This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet

Rudra Dutta, George N. Rouskas — North Carolina State University
Ilia Baldine — RENCI
Arnold Bragg, Dan Stevenson — RTI International

***Abstract*— We propose a new internetworking architecture that represents a departure from current philosophy and practice, as a contribution to the ongoing debate regarding the future Internet. Building upon our experience with the design and prototyping of the Just-in-Time protocol suite, 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. In this position paper, 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. The proposed architecture is flexible and extensible so as to foster innovation and accommodate change, it supports a unified Internet, it allows for the integration of security and management features at any point in (what is now referred to as) the networking stack, and it is positioned to take advantage of hardware-based performance-enhancing techniques.**

I. Introduction

The Internet, conceived in the era of mainframe computers and 56Kbps links, has evolved into a complex world-wide system of importance equal to that of the power grid and the transportation infrastructure. The Internet's explosive growth has been mainly due to its innate ability to incorporate easily new link and node technologies, and to accommodate seamlessly novel protocols, applications, and edge devices. The Internet's successful evolution from a free tool for academic pursuits to a key component of the global information and communications infrastructure, is a testament to the flexibility of its architecture and the fundamental principles underlying its design [10], [23]. The resulting combination of a simple, transparent network offering a basic communication service with end systems providing for a rich functionality, which lies at the foundation of the Internet architecture, has proven exceptionally adaptable to new and changing requirements.

While the network works well for common communication tasks, there have emerged communities of users with widely divergent needs for whom the current architecture is either not adequate or an overkill. For instance, the e-Science community, with its emphasis on high performance networking, is supporting the development of specialized protocols [16]. At the other end of the spectrum, the proliferation of network-enabled low-power mobile devices and sensors designed for simple, specific tasks, represents an increasing need for a rudimentary communication service; hence TinyOS does not implement the standard TCP/IP stack [19]. Continuing to ignore these divergent needs could result in "balkanization" of the Internet [22]. We feel that any new network architecture must possess a wide range of operating regimes and be able to deal gracefully with a broad spectrum of application requirements, network performance, and device capabilities, so as to ensure a unified, globally accessible Internet.

The suitability of protocol layering, as either an organizing or implementation principle for future network architectures, is also being questioned [7]. Protocols typically incorporate significant functionality, making them inflexible and difficult to evolve. New functionality is difficult to fit in the rigid structure of network stacks, resulting in the proliferation of 1/2 layer solutions (e.g., IPSec and MPLS). Also, the lack of mechanisms for cross-layer interactions has led to frequent layer violations. Moreover, protocols were not designed to take advantage of emerging hardware architectures such as the Cell [20] with one main and several synergistic processors in a single chip. A more modular design that makes it easier to selectively offload into hardware the most time consuming functions, might lead to significant performance gains.

As the limitations of the Internet architecture have become evident, the networking community has taken a variety of approaches in addressing the issues that arise. Typically, a solution for a specific problem is engineered within existing constraints; consider, for example, the research on TCP variants for high bandwidth-delay product networks [16], [18], earlier work on TCP over wireless networks [2], and efforts towards cross-layer optimization [24], [27], [28]. In other cases, addressing broader needs, such as IP address shortage or security, has led to more general solutions, which, however, may violate some of the original principles of the Internet or introduce new elements in its architecture; for instance, network address translation is not consistent with the end-to-end principle, whereas IPSec was introduced as layer 2.5.

Currently, a consensus appears to forming within the community regarding the need to think carefully about the requirements for the Internet in 15-20 years, to formulate a vision [26] for the future, and to carry out a focused research agenda to realize this vision, possibly starting with a "clean slate" [15]. Recent work in this direction has addressed a number of important issues regarding a future network architecture, including design principles [1], [13], accommodating "tussles" among different stakeholders [14], introducing a "knowledge plane" [12], overcoming the limitations of layering [7], and new routing [29] and addressing [11] architectures.

As our contribution to this debate, in this paper we propose

1-4244-0353-7/07/$25.00 ©2007 IEEE

a new network architecture that represents a departure from current philosophy and practice. Building on our experience with the design and prototyping of the Just-in-Time protocol suite [4], [5], [30], 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.

