Towards Networks as Formal Objects (Position Paper)

Timothy G. Griffin (AT&T Research), Randy Bush (Internet Initiative Japan, IIJ)

Abstract

The success of the end to end principle has led to a commonly held fallacy that the Internet core is simple and stupid. A corollary to this is that there is nothing interesting here for networking research. Our experience with the overwhelming difficulty of managing and operating core functionality tells us that exactly the opposite is true. At the heart of the problem is a lack of network level models and abstractions. The evolution of appropriate models and abstractions requires a deep understanding of the data networking domain. We claim that addressing these problems should be one focus of the networking research community.

1 Using Tweezers to Operate and Manage the Internet Core

The end to end argument [6] has been a very successful design principle for the Internet. This principle tries to minimize the functionality implemented in the core of the network, locating most functionality at the edge. This contrasts sharply with traditional telco networks where all important functionality is implemented in the network and the edge consists of very simple access devices.

By "internet core" we mean this architectural core, not what might be called the "provider core". The provider core consists of commercial and noncommercial entities that provide services that fall within the architectural core, such as transit services, and services that fall within the architectural edge, such as email and web hosting. Architecturally, the internet core functionality currently revolves around transport (packet forwarding) and routing. However, in campus networking, corporate networking, and networking within and between Internet Service Providers (ISPs), the task of realizing this core functionality is more complex than a simple architectural diagram might suggest. Networks must be *designed*, *deployed*, *configured*, *protected*, and *monitored*. Today these tasks are difficult, expensive, labor intensive, and prone to many types of error (for example, configuration errors in BGP are fairly common [13]).

Network operators are overwhelmed with complexity. They are trapped at the device level when talking about network design, network measurements, network monitoring, and network security. They must deal with low level configurations of proprietary interfaces. They are forced to configure devices when they should be configuring networks. The situation is likely to get worse before it gets better. For example, many ISPs must manage core functionality as well as edge functionality that has been

outsourced to them. In addition, the core will become more and more complex as additional functionality is added there (for example, trust management [5]).

The success of the end to end principle has led to a commonly held fallacy that the Internet core is simple and stupid. A corollary to this is that there is nothing interesting here for networking research. Our experience with the overwhelming difficulty of managing and operating core functionality tells us that exactly the opposite is true. We believe that at the heart of the problem is a lack of network level models and abstractions. The evolution of appropriate models and abstractions requires a deep understanding of the data networking domain. In other words, we hold that much more is involved than mere systems integration. We claim that addressing these problems should be one focus of the networking research community.

2 Need for Network Abstractions

When we look at the maturation path of many areas in Computer Science we see remarkable similarity. Areas as diverse as databases, programming languages, operating systems, and cryptography have all progressed along a familiar road. Each field begins in implementation muddle marked by proprietary solutions and a rather messy collection of low-level tricks. Then, slowly, this muddle begins to crystalize around a small number of key abstractions. Vendors resist, but the advantages of abstractions are too compelling — the benefits of portability and standardization start to outweigh the small gains in efficiency made possible by proprietary hacks. A new field is really born when its focus becomes the study of these new abstractions and their efficient implementation. The abstractions provide the vocabulary used by the specialist. What was a muddle is refined into the collection of "smoke-and-mirrors" techniques that allow implementation of the abstractions. Vendors get on board and modify their hardware and software to facilitate the implementations of the abstractions. The illusion is often so successful, that even technically astute professionals forget about what goes on behind the curtain. It all comes to seem so natural, so straightforward, that people slowly forget there ever was any confusion.

We believe that implementation, design, and management of the internet core is still in the muddle phase of its development. Great strides have been made in the development of abstractions related to network protocols. But we are still stuck in a device-level muddle in the core. We can configure devices, but we cannot configure networks. We have few network level abstractions and absolutely no network level programming languages. When we configure networks today, we must perform the moral equivalent of manual register allocation. For the field to mature we must isolate and develop the right network-level abstractions.

We would like to think of current vendor specific configuration languages in the way compiler writers think of machine languages. That is, as something we want to move away from as quickly as possible. We want to think about the *configuration* of standardized protocols (BGP, OSPF, SNMP, RADIUS) in vendor independent ways. But we view this as still very much at the level of assembly languages. To program *networks*, we need the equivalent of high level languages that treat networks as objects that arise out of the composition of other, smaller, objects. The compiler's job is then to translate such specifi-

cations into protocol specifications.

To build bigger, better, more secure, networks, we need higher levels of abstraction. We need to be able to layer abstractions and have the ability to move through levels of abstraction. And we need to do this with far more formal rigor.

