Report from Dagstuhl Seminar 15071

Formal Foundations for Networking

Edited by

Nikolaj Bjørner[1], Nate Foster[2], Philip Brighten Godfrey[3], and Pamela Zave[4]

1 **Microsoft Research – Redmond, US,** nbjorner@microsoft.com
2 **Cornell University – Ithaca, US,** jnfoster@cs.cornell.edu
3 **University of Illinois – Urbana-Champaign, US,** pbg@illinois.edu
4 **AT&T Labs Research – Bedminster, US,** pamela@research.att.com

Abstract

This report documents the program and outcomes of Dagstuhl Seminar 15071 "Formal Foundations for Networking." Networking is in the midst of a revolution being driven by rapidly expanding infrastructures and emerging software-defined networking architectures. There is a growing need for tools and methodologies that provide rigorous guarantees about performance, reliability, and security. This seminar brought together leading researchers and practitioners from the fields of formal methods, networking, programming languages, and security, to investigate the task of developing formal foundations for networks.

Seminar Feburary 8–13, 2015 – http://www.dagstuhl.de/15071
1998 ACM Subject Classification C.2.3 Computer-Communication Networks – Network Protocols, D.2.4 Software/Program Verification – Formal Methods, F.3.1 Logics and Meaning of Programs – Specifying and Verifying and Reasoning about Programs
Keywords and phrases Formal methods, logic, middleboxes, model checking, networking, program synthesis, security, software-defined networking, verification
Digital Object Identifier 10.4230/DagRep.5.2.44

1 Executive Summary

Nikolaj Bjørner
Nate Foster
Philip Brighten Godfrey
Pamela Zave

The scale and complexity of computer networks has increased dramatically in recent years, driven by the growth of mobile devices, data centers, and cloud computing; increased concerns about security; and generally more widespread and diverse uses of the Internet. Building and operating a network has become a difficult task, even for the most technologically sophisticated organizations.

To address these needs, the research community has started to develop tools for managing this complexity using programming language and formal methods techniques. These tools use domain-specific languages, temporal logics, satisfiability modulo theories solvers, model checkers, proof assistants, software synthesis, etc. to specify and verify network programs.

Yet despite their importance, tools for programming and reasoning about networks are still in a state of infancy. The programming models supported by major hardware vendors require

configurations to be encoded in terms of low-level constructs – e.g., hardware forwarding rules and IP address prefixes. To express richer policies, network operators must incorporate "tribal knowledge" capturing requirements that cut across different customers, service-level agreements, and protocols and can easily lead to contradictions. In addition, networks are rarely static, so operators must deal with updates to configurations and the complications that arise during periods of transition or when unexpected failures occur.

The goal of this seminar was to bring together leading practitioners from the areas of formal methods, networking, programming languages, and security, to exchange ideas about problems and solutions, and begin the task of developing formal foundations for networks. The seminar program was grouped into broad categories addressing the following issues:

- Networking Applications (Akella, Gember-Jacobson, Jayaraman, Rexford). What are the key concerns in enterprise, data center, and wide-area networks today? What kinds of modeling, verification, and property-checking tools are operators deploying? What kinds of scalability challenges are they facing?
- Emerging Areas (Papadimitriou, Rozier). What are the key issues in emerging areas such as crowd-sourced networks and aerospace engineering? Can existing tools be easily adapted to these areas? How can new researchers get involved?
- Distributed Systems (Canini, Cerny). What are some techniques for handling the distributed systems issues that arise in modeling and reasoning about networks? How can we exploit these insights to build practical tools for verifying properties in the presence of replicated state, asynchronous communication, and unexpected failures?
- Domain-Specific Tools (Chemeritskiy, Mahajan, Panda, Rybalchenko, Sagiv). What are the best approaches for verifying properties of real-world networks? How can we incorporate features such as dynamic control programs and mutable state? How can we make these tools scale to networks of realistic size?
- General Tools (Brucker, Ganesh, Guha, Jia, Nelson, Rosenblum, Rybalchenko). There is a rich literature on temporal logics, satisfiability modulo theories checkers, model checkers, proof assistants, Datalog, etc. What are the key differences between these tools and how can they be applied to networks?
- Platforms and Models (Guha, Schlesinger, Reitblatt, Walker, Zave). What is the state-of-the-art in network programming? How can we build compilers and hypervisors that correctly translate from high-level models to low-level implementations?
- Program Synthesis (Buchbinder, Chaudhuri, Cerny, Yuan). Synthesis is a promising approach to building correct software, since programs are generated automatically using a verification tool. What are the best current techniques for using model checkers and satisfiability-modulo theories solvers to generate network configurations, update protocols, and policies?

The seminar comprised four and a half days of presentations, interspersed with discussions, tool demonstrations, and working groups. This report collects the abstracts of the presentations, gives summaries of discussions and working groups, and lists open questions.