Among recent research, the work most closely related to ours is that on role-based architecture (RBA) [7], [25]. 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 [7], and to accommodate "middle boxes." We also advocate a non-layered architecture based on silos 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 [5]). 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.

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 [6], [21]. Both approaches are x-kernel specific, as they rely on the x-kernel [17] 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. In general, our proposed architecture goes beyond previous work in that (1) service silos are created on-demand, automatically, based on application requirements, local policies etc., and (2) mechanisms for adaptive control along with "cross-layer" interactions for the purpose of optimizing behavior are built into the framework.

Some of the ideas we present herein were borne out of our earlier research and development in three related areas. In the first area, we defined a novel and flexible transport layer architecture designed to support an unlimited set of new applications, services, and protocols [8], [9]. The architecture defines an open-ended set of standard transport services (i.e., as in ISO OSI layer 4) and novel transport services (to meet new requirements), and there is a fundamental distinction between a service and its implementation. E.g., "error-free delivery" is a service; "Reed-Solomon FEC" is one mechanism for implementing the service; "error detection and retransmission" is another. A subset of services comprises a virtual transport protocol instance. Industry-standard and experimental transport protocols can be emulated via an appropriate instance. The architecture encourages novel service combinations, and it supports adaptive controls in that the control plane may monitor a service's performance and invoke a more or less stringent implementation to meet performance or service quality requirements.

Supporting a rich set of services, and mapping an application's resource and QoS requirements to services has a number of performance implications. In the second area of research, we designed techniques for pushing services onto hardware, and for efficiently processing message headers that convey service requests [5], [30]. *JumpStart's* Just-in-Time (JIT) is an open protocol suite with multicast extensions which was developed under the assumption of an optical core and wireless access networks. JIT uses a novel message structure of flexible information elements (IEs). JIT IEs have a common header and separate hardware-parsable components for frequently executed functions, and software-parsable components for infrequent complex functions. The same IE format is used by all of the JIT management protocols – routing, connection management, network management, etc. This greatly simplifies hardware and software, and provides flexibility to accommodate future requirements.

In the third area of research, we deployed JIT prototypes in the ATDNet, a high-performance testbed in Washington, DC [3], [4]. Our team designed, built, and installed proof-of-concept JIT network controllers at three ATDNet sites as part of an experimental optical burst switching network overlay.

In the rest of this position paper, we articulate our view of an architecture that can supplant the current layer-based Internet architecture. While this structure is still a vision and we are likely to see changes in the details, it is our belief that the future Internet will exhibit the essential aspects of our silo architecture. At the end of the next section, we provide an example of how existing networking software can be accommodated within the silo architecture, while very different functionality can also be obtained. Our contribution consists of laying out the fundamental concepts of our architecture. In Section III we also discuss some research issues we are pursuing as part of a recent NSF FIND grant.

II. The SILO Architectural Framework

We now propose a flexible, service-oriented architecture for the future Internet. Our design was guided by the following goals for the network architecture of the future:

- **Interworking flexibility and extensibility.** Unlike the overly strict layering and tight integration of coarse-grain functions in current architectures, we advocate a framework of fine-grain building blocks along with explicit support for combining elemental functions in a highly configurable manner, so as to carry out complex communication tasks. The architecture does not limit either the number of functional building blocks or their

combinations, thus fostering experimentation and innovation and easily accommodating change.

