ABSTRACT

VELLALA, MANOJ. Stack Composition for SILO Architecture. (Under the direction of Associate Professor Rudra Dutta and Professor George Rouskas).

SILO is a new internetworking architecture that represents a significant departure from current philosophy and practice. The architecture consists of building blocks of fine-grain functionality, explicit support for combining elemental blocks to accomplish highly configurable complex communication tasks, and control elements to facilitate (what is currently referred to as) cross-layer interactions. It takes a holistic view of network design, allowing applications to work synergistically with the network architecture and physical layers so as to meet the application's needs within resource availability constraints. The SILO research advocates a non-layered architecture based on silos of services assembled on demand and specific to an application and network environment. With the goal to facilitate what in today's layered architecture is referred to as "cross-laye" interactions, in a manner that meets the exact user requirements and optimizes performance, the main focus of this research work is on developing mechanisms to optimize the construction of SILOs (stack of services) in a manner that takes into account service specific constraints, current network conditions and user policies.

Stack Composition for SILO Architecture

by

Manoj Vellala

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Master of Science

Computer Science

Raleigh, North Carolina

2008

Approved By:

Dr. Khaled Harfoush

Dr. George N. Rouskas

Dr. Rudra Dutta
Chair of Advisory Committee

DEDICATION

To my Teachers, Parents and God

BIOGRAPHY

Manoj Vellala graduated from PESIT with a Bachelor of Engineering degree in Information Science and Engineering, Visweshvaraiah Technological University, India in May 2003. He started his professional career at Infosys Technologies, Bangalore, India and spent the next 3 years as a Software Engineer at Infinera India Ltd. He then joined Master of Science program in Computer Science at North Carolina State University (NCSU).

ACKNOWLEDGMENTS

I would like to express my sincere thanks and gratitude to my advisors Dr. Rudra Dutta and Dr. George Rouskas for their continuous guidance, suggestions and the long hours they spent for me during the entire period of my research. I was able to fulfill my aspiration to contibute to research in networking with the motivation and inspiration instilled in me by my advisor. I am grateful to my committee members, Dr. David Thuente for their affable nature and constant support during the course of my research. I express special thanks to Dr. Ilia Baldine to his guidance in the prototye development and to Anjing Wang for working towards the success of this project.

I would like to thank my family members for their love and support throughout my life. I would also thank all my friends whose constant support and care has always kept me motivated.

Finally, I would also thank North Carolina State University for providing me an opportunity to pursue my Masters.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Context

The Internet, conceived as a free academic tool has evolved into a complex global system of importance equal to that of the power grid and the transportation infrastructure. The Internet has seen an explosive growth 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 into 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 [proposal17,44]. 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.

We are already seeing the beginnings of this: in sensor/actuator networks, in the increasing functionality of mobile handheld devices, and in the migration of many services to network appliances and the network itself. The dominant vision of networking in the future, and computing in general, has been called ubiquitous or pervasive networking. Even as computing technology reaches new heights of ubiquity, a crisis has been seen to be developing that can jeopardize this future vision. The pervasive network of the future is enabled by and must serve a new generation of communication endpoints that are very different from the personal computers and servers that form the bulk of the network endpoints in today's Internet. Communication devices are already appearing which are more integrated, more embedded, with sensors and other ubiquitous computing devices. Such devices often have unique characteristics, both advantages and limitations, which are not common in currently popular networking devices. Current internetworking protocols are flexible enough to handle them, but not gracefully. The overhead imposed in bending the capability of such devices to existing network architecture can make the use of such devices prohibitive and even useless.

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

At the same time, the suitability of protocol layering, as either an Organizing or implementation principle for future network architectures, is being questioned [9]. network 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, as witnessed by the proliferation of 1/2 layer solutions (e.g., IPSec and MPLS). Also, the lack of mechanisms for cross-layer interactions (e.g., for performance tuning) has led to frequent layer violations.

Moreover, protocols were not designed to take advantage of emerging hardware architectures such as the Cell by IBM [22] with one main and several synergistic processors in a single chip. Current TCP offload engines [3, 4, 6, 12, 30] are clumsy, having to deal with a

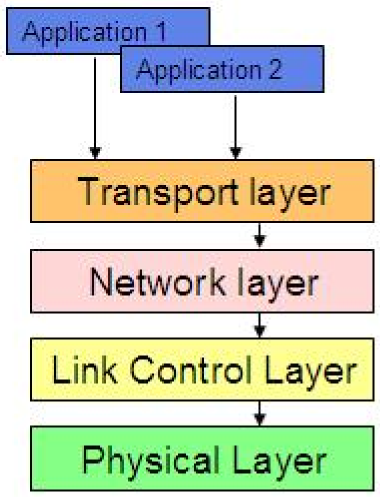


Figure 1.1: Traditional TCP Stack

multitude of contingencies forced by the presence of two or more separate networking stacks inside the node, and have to be carefully engineered for each specific hardware architecture. A more modular design that makes it easier to selectively offload into hardware the most time consuming protocol functions, might lead to significant performance gains.

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

An application specific stack is dynamically generated satisfying all the Qos requirements of that application and in confirmation with local polices and network constraints. The SILO stacks can be generated per flow or per class of flows.

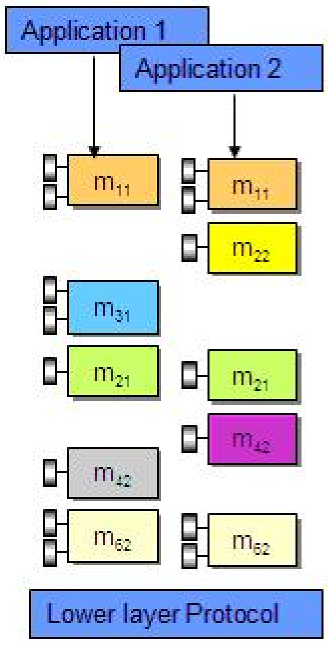


Figure 1.2: Per Flow Silo Stack

Despite the remarkable ongoing effects of the Internet, there is a widespread perception in the networking community that key limitations of its design might be bringing it close to a breakdown point and a sea-change is necessary in the next decade or so. Recently, the National Science Foundation issued a call for proposals for clean-slate Internet design. The SILO research is the effort of multi-organization collaborative research team that has been working on such a clean-slate approach to future Internet design funded by a grant from the NSF Future InterNet Design (FIND) program.

1.2 Review of parallel efforts

In recent years, the networking community has taken a variety of approaches in addressing the issues that arose, as the shortcomings and limitations of today's Internet architecture have become increasingly evident. Typically, a solution for a specific problem is engineered within the constraints of the current Internet architecture. Often, such a solution only applies to a specific context; consider, for example, the recent research on TCP variants for high bandwidth-delay product networks [16, 18, 19, 33] earlier work on TCP over wireless networks

Among recent research, the work most closely related to SILOS is that on role-based architecture (RBA) [9] carried out as part of the NewArch project [28]. RBA represents a non-layered approach to the design of network protocols, and organizes communication in functional units referred to as roles. 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 [9], 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 Chapter 2). However, the goal is to facilitate 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. Furthermore, a main focus of our proposed work is on developing control mechanisms to optimize the construction of silos of services in a manner that takes into account current network conditions and user policies.

Some earlier work also investigated more flexible frameworks for realizing protocols and services. The use of finer-grain protocol (micro-protocol) objects, each encapsulating a single function, to facilitate the development of protocol stacks was considered in [8]. The main goal of [23] is to dynamically select a set of protocol layers within a hierarchical framework, based on application semantics and the characteristics of the underlying network. The micro-protocols of [23] are coarser than our services, and we allow for a richer set of interactions among services beyond what is possible within the strict hierarchy considered there. The work in [8] focuses on protocol modularity and configurability as a means for ease and efficiency of implementation. Both these approaches are x-kernel specific, as they rely heavily on the x-kernel [17] environment and its mechanisms for communication between micro-protocols. In contrast, we do not tie our implementation to any special purpose environment; rather, we target our approach to a more common POSIX-like OS. The focus of [26] was on efficient construction of transport services with different QoS characteristics for multimedia applications. Their concept of services is much coarser than ours (e.g., "multicast" is considered a service in [27]), and the work is oriented toward hierarchical composition with limited interaction (information sharing) between services. In general, our proposed architecture goes beyond previous work in that (1) service silos are created on-demand, automatically, based on application requirements, local and core policies etc., and (2) mechanisms for adaptive control along with "cross-layer" interactions for the purpose of optimizing behavior are built into the framework.