2 Table of Contents

Executive Summary
Nikolaj Bjørner, Nate Foster, Philip Brighten Godfrey, and Pamela Zave 44

Overview of Talks

Abstractions for Network Functions
Aditya Akella 48

Modeling and Conformance Testing of Network Access Policies in HOL-TestGen
Achim D. Brucker 48

Distributed SDN controllers
Marco Canini 48

Program Synthesis for Network Updates
Pavol Cerny 49

Program Synthesis
Swarat Chaudhuri 49

VERMONT (Verifying Network Monitor)
Evgeny Chemeritskiy 49

The Impact of Community Structure on SAT Solver Performance
Vijay Ganesh 50

Machine-Verified Network Controllers
Arjun Guha 51

Management Plane Analytics
Aaron Gember-Jacobson 51

Network Verification in Microsoft Azure
Karthick Jayaraman 52

Verifying Network Protocols using Declarative Networking
Limin Jia 52

Configuration verification: A missing link toward fully verified networks
Ratul Mahajan 52

SAT Applications Tutorial (plus a pinch of Margrave)
Tim Nelson 53

Verifying Mutable Datapaths
Aurojit Panda 53

Software-Defined Crowd-Shared Networks
Panagiotis Papadimitriou 53

Proof-Carrying Network Code
Mark Reitblatt 54

SDN Applications
Jennifer Rexford 54

A Brief Look at Probabilistic Model Checking
David Rosenblum 54

Formal Methods Challenge: *E*fficient Reconfigurable *C*ockpit Design and Fleet *O*perations using Software Intensive, *N*etworked, and Wireless-Enabled Architecture (ECON)
Kristin Yvonne Rozier 55

Constrained Horn Clauses for Software, Hardware, Network Verification and Synthesis
Andrey Rybalchenko 55

Reasoning about Stateful Networks
Mooly Sagiv 56

Protocol-independent Packet Processing
Cole Schlesinger 56

Decentralizing SDN Policies
Sharon Shoham Buchbinder 56

Online Data Center Modeling
Robert Soulé 57

Implementing Path Queries
David Walker 57

Programming Network Policies by Examples
Yifei Yuan 58

A formal model of a cloud, featuring modular, expressive, re-usable abstractions
Pamela Zave 58

Working Groups

Beyond Verification 59

Abstractions for Capturing Intent 59

Network Verification: What's New? 60

Building Decomposable Control Abstractions 60

Open Problems 61

Participants 63

3 Overview of Talks

3.1 Abstractions for Network Functions

Aditya Akella (University of Wisconsin – Madison, US)

Joint work of Akella, Aditya; Gember-Jacobson, Aaron
URL http://opennf.cs.wisc.edu

Network functions (NFs), also called middleboxes, are devices that perform custom packet processing functions. With the advent of network functions virtualization (NFV), where NFs are deployed within VMs or as software processes, it has become markedly simpler to bring up of tear down NFs within networks. Furthermore, software defined networking (SDN) is being used to flexibly steer traffic through specific NF instances. The confluence of NFV and SDN opens up the door to two exciting sets of scenarios, namely distributed processing and service chaining. We argue that key attributes of NFs – which differentiate them from traditional routers/switches – impede our ability to realize the full potential of distributed processing and service chaining. We present abstractions and systems that help overcome these impediments.

3.2 Modeling and Conformance Testing of Network Access Policies in HOL-TestGen

Achim D. Brucker (SAP – Karlsruhe, DE)

Modern systems need to comply to large and complex security policies that need to enforced at runtime. This runtime enforcement needs to happen on different levels, e.g., ranging from high level access control models to firewall rules. We present an approach for the modular specification of security policies (e.g., access control policies, firewall policies). Based on this formal model, i.e, the specification, we discuss a model-based test case generation approach that can be used for both testing the correctness of the security infrastructure as well as the conformance of its configuration to a high-level security policy.

3.3 Distributed SDN controllers

Marco Canini (Université catholique de Louvain – Louvain-la-Neuve, BE)

This talk motivates and illustrates the challenges to correctly design and reason about distributed controllers. It then presents Software Transactional Networking, a distributed SDN control plane based on software transactional memory principles that supports concurrent policy updates while ensuring consistent policy composition and high availability.

3.4 Program Synthesis for Network Updates

Pavol Cerny (University of Colorado at Boulder – Boulder, US)

Joint work of McClurg, Jedidiah; Hojjat, Hossein; Cerny, Pavol; Foster, Nate
Main reference J. McClurg, H. Hojjat, P. Cerny, N. Foster, "Efficient Synthesis of Network Updates," arXiv:1403.5843v3 [cs.PL], 2015.
URL http://arxiv.org/abs/1403.5843v3

Software-defined networking (SDN) is revolutionizing the networking industry, but current SDN programming platforms do not provide automated mechanisms for updating global configurations on the fly. Implementing updates by hand is challenging for SDN programmers because networks are distributed systems with hundreds or thousands of interacting nodes. Even if initial and final configurations are correct, naively updating individual nodes can lead to incorrect transient behaviors, including loops, black holes, access control violations, and others. This talk presents an approach for automatically synthesizing updates that are guaranteed to preserve specified properties. We formalize network updates as a distributed programming problem and develop a synthesis algorithm that uses counterexample-guided search and incremental model checking to dramatically improve performance. We describe our prototype implementation, and present results from experiments on real-world topologies and properties demonstrating that our tool scales to updates involving thousands of nodes in a few seconds.