- **Support for a scalable, unified Internet.** We are witnessing a growing gap between commodity applications running in today's Internet, on the one hand, and high-performance e-Science applications and a wide range of wireless applications, on the other hand, which are tuned to run on isolated customized networks. A fine-grained modularization of networking functions opens up interesting opportunities for low-powered devices like network-enabled sensors, which do not need the full networking stack, as well as high-performance applications which require specialized protocols. These can be provided with customized functional blocks most appropriate for their requirements and network environment, all the while staying within a consistent architectural framework.
- **Holistic network design through explicit facilitation of cross-service interactions.** Existing protocol stacks lack well-defined control interfaces for cross-layer interactions, hence the latter have to be engineered in a piecemeal and *ad-hoc* fashion. We have explicitly built in the ability for functional blocks to interact with each other so as to optimize their behavior for the specific communication task at hand. To this end, the architecture requires all functional blocks to have well-defined interfaces and provides for a control entity that is able to tune the parameters of individual blocks in order to match the application QoS requirements and improve network resource utilization.
- **Smooth integration of security features.** We feel that it is critical to incorporate simple mechanisms into the network architecture to create barriers to miscreants. The SILO architecture allows for the integration of security and management features at any point in (what is now referred to as) the networking stack. By treating security functions as easily pluggable components, our framework makes it possible to include security into the design from the ground up.
- **Support for performance-enhancing techniques.** A finer modularization of the networking stack has the potential to facilitate faster integration of hardware-accelerated solutions. Our approach is positioned to take advantage of the capabilities of multiprocessor-on-a-chip architectures such as Cell [20] which are expected to be prevalent in the future, by offloading small but computationally intensive functions to secondary CPUs so as to achieve dramatic performance improvements. This goal may be further advanced by employing a message structure similar to the one we implemented for Just-In-Time [5] which facilitates hardware/software partitioning.

A. The SILO Network Architecture

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 [7]. 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).

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.

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.

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 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.

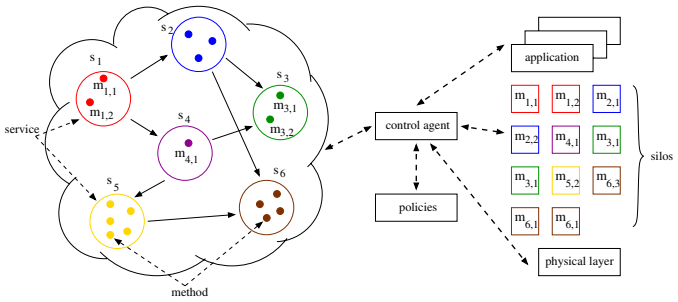

Fig. 1. Elements of the SILO architecture

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.

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.

We also point out the inherent strength of the SILO architecture which is explicitly designed with cross-service optimization in mind. Recently, it has become clear that cross-layer control is indispensable in many emerging environments, such as wireless multihop networks. 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", shows that such cross-layer interaction will span the entire gamut of layered architecture. 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. Either that, or cross-layer approaches would remain curiosities and special cases, never to affect 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.

B. SILO Examples

We present two examples showcasing some of the capabilities of the SILO architecture: we establish that it is relatively straightforward to implement the functionality of existing protocols within the framework, and we demonstrate the strengths of the framework in facilitating cross-layer interactions.

Figure 2(a) presents a silo equivalent to the current TCP/IP over Ethernet protocol stack. Each block in the silo contains the name of the corresponding service, as well as the name of the module (in italics and indented) implementing the service. Each module implements a polymorphic interface defined for the corresponding service. An example of an interface for the "segmentation" service could be: `In` – data buffers of arbitrary length; `Out` – buffer fragments not exceeding the maximum length; `Service-specific control knob` – maximum length of the fragment. The module "variable length sequential segmentation" implements the "segmentation" service in the traditional way of segmenting a stream of data. A possible alternative implementation (not suitable for TCP/IP emulation) could be one that stripes data from the input buffers across multiple outgoing fragments. While the silo shows a TCP SACK-like variant, any other TCP variation (e.g., Reno, Tahoe, BIC) can be emulated similarly by substituting the appropriate services inside the silo.

To illustrate the power of the SILO framework in facilitating "cross-layer" interactions, Figure 2(b) presents a silo for streaming video over a wireless network. An application at the top supplies raw video frames, and at the bottom, packetized video is transmitted over RF. This is similar to transmitting MPEG video over UDP in today's Internet, with one notable difference: it includes "video compression" as a service within the SILO architecture, not external to it, which allows it to interact with other services to achieve better results.

