Report of the DIMACS Working Group on Abstractions for Network Services, Architecture, and Implementation

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

Pamela Zave
AT&T Laboratories–Research
pamela@research.att.com

This article is an editorial note submitted to CCR. It has NOT been peer reviewed.
The authors take full responsibility for this article's technical content. Comments can be posted through CCR Online.

ABSTRACT

A workshop on Abstractions for Network Services, Architecture, and Implementation brought together researchers interested in creating better abstractions for creating and analyzing networked services and network architectures. The workshop took place at DIMACS on May 21-23, 2012. This report summarizes the presentations and discussions that took place at the workshop, organized by areas of abstractions such as layers, domains, and graph properties.

Categories and Subject Descriptors

C.2.0 [**Computer-communication Networks**]: General; C.2.1 [**Computer-communication Networks**]: Network Architecture and Design

Keywords

Abstractions, architecture, services, education

1. INTRODUCTION

A field of computer science progresses fastest when its researchers find the right abstractions. Progress comes from the many benefits of abstraction, including:

- Abstractions embody knowledge about the field, including vocabulary and principles, making it easier to talk about and teach.
- Abstractions provide modularity and separation of concerns, without which the complexity of the subject becomes overwhelming.
- Abstractions hide implementation details, so that good designs can be generalized and turned into reusable software.
- Abstractions can be formalized, leading to benefits such as automated reasoning, optimization, and composition of components.

In comparison to other fields of computer science, networking is in an unusual position because of the Internet. No other field has an artifact so dominant, and one that by its very nature demands global interoperability. Consequently, in no other field is it as challenging to envision and experiment with new abstractions.

Nevertheless, many people feel that the time is ripe for the field of networking to invest more effort in abstraction. Keshav points out that the recent emphasis on clean-slate design has exposed the paucity of useful abstractions [8]. Shenker has said that networking still focuses more on mastering complexity than on extracting simplicity [12]. Looking at the literature of networking, people could be forgiven for believing that networking is just a plethora of protocols, a heap of header formats, a big bunch of boxes, or a ton of tools [9].

With these issues in mind, we organized a workshop on abstractions for networking. It was held from noon to noon 21-23 May 2012 at Rutgers University in New Brunswick, New Jersey, with the generous support of the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS).

Our hope in organizing this workshop was to foster interest in abstractions for networking as a first-class topic of research, one that might work synergistically with all the various "next generation" goals and projects. The title "Abstractions for Network Services, Architecture, and Implementation" expresses our conviction that successful abstractions must cover, or at least recognize, concerns at all the levels touched by networking.

For the program, we organized talk sessions on the following topics:

- Applications
- Data-Plane Abstractions
- Software-Defined Networks
- Transport, Consistency
- Middleboxes, Failure Detection
- Routing

We scheduled approximately half the workshop time for discussion, organized around these three questions:

1. Why are abstractions needed?
2. What abstractions are needed?
3. How can research on network abstractions be fostered and helped to converge?

Answers to Question 2 are needed to link requirements for abstractions (Question 1) with network mechanisms and properties of formalisms. Answers to Question 3 are needed to make room for a kind of research that may not conform to the cultural norms of the networking community.

The participants in the workshop were Azer Bestavros, David Clark, Mary Fernandez, Bryan Ford, Nate Foster, Brighten Godfrey, Tim Griffin, Aaron Jaggard, S. Keshav, T. V. Lakshman, Boon-Thau Loo, Yun Mao, Vyas Sekar, Anees Shaikh, Scott Shenker, Robert Soule, David Rosenblum, Michael Walfish, David Walker, Alexander Wolf, and Rebecca Wright. Naga Praveen Katta served as a scribe.

In this report we do not attempt to summarize the talks, which were excellent, and which can be found on the Web (see bibliography). Rather, we attempt to capture a bit of the flavor of the talks and discussions together, by pulling out some of their themes. Each theme was represented in multiple talks and the discussion, so that we know it is important and got several perspectives on it. In the following sections, names and references indicate the general content of talks. But what we have written is a blend of multiple talks and discussions, and statements should not be attributed to specific persons (except perhaps us).