3.5 Program Synthesis

Swarat Chaudhuri (Rice University – Houston, US)

The field of program synthesis envisions a software design process where programmers write partial, nondeterministic specifications of programming tasks, and powerful algorithms are used to find correct implementations of these specifications. In this talk, I will describe some of my recent work in this area and its potential application in the domain of networks. Specific topics will include:

1. Lambda^2, a new algorithm for example-driven synthesis of higher-order functional programs.
2. Pliny, a new project on program synthesis and repair that utilizes knowledge implicit in large pre-existing corpora of programs.

3.6 VERMONT (Verifying Network Monitor)

Evgeny Chemeritskiy (Applied Research Center for Computer Networks – Moscow, RU)

VERifying MONiTor (VERMONT) is a software toolset for checking the consistency of network configurations with formally specified invariants of Packet Forwarding Policies (PFP).

Correct and safe management of networks is a very hard task. Every time the current load of flow tables should satisfy certain requirements. Some packets have to reach their destination, whereas some other packets have to be dropped. Certain switches are forbidden for some packets, whereas some other switches have to be obligatorily traversed. Loops are not allowed. These and some other requirements constitute a PFP. One of the aims of network engineering is to provide such a loading of switches with forwarding rules as to guarantee compliance with the PFP. VERMONT provides some automation to the solution of this task.

VERMONT can be installed in line with the control plane. It observes state changes of a network by intercepting messages sent by switches to the controller and command sent by the controller to switches. It builds an adequate formal model of a whole network and checks every event, such as installation, deletion, or modification of rules, port and switch up and down events, against a set formal requirements of PFP. Before a network update command is sent to a switch VERMONT anticipates the result of its execution and checks whether a new state of network satisfies all requirements of PFP. If this is the case then the command is delivered to the corresponding switch. Upon detecting a violation of PFP VERMONT blocks the change, alerts a network administrator, and gives some additional information to elucidate a possible source of an error.

VERMONT has a wide area of applications. It can be attached to a SDN controller just to check basic safety properties (the absence of loops, black-holes, etc) of the network managed by the controller. VERMONT may be also cooperated with software units (like FlowVisor) that aggregate several controllers. In this case VERMONT checks the compatibility of PFPs implemented by these controllers. This toolset can be used as a fully automatic safeguard for every software application which implements certain PFP on a SDN controller.

3.7 The Impact of Community Structure on SAT Solver Performance

Vijay Ganesh (University of Waterloo – Waterloo, CA)

Modern CDCL SAT solvers routinely solve very large industrial SAT instances in relatively short periods of time. It is clear that these solvers somehow exploit the structure of real-world instances. However, to-date there have been few results that precisely characterise this structure. In this paper, we provide evidence that the community structure of real-world SAT instances is correlated with the running time of CDCL SAT solvers. It has been known for some time that real-world SAT instances, viewed as graphs, have natural communities in them. A community is a sub-graph of the graph of a SAT instance, such that this sub-graph has more internal edges than outgoing to the rest of the graph. The community structure of a graph is often characterised by a quality metric called Q. Intuitively, a graph with high-quality community structure (high Q) is easily separable into smaller communities, while the one with low Q is not. We provide three results based on empirical data which show that community structure of real-world industrial instances is a better predictor of the running time of CDCL solvers than other commonly considered factors such as variables and clauses. First, we show that there is a strong correlation between the Q value and Literal Block Distance metric of quality of conflict clauses used in clause-deletion policies in Glucose-like solvers. Second, using regression analysis, we show that the the number of

communities and the Q value of the graph of real-world SAT instances is more predictive of the running time of CDCL solvers than traditional metrics like number of variables or clauses. Finally, we show that randomly-generated SAT instances with $0.05 \leq Q \leq 0.13$ are dramatically harder to solve for CDCL solvers than otherwise.

3.8 Machine-Verified Network Controllers

Arjun Guha (University of Massachusetts at Amherst – Amherst, US)

Joint work of Guha, Arjun; Reitblatt, Mark; Foster, Nate

In many areas of computing, techniques ranging from testing to formal modeling to full-blown verification have been successfully used to help programmers build reliable systems. But although networks are critical infrastructure, they have largely resisted analysis using formal techniques. Software-defined networking (SDN) is a new network architecture that has the potential to provide a foundation for network reasoning, by standardizing the interfaces used to express network programs and giving them a precise semantics.

This talk describes the design and implementation of the first machine-verified SDN controller. Starting from the foundations, we develop a detailed operational model for OpenFlow (the most popular SDN platform) and formalize it in the Coq proof assistant. We then use this model to develop a verified compiler and runtime system for a high-level network programming language. We identify bugs in existing languages and tools built without formal foundations, and prove that these bugs are absent from our system. Finally, we describe our prototype implementation and our experiences using it to build practical applications.