Another research worth noting is JumpStart's Just-in-Time (JIT) Architecture [1]. 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.

Chapter 2

The SILO Architectural Framework

2.1 Architectural objectives

The SILO design was guided by the following objectives than ensure flexible, service oriented architecture for future internet.

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.

2.2 SILO Architecture

Figure below shows the difference of the SILO approach and traditional networking stack. The traditional stack is shown in Figure 1 (a). There are multiple applications, to which transport layer provides sockets, but only single instance of all other layers. This makes it impossible to provide customized service to different applications. for example, if we want to use reliable transport, we have to use TCP/IP stack. It is quite hard to extract this functionality out gracefully, while eliminating some other TCP functionality, for some application that does not need it. Another downside of this traditional stack is that we cannot easily perform cross-layer tuning to optimize the stack; we have to write custom version of the layers for that purpose. However, the SILO architecture incorporates these two requirements into the architecture framework as Figure 1 (b) shows. The networking communication entity is dynamically composed by protocols of fine-grained functionalities, which we call services. m1;1, m1;2 etc refer to these functionalities. Every such service provides explicit interfaces for performance optimization, which can be used by Cross-Service Tuning for optimization purpose. We explain these architectural components in greater detail next.

2.2.1 Services

The fundamental building blocks in the SILO architecture are services. A service is a well defined and self-contained function performed on application data, and which is relevant to a specific communication task. In-order packet delivery, end-to-end flow control, packet fragmentation, compression, encryption, 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 [3]. 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 affects 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 (defined below) and their actions and constraints.

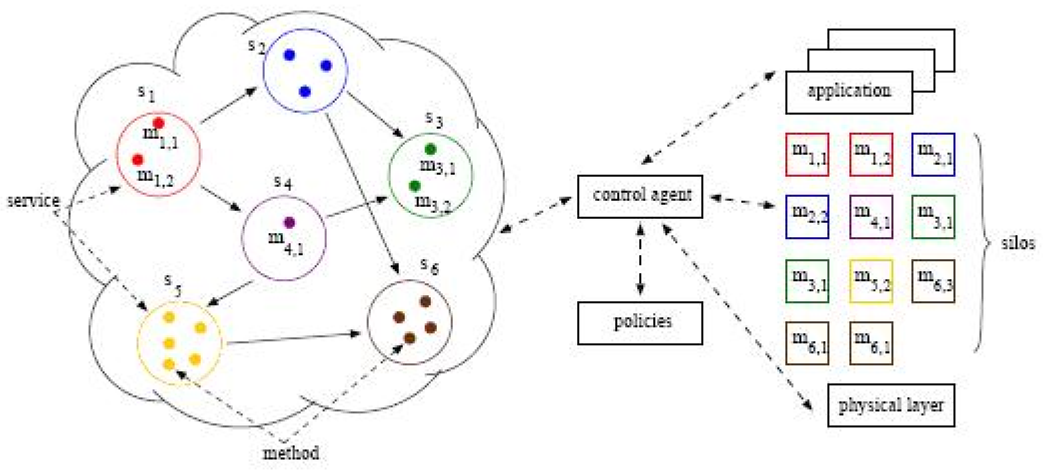


Figure 2.1: SILO Architecture Overview

2.2.2 Methods

We distinguish between a service and its realization. A method is an realization 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 tuning agent (defined in 4) 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 4 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.

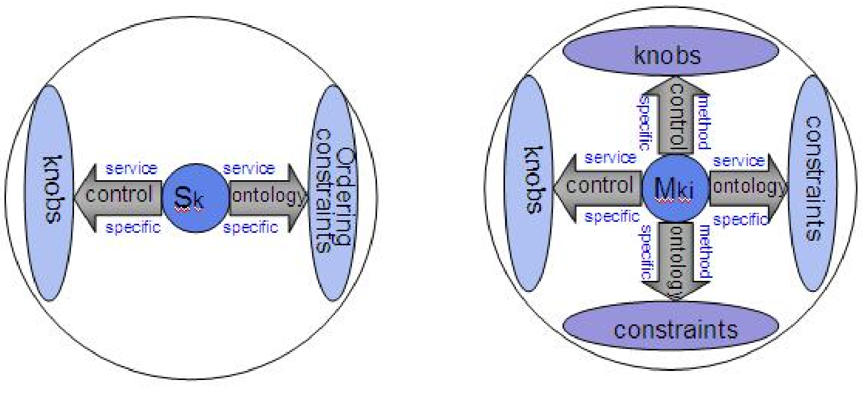


Figure 2.2: Services and Methods

2.2.3 Knobs

The knobs include read/write knobs and read-only knobs. The read/write knobs are adjustable parameters specific to the function performed by a service or a method, with a specified range of values or several enumerated values and a pre-defined relationship between these values and the perceived performance of the service or method. for instance, compression factor is a read/write knob for the compression service. The read-only knobs are values or states of a service or a method. for example, throughput is a read-only knob for the performance monitor service. The figure above presents a typical knob in SILO universe. It has maximum, minimum, default and current values, or has several enumerated values. With the help of relation between performances and knobs provided by Performance Table, the knobs are manipulated by the tuning agent (defined in 4) so as to optimize the performance of the subset of services selected for the specific task. If a knob is defined as enumerated values, the tuning agent is able to switch between them. Otherwise, the tuning agent can tune the value with the minimum gradient of tuning step. Service, Method and Knob designers can provide those parameters and Performance Table when they design the services, methods and knobs. Performance Table provides the relation between knob and performance in terms of relevance scale and mathematical expression.

2.2.4 Control Agent

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

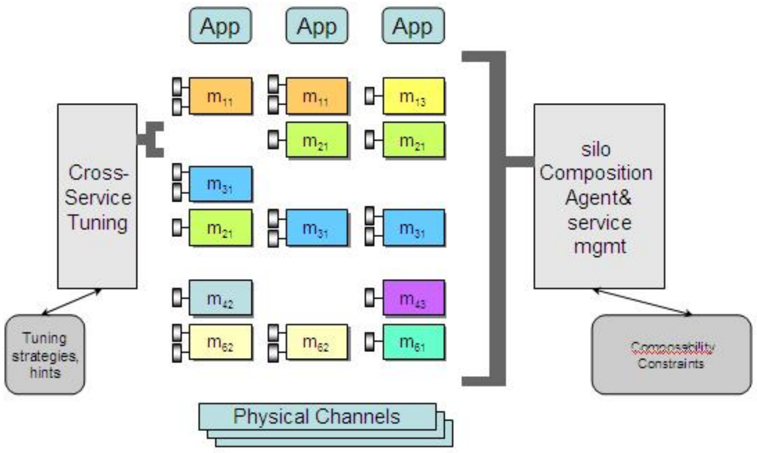


Figure 2.3: Detailed Overview Depicting Tuning agent and Composition agent

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.

Chapter 3

Problem definition

3.1 Stack composition problem

The protocol stack is dynamically on demand constructed per flow based on the flow characteristics and QoS requirements. The stack is composed of services, where every service interacts with an upper service and lower service. A set of compatibility rules and constraints determine the services that can be above or below a particular service in a stack. The rule set also captures other service/method dependencies. This opens up an opportunity for creating unique stacks on a per dataflow basis or distinct stacks for different class of applications.

The rule set does not limit the number of unique stacks that can be construction for a specific need (data flow or class of applications). The problem of composition is the problem of finding the optimal stack recipe satisfying all the service constraints and not violating any of the compatibility rules. The section on Silo Composition Agent discusses the algorithm used for Silo composition.

The silo construction problem thus becomes one of finding a valid ordering of a set of services which are self-consistent, and incorporate the functionality required by the user and application requirements, including derived ordering requirements. We can now conceive of an optimization problem, which further seeks to find that solution which maximizes some measure of user satisfaction, e.g., obtained from the prioritization of the user preferences. Clearly, an approach based on exhaustive search is not computationally scalable. Below, we provide a graph interpretation of part of the problem.

