Formal models of software-defined networks

Feng, H.

Chapter 8

Conclusions

With the explosive development of the Internet, traditional network architectures have struggled to meet the increasing demands for network scale, complexity, and dynamism. Software-defined networking (SDN) emerged in response, offering a novel architecture that separates the control plane from the data plane, achieving centralized control and flexibility of the network. SDN introduces a new paradigm for network management, making it more flexible and efficient, and capable of quickly adapting to changing network requirements and application scenarios.

The core advantage of SDN lies in its high programmability, allowing network administrators to directly control network behavior through software interfaces rather than relying on the physical configurations of traditional network devices. This design not only simplifies network configuration and management but also accelerates the deployment of new network services and policies, enabling the network to adapt more swiftly to new business requirements. Moreover, the openness standards and protocols of SDN promote innovation of third-party applications and services, providing possibilities for customized network functions and services.

However, as an innovative technology, SDN also faces several challenges, including network security issues, performance bottlenecks, and compatibility issues with existing network devices and protocols. To overcome these challenges, researchers and engineers are continuously exploring and developing new technologies and methods, such as using formal methods to verify and ensure the correctness of SDN controllers and applications and developing more efficient data plane technologies to enhance network performance.

Classic formal verification methods for Software-Defined Networking (SDN) can be implemented using model checking, a process where the properties of a system are verified against a formal model of the system. The formal model for SDN is indeed complex, as it needs to accurately represent various aspects of the network, including the behavior

of the control plane, the data plane, and the interactions between them. This complexity is further compounded by the dynamic and programmable nature of SDN, which allows network behavior to be modified at runtime through software. Model checking of SDN involves creating a formal model of the network's behavior, including all possible states and transitions that the network can undergo. This model must also incorporate the SDN controller's logic, which dictates how the network's configuration can change in response to different events. The verification process then checks if this model adheres to certain specified properties or invariants, such as connectivity, security policies, or absence of loops, under all possible scenarios.

In this thesis, we apply model checking to check SDN properties via the language Reo and its operational semantics constraint automata. Due to the language provided by Reo accurately describing the coordination and communication protocols in systems, it is more suitable for SDN, which separates the network control plane from the data plane. Also, the Reo is particularly suited to describing concurrent and synchronous processes, it can explicitly represent the synchronization and asynchronous transmission of data flows, as well as dependencies between components. In the following section, we summarize the main research contributions of this thesis and analyze the limitations or challenges that we encountered during the research.

8.1 Main contributions

In Chapter 2, we concretely analyzed the composition structure of SDN and contrasted it with the traditional network architecture by how data is transferred in different layers. By implementing the OpenFlow protocol used in SDN, we described three basic OpenFlow messages that communicated in the control plane and data plane. To formalize this process, we built a Reo model of SDN controllers and switches which answered the 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?*

Reo is a powerful tool for designing and reasoning about communication, coordination, and data flow in concurrent systems, which are key aspects of SDNs. As we discussed in Chapter 4, our SDN model is based on Reo circuits which take semantics on a novel definition of constraint automata with memory (defined in Chapter 4). The model is based on the OpenFlow protocol, which is formalized via PktIn, PktOut, and FlowMod messages. A small case study shows the details of the model, highlighting the complicacy of a stateful and concurrent behavior. To be able to use the model for verification purposes we answered the second research question.

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

To address this question, we gave a translation of symbolic constraint automata used in our model of SDNs to Promela in Chapter 5. Since Promela can be used for both model checking and simulation by SPIN, our answer to the research question is affirmative, We modeled a simple SDN and verified that a safety property was not satisfied.

While the Reo model of SDNs has typically very few states and many transitions, because of the memory involved, the translation to Promela generates a transition system with a very large number of states, making the verification of a real SDN challenging.

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

NetKAT was originally designed as a network programming language that uses a formal algebraic approach to describe and reason about network behaviors. It allows executions for packet filtering, modification, and forwarding rules, but it primarily operates under a stateless and sequential paradigm. Without changing the NetKAT automata model, in Chapter 6 we extended NetKAT with concurrent processes that communicate via shared ports. It turns out that the extended model is very similar to our symbolic constraint automata, and therefore can model a large subset of Reo. Our focus was purely semantics, and we left as a future direction an axiomatization for the extended NetKAT.

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

Another crucial aspect of SDN is causality, in a scenario where packets need to be processed and forwarded through the network, causality ensures that packets are handled correctly, and avoids any hazards that may cause their loss. We gave a partial answer to the above research question in Chapter 7, where we study causality in the context of ordinary automata. We presented an algorithm for detecting hazardous events returning, if possible, an alternative action sequence that will guarantee not to cause the occurrence of the event. Although the type of automata we considered are simpler than those used for modeling SDNs, we consider this result as the first step towards causal reasoning for SDN.

8.2 Future research directions

We conclude by presenting potential avenues for future research that build upon the work we presented in this thesis.

Verification for SDNs models

When answering our second research question we notice a state explosion problem because in Promela, explicit modeling of memory states can rapidly escalate the size of the state space, leading to computational infeasibility for the verification of large-scale SDNs. Transitioning from model checking in Promela to a symbolic model checker could offer a better approach to mitigate the state explosion problem, as it is based on a symbolic representation of data structures and variables, enabling the exploration of large state spaces more efficiently through symbolic manipulation. By abstracting memory operations and representing them symbolically, a symbolic model checker for SDNs could analyze system properties across entire classes of flow tables and messages rather than exhaustively enumerating individual ones. This paradigm shift would allow for a more scalable and tractable analysis of SDN verification in contrast to traditional model checking approaches including ours but also, for instance, ReoLive [29] and mCRL2 [65] as provided by the Reo framework [89].

Optimizing the Promela model

Another strategy aimed at reducing the computational complexity and state space of our Promela model for SDNs could be to simplify the Reo model of SDNs by abstracting away details that are not essential for the verification task at hand. This can involve removing transitions in the Reo model of a switch that we know will not be involved in the verification or simplifying the network structure by using smaller bounds for queues used between a switch and a controller. Additionally, employing partial-order reduction techniques can help reduce the number of explored states by considering only relevant interleavings of concurrent processes. Finally, exploiting symmetry in the model, such as identical processes or state transitions, can further reduce the state space by eliminating redundant exploration.

Formal verification of NetKAT

Our automata model for concurrent NetKAT with ports is a fine-grain description of the dynamics of a system through assertions, in terms of pre- and post-conditions. It would be interesting to see if a combination of symbolic techniques with deductive verification methods could be devised to enable more efficient and scalable verification of network properties.

Causality for SDNs

An obvious direction is to extend the causality framework we presented in Chapter 7 from ordinary automata to symbolic constraint automata and NetKAT automata. While this direction should not present too many technical difficulties, in practice the computational complexity of hazard discovering and avoiding could be intractable. A promising alternative could be combining a formal automata-based model of SDNs with artificial intelligence methods to explore causality within SDN environments. One possible strategy would be to utilize machine learning algorithms, to analyze the causal relationships inferred from (large) subsets of sequences accepted by the automata model to identify causal dependencies between events. Additionally, reinforcement learning algorithms could be utilized to infer causal relationships and optimize execution to avoid hazardous events.