3.9 Management Plane Analytics

Aaron Gember-Jacobson (University of Wisconsin at Madison – Madison, US)

While it is generally held that network management is tedious and error-prone, it is not well understood which specific management practices increase the risk of failures. To address this gap, we propose a management plane analytics framework that an organization can use to: (1) infer which management practices can impact network health, and (2) develop a predictive model of health, based on observed practices, to improve network management. We overcome the challenges of noisy data and insufficient samples by adopting a "big data" approach, in which we synthesize data from many networks and build predictive models over aggregates. Our current models can predict network health with an accuracy of 76–89%.

3.10 Network Verification in Microsoft Azure

Karthick Jayaraman (Microsoft Research – Redmond, US)

Network verification is problem of critical importance for managing large-scale data centers. In this talk, we will describe a system that we built to perform run-time monitoring of the data plane of Azure data center network, and assure its correctness and consistency. Our system uses a tool called SecGuru that leverages Z3, a SMT solver, to prove properties about access-control lists and routing tables. Our objective is to prove local correctness of devices, and most of the global properties can be decomposed to local properties. Our system is currently deployed, and its use has led to a positive measurable impact in assuring reliability of the network.

3.11 Verifying Network Protocols using Declarative Networking

Limin Jia (Carnegie Mellon University – Pittsburgh, US)

In this talk, I will first give a short tutorial on Declarative Networks. I will explain the syntax and semantics of NDlog, a network declarative language, and how to program in NDlog. In the second part of my talk, I will present our recent work on verifying secure network protocols using declarative networks. NDlog serves as a unified specification both for generating low-level implementations for empirical evaluation and for verifying formal properties. I will present a program logic that we developed for deriving invariant properties of NDlog programs that execute in an adversarial environment. These invariant properties are safety properties on traces that are expressive enough to express origin and path authenticity properties of networks.

3.12 Configuration verification: A missing link toward fully verified networks

Ratul Mahajan (Microsoft Research – Redmond, US)

What does it mean to be running a fully verified network? One perspective is that a verified network is one that is running verified 1) hardware; 2) software; 3) data plane; and 4) configuration. Given prior work in first three domains, the missing link is configuration verification. I'll describe our recent work on a logic-based approach for verifying the configuration of large, traditional (non-SDN) networks. I'll also describe how our approach can help bridge traditional and SDN networks.

3.13 SAT Applications Tutorial (plus a pinch of Margrave)

Tim Nelson (Brown University, Providence, US)

This talk provides an overview of using SAT-solvers for network analysis. For perspective, it includes a discussion of our prior work on configuration analysis in Margrave, and shows how rich questions about configuration behavior can be answered via SAT-solvers. It also shows examples of how to use Margrave-style analyses to resolve questions about the stateful behavior a configuration describes.

3.14 Verifying Mutable Datapaths

Aurojit Panda (University of California, Berkeley – Berkeley, US)

Recent work has made great progress in verifying the correctness of forwarding tables in networks. However, these approaches cannot be used to verify networks containing middleboxes such as caches and firewalls whose forwarding behavior depends on previously observed traffic. We explore how to verify reachability properties for networks that include such "mutable datapath" elements. Our work leverages recent advances in SMT solvers, and the main challenge lies in scaling the approach to handle large and complicated networks. While the straightforward application of model checking to this problem can only handle very small networks (if at all), our approach can verify invariants on networks containing 30,000 middleboxes in a few minutes.

3.15 Software-Defined Crowd-Shared Networks

Panagiotis Papadimitriou (Leibniz Universität Hannover – Hannover, DE)

Recently there has been an increasing interest in providing wider access to Internet. One opportunity for sharing Internet access in residential areas is to exploit the spare capacity in home broadband connections. Since such crowd-shared networks entail considerable configuration overhead both for home network users and ISPs, we leverage on SDN to outsource their configuration and management to third parties. Enabling a third party to federate wireless home networks can reduce the expenditure for network operators and enable new economic models for generating revenue from currently underutilized infrastructures.

3.16 Proof-Carrying Network Code

Mark Reitblatt (Cornell University – Ithaca, US)

Joint work of Reitblatt, Mark; Foster, Nate; Kozen, Dexter; Mamouras, Konstantinos; Silva, Alexandra

In many network settings, multiple parties interact to form a network program. For example, in PANE, different network users and applications are delegated with "shares" of the network which they can in turn control or delegate. To protect parties from one another, these shared networks usually enforce a rigid mode of interaction. Instead, we propose a mechanism (Proof Carrying Network Code) for a more flexible, policy-based interaction. Principals are able to specify requirements for other parties to meet when handling their traffic, and the other parties are required to show that their program meets these requirements. Analogous to Proof Carrying Code, in which binaries carry verifiable proofs of their own safety, Proof Carrying Network Code comes with a verifiable certificate that a certain policy is satisfied. In this talk I'll present our initial design and implementation of the framework, and discuss future directions for exploration.

