Formal models of software-defined networks

Feng, H.

Chapter 1

Introduction

In the field of modern networking, Software-Defined Networks (SDNs) have emerged as a novel paradigm transforming the way we conceptualize, deploy, and manage computer network infrastructures. Defined by their programmability, flexibility, and centralized control, SDNs represent a departure from traditional networking approaches, offering a more dynamic and agile framework to meet the evolving demands of today's digital systems.

The evolution of networking paradigms leading to SDNs can be traced back to the limitations of traditional network architectures [111]. Conventional networks are characterized by their rigid and hardware-centric nature making them hard to adapt to the rapidly changing requirements of modern applications and services. The emergence of virtualization technologies, coupled with the growing complexity of network management tasks made clear the need for more scalable, efficient, and programmable networks.

SDNs address these challenges by decoupling the control plane from the data plane and centralizing network intelligence and management in software-based controllers [42]. Unlike traditional networks where control mechanisms are decentralized and embedded within network switches and routers, SDN logically centralizes the control and thus simplifies network programming. Specifically, SDN concentrates network control within one or more SDN controllers, enabling greater programmability and automation, and facilitating dynamic provisioning, configuration, and optimization of network resources. Moreover, the introduction of separated layers of abstraction simplifies network management tasks and promotes interoperability across heterogeneous networking environments.

The main defining feature of a Software-Defined Network is the centralized network control and management functions, which enhance orchestration and policy enforcement across the network. This means that tasks like configuring and managing network devices can be done more efficiently from a central controller rather than locally on switches. SDN architectures are designed to be programmable. They expose APIs and interfaces that allow for the programmable control of network behavior. This flexibility enables networks to adapt to specific application requirements more effectively. Additionally, SDNs abstract the underlying network infrastructure separating the logical network from its physical implementation, offering more flexibility and scalability [53, 52, 42].

In terms of benefits, SDNs offer numerous advantages. They enable agility and better scalability by decoupling network control from hardware. This leads to greater flexibility and resource utilization as well as enabling rapid deployment [52], as network policies can be updated and enforced across the network almost in real-time, with no need for manual and error-prone configuration of individual devices. SDNs also contribute to improved performance and reliability by allowing for the automation of many routine network management tasks. Finally, SDNs enhance network security by providing centralized visibility and control over network traffic. This enables more effective threat detection, mitigation, and policy enforcement, safeguarding against cyber threats and breaches [83].

Although widely applied with success in many large networks, SDNs present a multitude of challenges inherent to the complexity of SDN architectures themselves. As SDNs decouple the control plane from the data plane and centralize network intelligence, they introduce new layers of abstraction and dependencies, which can lead to increased complexity in design, implementation, and management. The dynamic and highly concurrent nature of SDNs amplifies the challenge of ensuring the correctness and reliability of the controllers. This can result in configuration errors, inconsistencies, and vulnerabilities that undermine the reliability and security of the network.

Current methods for verifying and validating SDN configurations often rely on manual testing, ad-hoc approaches, and proprietary tools, which are not scalable or comprehensive enough to address the complexities of modern SDN deployments. As a result, there is a pressing need for rigorous formal methods that can provide mathematical rigor and systematic approaches to analyzing, designing, and verifying SDN systems [98, 101].

1.1 Formal methods in SDN

Emphasizing the need for rigorous formal methods is essential in addressing these challenges effectively. Formal methods offer a systematic approach to modeling, analyzing, and reasoning about SDN behaviors and properties. By leveraging mathematical techniques such as formal specification languages, model checking, and theorem proving, formal methods can provide rigorous guarantees of correctness, reliability, and security in SDNs.

Formal models applied to SDN not only enhance our understanding of SDN systems but also facilitate precise analysis. Controllers of SDNs are essentially software entities, faults in a controller could lead the entire network to a catastrophic situation. Since networks are critical infrastructures, rigorous validation is the only way to ensure that network policies are correctly implemented before deployment and do not lead to unintended consequences. Techniques such as model checking and theorem proving enable exhaustive exploration of system behaviors under various abstractions, ensuring that SDN deployments used, for example, in data centers, enterprise networks, and cloud computing adhere to specified requirements and constraints.