The fact that little progress has been made in this area is not due entirely to a lack of awareness on the part of the research community. There has been a lack of communication and visibility concerning these problems since they are often in the domain of service providers and large organizations without strong ties to the research community. Even within those rare providers supporting research there are often organizational impediments to a more unified treatment of these issues. For example, a provider may evolve different organizations to handle fault management, routing management, and traffic measurements, thus inadvertently creating barriers to a more unified approach. There is also a natural resistance to abstraction from device vendors and the specialists that configure these devices. Put simply, vendors are not interested in making it easy to abstract away from their devices. Any move in this direction decreases an operator's dependence on any given vendor. In addition, many vendors have instituted very successful training and certification programs for configuration specialists that in effect tie job security to mastery of obscure, low level, vendor specific implementation details. However, there is something of a chicken and egg problem here, since this level of management is currently *required* given our lack of higher level abstractions. Simple standards often require good theory, but in the absence of theory, sufficient social and business pressure will always produce *something that works*, no matter how complex, unscalable, or expensive.

3 Looking for Guidance

We now look for interesting work in various fields which seem as if they might lead us toward useful paths on our search.

3.1 Programming Language Theory and Concepts

Clearly, the field of programming languages has much to offer. Programming languages give us higher level abstractions that can be composed and reasoned about. They provide mechanisms for information hiding and modularity. We should think about domain specific languages that capture the essential components of networking. Then compilers can be implemented to translate high level specifications into low level implementations. This may require different languages for different layers of abstraction.

Although networks are distributed systems, we think this challenge should be much easier than that of general distributed systems programming. The reason is that core functionality is restricted to supporting small number of core functions. The network-specific languages may be simple enough that we can reason about them effectively. In fact, most network configurations should be entirely declarative. Logival constraints on what should hold true in a network could themselves be used to drive automated configuration.

This could be extended to network monitoring as well. Constraint-based models of the network could be used to form assertions about allowable states of the network which could then be used to configure network measurement systems to monitor those assertions and take action if they are stressed or fail.

While we hope that formal languages, mod-

els, and constraints will allow us to deal with networks at a useful and scalable level of abstraction, we have the 'unfortunate' reality of complex network devices such as routers. Their complexity is multi-dimensional and often produces unexpected results. Therefore, describing them formally as they exist today is too often not tractable.

Hence, we are inclined to see complex devices as black boxes at the edge of the formal model. The next step would be to consider their behavior exogenous to the constrainable model, which would be a serious detriment to formalizing network management.

But what if we were to formalize the constraints that the network model(s) need to place on these black boxes? I.e., consider the network-imposed constraints to be specification and design criteria for the development of complex network devices.

This path leads us to consider the design of such devices as formal software engineering, which could be a very good thing. And this would allow us to make assertions about the parameters of their reliability.

3.2 Roles and rules of Behavior

An example of a language-based approach is Firmato [3], a firewall management toolkit. Firmato has a simple language for defining network-wide policies for access security, which is compiled into an instance of a relational database model. This model is then compiled into vendor-specific firewall configurations.

This is a very powerful paradigm and it is not a large leap to imagine it applied to the network as a whole. It may then be possible to reason about the correctness of the specification and proving that a compiler implements specifications correctly in the network. And, as is done in the area of chip design, it may be possible to check various design rules against a network once it has been composed. Today this type of work is too often done with post-it notes.

In the interaction between domains, perhaps logics of roles and obligations, such as deonitc logic [9], would be appropriate. Deontic logic is is a field of philosophic logic that attempts to reason about normative behavior, obligations, and permissions. It is often applied or extended to reasoning about legal contractual relationships [19, 16].

3.3 Trust Management and Logics of Trust

The area of trust management [4] is a critical area requiring increased automation and network wide layers of abstraction. In the an enterprise, trust management is fairly hierarchic with some cross-hierarchy authority. In an ISP, trust management tends to be flat with multiple strata. Different techniques are used at layers 1, 2 and 3, for customer data, for DNS management, and address management. For trust management between providers, there is a web of trust based on personal and business relationships between network operators. What all of these share is that currently they are often informal relationships implemented, if at all, in ad hoc ways.

Interdomain routing is a very interesting example of the need for automated trust management. The BGP routing protocol has many well documented security holes [14]. In response to this an extension to BGP, Secure BGP [12] (S-BGP), has been defined with countermeasures that provide authentication and authorization techniques that address many of BGP's security

problems.

However, S-BGP has not yet been deployed. There are concerns that there is an impedance mismatch between the trust model implicit in the BGP protocol (web-like inter-provider relationships) and the more hierarchical trust model implicit in the centralized address allocation organizations to witch S-BGP currently links its certification mechanism. Service providers distrust centralization, especially when trust is being centralized. However, we note that S-BGP can function perfectly well in environments that do no have a hierarchical mechanism of distributing certificates. This brings out the need to explore means of propagating and implementing trust in a decentralized manner using trust exchange protocols. How can we reason about trust and about the distribution of trust in a network [11]? Can we compose units of trust into larger units of trust?

3.4 The Transactional Network

One difficult problem encountered in network configuration is that one conceptually simple change may require "touching" the configuration of a large number of devices. Currently, these individual devices may not provide interfaces that allow "all or nothing" changes to their configurations. And at the network level, backing out of changes is often extremely difficult.

What is required is a layer of software that presents the network as a collection of abstract objects performing in certain roles. Simple network updates may be compiled to updates to multiple objects, and this layer of software should provide the network with transactional semantics. In addition, it should provide rollback facilities.