3.17 SDN Applications

Jennifer Rexford (Princeton University – Princeton, US)

Software-Defined Networking (SDN) is changing how networks are designed and managed. By offering a simple, open interface to the data plane, SDN enables data-plane verification techniques to check that a network-wide snapshot of the forwarding rules satisfies key network invariants. In this presentation, we discuss several example SDN applications, with an eye toward new research opportunities in verification and programming languages. We consider toy applications like MAC learning, stateful firewalls, and server load balancing to illustrate subtle bugs that can be hard to prevent or detect. We also survey several prominent commercial SDN applications (e.g., wide-area traffic engineering, network virtualization for multi-tenant data centers, and traffic steering through middleboxes) to illustrate further opportunities for research in language abstractions and verification techniques. Example challenges include (i) performing multiple tasks simultaneously with a single set of rules, (ii) policies (and network invariants) that change over time, (iii) uncertainty in the ordering of events, (iv) limitations on rule-table space, (v) non-deterministic applications, and (vi) interactions with other network protocols (such as TCP and BGP).

3.18 A Brief Look at Probabilistic Model Checking

David Rosenblum (National University of Singapore – Singapore, SG)

In this talk I will give an overview of the motivation and main ideas behind probabilistic model checking, and I will discuss recent research on the use of perturbation theory as a

way of dealing with uncertainty about probability parameters in stochastic models and the effect of perturbations on verification results. I will conclude with a discussion of some of the challenges in applying probabilistic model checking to problems in networking.

3.19 Formal Methods Challenge: *E*fficient Reconfigurable *C*ockpit Design and Fleet *O*perations using Software Intensive, *N*etworked, and Wireless-Enabled Architecture (ECON)

Kristin Yvonne Rozier (University of Cincinnati – Cincinnati, US)

We are at the dawn of a new generation of aircraft! Current aircraft are consistently built over-weight and over-budget and suffer from limitations that cannot carry forward into our increasingly digital age. In order to meet the demands placed on future aircraft, we need to design a new type of *net-enabled* aircraft by reducing aircraft weight, increasing automation and modularity, moving from hardware to software systems, moving from on-board/aircraft-based systems to cloud/fleet-based systems, and thereby facilitating easier and more responsive maintenance and system health management procedures.

For example, the wiring alone on the A-380 weighs approximately six tons, while each ton of weight in the aircraft infrastructure costs an estimated $7.2 billion in fuel across a fleet in the U.S. each year. We ask the question: can we replace some of these wires with wireless systems? Can we add wireless backup systems for wired systems that don't currently have backups due to weight constraints? But then how do we design these new hybrid systems in a way that allows us to rigorously reason about their safety, reliability, availability, and security? For another example, can we replace heavy, customized, and therefore hard-to-maintain cockpit systems by lighter, more modular, alternatives using wireless, software, or cloud-based technologies? We have not solved the formal verification problem for current cockpits; how will we scale to net-enabled cockpits?

We must reason about an aircraft as a network of avionics sub-systems, about a fleet as a network of aircraft, and about both of these being located in the cloud. What restrictions do we need to make to enable formal verification, from design-time to runtime? Join this NASA-lead team of government, academia, and industry experts as we attempt to answer the ultimate question: how can we design and verify a new class of net-enabled aircraft that are *lighter*, *cheaper*, and *safer* than ever before?

3.20 Constrained Horn Clauses for Software, Hardware, Network Verification and Synthesis

Andrey Rybalchenko (Microsoft Research UK – Cambridge, GB)

We show how models of software, hardware, and networks can be represented using logical formulas in a way that facilitates deductive reasoning about them using verification conditions. These verification conditions appear in form of constrained Horn clauses and can be solved efficiently using state-of-the-art automated deduction techniques.

3.21 Reasoning about Stateful Networks

Mooly Sagiv (Tel Aviv University – Tel Aviv, IL)

We describe techniques for verifying properties of networks of middleboxes.

3.22 Protocol-independent Packet Processing

Cole Schlesinger (Princeton University – Princeton, US)

The OpenFlow protocol initially provided a compelling but simple abstraction for network hardware: A switch is a match-action table configured by a controller. Later versions recognized that switch architectures comprise pipelines of tables, each table capable of operating on a fixed set of header fields, and fine-tuned the interface to allow for control of the pipeline.

But in some switches, the packet parser and pipeline are not fixed; rather, they can be configured in advance to extract and operate on arbitrary header fields. Architectures like Intel's FlexPipe and Cisco's Doppler already have this functionality under the hood – after all, it would be impractical to fabricate a new ASIC each time a new protocol is adopted. And new hardware has been proposed to expose parser and pipeline configuration directly to the controller in an SDN setting.

This talk explores the capabilities of reconfigurable switches as well as emerging languages and compilation techniques for guiding reconfiguration.

3.23 Decentralizing SDN Policies

Sharon Shoham Buchbinder (Academic College of Tel Aviv-Yaffo – Tel Aviv, IL)

Joint work of Padon, Oded; Immerman, Neil; Karbyshev, Aleksandr; Lahav, Ori; Sagiv, Mooly

Software-defined networking (SDN) is a new paradigm for operating and managing computer networks. SDN enables logically-centralized control over network devices through a "controller" – software that operates independently of the network hardware. Network operators can run both in-house and third-party SDN programs on top of the controller, e.g., to specify routing and access control policies.