We denote the set of services S available to the silo manager as $s_1, s_2, ...s_n$. for each service s_k, let there be nk methods available, we denote them by $m_{k,1}, m_{k,2}, ...m_{k,n_k}$ .We denote the set of all methods by M. for each $r \epsilon S \bigcup M$, $p(r)$ denotes a set of subsets of $S \bigcup M$, such that at least one service or method from each element of $p(r)$ must also be included in any silo which includes r. For example, if $p(m_{1,2}) = \{\{s2, s3, s4\}, \{m6, 1, m6, 2\}, \{m5, 2\}\}$, then to include the method m1;2 in a silo, we must include also a method for either services 2, 3 or 4, and either methods $m_{6,1} or m_{6,2}$, and method $m_{5,2}$. Also for each $r \epsilon S \bigcup M$, $a(r)$ and $b(r)$ are two sets expressing ordering constraints; $a(r)$ is the set of elements of $S \bigcup M$ that must be ordered *after* r; $b(r)$ the set of those that must be *ordered* before r. Given a subset S_r of S representing user requirements, and a set O_R of ordered pairs of elements representing derived ordering constraints, the problem is to find two ordered sets S_a and M_a, the augmented service and corresponding methods lists. for every element in S_a, the corresponding element (in the same ordered position) of M_a must be a method implementing the service. S_a must be a superset of S_r that satisfies the dependencies $p(r)$, obeys the constraints $a(r)$ and $b(r)$ for every element $r \epsilon S_a$, and obeys the constraints O_R.

3.2 Representing the Composibility rules and Constraints: Ontology

The compatibility rules and constraints need to be captured and represented in the computer. There needs to be query based interface that allows the composition agent to query online for allowed services based on specified conditions. 'the Ontology' is the global set of services and rules set and provides the querying interface. To Ontology facilitates flexible maintenance, supports customized expressive specifications and semantic comparisons, reasoning, understanding.

The existing ontology framework and infrastructure for web semantics can be taken advantage of. RDFS is used as the ontology language and JENA is the java based querying and reasoning library is used by the composition agent to query the ontology. The section on ontology framework discusses the implementation in detail.

Silo Composition Problem The Problem of SILO composition constitutes two subproblems 1. Ontology: Logical representation of the constraints and conditions that services and methods impose on themselves and on other services and methods in a format that can be used to programmatically deduct/infer complex implications. 2. Composition: Composing a silo recipe of services and methods such that all the application's requirements are met and is in accordance with the above said constraints and conditions.

The composition problem can be further defied formally in the following way: Inputs: a) Application required services/methods b) Application imposed ordering constraints both loose and strict c) Ontology Output: a) A minimal silo that does not violate any of the conditions/constraints specified in the input and meets all the input requirements.s OR Error indicating that such a recipe is not possible.

3.2.1 Types of Constraints

The constraints are defiend as a triple, the subject service, the relation/constraintand the object service. The constraints are defiend in sucg a way they are to be confirmed by every inatance of both the subject and object serivce. For instance, Service-A constrsint service-B, would mean that, the constraint should not be violated for any pair of instances of A and B. When a SILO has no repeated service then, the implications are strainght forward. The initial set of constrinsts identifed are enumerated below.

Simple Constraints

- Only_Above
- Only_Below
- Only_Imm_Above
- Only_Imm_Below
- Primitive_Depedency / Requires

Complement Constraints

- Not_Above {equivalent to Only_Below}

- Not_Below { equivalent to Only_Above}
- Not_Imm_Above
- Not_Imm_Below

Compound Constraints

- AND_Connector
- OR_Connector
- Forbids

3.2.2 Constraint Definitions

Simple Constraints

- **Only_Above**
  A Only_Above B : If A is present AND if B is present, then A has to be above B
  (If A then A above B if B)
- **Only_Below**
  A Only_below B : If A is present AND if B is present, then A has to be below B
  (If A then A below B if B)
- **Only_Imm_Above**
  A Only_Imm_Above B : If A is present AND if B is present, then A has to be Immediately above B
  (If A then A immediately above B if B)
- **Only_Imm_Below**
  A Only_Imm_Below B : If A is present AND if B is present, then A has to be Immediately below B
  (If A then A immediately below B if B)

- **Needs**
  A Needs B : If A is present then B needs to be present
  (If A then B)
- **Forbids**
  A forbid B : If A is present then B cannot be present
  (If A then not B)

Complement Constraints

- **Not_Above**
  A Not_Above B : If A is present AND if B is present, then A Should not be above B
  (If A then A not above B if B)
- **Not_Below**
  A Not_below B : If A is present AND if B is present, then A should not be below B
  (If A then A not below B if B)
- **Not_Imm_Above**
  A Not_Imm_Above B : If A is present AND if B is present, then A should not be immediately above B
  (If A then A not immediately above B if B)
- **Not_Imm_Below**
  A Not_Imm_Below B : If A is present AND if B is present, then A should not be immediately below B
  (If A then A notimmediately below B if B)

Compound Constraints

OR_Connector acts as OR operator over a set of simple, complement and compound constraints.
The AND_Connector is not needed as each of the ANDed constraints can be represented as individual constraint on the same service.

Allowed ordering between two services and Their realization:

The following ten cases are the complete set of orderings that are possible between two services (A, B).

1. Neither A nor B is present
2. Only A is present
3. Only B is present
4. A appears immediately above B
5. B appears immediately above A
6. A appears above B and A,B are not consecutive
7. B appears above A and A,B are not consecutive

Realization of each of the above cases and valid combinations of cases is discussed below:

Only Case 1. Neither A nor B is present

A needs B

B needs A

A forbids B

Only case 2. Only A is present

A forbids B

B needs A

Only case 3. Only B is present

B forbids A

A needs B

Only case 4. A appears immediately above B

A needs B

A only immediately above B

B needs A

B only immediately below A

only case 5. B appears immediately above A

B needs A

B only immediately above A

A needs B

A only immediately below B

only case 6. A appears above B and A,B are not consecutive

A needs B

A only above B

B needs A

B only below A

A not immediately above B

Only case 7. B appears above A and A,B are not consecutive

B needs A

B only above A

A needs B

A only above B

A not immediately below B

Apart from these ten cases few combination of the cases are valid. The following are some of the valid combinations.

Case 4 OR 6: A appears above B

A needs B

A only above B

B needs A

B only below A

Case 5 OR 7: B appears above A

B needs A

B only above A

A needs B

A only below B

Case: 1 OR 2

B needs A

A forbids B

Case: 1 OR 2

A needs B

B forbids A

Case: 2 OR 3

A forbids B

B forbids A

Case: 4 OR 5

A needs B

B needs A

A immediately below B OR A immediately above B

B immediately below A OR B immediately above A

Case: 2 OR 4

B needs A

B only immediately below A

Case: 2 OR 7

B needs A

A Not above B

It can also be realized as B needs A, A only Below B Similarly other combinations can be realized using the constraint set specified in the previous section. We note that most of the ten cases discussed can be realized in more than one way. for example:

Case: A appears immediately above B

Realization 1:
A needs B
A only immediately above B
B needs A
B only immediately below A

Realization 2:
A needs B
A only immediately above B
B needs A

Realization 3:
A needs B

B needs A

B only immediately below A

Case: B appears above A

Realization 1:

B needs A

B only above A

A needs B

A only below B

Realization 2:

B needs A

B not below A

A needs B

A not above B

This indicates that a minimal set of constraints that can be used to realize all orderings between two services is possible. The next section explores this.

3.2.3 Minimal Constraints set

Some constraints are redundant. This means two constraints mean the same. Such constraints can be called to belong to same equivalent class.

Equivalent Constraint Instance Class 1

{A Only_Above B, A Not_Below B, B Not_Above A, B Only_Below A}

A Not_Below B : This means that A appears above B since two services cannot be at the same position on a stack. So this can be represented as A above B.

B Not_Above A : This means that B appears only below A for the same reason as above.

B Only_Below A : This means that A appears above B. So this can be represented by A Only_Above B.

Hence, of the four constraints Only_Above , Not_Below, Not_Above, Only_Below, any one would suffice. Similarly, it can be observed that the following two equivalent constraint set are also possible.

Equivalent Constraint Instance Class 2

{A Only_Imm_Above B, B Only_Imm_below A }

Equivalent Constraint Instance Class 3

{A Not_Only_Imm_Above B, B Not_Only_Imm_below A }

Implication: Equivalent Constraint Instance Class 2 and Equivalent Constraint Instance Class 3 imply: Only_Imm_Above and Not_Imm_above should suffice.