2. THE CHALLENGES OF ABSTRACTION

Not surprisingly, many people spoke on why it is so difficult to formulate successful abstractions. At the level of application support, David Rosenblum talked about publish/subscribe systems, which have a very general functional abstraction that is useful for a wide range of Internet applications [10]. Implementing the abstraction at Internet scale, however, requires many difficult trade-offs. After the decisions are made, the result is an implementation that matches the operating characteristics of only a small subset of the original range of applications. This subset may be too small to justify the overall effort, while for applications outside the subset, the implementation performs poorly or is not cost-effective. The same phenomenon has been observed with multicast systems and (arguably) DHTs.

Bryan Ford's talk focused on the difficulty of abstracting transport protocols—let alone designing and deploying them—because they are currently at the center of a vortex of conflated concerns [3]. Transport protocols must be application-oriented in providing well-defined application interfaces, in defining the units of data movement and reliable transmission, and in providing secure transmission. Transport protocols must be network-oriented because they define the units of rate control and resource sharing, and consequently interact with routing. Transport protocols also interact with firewalls, NAT boxes, endpoint naming, and mobility.

David Clark observed that since networks deviate from perfect performance because of real-world impairments, an *efficient* network is one that has only fundamental impairments, and a *general* network is one that allows its users to choose trade-offs among impairments [1]. TCP alone does not provide a general transport service, because it fixes the trade-offs. A multiplicity of different services seems better in this regard, but it is dangerous to specify them with performance guarantees. Research has shown that, for a performance guarantee to be achievable in a general-purpose network, it must be very unattractive. Attempts to enforce guarantees in such contexts usually lead to silly behavior.

On a more positive note, a high-level abstraction helps us reason modularly and reusably about a network function, independently of its implementation. A general-purpose implementation of the function would allow its users to choose their cost and performance trade-offs. Even if the function does not have a single general-purpose implementation, it can have multiple implementations to cover the design space.

People believe that probabilistic performance models are the key to achieving reasonable bounds at reasonable cost, and that much more research on them is needed to improve the flexibility of our implementations.

3. LAYERS

Layers are the mother of all network abstractions. The layered abstraction of the data plane in which the physical layer moves bits, the link layer provides best-effort local packet delivery, the network layer provides best-effort global packet delivery, the transport layer provides reliable (or unreliable) transport, and the application layer provides applications and mnemonic names, has been a major factor in the success of the Internet so far [12].

Despite this success, the five-layer model has many shortcomings as an abstraction of the current and future Internet. The slow pace of migration to IPv6 shows that it has not (at least as implemented) provided enough modularity. Inspection of packet headers shows that it is a gross oversimplification of how real Internet traffic is handled. Each layer is global and its abstraction would ideally have performance properties as well as logical properties, yet all the evidence (as noted above) suggests that this is infeasible.

An alternative presented by Pamela Zave is the description of Internet architecture as a hierarchy of layers without a fixed number of levels [14]. Layers have varying scopes, so a layer need not be global or universal. Instead of providing a particular function, each layer is a microcosm of networking, containing (at least in principle) all the major functional components such as a name space, a session protocol, a routing protocol, and a forwarding protocol. Because layers appear at many different levels of the hierarchy, with many different scopes, and for many different purposes, their versions of each of the basic layer components also vary accordingly. This "geomorphic" view can, at least, describe what the Internet is really doing.

For example, Bryan Ford spoke of the need to break the transport layer into sublayers (semantic, end-to-end security, flow, endpoint) to separate some of the conflated concerns [3]. For each sublayer, there are many possible choices. A particular combination of choices (*i.e.,* sublayer instances) can be seen as forming a "geomorphic" layer that is used by some applications and not by others. A different set of endpoints, communicating for the purpose of a different application, could use a different layer that combines a different set of choices.

Keshav's talk on universal switching machines (USMs) takes this idea further [7]. As in the geomorphic view, each layer contains the same basic functions as other layers, just customized and instantiated for different purposes. In the USM view, in addition, each layer consists only of USMs, each of which is a programmable machine with a prescribed set of capabilities. This provides much more structure, with which fundamental questions of reachability, naming, and state creation can be explored and reasoned about.

