SILO code from our project website [26], try using it, and send us news about their positive and negative experiences.

V. Ongoing SILO Research: Virtualization and Cross-Layer Experimentation

The SILO architecture has the potential to provide new insight into a number of research areas and enable new and fruitful directions of investigation and experimentation. In this section we address two topics that are the subject of active research and development within the SILO project.

A. Virtualization

Network virtualization efforts have attracted growing interest recently. Virtualization allows the same resource to be shared by different users, with independent and possibly different views. In network virtualization, a substrate network is shared by a virtualization system or agent which provides interfaces to different clients.

Testbeds such as PlanetLab have demonstrated network virtualization, and other efforts such as Emulab have allowed investigation of a virtualized network through an emulated environment. The Global Environment for Networking Innovation (GENI) has identified virtualization as a basic design strategy for the envisioned GENI facility to enable experimentation support for diverse research projects. More importantly, it has been conjectured that virtualization itself could become an essential part of the future Internet architecture. The FIND portfolio also contains projects on virtualization [29], [2]. Nevertheless, network virtualization is comparatively less mature than OS virtualization, being a significantly more recent field. We can expect there will be substantial ongoing work in increasing the isolation, generality and applicability of this area in the short- to mid-term. As such, it is an important area for any new architecture to consider.

Accordingly, we now describe our approach to realize virtualization within the SILO framework in order to achieve greater reusability. We go on to conjecture that such a realization might allow the concept of network virtualization to be generalized.

1) Virtualization as Service: So far, network virtualization has been strongly coupled to the platform and hardware of the substrate. Logically, however, network virtualization consists of many coordinated individual virtualization capabilities, distributed over networking elements, that share the common functionality of maintaining resource partitions and enforcing them. In keeping with the SILO vision, we can view these functions as separate and composable services.

The most basic of these services is that of *splitting* and *merging* flows; these services must obviously be paired. This is no more than the ability to mux/demux multiple contexts. Note that this service is a highly reusable one, and can be expected to be useful in diverse scenarios whenever there is aggregation/dis-aggregation of flows, such as mapping to/from physical interfaces, or at intermediate nodes for equivalence classes on a priority or other basis, or label stacking.

In the networking context, virtualization is usually interpreted as implying two capabilities beyond simple sharing. The first is *isolation:* each user should be unaware and unaffected by the presence of other users, and should feel that it operates

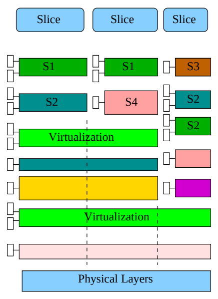

Fig. 5. Generalizing virtualization via successive virtualization services

on a dedicated physical network. This is sometimes also called "slicing." This capability can be broken down into two services: (i) slice maintenance, which keeps track of the various slices and the resources used by them, and (ii) access control, which monitors and regulates the resource usage of each slice, and decides whether any new slices requested can be allowed to be created or not; for example, rate control such as leaky bucket would be an access control function.

The second capability is *diversity:* each user should be able to use the substrate in any manner in which it can be used, rather than being restricted to use a single type of service (even if strictly timeshared). This is akin to the ability to run different operating systems on different virtual machines. In SILO, this capability is natively supported, through the composable nature of the stack (silo). Not only do different silos naturally contain different sets of services, but the composability constraints provide a way to indicate what set of upper services may be chosen by different slices when building on a particular virtualized substrate.

The definition of a standard set of services for virtualization means that every realization of this service (for different substrates) would implement the functional interfaces specified by the service itself, thus any user of the virtualization agent would always be able to depend on these standard interfaces. Articulating this basic interface is part of our goal in this regard. For example, consider the case of virtualizing an 802.11 access point with the use of multiple SSIDs; the interface must allow specification of the share of each slice. However, since the different slices can use different varieties of 802.11 with different speeds, the sharing must really be specified in terms of time shares of the wireless medium, which is the appropriate sharing basis in this context.

2) Generalizing Virtualization: Following the principle that a virtual slice of a network should be perceived just like the network itself by the user, we are led to the scenario that a slice of a network may be further virtualized. A provider who obtains a virtual slice and then supports different isolated customers may desire this scenario. The current virtualization approaches do not generalize gracefully to this possibility, because they depend on customized interfaces to a unique underlying hardware. If virtualization is expressed as a set of services, however, it should be possible to design the services so that such generalization is possible simply by re-using the services, as illustrated in Figure 5.