Implication: forbids is implemented as two contradicting instances of Only_Above : A forbids B is implemented as A Only_Above B, B Only_Above A

Thus we can conclude that the following set of constraints would suffice:

Minimal Set of constraints:

1. Only_Above
2. Only_Imm_Above
3. Not_Imm_Above
4. Requires

To verify if this is the sufficient set of constraints, realizing all the above said seven orderings between two services with the minimal set can be tested.

1. Neither A nor B is present

   1. A requires B
   2. B requires A
   3. A above B
   4. B above A

2. Only A is present

   1. A above B
   2. B above A
   3. B requires A

3. Only B is present

   1. B above A
   2. A above B

   3. A requires B

4. A appears immediately above B

   1. A requires B
   2. A only immediately above B
   3. B requires A

5. B appears immediately above A

   1. B requires A
   2. B only immediately above A
   3. A requires B

6. A appears above B and A,B are not consecutive

   1. A requires B
   2. A only above B
   3. B requires A
   4. A not immediately above B

7. B appears above A and A,B are not consecutive

   1. B requires A
   2. B only above A
   3. A requires B
   4. A only above B
   5. B not immediately above A

We notice that all the seven possible orderings can be implemented by the minimum set of constraints. By excluding any one of them its not possible to realize atleast one of these. for example, if we exclude “not immediately above” from the set, then Consider ordering 7 : B appears above A and A,B are not consecutive Realizing the condition that A and B are not consecutive cannot be done. If we exclude “only immediately above” then ordering 5 (B appears immediately above A) cannot be realized. The ordering where B appears above A can be realized but, the consecutiveness cannot. If we exclude the ‘requires’ constraint, most of the orderings cannot be realized. for example consider ordering 2: Only A appears on the stack. A above B and B above A ensure that A and B do not occur on the same stack. But cannot ensure the case where only B appears on the stack is invalid.

Representing the composition problem as a graph problem

A digraph can be constructed where the services are vertices and the constraints are edges. Such a graph will have edges of different colors each color representing a constraint type. A specific color could mean, that such an edge (or any parallel edge between the two vertices) should never be traversed. Another color could mean that the specific edge (or any parallel edge) has to be traversed if the edge's source node is in the path. The graph problem if multi colored edges can be also be represented as multiple graphs of single color with the condition that a path in one graph does not violate any constraints imposed by other graphs.

This approach works when we have constraints that define ordering of two vertices when they are adjacent in the path. A constraint like "A above B" which is a loose ordering constraint cannot be represented by a graph as described above.

When each of the constrainsta or a subset of constrainst are consider and represented as graph problem we note that the silo composition is polynomiyal time problem. But the most general case is un-charecterised.

a) Above: Between all services excluding Ss and Se,the edges are of type Only_Above if an edge exists. If no edge exists between two nodes then there is no ordering restriction between them. The problem is finding a directed ordering from Ss to Se. As are the services that can go immediately below Ss and Bs are services that can be immedately above Se. If one of As is also one of Bs, then Ss, As, Se is a valid ordering and this take linear time in number of services to check. For the case that an edge exists from a Bs to an As, if there exists any vertex v such that, $v \rightarrow As$ or $Bs \rightarrow v$ or both are not true, then As, v,Bs is a valid ordering. If no such v exists then there is no ordering satisfying the constraints. Finding such a v is again linear in number of services for a given pair of As and Bs. In the first two cases the ordering is of length 3 at maximum and in the second case it is of length 4. Each of the sub-cases is of polynomial time complexity.

b) Imm_Above: The proof is exactly the same as for the case above. An easy

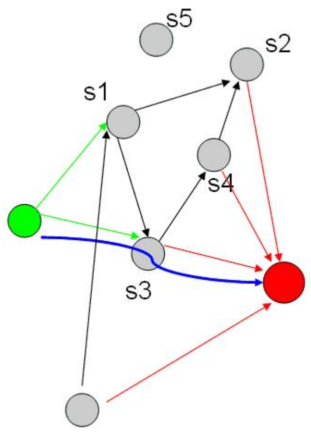


Figure 3.1: Only Above Constraint Graph: Silo size 3

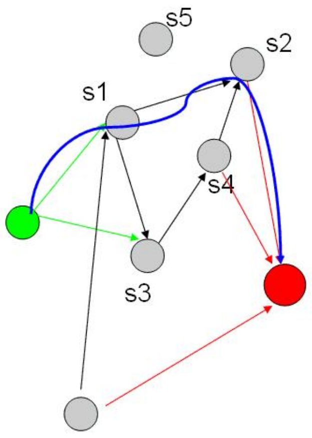


Figure 3.2: Only Above Constraint Graph: Silo size 4

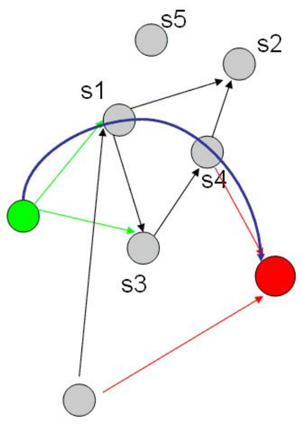


Figure 3.3: Only Above Constraint Graph: Silo size 4

way to see this is as follows: if between two services As, Bs there are no edges then we can include the edges $As \rightarrow Bs$ and $Bs \rightarrow As$ and then run Dijkstras algorithm. Here the services Ss and Se are treated like any other service.

c) Not_Imm_Above: Here we construct another graph in which every edge is an ImmAbove edge. Initially this is a complete bi-directed graph on the nodes (services). Then remove the directed edges between any two services u, v if a constraint u Not_Imm_Above v is specified. Now run Dijkstras algorithm. Obviously, even when Imm_Above and Not_Imm_Above are both present, the problem can be solved in polynomial time by removing the (directed) edges corresponding to the contradicting constraints. (Such contradicting constraints should not exist if the ontology is consistent). Finally, we note that in the more general case when Imm_Above and Above are both present, the problem can still be solved in polynomial time by the same procedure as in the Above only case. However, the most general case remains beyond our reach.

A graph with Only_Imm_Above, And Not_Imm_Above can be solved in polynomial time , as we can remove directed edges accordingly and run Dijkstras algorithm. A graph with Only_Imm_above,And Only_Above can be solved in polynomial time , exactly the same way as in only Only_above case. Any of the above cases with “requires constraint can also be solved in polynomial time. However, The most general case remains uncharacterized.

Application Specified inputs

The application can specify required and recommended services and some ordering constraints between them. Let us consider only the required services. These are the services that need to be present in the path constructed. An ordering constraint makes path computation more complicated in that the vertices need to be visited in a specified order.

The Opening and closing services

The opening service and the closing service interface the silo with the application and the lower layer protocols respectively. The opening service helps in identifying the services from which the silo construction can be started. The closing service helps in terminating the silo-search. The closing service also helps in multiplexing and de-multiplexing the flows.

The diagram above depicts a bidirectional connection as an incoming and an outgoing flow. In case of an incoming flow, the closing service being the lowest on the silo stack identifies the right silo which needs to service the flow. In the outgoing direction it multiplexes multiple silo flows on to one single lower layer protocol stack.

3.2.4 Mathematical formulation

By trying to formulate each of the constraint as a mathematical expression we tried to formulate the composition problem as an ILP. Our attempt was reasonably sucessful in that we were able to formutlae each constraint as a quadratic inequation.

Following is the formutaion:

Parameters:

A_{ij} = 1 if service i should be above service j

= 0 otherwise

R_{ij} = 1 if service i needs service j to be in the stack.

IA_{ij} = 1 if service i should be immediatley above service j

= 0 otherwise

Table 3.1: Required constraint truth table

R_{ij}	A_{ij}	S_{ij}	ϕ
0	0	0	1
0	0	1	1
0	1	1	1
0	1	0	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Variables:

$S_i = 1$ if i^{th} service is in the stack

$= 0$ otherwise

K_i is the position of the i^{th} service in the stack.

$X_{ij} = 1$ if service i is above service j

$= 0$ otherwise

The problem is to find a SILO such that $\sum S_i$ is minimum and $\sum S_i \geq 1$. This means the smallest SILO with atleast one service. (Note here we do not consider the opening and closing service.)