The potential benefits of employing formal methods in SDN infrastructures are manifold. Increased reliability is achieved through rigorous verification and validation of SDN configurations, reducing the likelihood of configuration errors, protocol conflicts, and performance bottlenecks. VeriCon [14] is a tool designed for the verification of network configurations and behaviors in SDN environments. It provides a comprehensive platform that checks whether a given network configuration satisfies certain safety and correctness properties. VeriCon employs a combination of symbolic execution and model-checking techniques to verify properties such as security policies, reachability, and the absence of loops across all possible packet transmissions in the network. Enhanced security is realized by systematically analyzing and verifying security properties, ensuring that SDN deployments are resilient to cyber threats, attacks, and intrusions [86, 103]. Finally, improved maintainability is attained by precise modeling, enabling a deeper understanding of SDN systems and their behaviors, more efficient troubleshooting, debugging, and evolution of SDN infrastructures over time. As a modeling language designed for concurrent object-oriented systems, the Abstract Behavioral Specification (ABS) language focuses on simulating and verifying network behaviors. ABS tool suite [2] is a collection of tools designed to analyze and verify

SDN, includes capabilities for model checking, simulation, and deductive verification, providing a robust environment for testing and verifying the correctness and performance of SDN configurations and policies.

In the context of SDNs, existing model-checking tools can be applied to verify various properties and behaviors of SDN architectures, protocols, and deployments. Examples include TLA+ [72], a high-level language for modeling programs and systems, especially for concurrent and distributed ones. They can be checked for logical consistency and correctness by writing precise specifications. Its model checker TLC [112] can handle a subclass of TLA+ specifications that is suitable for the behavior of a single switch in a SDN [61]. The Model checker SPIN [48] uses Promela as its input modeling language, and the full Linear Temporal Logic (LTL) as a specification language. We will see in this thesis how SPIN can be used for modeling SDNs. Alloy [56] is a formal design language and a model-checking tool, it uses the Alloy for specifying models and properties, and Alloy's specification language is based on a subset of first-order logic with relational operators. It has been used to formally model the OpenFlow switch verifying a great part of the properties of an OpenFlow switch [94].

Both PRISM [70] and UPPAAL [16] have been utilized for verifying quantitative properties of SDNs such as performance, timing, reliability, and resource utilization. PRISM is a probabilistic model checker supporting Probabilistic Computation Tree Logic and Continuous Stochastic Logic. Although it is primarily designed for probabilistic systems, it has been used to analyze performance, reliability, Quality of Service (QoS), and security aspects of SDN deployments. UPPAAL supports temporal logic specifications expressed in Timed Computational Tree Logic (TCTL). It is specialized in real-time systems and has been employed for timing analysis, performance evaluation, resource allocation, and scheduling and synchronization analysis in SDNs.

SDN testing strategies can increase our trust in SDN deployments ultimately improving the reliability and robustness of SDN systems. However, the complexity and diversity of SDN environments make it challenging to create comprehensive test scenarios that accurately reflect real-world conditions. Additionally, the dynamic nature of SDN infrastructures introduces uncertainties and variability that may not be fully captured in traditional testing approaches, leading to potential gaps in test coverage and effectiveness [87].

1.2 Research objectives

In this thesis, we explore the use of the coordination language Reo as a model of SDNs. Reo [5] is a coordination language designed for specifying and orchestrating the behavior of reactive and distributed systems. It provides a powerful and visually appealing framework for expressing complex coordination patterns among system components, facilitating the modeling of communication, synchronization, and interaction protocols in a concise and modular manner.

Reo is characterized by its formal semantics given in terms of constraint automata [13]. It comes equipped with several tools for the rigorous analysis and verification of system specifications, ensuring correctness and reliability. With support for hierarchical composition and modularity, Reo promotes the reuse and maintainability of coordination patterns, facilitating the development and evolution of distributed systems.

Our choice for Reo is justified by the easy and intuitive visual representation via user-defined channels and their composition. Reo offers a versatile tool for designing and coordinating reactive systems, with applications ranging from network protocols and distributed algorithms to software architectures and concurrent processes.

While Reo was not explicitly developed for SDN, we investigate in this thesis if its principles and capabilities could potentially be applied to model and verify certain aspects of SDN architectures, leading to our first research question:

Research question 1. *Can the coordination language Reo and its semantic model of constraint automata be used as a formal model of SDNs?*

Because Reo was primarily designed for modeling reactive systems and distributed protocols, we found that certain features and concepts in SDN, such as switch flow tables and specific communication messages between switches and controllers, are not well-suited for being directly modeled using Reo. This motivated us to investigate Chapter 3 for a non-trivial extension of Reo with variables while maintaining the idea behind the synchronization operation of the original Reo language.