Clearly, there are open issues and challenges with this approach. One obvious question is whether the multiple levels of virtualization should be mediated by a single SMA or whether the SMA itself should run within a virtualization, and thus multiple copies of SMA should run on the multiple stacks. Either approach is possible to proceed with, but we believe the former is the correct choice and is the one we are in the process of implementing. In OS virtualization, the virtualization agent is itself a program and requires some level of abstraction to run, though it maps virtual machine requests to the physical machine down to a high level of detail. Successive levels of virtualization with agents at all levels being supported by the same lower level kernel are difficult to conceive. However, networking virtualization agents do not seek to virtualize the OS which supports them. As such, the kernel support they require can be obtained through a unique SMA.

It may appear from this discussion that with per-flow silo states, there is no need to virtualize, and in fact it is possible to extend all the slices to the very bottom (as indicated with the dotted lines in Figure 5). However, the advantage of our virtualization-as-a-service approach lies precisely in state maintenance; a service which is not called upon to distinguish between multiple higher level users can afford to keep state only for a single silo, and the virtualization service encapsulates the state keeping for the various users.

3) Cross-Virtualization Optimization: Finally, it is possible to conceive of cross-layer interaction across virtualization boundaries, both in terms of composability constraints, and tuning. Returning to Figure 5, the service S1 may require the illusion of a constant bit-rate channel below it, and the virtualization below it may be providing it with this by isolation. If, however, there is some service still further down that does not obey this restriction (some form of statistical multiplexing, for example), then the illusion will fail. It must be possible to express this dependence of S1 as a constraint, which must relate services across a virtualization boundary. It appears harder to motivate the need to tune performance across boundaries, or even (as the SMA could potentially allow) across different slices. Although we have come up with some use cases, they are not persuasive. However, we recall that the same was true of the layering abstraction itself, and it is only recently that cross-layer interactions have come to be perceived as essential. We feel that cross-virtualization optimization is also an issue worth investigation, even if the motivation cannot be clearly seen now.

B. Cross-Layer Experimentation

In today's networks, the physical layer is typically considered as a black box: sequences of bits are delivered to it for transmission, without the higher layers being aware of exactly how the transmission is accomplished. This separation of concerns imposed by the layering principle has allowed the development of upper layer protocols that are independent of the physical channel characteristics, but it has now become too restrictive as it prevents other protocols or applications from taking advantage of additional functionalities that are increasingly available at the physical layer.

Specifically, in the optical domain, we are witnessing the emergence of optical layer devices that are:

1) *intelligent and self-aware*, that is, they can sense or measure their own characteristics and performance, and
2) *programmable*, that is, their behavior can be altered through software control.

The software logic defining more and more of these devices requires cross-layer interactions, hence the current strictly layered architecture cannot capture the full potential of the optical layer. For instance, the optical substrate increasingly employs various optical monitors and sensors, as well as pools of amplifiers and other impairment compensation devices. The monitoring and sensing devices are capable of measuring loss, polarization mode dispersion (PMD), or other signal impairments; based on this information, it should then be possible to use the appropriate impairment compensation to deliver the required signal quality to the application. But such a solution cannot be accomplished within the current architecture, and has to be engineered *outside of it* separately for each application and impairment type; clearly, this is not an efficient or scalable approach.

Reconfigurable optical add-drop multiplexers (ROADMs) and optical splitters with tunable fanout (for optical multicast) are two more examples of currently available devices whose behavior can be programmed according to the wishes of higher layer protocols. Looking several years into the future, one can anticipate the development of other sophisticated devices such as programmable MUX-DEMUX devices (e.g., that allow the waveband size to adjust dynamically), or even hardware structures in which the slot size can be adjustable

In the SILO architecture, all these new and diverse functionalities within (what is currently referred to as) the physical layer will typically be implemented as separate services, each with its own control interfaces (knobs) that would allow higher-level services and applications direct access to, and control of, the behavior of the optical substrate. Hence, the SILO architecture has the ability to facilitate a diverse collection of important cross-layer functions, including traffic grooming [10], impairment-aware routing [25], [31], and multi-layer network survivability [22] that have been studied extensively, as well as others that may emerge in the future.

As a first step towards realizing this vision, we have completed the design and implementation of a virtual concatenation (VCAT) and the link capacity adjustment scheme (LCAS) [15] as a set of SILO services within the software prototype. In this initial stage, the services operate over an environment that emulates SONET transport. Nevertheless, we have been successful in demonstrating cross-layer interactions by having the VCAT service increase or decrease the number of concatenated circuits in response to application requests and/or (emulated) transport network failures.