3.2.5 Deriving expression for Required constraint

The following truth table points out the conditions under which including two services in the silo would be invalid.

$\neg\phi = \mathrm{S}_i \ . \ \neg\mathrm{S}_j \ . \ R_{ij}$

$\Rightarrow \phi = \neg(\mathrm{S}_i \ . \ \neg\mathrm{S}_j \ . \ R_{ij})$

$\Rightarrow \phi = \neg\mathrm{S}_i \vee \mathrm{S}_j \vee \neg\mathrm{R}_{ij})$

$\phi \geq 0 \Rightarrow (1-\mathrm{S}_i)+S_j+(R_{ij}) \geq 1$

$\Rightarrow 1+\mathrm{S}_j - S_i + R_{ij} \geq 0$

This is a linear integer inequation.

Table 3.2: Only_Above constraint truth table

X_{ij}	A_{ij}	S_{ij}	S_{ij}	ϕ
0	0	0	0	1
0	1	0	0	1
0	1	1	0	1
0	1	0	1	1
0	1	1	1	X
....				
....				
....				
0	1	1	1	0

3.2.6 Deriving expression for Above constraint

Here we use a new varibale X whose value depends on the position of two services under consideration. Deriving expression for Required constraint result sin two inequations.

One is derived form the definition of X and other the above constraint. Definition of X:

$X_{ij} = 1$ if $K - i$ - $K_j \geq 1$

$\Rightarrow$ (K$_i$ - K_j) X_{ij} + $\neg$(K$_i$ - K_j) ≥ 1

$\Rightarrow$ (K$_i$ - K_j) X_{ij} + (K_j - K_i) $\geq$ 1........$EQ2$

Definition of Only_Above constraint results in the following truth table:

$\neg\phi = \neg$X$_{ij}$. A_{ij} . S_i .Sj

$\Rightarrow \phi$ =X$_{ij}$ + (1 - A_{ij}) + (1 - S_i) + (1 - S_j)

$\Rightarrow$ (X$_{ij}$ + 3 - A_{ij}- S_i - S_j) $\geq$ 0........$EQ3$

3.2.7 Only Immediately Above

Here we use a new varibale X whose value depends on the position of two services under consideration. Deriving expression for Required constraint results in two inequations.

One is derived form the definition of X and other the above constraint. Defition of X:

$X_{ij} = 1$ if $K - i$ - K_j 1

= 0 other wise.

$\Rightarrow$ (K$_i$ - K_j -1)$X_{ij} \geq$ 1........$EQ4$

$\Rightarrow$ (1$-$K$_i$ + K_j) $X_{ij} \geq$ 1........$EQ5$

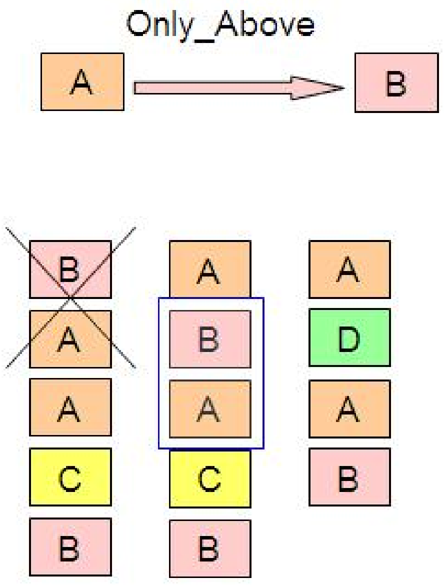


Figure 3.4: Only Above Constraint

Definition of Only_Immediatley_Above constraint results in the truth table 3.3.

$\neg\phi = \neg X_{ij}$. A_{ij} . S_i .Sj

$\Rightarrow \phi = X_{ij} + (1 - A_{ij}) + (1 - S_i) + (1 - S_j)$

$\Rightarrow (X_{ij} + 3 - A_{ij} - S_i - S_j) \geq 0$........$EQ6$

3.3 SILO Composition Algorithm

The composition problem then can be stated as follows: Given a set of services in the ontology, the problome is to obtain an ordering that is consistent with the precedence constraints in the ontology, possibly augmenting the set of services for the purpose. A straightforward approach can easily guarantee that correct stacks (obeying all constraints) are constructed. The approach is to do a DFS starting from the Start Service untill Closing service is reached, then backtrace one step and try to extend. At every step of DFS, the services that are forbidden or services that violate a constrsint withrespect to services already in the stack are eleminated. When a step is backtraced all services that were

Table 3.3: Only_Immediatley_Above constraint truth table

X_{ij}	A_{ij}	S_{ij}	S_{ij}	ϕ
0	0	0	0	1
0	1	0	0	1
0	1	1	0	1
0	1	0	1	1
0	1	1	1	X
....				
....				
....				
0	1	1	1	0

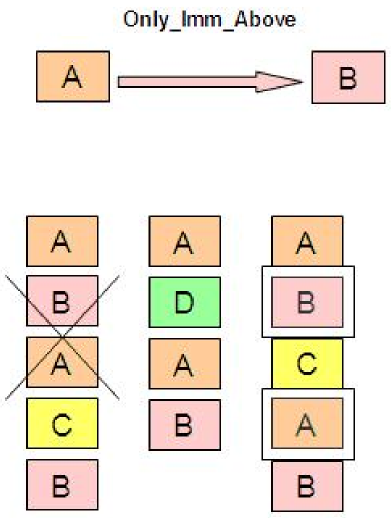


Figure 3.5: Only Imm Above Constraint

eleminated would agin become allowed.

Briefly, the steps are:

- Initialize essential services list from application specification
- Designate any service S1 as the top service
- Recursively, build the stack below
- S_i := service last added
- If S_i Requires any service, add to essential
- If S_i has an ImmAbove constraint, add it to stack (unless marked backtrack)
- Otherwise, add any other service which can be added
- Recurse // (No other service can be added)
- Check to see if stack violates any Above condition
- OR any essential service is missing
- If not, output stack and exit
- Else backtrack to remove last service added
- Start with some other service as top service

Clearly, this algorithm will produce correct stacks; equally clearly, it has a very long running time. Many improvements to running time can be made in the form of sophistications such as checking each service for violations with services already on the stack, backtracking several steps when removing a service required by another above, starting with essential services as top service choice, etc. However, they leave the algorithm essentially the same, and not guaranteed to complete in polynomial time. for this reason as well as some indications from a graph-theoretic modeling, we conjecture that the problem is NP-complete; however, we do not have a formal proof at this time. In practice, in ontologies with reasonable sets of constraints, the version of the algorithm with all the accelerations runs with very little delay (a few seconds with an ontology of around fifty services) if there are a reasonable number of constraints to prune the search. However, this changes with further drastic simplification of the set of precedence constraints. If there are no essential services designated by the application, and only one of the three ordering constraints are allowed, together with Requires, the problem can be solved polynomially. for ease of discussion, w.l.o.g. we consider a unique top service Ss and a unique bottom service Se. Further, we consider that the set of services As that can follow Ss are known, as is the set of services Bs that can precede Se. In the general case, these can of course be the set of all services. We consider a digraph where every service can be represented as a node and the constraints as edges between them.

Chapter 4

SILO Prototype Architecture

A high-level view of the prototype software architecture is shown in Figure ??. The architecture consists of the following major components:

* the SILO API,
* the SILO ontology of services and methods,
* the SILO-enabled application (APP),
* the SILO management agent (SMA),
* the SILO tuning agent (STA),
* the SILO construction agent (SCA),
* the universe of services storage (USS), and
* the control strategies storage (CSS).

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

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

4.1 Over view of the implementation approach

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

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

4.2 High-level Architecture

Below, we describe the individual components in greater detail.

In the prototype, we have implemented SILO API, SILO-enabled Application, SILO Management Agent and SILO Tuning Agent.