Let us explain the topmost services in this silo in more detail. Raw video is processed by the "video compression" service, implemented by a module according to the H.264 VBR standard. A service-specific control knob allows the control agent to dynamically tune the compression level as a tradeoff to video quality. The bitstream exiting the "H.264" module goes through the "video encapsulation" service which is implemented by the "raw encapsulation" module, meaning that the bitstream proceeds unmodified. It would be easy to substitute the "raw encapsulation" module with one that packages the bitstream as, say, an MPEG TS stream. The raw MPEG bitstream is segmented using the "segmentation" service; this service is implemented by the "fixed length sequential segmentation" module, which always creates segments of equal, pre-specified length, waiting for more data if necessary. The "rate control" service is responsible for ensuring that the bitstream does not exceed certain rate and burst size parameters. The desired rate is originally specified by the application. Since MPEG encoders can at times exceed specified bit rates, the control agent can dynamically tune the video compression level to achieve the highest quality under the specified rate. Finally, the "error-free delivery" service is implemented by the "Reed-Solomon FEC" module; the latter includes a module-specific control knob which allows the control agent to dynamically tune the length of the FEC.

The knobs provide the control agent with options in maintaining video quality based on levels of RF interference and/or contention. For example, in response to higher noise level, the control agent may shorten the fragment size or lengthen the FEC to increase the probability of successful packet delivery. Similarly, with higher level of contention, the control agent may increase the fragment size or opt for lower bit rate and higher video compression, depending on the bit rate and quality constraints specified by the application.

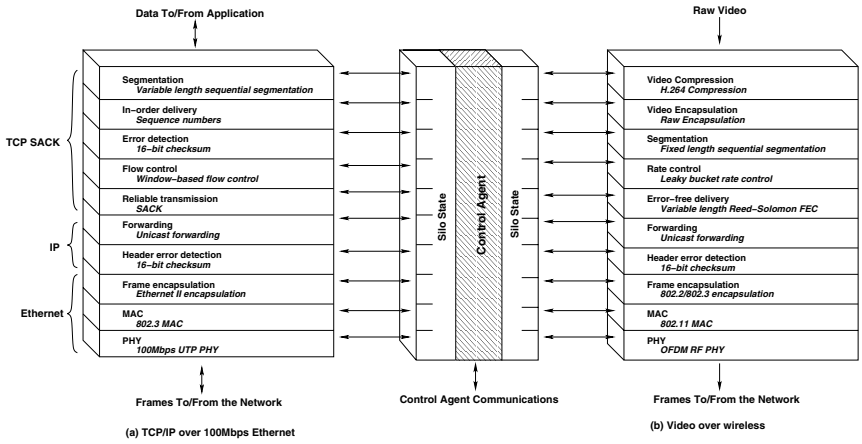

Fig. 2. SILO examples: (a) TCP/IP emulation (b) MPEG video transmission over wireless

III. Concluding Remarks

The SILO architecture is our contribution to the ongoing debate on the future of the Internet. The proposed framework of automatically assembled fine-grain functional building blocks with its emphasis on explicit control interfaces, can be the centerpiece of a new architecture which interacts harmoniously with applications and the physical layer to optimize user experience. There are many research questions that must be answered to make the SILO architecture a workable reality, such as: How would building blocks be selected and what should their granularity be? How are services defined? How are silos constructed and optimized? How are the silo optimizing choices communicated to the user, and user preferences captured? How do nodes interact in the SILO architecture and how can proper interaction between nodes be guaranteed? We are working on several of these research questions as part of a recent NSF FIND grant, and hope to report positive results in the near future. We are also working on proof-of-concept prototypes of silo-based systems, on which we shall demonstrate the approaches yielded by our research.

References

[1] B. Ahlgren *et al.* Invariants – a new design methodology for network architectures. In FDNA-04, pages 65–70, Portland, OR, 2004.

[2] H. Balakrishnan, , V. N. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *SIGCOMM 1996*, pages 256–269, August 1996.

[3] I. Baldine, A. Bragg, G. Evans, M. Pratt, M. Singhai, D. Stevenson, and R. Uppali. JumpStart deployments in ultra-high-performance optical networking testbeds. *IEEE Commun. Mag.*, 43(11):S18–S25, Nov. 2005.

[4] I. Baldine, M. Cassada, A. Bragg, G. Karmous-Edwards, and D. Stevenson. Just-in-time optical burst switching implementation in the ATDnet all-optical networking testbed. In *Globecom 2003*, pages 2777–2781.

