Architectural Support for Internet Evolution and Innovation (Invited Paper)

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 introduce the SILO architecture, a *meta-design* framework within which the system design can change and evolve. We describe the actual architecture itself, as well as the "proof-of-concept" software prototype we have built.**

I. Toward a new Internet Architecture

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 [4], NSF FIND [1], EU FIRE [2] and China's CNGI all addressing the question of the "new" Internet architecture. So far this architecture has managed to survive and adapt to the changing requirements and technologies while providing opportunities for innovation and growth. On the one hand, such adaptability seems to conŢrm that some of the original principles have allowed 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. 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" for the networking researcher community.

The diversity of points of view within the community makes it difŢcult to see clearly the fundamental elements of the architecture and their inţ uence 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 speciŢc design decisions, comparatively more mutable? In our SILO project [3] we came to recognize that the important problem is *not* to obtain a particular design or arrangement of speciŢc features, but rather, to obtain a *meta-design* that explicitly allows for future change. 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 characteristics of an architecture that will be successful in doing so.

We describe the SILO architecture and software prototype in Sections II and III, and we conclude in Section IV.

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

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

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. 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 difŢcult 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 ţ ow control vs. the various ways in which the window size may change) preventing the reuse of various components; and (3) the lack of a pre-deŢned 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. Based on this observation, it became clear that what is needed is a new architectural framework that will address these deŢciencies and allow for a continuing evolution of protocols and their adaptation to new uses and media types.

A. Design Principles

As a starting point, we adopted a view that layering of protocol modules is a desirable feature that has withstood the test of time, as it made data encapsulation easy, and simpliŢed buffer management. The layer *boundaries*, on the other hand, do not have to be in speciŢc places; to our minds, this caused entrenchment of existing protocols, and is one of the causes of the identiŢed ossiŢcation 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 ţ ow should have its own arrangement of layered modules, such that the application or the system could create such arrangements based on application needs

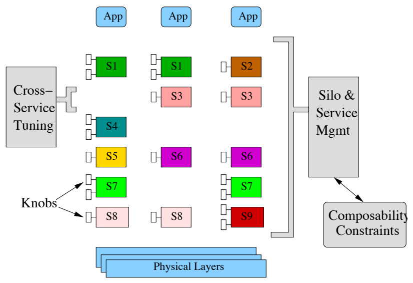

Fig. 1. Generalization of layering in SILO

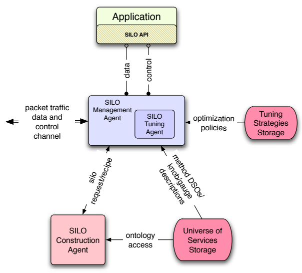

Fig. 2. SILO functional architecture.

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 deŢned set of interfaces.

These three principles became the basis of the SILO architecture. We refer to each individual layered arrangement serving a single data ţow as a *silo* and we refer to individual layers within a silo as *services* and *methods* (i.e., speciŢc implementations of services). Figure 1 depicts three application-speciŢc 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 Ţgure 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 fulŢll the changing requirements of various applications, while allowing the reuse of existing ones.

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 speciŢed 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 speciŢcation. 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 ţow, 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 modifying their state via *knobs*. Both gauges and knobs must be well-deŢned 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 identiŢed deŢciency 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. SILO Functional Blocks

Another way to look at the SILO architecture is from the point of view of functional blocks. At the heart of the system is the *Silo Management Agent* (SMA), which is responsible for maintaining the state of individual data ţows 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 methods. All service descriptions, method implementations, constraints and interface deŢnitions 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 for speciŢc objectives. This optimization is achieved by manipulating knobs that methods expose. This functional view is shown in Figure 2.

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 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 ţow, 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.

E. 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

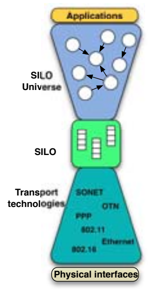

Fig. 3. The SILO hourglass.

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 speciŢc services be deŢned or methods implemented. It does not dictate that the services be arranged in a speciŢc fashion, and leaves a great deal of freedom to the implementors of services and methods. What it does deŢne is a generic structure in which these services can coexist to help applications fulŢll 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 3).

III. The SILO Software Prototype

We have developed a prototype implementation which serves as proof-of-concept demonstration of the feasibility of the SILO framework. This prototype, available from the project website [3], is implemented in portable C++ and Python as a collection of user-space processes running on a recent version of Linux. 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 2.

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 speciŢcations 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*, speciŢcally, 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 deŢning 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 deŢned 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 [5] that operate on these ontologies and create silo recipes.

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"), as well as their data effects (examples include cloning of a buffer, splitting or combining of buffers, etc). 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 speciŢcations in the request. Using this additional information in the composition algorithm is an active area of our research.

IV. Conclusions

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.

References

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

[2] A. Gavras, A. Karila, S. Fdida, M. May, and M. Potts. Future internet research and experimentation: The FIRE intitiative. *ACM Computer Communication Review*, 37(3):89–92, July 2007.

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

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

[5] 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.