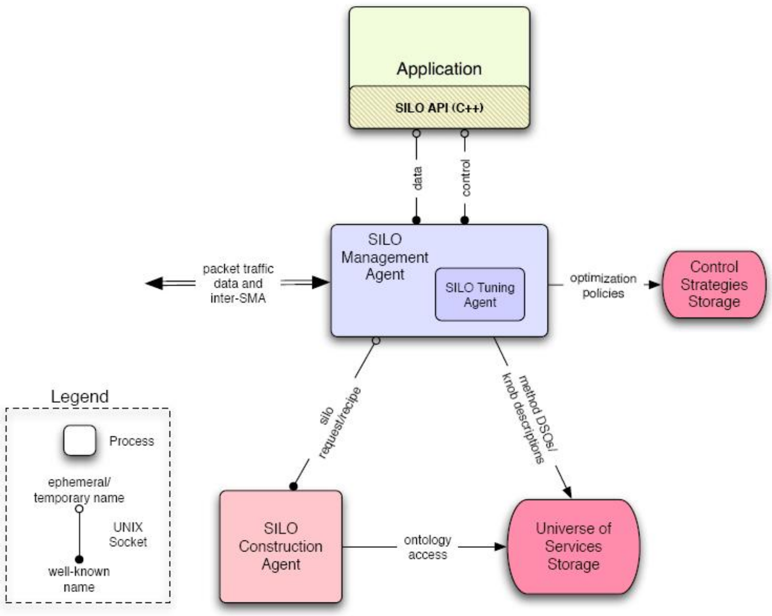


Figure 4.1: High Level Architecture

4.3 Component Description

We have described SILO Construct Agent, SILO Management Agent and SILO Tuning Agent in pervious sections. Besides these control agents, other components are described below.

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

SILO Construction Agent SILO Construction Agent (SCA) is a major component of the architecture whose responsibility it is to assemble a silo based on application's request. It utilizes the SILO Ontology (discussed in 4.3), an inference Engine, and a collection of custom algorithms in order to turn the application request into a silo recipe. The silo recipe is used by the SMA to construct the custom silo for the application.

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

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

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

The SILO API The SILO API consists of C++ header files and library code. Two types of API and libraries are implemented:

∗ Application API - for creating SILO-enabled applications

∗ Internal API - library code common to the individual SILO components (e.g. to facilitate inter-component communications)

The SILO API can help application to add application-specified constraints or ontology as part of the request. It maintains the requests from Application to SILO Construct Agent, whose purpose is requesting SCA to construct a silo and to release a silo.

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

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

Universe of Services Storage The USS serves as the main repository of information about the SILO framework. It contains
(a) The ontology that describes relationships between services and service interfaces,
(b) A database of method implementations which helps the SMA locate the executable code necessary to construct a given silo, (c) Current policy setting which affect the operation of the SILO framework. These can be application-, node- and network-specific. The USS has a query-based interface, which allows other components of the SILO framework to utilize its functionality.

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

4.4 Component Interactions

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

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

4.5 Ontology in detail

The ontology, which is the universe of services and constrsints acts like a database that can be queried for services that confirm to some specified conditions. The Ontology is implemnted by defining services and constraints as objects of specifc object types or classes. The classes implemented are:

- ∗ **Service** : Every service has the following attributes:

- Has_Constraint : This attribute is defined by an object of type Constraint.
- Has_Data_effect : This attribute is defined by an object of type Data effect.
- Has_Effect : This attribute is defined by an object of type Performance effect.
- Has_Function : This attribute is defined by an object of type Service Function.
- Is_of_type : This attribute is defined by an object of type ServiceType.
- Name : The service's name.

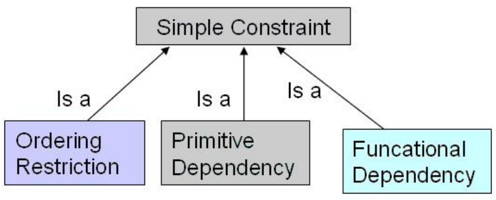


Figure 4.2: Constraint Classes in RDF

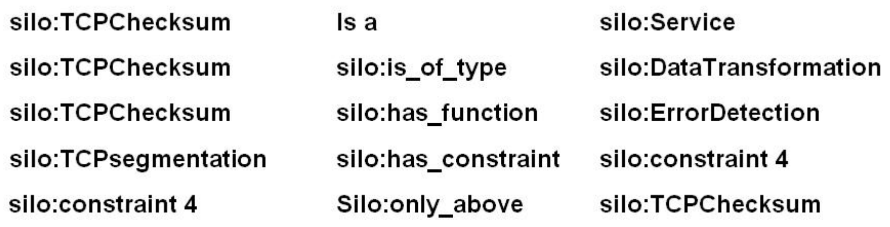
∗ **Method** Every method has the following attributes:

- Has_Constraint : This attribute is defined by an object of type Constraint.
- Has_Data_effect : This attribute is defined by an object of type Data effect.
- Has_Effect : This attribute is defined by an object of type Performance effect.
- Implements : This attribute defines the service that this method implements.
- Name : The method's name.

∗ **SimpleConstrinst** : The simple constrinst can be a PrimitiveDependency or an orderingRestriction.

* **PrimitiveDependency** : This type of constraint is the representation of 'Requires constraint' in the ontology. Its the dependency of a primitive (service or a method) on another primitive.
* **OrderingRestriction** : Objetcs of this constraint represents Above or Immedately_Above or Not_Immediately_Above kind of constraints.
* **Compount constraint** : Is an OR-ed combination of simple constaints on a specific primitive.

Once the services and the constraints are defined and captured in the ontology, a reasoner or a query language can be used to query the ontology. JENA is a javabased query library that implements a reasoner for ontology defined RDFS or OWL. See appendix for more information on Jena. RDF defines every relation as triad as depicted in the picture..



silo:TCPChecksum	Is a	silo:Service
silo:TCPChecksum	silo:is_of_type	silo:DataTransformation
silo:TCPChecksum	silo:has_function	silo:ErrorDetection
silo:TCPsegmentation	silo:has_constraint	silo:constraint 4
silo:constraint 4	Silo:only_above	silo:TCPChecksum

Figure 4.3: Service, constraint instances as Traids in RDF

Chapter 5

Algorithm Variations and Results

The algorithm presented in section 3.3 gives the over view of a brutforce method of composing a stack. Few variations of this algorithm has been implemented.One various does not allow any service to be repeated on the stack, while one does not allow more than ‘r’ repititions of service. Another variation restricts loop repittion in a stack. While none of the flavours run in polynomial time, the first two have deterministic running time. They are exponential on number of services ‘n’ in terms of running time. The latter variations running time or complexity was not quantifiable. The following sections discuss the nature of each of the variations and some results.

5.1 Stack composition with fixed length

The stack size is fixed and is defined by the application or some local environment policy. Further variation of this is fixed length and no repetitions allowed or repetitions allowed.

Given an ontology of ‘n’ services excluding the opening and closing services, in the cae where no repetitions are allowed, and when the length is ‘l’ the total number of possible stacks are ${}^{n}P_{l}$.

When repetitions are allowed, the toatl number of possible stacks are l^n and is of exponential complexity.

The resriction on services that can be immediately below Starting Service and on services that can be just above a closing service makes the running time less, but does not change the complexity. For, example, if s services can be starting services, c services can be closing services, the number of possible stacks are $s.(l-2)^n.c$, which is still exponential.

5.2 Stack composition with no repetitions and unrestricted length

Since the stack size can vary from one to 'n', the total number of possible stacks are $^nP_1 +^n P_2 +^n P_3 +^n P_4 .... +^n P_n$.

Had the order of the services with in a stack not mattered then number of stacks possible are

$^nC_1 +^n C_2 +^n C_3 +^n C_4 .... +^n C_n = 2^n$.