In practice, having the controller handle events limits the network scalability. Therefore, the feasibility of SDN depends on the ability to efficiently decentralize network event-handling by installing forwarding rules on the switches. However, installing a rule too early or too late may lead to incorrect behavior, e.g., (1) packets may be forwarded to the wrong destination or incorrectly dropped; (2) packets handled by the switch may hide vital information from the controller, leading to incorrect forwarding behavior. The second issue is subtle and sometimes missed even by experienced programmers.

The contributions of this paper are two fold. First, we formalize the correctness and optimality requirements for decentralizing network policies. Second, we identify a useful class of network policies which permits automatic synthesis of a controller which performs optimal forwarding rule installation.

3.24 Online Data Center Modeling

Robert Soulé (Università della Svizzera italiana – Lugano, CH)

Modern enterprise data centers are crucial infrastructure, but are also complex, dynamic, highly networked systems. As such, their capacity, performance, behavior, and failure modes are difficult to predict, understand, and plan for. A major reason for this complexity is that the many conceptual layers involved in an enterprise data center (physical network connectivity, physical and virtual machines, link layers, VLANs, routing, service-oriented architectures, application deployment, etc.) are managed today by tools and techniques which focus on only one or a few layers.

Rather than focusing on mechanisms to control and manage subsets of a data center, we will create a common data model and representation for the state of an operational data center which can be populated and driven by logs, traces, and configuration information; queried by operators to determine global properties of the system (such as traffic matrices), and drive online workload-driven simulations to explore the effects of configuration changes. Our goal is to provide a shared substrate for diverse data center management functionality, analogous to the way that the relational model of databases provided a common substrate for tabular data.

3.25 Implementing Path Queries

David Walker (Princeton University – Princeton, US)

Joint work of Narayana, Srinivas; Arashloo, Mina; Rexford, Jennifer; Walker, David

Decades of experience suggests that complex programming systems should be implemented using a stack of compiler intermediate languages. Indeed, virtually all modern compilers use such an architecture, and we see no reason that network programming systems should deviate from this well-established trend. In this talk, we illustrate this idea via a case study involving the implementation of an expressive new query language for software-defined networks. This query language allows users to measure the flow of packets along user-specified paths in a network. It can help diagnose link congestion, implement traffic engineering algorithms, or mitigate DDoS attacks. More specifically, the language allows network operators or control software to issue queries specified as regular expressions over predicates on packet locations and header values. It also uses SQL-like "groupby" constructs to aggregate results anywhere along a path. A run-time system compiles the high-level queries into a deterministic finite automaton, which is encoded in the NetKAT intermediate language. The NetKAT program is then compiled into a set of OpenFlow rules. Finally those OpenFlow rules are distributed

to a set of switches. As packets flow through the network, switches stamp packets with automaton states, which tracks the packets' progress towards fulfilling a query. Only when a query is satisfied are packets counted, sampled, or sent to collectors for further analysis. By processing queries inline in the data plane, users "pay as they go" as data-collection overhead is limited to only those packets that satisfy the query. We have implemented our system on top of Pyretic, an open source SDN platform, and evaluated its performance on a campus topology.

3.26 Programming Network Policies by Examples

Yifei Yuan (University of Pennsylvania – Philadelphia, PA)

The emergence of programmable interfaces to network controllers offers network operators the flexibility to implement a variety of policies. We propose NetEgg, a programming framework that allows a network operator to specify the desired functionality using example behaviors. Our synthesis algorithm automatically infers the state that needs to be maintained to exhibit the desired behaviors along with the rules for processing network packets and updating the state. We report on an initial prototype of NetEgg. Our experiments evaluate the proposed framework based on the number of examples needed to specify a variety of policies considered in the literature, the computational requirements of the synthesis algorithm to translate these examples to programs, and the overhead introduced by the generated implementation for processing packets. Our results show that NetEgg can generate implementations that are consistent with the example behaviors, and have performance comparable to equivalent imperative implementations.

3.27 A formal model of a cloud, featuring modular, expressive, re-usable abstractions

Pamela Zave (AT&T Labs Research – Bedminster, US)

Today's Internet has many layers at different scopes and levels, instead of the fixed 5 or 7 prescribed by classic reference models. This talk presents a new architectural model suitable for today's Internet, in which each layer instantiates and constraints the same template of mutable state to carry out its particular purposes. The model is illustrated with four formally-specified layers implementing a realistic large-scale cloud with rich functionality including middlebox policies, live VM migration, tenant isolation, and multiple data centers. The model supports formal, modular verification of a wide variety of properties including reachability, security, consistent updates, and safe optimizations.

4 Working Groups

4.1 Beyond Verification

This working group was motivated by the prevalence of discussions focused on verification during the early days of the seminar. Many people felt that the goal of users such as network operators is not verification, but rather proper, predictable functioning of a network. To achieve the real goal, it is necessary to reason about less well-understood "ility" properties, as well as well-understood logical properties. Reasoning about "ility" properties may require different frameworks from logical properties. Even for logical properties, verification is just one type of formal reasoning, and one of the most expensive. Consequently, we decided that "before and besides verification" describes our concerns more accurately.