4. DOMAINS

Another important concept, and one that has not received enough attention as a basis for abstraction, is domains. In relation to the classic five-layer model, domains have vertical boundaries while layers have horizontal boundaries. Domains include trust domains [1] and autonomous domains (defined by ownership). There are also domains defined by capability or interoperability, *e.g.,* the domain of nodes that understand a specific protocol. In the geomorphic view layers have both horizontal and vertical boundaries, so the relationship between geomorphic layers and trust domains is not yet clear.

David Clark pointed out that, while encryption is a good security tool, it is only useful to protect communication between two parties that trust each other—an increasingly rare scenario. Communication between mistrustful parties is communication between trust domains, and requires additional security safeguards such as virus detection.

Dependability (in the broad sense) is another issue that may best be studied from the viewpoint of domains. For example, Michael Walfish argued that today's failure-detection mechanisms hide too much information from the end hosts, making it difficult for applications to respond effectively, making their own decisions about whether and how to recover [13]. He also pictured failure-detection mechanisms as orthogonal to layers and funneling multi-layer reports to applications. To reduce the overhead for conveying fine-grained failure information, he proposed piggybacking failure information on existing protocols.

This argument has several aspects related to domains. Defining, monitoring, and reporting failures all have security implications. Extending protocols to convey failure information has capability implications. In addition, the foundation of availability, beyond failure detection, is failover to trustworthy alternative components. Without trust domains, there is no sound basis for choosing failover components.

At one point it was proposed that all security attacks be characterized as "above" or "below" the security abstraction. If domains turn out to be the major abstraction for security, then the partition should be amended to "outside" or "inside" the security abstraction.

Vyas Sekar's talk on middleboxes presented a practical perspective on trust domains, such as those corresponding to ISPs and enterprise networks [11]. These domains are full of middleboxes supporting valuable services. Yet legitimate users who are not members of these domains may have no access to them. Even within the domains, the middleboxes present many challenges with respect to configuration and resource management.

5. GRAPH PROPERTIES

The core network functions—routing, forwarding, monitoring, and access control—are already better-understood and more general than the functions closer to applications. Discussion of these functions tends to be based, whether implicitly or explicitly, formally or informally, on properties of graphs and paths through graphs.

Scott Shenker's talk on Software Defined Networking (SDN) argued for a global network view in the form of an annotated graph (constructed from distributed state with different consistency and durability requirements). The graph can then support other useful abstractions. For example, an access-control policy can be expressed by abstracting a subnetwork as one big virtual switch to simplify configuration.

The graph abstraction is a foundation for defining path metrics such as length and width. It is also a foundation for defining qualitative properties of sets of paths, including connectivity, no forwarding loops, no security holes, no black holes, and waypointing. These are abstractions of proven value, and several speakers talked about how to build on these foundations.

In addition to proposing an abstraction for SDN, Shenker's talk discussed ways to confront the scalability challenges of managing large wide-area networks. He proposed a hierarchical representation of the network topology, where portions of the topology are combined together into a single logical node (and its external ports) at the next level. Each group of nodes has a logical SDN controller that installs the forwarding state in these nodes (based on a single forwarding table from the parent controller) and provides an aggregated view of network events and traffic statistics about the group to the parent controller.

Nate Foster's talk [4] proposed programming abstractions for software-defined networks, with an emphasis on abstractions for updating a network-wide forwarding policy consistently. In a *per-packet* consistent update, every packet follows either the old policy or the new policy for every hop in its journey, rather than some mixture of the two policies. Foster presented several ways to implement consistent updates using OpenFlow-compatible mechanisms. The talk also described how today's OpenFlow API makes it difficult to compose multiple software modules into a single application, and presented higher-level programming abstractions that improve modularity.

Tim Griffin's talk [6] focused on how to separate *what* path is computed, which is usually an optimal path with respect to some metric (*e.g.,* shortest, widest), from *how* a routing protocol computes it. This is valuable because of the mathematical theory of semirings, which shows that if the *what* has certain properties, the *how* can be accomplished with generic algorithms. The MetaRouting project exploits this by offering a language of combinators for specifying complex metrics, combined with a library of verified algorithms for routing based on these metrics.