Our Reo model of SDN switches and controllers together with their network topology includes an abstraction of the OpenFlow communication protocol. While OpenFlow is not the same as an SDN, it enables the programmability and centralized control of its network devices, such as switches and routers. It defines

a standardized interface between the control plane and the data plane of network devices, allowing a centralized SDN controller to dynamically manage the forwarding behavior of network switches and routers [42]. Our Reo model of the OpenFlow protocol serves as a foundational study providing a formal model for implementing and orchestrating network control in the programmable and dynamic environments of SDNs.

The advantage of using Reo as a formal semantics, of OpenFlow-based architectures of SDNs is the modularity of the approach and the possible leverage of verification tools for SDNs. Reo is supported by various tools and frameworks for simulation, model checking, and verification, however, that cannot be directly used with the extension of Reo we considered for SDNs. This brings us to our second research question:

Research question 2. *Can we use existing model checkers to verify properties of SDNs modeled by Reo?*

We answer this question in Chapter 4, where we show how to generate automatically Promela code from symbolic constraint automata, the semantic model of the extension of Reo we use for SDNs. Promela (PROcess MEta LAnguage) is a process modeling language specifically designed for modeling and verifying concurrent and distributed systems using SPIN (Simple Promela Interpreter), a powerful model-checking tool [48]. With SPIN, Promela models can be analyzed for various linear time temporal properties, such as deadlock freedom, liveliness, and safety, through exhaustive state space exploration. SPIN provides an interface for specifying Promela models and defining verification properties, allowing a systematic validation of the correctness of complex concurrent systems.

By modeling the SDN control plane and data plane in Reo and automatically obtaining a Promela code, we can use the SPIN model checking tool to explore the SDN model and verify properties. Another high-level domain-specific language for expressing and reasoning about network policies and behaviors in Software-Defined Networking (SDN) environments is NetKAT [4]. NetKAT is inspired by Kleene algebra, a mathematical framework for describing the behavior of regular expressions, and is extended to the domain of network policies. In NetKAT, network policies are expressed as compositions of basic primitives, such as packet filters, forwarding actions, and logical operators, allowing for the specification of complex routing and forwarding behaviors in a concise and modular manner.

Similar to our Reo model, NetKAT provides formal semantics and mathe-

matical foundations for reasoning about network policies, enabling verification techniques such as equational reasoning and model checking to ensure correctness and consistency. Since both Reo and NetKAT use automata-based semantics, a natural question is whether the two models can be unified in a single framework:

Research question 3. *Can we extend the automata-based model of NetKAT to be stateful and allow concurrency in a way similar to Reo?*

In Chapter 5 we extend NetKAT with ports, which basically are shared variables used by processes to communicate with each other. Differently from a traditional variable, reading a value from a port is destructive, while writing is only possible if the port is empty. The addition of ports to NetKAT and a simple modification of its automata model are enough to be comparable with the Reo model. In particular, the extension of NetKAT with ports allows for concurrency, it is stateful, and it is backward compatible with the original NetKAT model.

So far, the automata model of Reo (and NetKAT) we introduced has been mainly geared toward the verification of SDNs. However, another important aspect of SDN is causality, particularly in understanding and managing network events, behaviors, and dependencies. In SDNs, where the control plane is centralized and programmable, causality helps determine the order of events and actions that influence network behavior. Understanding causality enables SDN controllers to make informed decisions about network policies, routing, and resource allocation based on the sequence of events and their causal relationships. Moreover, causality is crucial for troubleshooting and debugging network issues, as it helps identify the root causes of anomalies or failures by tracing the sequence of events leading to the observed behavior. Our final research question is related to causality:

Research question 4. *Can we use Reo or NetKAT and their associated automata-based models for avoiding hazard events in SDN using causality?*

We partially answer this question in Chapter 6, where we investigate an ordinary automata model incorporating causality into the design of the safety properties so that if a hazard event may occur then not only we must avoid it, but we also offer an alternative sequence of action that does not cause that event to happen. While the automata we consider are simpler than the one used as the model for SDNs via either NetKAT or Reo, we see this result as a first step toward

causality reasoning for SDNs.

1.3 Publications

In this section, we present all publications upon which our research presented in this thesis is based. They constitute the primary source of our theoretical research and methodological framework.