Device vendors like to say that "the network is the database of record", but nothing could be more misguided. This is only a ploy to lock service providers into their proprietary device management systems. We need layers of abstraction to move away from vendor proprietary solutions.

4 Interdomain Routing

Today interdomain routing in the Internet is accomplished with the Border Gateway Protocol (BGP) [15]. This protocol is quite simple, yet its configuration is very complicated [8, 17].

We would like to be able to give a complete, rigorous treatment of interdomain routing without ever mentioning the BGP protocol. We want to think of routing protocols as low-level implementation detail. Today we require a large number of specialists in the configuration of BGP. We believe that some day only compiler writers will need to know about the details of this protocol. But how can we model routing policies in a way that is independent of implementation details?

Some efforts have been made in this direction. For example, the Routing Policy Specification Language (RPSL) [1] is a language for used to describe routing policies between autonomous systems. A set of tools [2] is available that can automate the generation of fragments of router configuration files given RPSL specifications stored in databases. RPSL is AS-wide, but we find that it's details are too closely tied to BGP and so are very low level. Many BGP related configuration tasks are not described by RPSL, and so in practice it often requires an awkward mixture of semi-automation and manual configuration.

We can imagine a specification language that is more in the style of Firmato, and based on roles and relationships. The compiler will worry

about how BGP is configured. For example, BGP makes few assumptions about how attributes will be used by operators to implement routing policies. There is a tradeoff between increased policy expressiveness and the ability to provide guarantees that the routing system will actually converge. It has been shown that local routing policies may together be globally inconsistent in a way that causes the BGP to diverge [18]. However, a higher level language may be able to guarantee correctness of the configurations generated by a compiler. For example, a scheme for implementing complex backup routing policies was described in [7]. Although the low level BGP implementation of this scheme may be too complex to implement manually, it would pose not difficulty for a compiler writer.

There has been some interesting work on what might come after BGP [10]. However, we believe that an approach that *hides* BGP and other routing protocols by providing higher levels of abstraction may achieve many of these goals, at least in the short run. And it will certainly be easier to roll out a replacement of BGP if the configuration of networks do not depend on its details.

5 Moving Forward

This paper calls on the data networking research community to bring network core studies into the data networking curriculum. This will be critically important in order to ensure that internet does not become balkanized with vendor proprietary solutions and islands of provider proprietary solutions. The core needs to be considered a legitimate and full blown part of data networking "science", not a neglected muddle left to providers and vendors to sort out as it is today. This will only happen if we can abstract away from the low level details, and arrive at network wide abstractions.

References

[1] C. Alaettinoglu, T. Bates, E. Gerich, D. Karrenberg, D. Meyer, M. Terpstra, and C. Villamizar. Routing Policy Specification Language (RPSL). RFC 2280, 1998.

[2] C. Alaettinoğlu. RAToolSet : A Routing Policy Analysis Tool Set. `http://www.isi.edu/ra/RAToolSet`.

[3] Yair Bartal, Alain J. Mayer, Kobbi Nissim, and Avishai Wool. Firmato: A novel firewall management toolkit. In *IEEE Symposium on Security and Privacy*, pages 17–31, 1999.

[4] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. Technical Report 96-17, 28, 1996.

[5] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: The end to end arguments vs. the brave new world. In *Communications Policy in Transition: The Internet and Beyond*. MIT Press, 2001.

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

[7] T. Griffin, Lixin Gao, and Jennifer Rexford. Inherently safe backup routing with BGP. In *Proc. IEEE INFOCOM*, April 2001.

[8] Bassam Halabi. *Internet Routing Architectures*. Cisco Press, 1997.

[9] Risto Hilpinen. *Deontic Logic: Introductory and Systematic Readings*. Kluwer, 1981.

[10] Geoff Huston. Scaling interdomain routing. *Internet Protocol Journal*, 4(4), December 2001.

[11] Li, Feigenbaum, and Grosof. A logic-based knowledge representation for authorization with delegation. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[12] Charles Lynn, Joanne Mikkelson, and Karen Seo. Secure BGP (S-BGP). draft-clynn-s-bgp-protocol-00a.txt. work in progress.

[13] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration, 2002.

[14] Sandra Murphy. BGP security analysis. draft-murphy-bgp-secr-02.html. work in progress.

[15] Y. Rekhter and T. Li. A Border Gateway Protocol. RFC 1771 (BGP version 4), March 1995.

[16] L. M. M. Royakkers. *Extending Deontic Logic for the Formalization of Legal Rules*. Kluwer, 1998.

[17] John W. Stewart. *BGP4: Inter-Domain Routing in the Internet*. Addison-Wesley, 1998.

[18] Kanan Varadhan, Ramesh Govindan, and Deborah Estrin. Persistent route oscillations in inter-domain routing. *Computer Networks*, 32:1–16, 2000.

[19] Roel J. Wieringa and John-Jules Ch. Meyer. *Applications of Deontic Logic in Computer Science: A Concise Overview*, pages 17–40. John Wiley & Sons, 1993.