INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Evaluating the State of Security in Software-Defined Networks

Fabian Ruffy Varga

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Bachelor's Thesis

Evaluating the State of Security in Software-Defined Networks

Fabian Ruffy Varga

Supervisor:	PD Dr. Wolfgang Hommel
Advisor:	Felix von Eye
Submission Date:	26. June 2015

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

I hereby guarantee to have authored the present thesis on my own accord. I did not use any other means other than the listed sources and material.

Munich, 26. June 2015

. . . . . .
(Unterschrift/Signature)

Abstract

Software-Defined Networking (SDN) is currently a much discussed topic, as it promises to free network operators from the proprietary and decentralised restrictions imposed by legacy networks. The new approach to network architecture shifts the configuration and routing mechanisms from routers and switches to a central controller, a single programmable software device which is able to view and command the entirety of the network. The key player in this evolution is the OpenFlow protocol, propagated by the Open Networking Foundation (ONF). However, in the process of growing popularity and surge of interest, the security aspect of SDN has been neglected. This circumstance may become a major hindrance in the acceptance and adoption of the new paradigm.

The goal of this thesis is to compile research on security regarding both vulnerabilities and opportunities and to infer requirements for a secure software-defined network. The first section aims to provide a thorough background of SDN, its architecture and main components. The general design is then inspected for flaws by analysing and identifying several security vulnerabilities and problematic trends in the attack fields of Spoofing, Tampering, Repudiation, Denial of Service, and Elevation of Privilege. The threats are summarised and visualised in attack tree models. The results of this security assessment reveal that the software-defined network based on current standards and popular control software can not be considered secure. Consequently, the second section of the thesis utilises and augments contemporary approaches to enhance the security of the OpenFlow protocol as well as the general SDN infrastructure. The security principles and concepts demonstrate that the design of SDN is ultimately capable of preventing many of the identified vulnerabilities and even selectively enhances security compared to legacy infrastructure. Nevertheless, due to the software-based and virtual nature of SDN, the network may be exposed to the constant looming threat of software bugs and exploits that may facilitate Denial of Service and Elevation of Privilege in the central controller. Furthermore, the multitude of different required solutions may heavily impact the performance and latency of the control plane or introduce new previously unconsidered vulnerabilities.

Contents

1 Introduction

Although network technology and capabilities have evolved considerably over the course of time, the underlying design of the network itself has not adapted to accommodate the continuously changing demands. While a decentralised approach was deemed necessary to guarantee sufficient resilience in the early days of networking [1], it is now considered a major hindrance in terms of innovation and overall efficiency. Network operators have to handle complex vendor-specific software implementations and operational commands and constantly risk dependency on a single manufacturer. Proprietary routers and switches are closely intertwined with specialised middleboxes, including firewalls, intrusion detection systems (IDS) or load-balancers. The overall consequence is a heavily complex and rigid, almost completely stagnant, base framework. To address these limitations a new paradigm has emerged: Programmable Networks. The concept separates the network into two decoupled, logical layers. The data plane, where traffic is being forwarded on, and the control plane, which is responsible for overseeing and directing the elements of the data plane. In the additional management plane the individual devices of the network can be accessed, reprogrammed and configured. As a result, the intelligence of the network is largely centralised and comfortable to manage. Although research communities have been working on the concept for over two decades [2], programmable networks have only recently gained traction in both academia and industry. Big data and the enormous growth of cloud services, which require adaptable and flexible solutions, have been fuelling research in the past years. Furthermore, the publication of influential OpenFlow protocol [3] has made a considerable impact among research communities [4]. Developed in Stanford University in 2008 to simplify experimental research, it is widely regarded as the foundation of the modern variant of programmable networks, Software-Defined Networking (SDN) [5].

However, new technologies also bear new risks. A software-based and centralised supervision and control exposes a network to a wide range of attacks and dangers, which may have not been considered in conventional networks. Moving into industry territory after the initial hype, the largely academic-born software-defined network has come under close scrutiny. Various committees and research institutions are currently avid to poke holes into the concept of the future architecture. Nevertheless, a substantial amount of solutions mitigating the current security flaws has been presented as well.

1.1 Objective

The aim of this thesis is to carry out a comprehensive security review of the newest version of programmable networks. The fundamental design and maturity of SDN is reviewed based on contemporary literature, discussion and research findings.

A theoretical traditional network, as well as a software-defined network, are constructed, serving as point of reference for an assessment. The legacy structure is built upon general information security and networking foundations, whereas the OpenFlow standard specifications define the design and technical details of the software-defined network. The SDN

blueprint is augmented with common features of relevant control devices and general recommendations of the Open Networking Foundation (ONF), the standard committee devoted to the development of the OpenFlow protocol.

Empirical research and critical evaluation are consulted to examine the security of the individual segments of the network. An extensive threat model tries to locate shortcomings and design flaws and additionally demonstrates possible points of vantage for an attacker. The thesis attempts to mitigate the encountered vulnerabilities by building on research proposals of secure designs and individual solutions while addressing the threat model in all of its aspects. Opportunities and deficiencies are highlighted and the results are compiled accordingly. Lastly, a sketch of a secure software-defined network is developed and provides the basis for an effort to estimate final security characteristics in comparison to conventional architecture.