The most important research questions concern formal descriptions: What is a basic set of formal descriptions that will cover all needs? What are the relationships among these descriptions? Given such a set of descriptions, the group produced the following list of (non-verification) goals or ways of reasoning about them:

- safe optimization
- semantic-difference analysis
- change analysis
- root-cause analysis
- test-case generation
- simulation
- performance analysis
- understanding design principles
- understanding trade-offs in a design space

We agreed that the distinction between networks and distributed systems is institutional only. That being the case, we wondered about what part of networking is missing from distributed systems, to the disadvantage of the work? We also talked about the well-known "design principle" papers, namely "End-to-end arguments in system design" (Saltzer, Reed, and Clark) and "The design philosophy of the DARPA Internet protocols" (Clark). If we had up-to-date formal descriptions as a basis for reasoning, how would these principles hold up? Or would they be replaced by others?

4.2 Abstractions for Capturing Intent

This working group discussed the design of higher-level abstractions that capture operator intent. The discussion was motivated by the observation that many problems in networking stem from the fact that network software is expressed at a low-level of abstraction – one that does not necessarily match intent.

The first part of the discussion enumerated aspects of network behavior that are important to operators. Starting with existing configuration languages, the group noted a focus on two features: (i) forwarding paths and (ii) security policies. However, precisely capturing security policies often requires describing both what is allowed and what is not allowed, which is difficult using mechanisms such as access control lists. Similarly, working with concrete forwarding paths can create a mismatch with intent – the operator may specify concrete paths when actually any of a much large set of paths would suffice. Performance is another important aspect of intent, but is not typically captured in low-level configuration

languages. Operators often want to optimize for objectives such as low latency or congestion. Finally, the group noted that service-level agreements often state resilience to failures and incorporate temporal notions.

The second part of the discussion investigated concrete mechanisms for expressing intent. Some participants suggested reverse engineering intent from legacy deployments. The idea here would be to extract information from low-level configurations and put them into higher-level analytical models that are designed to capture intent. Other participants highlighted languages based on path formalisms such as NetKAT and FlowLog, languages that incorporate bandwidth guarantees such as Merlin, and platforms such as Click and P4. The group closed with a more speculative discussion on whether networks will be built using domain-specific software stacks or Turing-complete languages.

4.3 Network Verification: What's New?

Most current network verification tools are either based on domain-specific decision procedures (e.g., Header Space Analysis, VeriFlow, etc.) or encode networking problems as inputs for existing tools (e.g., Alloy, Z3, etc.). This working group explored the question: what, if anything, is new in network verification?

Many participants felt that although it might be possible in principle to encode networking problems as inputs for other solvers, there may be value in developing domain-specific decision procedures – typically there is domain knowledge that can be exploited to improve precision and performance. However, others noted that domain knowledge can sometimes be incorporated into the encoding itself, although this makes the encoding more complicated and susceptible to bugs. The group also discussed properties and noted that building on existing tools may make it difficult or impossible to express certain features such as quantitative constraints or probabilistic bounds. Moreover, there may be symmetries based on topology that can be exploited to obtain parametric and compositional reasoning, which is quite powerful. Another pragmatic challenge stems from the fact that network behavior is defined both by a program and a configuration. Almost all participants agreed that having open repositories of benchmarks and challenge problems would help focus community effort on the right problems. The group closed with a discussion of testing tools and security properties such availability, confidentiality, and anonymity.

4.4 Building Decomposable Control Abstractions

The question posed to this working group was: What abstractions can allow us to decompose a logically-centralized controller into modules? The discussion identified that controller modularity might come in several flavors: a "horizontal decomposition" into a collection of distributed controllers, or a "vertical decomposition" pushing certain elements of control downward into the data plane. These decompositions might be combined. For example, a logically-centralized control program could be fragmented across a central controller, end-hosts, and software in the data plane of switches, taking advantage of different information and reaction times available to each type of component.

Why would one want to decompose a controller? The participants saw many potential opportunities: eliminating reactivity at the controller, avoiding latency to the controller, offloading work from the hypervisor (e.g. via vertical decomposition), measurement and

monitoring, real-time congestion avoidance inside network switches or end-hosts (as in several recent papers), and flexible placement and relocation of middlebox functions. Some application-level processing such as MapReduce aggregation could even be offloaded to network devices for greater efficiency.[1]

Discussion participants pointed out a growing diversity of hardware and software providing substrates for network control. For example, Facebook's "6-pack" is a modular switch that runs with split control, using a local controller on the switch that communicates with a centralized controller. Several SDN and traditional switch vendors have released whitebox hardware enabling computation directly on switches, and Open Network Linux provides an operating system for such gear. The rise of "disaggregation" and rack-sized machines will provide new opportunities for control of their networks. Even the OpenFlow protocol, typically thought of as programming dumb switches with simple instructions, can encode perhaps surprisingly complex behavior in the forwarding plane.[2] Bianchi et al.[3] propose to further enrich OpenFlow with a minimal amount of stateful processing – taking a step in a finite state machine – to build functionality such as firewalls and load balancers. All these provide interesting new options for factoring a control application.