Since the algorithm is a brute-force algorithm, with some optimizations, the 'no repetitions' variation is atleast $\Omega(2^n)$ and atmost $O\sum^n P_r r \rightarrow$\{1,2,...n\}

5.2.1 Example Stack composition

An ontology in which TCPHandshake is the opening service and TCPChecksum is the closing service, and total number of services are 9 and with 4 constraints, apart of the algorithm out put is shown below.

```
New Silo:
silo:TCPHandshake , silo:IPv4SegReassy , silo:OrderedDelivery ,
silo:TCPReliableDelivery , silo:TCPInOrderDelivery , silo:TCPChecksum ,

New Silo: silo:TCPHandshake , silo:IPv4SegReassy , silo:OrderedDelivery ,
silo:TCPReliableDelivery , silo:TCPInOrderDelivery , silo:TCPSegmentation ,
silo:TCPChecksum ,

New Silo:
silo:TCPHandshake , silo:IPv4SegReassy , silo:OrderedDelivery ,
```

silo:TCPReliableDelivery , silo:TCPInOrderDelivery , silo:TCPSegmentation , silo:TCPFlowControl , silo:TCPChecksum ,

New Silo: silo:TCPHandshake , silo:IPv4SegReassy , silo:OrderedDelivery , silo:TCPReliableDelivery , silo:TCPInOrderDelivery , silo:TCPSegmentation , silo:TCPFlowControl , silo:TCPCongestionControl ,
silo:TCPChecksum ,

New Silo: silo:TCPHandshake , silo:IPv4SegReassy , silo:OrderedDelivery , silo:TCPReliableDelivery , silo:TCPInOrderDelivery , silo:TCPSegmentation , silo:TCPFlowControl , silo:TCPCongestionControl ,
silo:IPv4Checksum , silo:TCPChecksum ,

New Silo: silo:TCPHandshake , silo:IPv4SegReassy , silo:OrderedDelivery , silo:TCPReliableDelivery , silo:TCPInOrderDelivery , silo:TCPSegmentation , silo:TCPFlowControl , silo:TCPCongestionControl , silo:IPv4Checksum , silo:IPv6SegReassy , silo:TCPChecksum ,

5.3 Stack composition with repetitions and unrestricted length

Unrestricted length and unrestricted repetitions clearly result in infinate possibilities. The-ordering constraints can only reduce some of the possibilities. With restricted repetitions, the problem becomes deterministic.

For example the number of possible stack compositions to be verified for validity in the case when at most one reprtitions of a service is allowed, is ${}^{2n}P_1 + {}^{2n}P_2 + {}^{2n}P_3 + {}^{2n}P_4 .... + {}^{2n}P_{2n}$. This is equivalent to the case where there are ‘2n’ services and no repetitions are allowed.

5.4 Stack composition with repetitions and restricted loops

In this case a service can appear any number of times. But a particular loop of services can loop only a specified number of times. This is a tricky situation, since in the algorithm, even if an addition of a repeated service to the stack doesnot result in a repeated loop, part of the stack that includes the last added service can be considered and any of its previous instance can be considered as the first appearance of a loop. The loop restriction algorithm is presented in the appendix.

The complexity or running time of this algorithm is non-deterministic. Even with a restriction on the number of times a loop can be repeated, say no loop repetitions are allowed, the every first possible SILO realization can take a non-deterministic time. Cosider a subset of services with no ordering constraints. Number them as 0,1, 2, 3 ..and so on.

For instance let the number such services be 10. A non deterministic series of these ten symbols can be generated. For example consider Pi.It is an irrational number, which means that its decimal expansion never ends or repeats. Indeed, beyond being irrational, it is a transcendental number, which means that no finite sequence of algebraic operations on integers could ever produce it. Such transcendental series can be generated even with two symols (or digits), since Pi (or any transcendental number) can be realised even in Binary number system.

Since no sequence of algebraic operations on these integers can ever produce such series of symbol,no formal language can be defined. Thus the number of steps in the algorithm and hence the running time can be concluded to be non-deterministic.

Chapter 6

Future Work

6.1 Peering Silo Construction

Silo composed on one end of a connection confirms to the global ontology, local polocies and is dependenton the availability of the services an the end. An identical silo may not be possible atthe remote end. This necessiates peering the Silo ie, negotictaing the SILO reciepe by both the ends. Also, at every hop a partial SILO needs to constructed. While neotiating the SIloreciepe across the network, the following things need to be considered:

1. End to End silo peering Vs Hop by Hop (E2E Vs HbH)
2. Non-homogeneous network
3. Different addressings/Address translation
4. Standard Bootstrap Silo(B-Silo). Is a universal B-Silo possible?
5. Does every silo need a peer-silo negotiation (before sending data)?
6. Does every service on a silo need a peer (E2E or HbH)?
7. Negotiation Similar to HandShake.

6.1.1 End to end silo peering Vs Hop by Hop ((E2E Vs HbH))

E2E silo peering makes heavy assumptions about the capabilities of intermediate nodes. for E2E to be sufficient every intermediate node should be able to build any silo (or that part of the negotiated Silo that is needed for correct forwarding).

In case of HbH peering, there is considerable overhead introduced. Here every node negotiates with the node at next hop. This means that the silos constructed at the either ends need not be identical.

6.1.2 Non-homogeneous network

If the network is not homogeneous, and if the two ends are in different networks, the peer silos cannot be identical. for example, if the source end is in wired n/w and if the destination is in wireless network, the service (or method of the service) that serves Data Link Layer kind of services cannot be the same at both the ends.

6.1.3 Different addressing

If the addressing for the source and destination are different, then again the peer silos cannot be same. This is similar to the non-homogeneous network case. Silo needs to include address translation on some intermediate node.

6.1.4 Bootstrap Silo

for peer silo negotiation to happen, there should be some way of communication. This is enabled by a bootstrap silo. However, for the same reasons as above (and below) a universal bootstrap silo is not possible.

6.1.5 Does every silo need a peer-silo negotiation

If the packet headers contain the silo information, then the intermediate nodes and the endnode can attempt to construct a 'compatible' silo. But, if there are any QoS, or application req constraints, then such the intermediate/end nodes should be mind ful of such constraints.

This means there is a default silo used temporarily to read the header info and the constraints in the first packet. The node at the next hop makes a best effort attempt to construct a compatible silo and forwards a similar packet to the next hop.

Should the next hop deny, how many alternate next hops should be tried?

6.1.6 Does every service on a silo need a peer (E2E or HbH)?

In HbH, Source and next hop, and Destination and previous hop cannot negotiate on all services in a silo since the intermediate nodes need to construct only on a part of the silo. The actual question is should every service have a peer either at next hop or at the other end?

We think not all services need to be peered.

1. Services that modify the data or the header need to be peered.

2. Services that generate control messages (that help other nodes on the 'flow' tune services/methods) need to be peered

6.1.7 Default silo

1. As discussed above, when no negotiation is required, a default silo is needed to read the header info and the constraints in the first packet. The node at the next hop makes a best effort attempt to construct a compatible silo and forwards a similar packet to the next hop.

2. Boot Strap silo on each node, that enables peer Silo negotiation (three way handshake).

3. The default silo should facilitate the node to communicate to other nodes, its participation in the network.

6.1.8 Hop By Hop peering

Assuming that all the nodes have a bootstrap Silo (B-Silo), the peer negotiation can happen as below:

The source constructs a silo S_s and communicates it to the next hop N_{h1}. The next hop builds the silo peering with minimum number of services on the Silo S_s so that it can forward the packet to the second hop N_{h2}. This means the services below the first forwarding silo is peered.

If the Node N_{h1} acts as a gateway for two different types of networks, then it adds more services corresponding to the different network. If the 'other network' is a sonet/atm kind of network, the silo will have multiple forwarding services.

Consider Ip over Sonet.

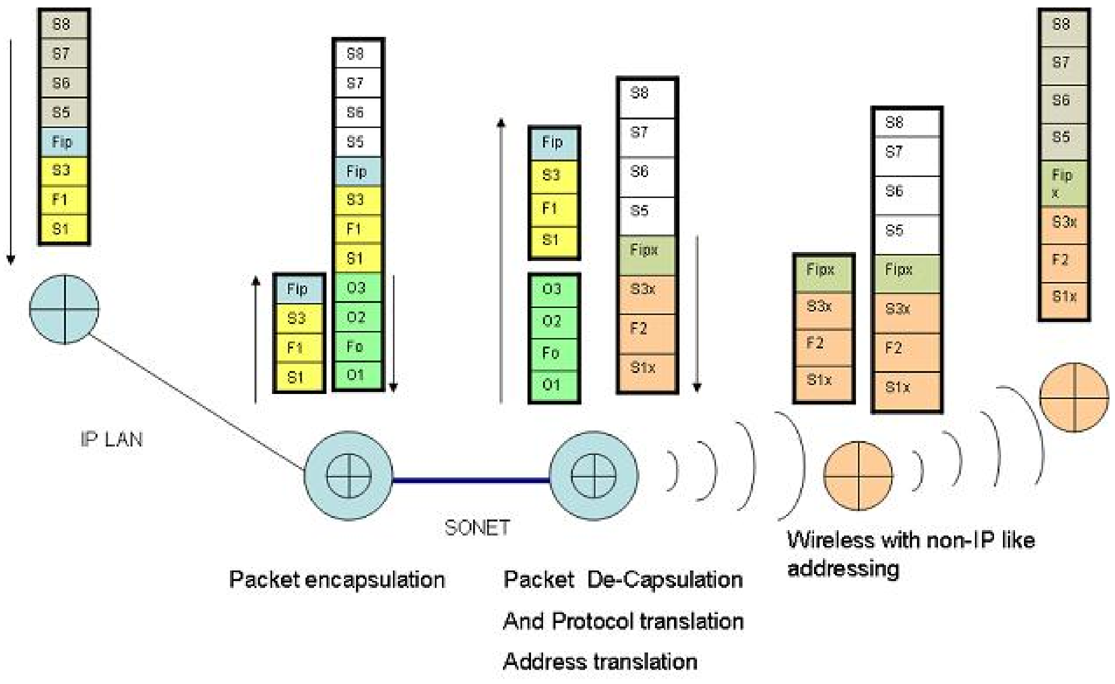


Figure 6.1: Peering Silos in a Non-homogeneous network

For the Proof of Concept model, TCP/UDP like Silo is stacked over IP.