Despite its promise, the theory of semirings has current limitations requiring further research. A focus on global (rather than local) optimality is too restrictive. Some realistic metrics do not have the algebraic properties on which semiring theory relies. The relationship between routing and forwarding may not be as straightforward as is usually supposed.

Brighten Godfrey's [5] talk on abstractions for network routing focused on how to create routing systems that are flexible and extensible, and how to compare different architecture quantitatively. Godfrey discussed how *pathlet* routing, which stitches together path segments into an end-to-end path, can capture a wide range of routing architectures from source routing to today's BGP, and a variety of solutions in between. Pathlet routing also gives network owners the flexibility to define how their networks can be used.

6. PRESERVING AND PROMOTING ABSTRACTIONS

If good abstractions are hard to come by and the current Internet does not have enough of them, it follows that we need to understand how to preserve and promote abstractions. Although the working group did not come to any consensus on the best ways to preserve and promote abstractions, some general themes emerged from the discussions.

One principle for preserving abstractions was a subtext of almost every talk: "Design for variation in outcome, so that the outcome can be different in different places, and the tussle takes place within the design, not by distorting or violating it" [2].

Brighten Godfrey introduced a quantitative model of evolvability that may help us think about how to promote new abstractions. His model uses quantities such as percentage of nodes deploying a new protocol, the probability of an attack (in the sense of rejection) on a new protocol, utility to deployer, and deployment cost. Simulations provide insight into the relationships among these quantities.

Everyone agreed that education is an important means of promoting abstraction, and would welcome more use of abstraction in the networking curriculum. There may also be other ways to communicate the value of abstractions. For example, graduate students and postdocs know they need help with designing valid experiments. A design for an experiment is all about controlling some factors and varying other factors; the controlled factors can be thought of as an abstraction, while the varying factors represent its range of possible implementations.

It was felt that theory shines most brightly when it confirms and explains good practical intuitions. This kind of theory results from close cooperation between theoreticians and practitioners. In the field of databases, PODS and SIGMOD are always co-located; perhaps the field of networking could find similar ways to foster such productive cooperation.

7. CONCLUSION

The workshop showed that there is growing enthusiasm for abstractions of network services, architecture, and implementation. We cannot say, however, that this enthusiasm has taken a coherent shape as yet. We hope that the participants and others will continue discussing these issues, so that in a year or two another meeting on the subject will give evidence of much progress.

Acknowledgments

We would like to thank the participants for their enthusiasm, including those who encouraged us but could not, in the end, attend the meeting.

We would also like to thank the DIMACS staff for their splendid organization and their warm welcome to our participants.

8. REFERENCES

[1] D. D. Clark. Applications and abstractions. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/Dave_Clark.pdf`.

[2] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475, June 2005.

[3] B. Ford. How should we think about transport abstractions? `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/Bryan_Ford.pdf`.

[4] N. Foster, J. Rexford, and D. Walker. Abstractions for software-defined networking. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/Nate-dimacs.pdf`.

[5] B. Godfrey. Abstractions for network routing. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/godfrey-DIMACS.pdf`.

[6] T. Griffin. Algebraic path finding. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/Tim_Griffin.pdf`.

[7] S. Keshav. Net working? `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/Keshav.pdf`.

[8] S. Keshav. Editor's message: Modeling. *Computer Communication Review*, 42(3):3, July 2012.

[9] J. Rexford. The networking philosopher's problem. *Computer Communication Review*, 41(3):5–10, July 2011.

[10] D. S. Rosenblum. Applications and abstractions: A cautionary tale. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/DSRDIMACS.pdf`.

[11] V. Sekar and S. Ratnasamy. Abstractions for middleboxes. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/Vyas-dimacs.pdf`.

[12] S. Shenker. The future of networking, and the past of protocols. `http://opennetsummit.org/talks/shenker-tue.pdf`.

[13] M. Walfish. Failure detection as a network abstraction for end-host applications. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/dimacs12-walfish-talk.pdf`.

[14] P. Zave. Abstractions of the data plane. `http://dimacs.rutgers.edu/Workshops/NetworkServices/Slides/absDataPlaneTalk.pdf`.