But hardware and protocols by themselves are not enough. The question originally posed to the working group – that of the right abstractions for decomposition – is an open problem. Several recent works provide some steps: Kandoo (Yeganeh and Ganjali, HotSDN 2012) decomposes control into two layers, global centralized control and local control at each switch, with applications such as network measurement. Recursive SDN (work by McCauley, Panda, Liu, Kazemkhani, Koponen, Raghavan, and Shenker) builds a hierarchy of SDN controllers, tackling problems of network repair and traffic engineering. Beehive (also Yeganeh and Ganjali, HotNets 2014) provides a programming abstraction similar to a centralized controller, and automatically decomposes the application into distributed execution across multiple controllers. The participants in the working group speculated about further sources of inspiration for solutions: OSPF areas, FlowLog, and work on automated parallelization of programs. From the latter research area, the key take-away is that the data dependencies are the trickiest aspect of the problem to grapple with.

5 Open Problems

Several open problems emerged from discussions throughout the seminar:

- What are the most important verification challenges in networking from the perspective of practitioners? Can the community assemble repositories of benchmarks and challenge problems to focus attention on the right issues?
- What are the right high-level language abstractions for programming networks, and what guarantees could we expect a compiler to provide – reachability, security, or even properties

[1] Luo Mai, Lukas Rupprecht, Abdul Alim, Paolo Costa, Matteo Migliavacca, Peter Pietzuch, and Alexander Wolf, "NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres", in ACM CoNEXT 2014.

[2] Michael Borokhovich, Liron Schiff, and Stefan Schmid, "Reclaiming the Brain: Useful OpenFlow Functions in the Data Plane", in ACM HotNets 2014.

[3] Giuseppe Bianchi, Marco Bonola, Antonio Capone, and Carmelo Cascone, "OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch", in ACM SIGCOMM Computer Communication Review, April 2014.

as detailed as the correct use of cryptography? Can these abstractions streamline some of the distributed aspects of network programming?

- What are appropriate formalisms for expressing and automatically verifying network properties? Reachability properties are ubiquitous in networking, being useful for characterizing connectivity, routing, and access control policies. But operators also care about quantitative properties such as latency, congestion, and resilience.
- What is the right division of labor between static and dynamic verification? Static tools find errors earlier in the development process and do not impose a run-time overhead. But dynamic techniques can be simpler and are able to detect bugs in control software and hardware errors.

Participants

- Aditya Akella
University of Wisconsin – Madison, US
- Nikolaj Bjørner
Microsoft Res. – Redmond, US
- Achim D. Brucker
SAP Research – Karlsruhe, DE
- Marco Canini
University of Louvain, BE
- Pavol Cerny
Univ. of Colorado – Boulder, US
- Swarat Chaudhuri
Rice University, US
- Evgeny Chemeritskiy
ARCCN – Moscow Region, RU
- Shiu-Kai Chin
Syracuse University, US
- Bryan Ford
Yale University, US
- Nate Foster
Cornell University – Ithaca, US
- Vijay Ganesh
University of Waterloo, CA
- Aaron Gember-Jacobson
University of Wisconsin – Madison, US
- Philip Brighten Godfrey
Univ. of Illinois at Urbana, US
- Arjun Guha
University of Massachusetts – Amherst, US
- Arie Gurfinkel
Carnegie Mellon University – Pittsburgh, US
- Karthick Jayaraman
Microsoft Res. – Redmond, US
- Limin Jia
Carnegie Mellon University – Pittsburgh, US
- Ethan Katz-Bassett
Univ. of Southern California – Los Angeles, US
- Shriram Krishnamurthi
Brown University, US
- Ori Lahav
MPI-SWS – Kaiserslautern, DE
- Nuno Lopes
Microsoft Research UK – Cambridge, GB
- Ratul Mahajan
Microsoft Res. – Redmond, US
- Tim Nelson
Brown University, US
- Aurojit Panda
University of California – Berkeley, US
- Panagiotis Papadimitriou
Leibniz Univ. Hannover, DE
- Mark Reitblatt
Cornell University – Ithaca, US
- Jennifer Rexford
Princeton University, US
- Timothy Roscoe
ETH Zürich, CH
- David Rosenblum
National Univ. of Singapore, SG
- Kristin Yvonne Rozier
University of Cincinnati, US
- Andrey Rybalchenko
Microsoft Research UK – Cambridge, GB
- Mooly Sagiv
Tel Aviv University, IL
- Cole Schlesinger
Princeton University, US
- Stefan Schmid
TU Berlin, DE
- Sharon Shoham Buchbinder
Academic College of Tel Aviv, IL
- Robert Soulé
University of Lugano, CH
- David Walker
Princeton University, US
- Alexander L. Wolf
Imperial College London, GB
- Burkhart Wolff
University Paris South, FR
- Yifei Yuan
University of Pennsylvania, US
- Vladimir Zakharov
Moscow State University, RU
- Pamela Zave
AT&T Labs Research – Bedminster, US