Hui Feng, Farhad Arbab and Marcello Bonsangue. (2019). A Reo Model of Software Defined Networks. In: Ait-Ameur, Y., Qin, S. (eds) Formal Methods and Software Engineering. ICFEM 2019. Lecture Notes in Computer Science, vol 11852. Springer, Cham. https://doi.org/10.1007/978-3-030-32409-4_5

In this paper, we use Reo to model SDN switches and controllers, according to the OpenFlow protocol specification. We use a similar abstract syntax and semantics of OpenFlow messages to model the interactions between the control plane and data plane in an OpenFlow-based SDN architecture. We also implement the data flow between different controllers and switches. The formal semantics of Reo is given by a novel version of constraint automata with memory, that we called symbolic constraint automata. Most of the content of Chapters 3 and 4 is based on this paper.

Hui Feng, Marcello Bonsangue and Benjamin Lion. (2022). From symbolic constraint automata to Promela. Journal of Logical and Algebraic Methods in Programming. vol 128. 100794. https://doi:10.1016/j.jlamp.2022.100794

In this paper, we implement an automatic translation from symbolic constraint automata to Promela code, the programming modeling language of the model checker SPIN. The translation enables the analysis of packet forwarding according to OpenFlow controllers, The focus is on verifying the functional properties of the SDN model, such as reachability, consistency, and correctness of network policies and configurations using linear temporal logic. As expected, the model suffers from the state explosion problem, and more research needs to be done on the scalability of our approach. This could involve simplifying the OpenFlow protocol to reduce the state space to be analyzed. On the positive side, our translation is not only for SND models but for any Reo circuit (with

and without variables). The main results of this paper are presented in Chapter 5.

Hui Feng and Marcello Bonsangue. (2024). Concurrent NetKAT with ports. In: Juw Won Park and Adam Przybyłek and Hossain Shahriar. (eds) SAC'24: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing. Association for Computing Machinery. https://doi.org/10.1145/3605098.3636048

The starting point of this paper is the original definition of NetKAT automata, which we extend so to support concurrency. The main idea is to use NetKAT automata as a model of propositional Hoare logic for concurrent processes communicating via shared ports.

Marcello Bonsangue, Georgiana Caltais, Hui Feng, and Hünkar Can Tunç (2022). A Language-Based Causal Model for Safety. In: Yamine Aït-Ameur, Florin Crăciun. (eds) Theoretical Aspects of Software Engineering. TASE 2022. Lecture Notes in Computer Science, vol 13299. Springer, Cham. https://doi.org/10.1007/978-3-031-10363-6_20

This paper is a first step towards a notion of causality for SDNs. Using ordinary automata we implemented an algorithm for finding and removing hazards specified as regular expressions. Since Reo can be modeled by NetKat automata with concurrency, causality analysis for SDN should be relatively easy to reach. By understanding the causal links between network events and configuration changes, and resulting outcomes, causality analysis could be used for diagnosing performance bottlenecks and troubleshooting issues helping in identifying the root causes of anomalies and failures. The content of this paper forms the basis for Chapter 7.

1.4 Overview

In this thesis, we applied Reo and constraint automata to formally model SDN architectures based on OpenFlow, We translated our formal model into Promela, so to allow for the use of state-of-the-art model-checking tools. More semantically, we extend the automata semantics of NetKAT to handle concurrent components communicating via shared ports. The latter model is exactly our Reo semantic model of SDNs. Finally, we took a few first steps into a causal analysis for SDNs.

The thesis is organized as follows: In Chapter 2, we introduce the main concepts of software-defined networking (SDN), paying specific attention to the three-layer framework and OpenFlow protocol. Consequently, in Chapter 3 we present the coordination language Reo and its formal semantics. Throughout this thesis, Reo is our main modeling language for the analysis of SDNs. The focus is on composition methods at the semantic level. Subsequently, in Chapter 4 we address **research question 1** by constructing a Reo model that faithfully encapsulates the data plane and control plane of an SDN, including the modeling of individual OpenFlow switches and their interconnecting channels. In Chapter 5, we propose a compositional translation from symbolic constraint automata to Promela, aligning with **research question 2**. In Chapter 6 we address the limitations of NetKAT in handling concurrency and relating it to Reo, thus resolving **research question 3**. Finally, Chapter 7 addresses **research question 4**, wherein we introduce an automata-based causal model where hazards are expressed in terms of regular expressions.