6.2 Shortest SILO First

Algorithm to geneate shortest SILO first and then sucessive shortest SILO is one areas of future research focus. This problem sounds similar to sucessive shortest path algorith bit is diffrent and complex due to the inherent dynamic nature of the graph representation of the ontology. The Ontology can be represented as graph with multiple cloured edges, and traversing any edge can result in modification of the graph due to the constraints. Since the graph possibly changes on every edge traversal, a decision made at one step of a graph algorithm can no more be valid. Even a sigle shortest pathalgorithm like Dijkstra's algorithm proves to be futile.

Bibliography

[1] I. Baldine, G. N. Rouskas, H. G. Perros, and D. Stevenson. “JumpStart: A just-in-time signaling architecture for WDM burst-switched networks”. IEEE Communications Magazine, February 2002.

[2] Anjing Wang. “Cross-Service Tuning and Optimization for Networking Services:The SILO System”. Phd qualifer exam, NCSU, 2007.

[3] B. Ahlgren, M. Brunner, L. Eggert, R. Hancock, and S. Schmid. “Invariants: a new design methodologyfor network architectures”. In Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA-04), pages 6570, Portland, 2004.

[4] Gigabit Ethernet Alliance. “Introduction to TCP/IP offload engine (TOE)” version 1.0. 10GEA White Paper, April 2002.

[5] C. Assi, A. Shami, M. Ali, R. Kurtz, and D. Guo. “Optical networking and real-time provisioning: An integrated vision for the next-generation internet”. IEEE Network, July/August 2001.

[6] P. Balaji, H. V. Shah, and D. K. Panda. “Sockets vs RDMA interface over 10-Gigabit networks: An indepth analysis of the memory traffic bottleneck”. In Proceedings of the RAIT Workshop, September2004.

[7] H. Balakrishnan, , V. N. Padmanabhan, S. Seshan, and R. Katz. “A comparison of mechanisms for improving TCP performance over wireless links”. In Proceedings of the 1996 ACM SIGCOMM Conference,pages 256-269, August 1996.

[8] N. T. Bhatti and R. D. Schlichting. “A system for constructing configurable

high-level protocols”. In Proceedings of the 1995 ACM SIGCOMM Conference, pages 138-150, Cambridge, MA, August 1995.

[9] R. Braden, T. Faber, and M. Handley. “From protocol stack to protocol heap - role-based architecture”.ACM Computer Communication Review, 33(1):17-22, January 2003.

[10] K. Brown and S. Singh. “M-TCP: TCP for mobile cellular networks”. Computer Communications Review,27(5):19-43, October 1997.

[11] R. Caceres and L. Iftode. “Improving the performance of reliable transport protocols in mobile computing environments”. IEEE Journal in Selected Areas on Communications, 13(5):850-857, June 1995.

[12] J. S. Chase, A. J. Gallatin, and K. G. Yocum. “End system optimizations for high-speed TCP”. IEEE Communications Magazine, 39(4):68-74, April 2001.

[13] D. D. Clark, R. Braden, A. Falk, and V. Pingali. “FARA: Reorganizing the addressing architecture”. In Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA-03), pages 313-321, Karlsruhe, Germany, 2003.

[14] D. D. Clark, K. Sollins, J. Wroclawski, and T. Faber. “Addressing reality: An architectural response to real-world demands on the evolving internet”. In Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA-03), pages 247-257, Karlsruhe, Germany, 2003.

[15] D. D. Clark, J. Wroclawski, K. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow’s internet. In Proceedings of the 2002 ACM SIGCOMM Conference, pages 347-356, Pittsburgh, PA,August 2002.

[16] S. Floyd. “HighSpeed TCP for large congestion windows”. IETF Internet Draft, 2003.

[17] N. Hutchinson and L. Peterson. “The x-kernel: An architecture forimplementing network protoccols”.IEEE Transactions on Software Engineering, 17(1):64-76, 1991.

[18] C. Jin, D. X. Wei, and S. H. Low. “FAST TCP: Motivation, architecture, al-

gorithms, performance". In Proceedings of the 2004 IEEE INFOCOM Conference, pages 2490-2501, Hong Kong, March 2004.

[19] D. Katabi, M. Handley, and C. Rohrs. "Tussle in cyberspace: Defining tomorrow's internet". In Proceedings of the 2002 ACM SIGCOMM Conference, pages 89-102, Pittsburgh, PA, August 2002.

[20] M. Kuznetsov et al. "A next-generation optical regional access network". IEEE Communications, 38(1):66-72, January 2000.

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

[22] S. K. Moore. "Multimedia monster". IEEE Spectrum, 43(1):20-23, January 2006.

[23] S. O'Malley and L. Peterson. "A dynamic network architecture". ACM Transactions on Computer Systems, 10(2):110-143, May 1992.

[24] C. Qiao and M. Yoo. "Optical burst switching (OBS)-A new paradigm for an optical Internet". Journal of High Speed Networks, 8(1):69-84, January 1999.

[25] V. T. Raisinghani and S. Iyer. "Cross-layer feedback architecture for mobile device protocol stacks". IEEE Communications Magazine, 44(1):85-92, January 2006.

[26] D. Schmidt, D. Box, and T. Suda. "ADAPTIVE: a dynamically assembled protocol transformation,integration, and evaluation environment". Concurrency: Practice and Experience, 5(4):269-286, June 1993.

[27] V. Srivastava and M. Motani. "Cross-layer design: A survey and the road ahead". IEEE Communications Magazine, 43(12):112-119, December 2005.

[28] D. D. Clark et al. Newarch project: "Future-generation internet architecture".

[29] J. Wang, L. Li, S. H. Low, and J. C. Doyle. "Cross-layer optimization in TCP/IP networks". IEEE/ACM Transactions on Networking, 13(3):582-595, June 2005.

[30] W-F. Wang, J-Y. Wang, and J-J. Li. "Study on enhanced strategies for TCP/IP offload engines". In Proceedings of the 11th International Con-

ference on Parallel and Distributed Systems (ICPADS ’05),pages 398-404, 2005.

[31] R. Winter, J. H. Schiller, N. Nikaein, and C. Bonnet. “CrossTalk: Cross-layer decision support based on global knowledge”. IEEE Communications Magazine, 44(1):9399, January 2006.

[32] Y. Wu, P. A. Chou, Q. Zhang, K. Jain, W. Zhu, and S-Y. Kung. “Network planning in wireless ad hoc networks: A cross-layer approach”. IEEE Journal on Selected Areas in Communications, 23(1):136-150,January 2005.

[33] L. Xu, K. Harfoush, and I. Rhee. “Binary increase congestion control (BIC) for fast long-distance networks”. In Proceedings of the 2004 IEEE INFOCOM Conference, pages 25142524, Hong Kong,March 2004.

[34] G. Xylomenos, G. C. Polyzos, P. Mahonen, and M. Saaranen. “TCP performance issues over wireless links”. IEEE Communications Magazine, 39(4):5258, April 2001.

[35] X. Yang. “NIRA: A new internet routing architecture”. In Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA-03), pages 301-312, Karlsruhe, Germany, 2003.