We also note that there is considerable interest within the GENI community to extend the programmability and virtualization functionality that is core to the GENI facility, all the way down to the optical layer so as to enable meaningful and transforming optical networking research. Currently, however, a clear road map on how to achieve such a "GENI-ized" optical layer has not been articulated, mainly due to the lack of interfaces that would provide GENI operations access to the functionality of the optical layer devices.

We believe that the SILO architecture would be an ideal vehicle for enabling optical-layer-aware networking within GENI, as well as enabling cross-layer research through explicit control interfaces (e.g., such as SILO knobs). Therefore, we are in the process of outlining specific strategies for incorporating the SILO concepts within the GENI architecture.

VI. Prior Related Work

One of the earliest attempts to impose orderly design rules on networking protocols is definitely the x-kernel project [14]. While SILO is similar to the x-kernel in introducing well-defined interfaces for protocol modules and organizing module interactions, it is important to recognize several major differences: (**a**) x-kernel was an OS-centric effort in implementing existing network protocols as sets of communicating processes inside the novel kernel, while SILO is an attempt to introduce a network protocol meta-design that is independent of any assumptions about the underlying OS; (**b**) x-kernel made an early attempt at streamlining some of the cross-layer communications mechanisms; SILO makes cross-layer tuning and optimization enabled by such mechanisms an explicit focus of the framework, and finally (**c**) SILO is focused on the problem of automated dynamic composition of protocol stacks based on individual application requests and module composability constraints, while the x-kernel protocols are pre-arranged statically at boot time.

Among recent clean-slate research, there are two projects whose scope extends to include the whole network stack and hence are most closely related to our own project. The first is work on the role-based architecture (RBA) [5], carried out as part of the NewArch project [27]. 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, and thus may interact in many different ways; as a result, the metadata in the packet header corresponding to different roles form a "heap," not a "stack" as in conventional layering, 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 [5], and to accommodate "middle boxes."

The second is the recursive network architecture (RNA) [28], [23] project, also funded by FIND. RNA introduces the concept of a "meta-protocol" which serves as a generic protocol layer. The meta-protocol includes a number of fundamental services, as well as configurable capabilities, and serves as a building block for creating protocol layers. Specifically, each layer of a stack is an instantiation of the same meta-protocol; however, the meta-protocol instance at a particular layer is configured based on the properties of the layers below it. The use of a single tunable meta-protocol module in RNA makes it possible to support dynamic service composition, and facilitates coordination among the layers of the stack; both are design goals of our own SILO architecture, which takes a different approach in realizing these capabilities.

VII. Conclusions and Open Problems

We have presented the SILO network architecture, a meta-design that provides a framework within which the system design can change and evolve. Our prototype software implementation realizes all key SILO concepts, demonstrating the feasibility of SILO and validating the design principles. An early case study also indicates that SILO has the potential to lower the bar in terms of implementing and experimenting with network protocols. Nevertheless, there are additional open problems associated with the SILO architecture that we plan to study, including:

- *Agreement on silo structure with remote end.* As we mentioned in Section III, the flexibility offered by the SILO architecture comes at a price: the need for an agreement between communicating applications about the structure of silos on both ends. We have already identified several solutions to this problem, however it is an interesting enough problem to continue keeping it open. Some desirable characteristics of an ideal solution are: low overhead of the agreement protocol, high degree of success in establishing agreement and security of the agreement process.
- *Stability and fairness.* The stability of today's Internet is in part guaranteed by the fact that the same carefully designed algorithms govern the behaviors of TCP flows to achieve fairness among them. As demonstrated in the literature, this stability is fragile and can be taken advantage of by non-compliant TCP implementations to achieve higher throughput rates compared to unmodified versions. SILO allows a plug-and-play approach to substituting optimization policies into protocol stacks, thus ensuring stability and fairness of the system within some pre-defined envelope of behavior becomes paramount.
- *SILO in the core and associated scalability problems.* Most of our work so far has concentrated on the edge of the network where applications construct silos to communicate with one another, and we have not yet carefully considered the structure of the networking stacks in the network core. It is clear that the SILO concept can be extended to the core, by providing value addition modules/services to individual flows or groups of flows, as long as the scalability issues are addressed.
- *Silo composition based on fuzzy application requests.* As we indicated in Section III, the problem of silo composition based on application requests remains open. One of the important areas to be studied is the ability to construct silos based on vague specifications from the application which may provide minimal information about its needs, e.g., "reliable delivery with encryption". This type of fuzzy or inexact specification requires an extended ontology of services in which some reasoning can take place. The solutions will be multiple and the system must pick the one that by some criteria optimizes overall system behavior or some other objective.