[5] I. Baldine, G. N. Rouskas, H. G. Perros, and D. Stevenson. `JumpStart`: A just-in-time signaling architecture for WDM burst-switched networks. *IEEE Commun. Mag.*, 40(2):82–89, Feb. 2002.

[6] N. T. Bhatti and R. D. Schlichting. A system for constructing configurable high-level protocols. In *SIGCOMM 1995*, pp. 138–150, 1995.

[7] R. Braden, T. Faber, M. Handley. From protocol stack to protocol heap - role-based architecture. *Comp. Comm. Rev.*, 33(1):17-22, Jan. 2003.

[8] A. Bragg, I. Baldine, and D. Stevenson. A transport layer architectural framework for optical burst switched networks. In *WOBS 2003*.

[9] A. Bragg, S. Bryant, B. Hurst, and S. Thorpe. Transport protocols for optical burst switched networks. In *PFLDnet '04*, Chicago, IL, 2004.

[10] D. D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM 1988*, pages 106–114, Stanford, CA, August 1988.

[11] D. D. Clark, R. Braden, A. Falk, and V. Pingali. FARA: Reorganizing the addressing architecture. In *FDNA-03*, pages 313–321, Karlsruhe, Germany, 2003.

[12] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A knowledge plane for the internet. In *SIGCOMM 2003*, pages 3–10, Karlsruhe, Germany, August 2003.

[13] D. D. Clark, K. Sollins, J. Wroclawski, and T. Faber. Addressing reality: An architectural response to real-world demands on the evolving internet. In *FDNA-03*, pages 247–257, Karlsruhe, Germany, 2003.

[14] D. D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's internet. In *SIGCOMM 2002*, pages 347–356, Pittsburgh, PA, August 2002.

[15] National Science Foundation. Future internet network design meeting. http://find.isi.edu/.

[16] Data Transport Research Group. Survey of transport protocols other than standard grid TCP. Global Grid Forum, 2005.

[17] N. Hutchinson and L. Peterson. The x-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, 1991.

[18] C. Jin, D. X. Wei, and S. H. Low. FAST TCP: Motivation, architecture, algorithms, performance. In *Proceedings of the 2004 IEEE INFOCOM Conference*, pages 2490–2501, Hong Kong, March 2004.

[19] P. Levis *et al.* TinyOS: An operating system for wireless sensor networks. In *W. Weber, J. Rabbey, and E. Aarts, editors, Ambient Intelligence*, New York, NY, 2004. Springer-Verlag.

[20] S. Moore. Multimedia monster. *IEEE Spectrum*, 43(1):20–23, Jan. 2006.

[21] S. O'Malley and L. Peterson. A dynamic network architecture. *ACM Trans. Computer Systems*, 10(2):110–143, May 1992.

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

[23] J. Saltzer, D. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Computer Systems*, 2(4):277–288, Nov. 1984.

[24] V. Srivastava and M. Motani. Cross-layer design: A survey and the road ahead. *IEEE Commun. Mag.*, 43(12):112–119, Dec. 2005.

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

[26] D. D. Clark *et al.* Making the world (of communications) a difference place. *ACM Computer Communication Review*, 35(3):91–96, July 2005.

[27] S. C. Visweswara, A. A. Goel, and R. Dutta. An adaptive ad-hoc self-organizing scheduling method for quasi-periodic sensor traffic. In *Proceedings of SECON 2004*, pages 342–351, 2004.

[28] J. Wang, L. Li, S. Low, and J. Doyle. Cross-layer optimization in TCP/IP networks. *IEEE/ACM Trans. Networking*, 13(3):582–595, June 2005.

[29] X. Yang. NIRA: A new internet routing architecture. In *FDNA-03*, pages 301–312, Karlsruhe, Germany, 2003.

[30] A. H. Zaim, I. Baldine, M. Cassada, G. N. Rouskas, H. G. Perros, and D. Stevenson. `JumpStart` just-in-time signaling protocol: A formal description using extended finite state machines. *Optical Engineering*, 42(2):568–585, Feb. 2003.