References

[1] M. Allman, V. Paxson, K. Christensen, and B. Nordman. Architectural support for selectively-connected end systems: Enabling an energy-efficient future Internet.

[2] T. Anderson *et al.* Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38(4):34–41, April 2005.

[3] B. Bhattacharjee, K. Calvert, J. Griffioen, N. Spring, and J. Sterbenz. Postmodern internetwork architecture.

[4] D. Blumenthal, J. Bowers, N. McKewon, and B. Mukherjee. Dynamic optical circuit switched (docs) networks for future large scale dynamic networking environments.

[5] R. Braden, T. Faber, and M. Handley. From protocol stack to protocol heap – role-based architecture. *ACM CCR*, 33(1):17–22, January 2003.

[6] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1):255–312, January 2007.

[7] D.D. Clark *et al.* Tussle in cyberspace: Defining tomorrow's internet. *Proc. of ACM SIGCOMM*, pp. 347-356, Pittsburgh, PA, Aug. 2002.

[8] C. Dovrolis. What would Darwin think about clean-slate architectures? *ACM Computer Communication Review*, 38(1):29–34, January 2008.

[9] D. Duchamp. Session layer management of network intermediaries.

[10] R. Dutta, A.E. Kamal, G.N. Rouskas, Eds. *Traffic Grooming in Optical Networks: Foundations, Techniques, and Frontiers*. Springer, 2008.

[11] D. Fisher. US National Science Foundation and the Future Internet Design. *ACM Computer Commun. Review*, 37(3):85–87, July 2007.

[12] B. Ford. Structured streams: A new transport abstraction. In *Proceedings of ACM SIGCOMM*, 2007.

[13] A. Gavras *et al.*. Future internet research and experimentation: The FIRE intitiative. *ACM CCR*, 37(3):89–92, July 2007.

[14] N. Hutchinson, L. Peterson. The x-kernel: An architecture for implementing network protoccols. *IEEE Trans. Softw. Eng.*, 17:64-76, 1991.

[15] International Telecommunication Union (ITU). Link capacity adjustment scheme (LCAS) for virtually concatenated signals. *ITU-T G.7042*, 2004.

[16] R. Kahn, C. Abdallah, H. Jerez, G. Heileman, and W. W. Shu. Transient network architecture.

[17] D. Krioukov, K.C. Claffy, and K. Fall. Greedy routing on hidden metric spaces as a foundation of scalable routing architectures without topology updates.

[18] X. Lin, N. B. Shroff, R. Srikant. A tutorial on cross-layer optimization in wireless networks. *IEEE JSAC*, 24(8):1452-1463, Aug. 2006.

[19] R. Morris and F. Kaashoek. User information architecture.

[20] Computer Business Review Online. ITU head foresees internet balkanization, November 2005.

[21] C. Partridge and R. Hinden. Version 2 of the reliable data protocol (RDP). RFC 1151, April 1990.

[22] M. Pickavet *et al.* Recovery in multilayer optical networks. *Journal of Lightwave technology*, 24(1):122–134, Janurary 2006.

[23] The RNA Project. RNA: recursive network architecture. http://www.isi.edu/rna/.

[24] K. Sollins, J. Wroclawski. Model-based diagnosis in the knowledge plane.

[25] J. Strand, A. L. Chiu, and R. Tkach. Issues for routing in the optical layer. *IEEE Communications*, 39:81–87, February 2001.

[26] The SILO Project Team. The SILO NSF FIND project website. http://www.net-silos.net/, 2007.

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

[28] J. Touch and V. Pingali. The RNA metaprotocol. In *Proceedings of the 2008 IEEE ICCCN Conference*, St. Thomas, USVI, August 2008.

[29] J. Turner, P. Crowley, S. Gorinsky, and J. Lockwood. An architecture for a diversified Internet.

[30] M. Vellala, A. Wang, G. N. Rouskas, R. Dutta, I. Baldine, and D. Stevenson. A composition algorithm for the SILO cross-layer optimization service architecture. In *Proceedings of ANTS*, December 2007.

[31] Y. Xin and G. N. Rouskas. Multicast routing under optical layer constraints. *Proc. IEEE INFOCOM 2004*, pp. 2731-2742, March 2004.