1.2 Outline

The thesis is organised as follows: Chapter 2 outlines and portrays the status quo of network architecture and security and subsequently clarifies the motivation and history of Software-Defined Networking. Section 2.4 covers the technical aspects and structure of SDN, based on the specifications of the Open Networking Foundation (ONF) and the OpenFlow v1.5 standard. In addition to a summary of standardisation efforts, relevant controller designs are abstracted and introduced to establish a generic framework of the control plane. Chapter 3 and 4 are the core of the thesis. Drawing from current research findings, Chapter 3 surveys the security of SDN. A Data Flow Diagram (DFD) of the individual SDN components, all of which were defined in the preceding section, is constructed to represent a generic software-defined network. Microsoft's STRIDE [6] is the main threat model to provide an in-depth security analysis of the DFD and thus the SDN architecture. The model is enhanced with attack trees [7] to provide an overview and to visualise the approach of an attacker for each individual STRIDE aspect.

Based on the results of the assessment, Chapter 4 reviews attempts to mitigate and prevent the deficiencies and examines two security proposals for software-defined networks. First, an official amendment recommendation and best practice guide for the OpenFlow protocol is presented and inspected for solutions and negligences. The second part of the chapter then leverages the nine secure design principles proposed by Kreutz, Ramos and Verissimo [8] and provides an overview of security solutions developed for SDN. For each of the nine principles corresponding research and individual vulnerability and threat solutions are summarised and inspected for their practicality.

Chapter 5 follows the literature review of Chapter 3 and 4 and merges the research findings into a single secure design to demonstrate the security potential of SDN. In Section 5.2 the current state of security in software-defined networks to traditional architecture and drawbacks and opportunities are addressed. The thesis concludes in Chapter 6 with a summary of the literature findings, finishing remarks, and indications of future work.

2 Software-Defined Networking

Software-Defined Networking is a fundamentally different approach to the conventional method and harbours a lot of potential. However, a change in network structure has substantial security implications for operators. To understand the motivation and to be able to gauge the new security risks it is necessary to define the structure, layout and security of conventional networks. Likewise, SDN has to be dissected into its base components. This chapter establishes background, motivation, and understanding of SDN. It provides the basis for the security evaluation in Chapter 3.

2.1 Traditional Network Architecture

Modern networks are largely decentralised and autonomous. The history of this structural development traces back to design choices for telephone networks during the era of the Cold War. Paul Baran, a researcher at the RAND corporation, raised concerns that the prevailing transport method of telephone networks lacks a satisfactory resilience and exposes the infrastructure as an easy target for military strikes. A failing element could easily take out a large section of the network. [1]

He suggested an autonomous travel of communication over the network by giving the individual elements independence in their behaviour.
If one of the network entities failed, network points would utilise the adjacent information to forward data on an alternative path. This approach ensures that communication traffic could navigate through the network without relying on static path specification. During the development of this project, another researcher, Donald Davies, built the foundation of the modern packet switching by publishing his concept of separating networking communication into smaller chunks and sending individual data through the network. Both proposals saw a large success in the future design of telecommunication, ultimately resulting in prominent representatives such as the internet and IP networks. [9], [10]

Today's switches and routers rely on a plethora of protocols to guarantee efficient and robust communication. To generalise behaviour and to approximate different functionality, networks are divided into three abstraction layers, commonly referred to as the data, the control, and the management plane. The data plane, also known as forwarding plane, carries and transports the data of the network. Temporary storing, queueing, and encapsulation of information is done by consulting prevalent routing tables, policies, and data sets written by the control plane. The control plane computes the network and forwarding paths, decides actions for incoming packets, and synchronises the general data flow of the network. The access interfaces of the hardware devices and the administration stations constitute the last logical part of the network, the management plane. Operations which are performed outside the normal automated scope of the network, such as general policy definition, troubleshooting, device configuration or manual actions, are part of this classification. In summary, the management defines and configures a general policy, which the control plane translates and the data plane executes. [11]

In line with the philosophy of Paul Baran and his colleagues, conventional networks combined the three planes into a single device, vertically integrating the control and data plane and assuring the independence of every node of the network. [12] Each individual router or switch calculates a personal transport path through the network and does scarcely rely on central control. Several routing protocols maintain information about the network. The devices configure their paths dynamically with the help of routing protocols such as Open-Shortest-Path-First (OSPF) or Intermediate System to Intermediate System (IS-IS), which gather link state information from all the devices of the network and construct an optimal network topology. In order to avoid transport loops in the switch topology, ports are blocked using path computation protocols, most commonly the Spanning Tree Protocol (STP) or Shortest Path Bridging (SPB). [13] To discover new elements in the network the devices may broadcast ARP request messages or utilise the Link Layer Discovery Protocol (LLDP), which periodically sends and collects information about routers in the network. [14] A group of routers connected over these (interior gateway) protocols is referred to as autonomous system (AS). Routers seated on the edge of the AS utilise the Border Gateway Protocol (BGP) to connect and transport traffic from autonomous systems to autonomous system, while simultaneously adhering to the policies implemented by network operators. [13] All protocols operate independently in every switch and router of the network. As each device maintains its own information and protocol stack the control plane is distributed over the entire AS. If the data plane receives an unknown packet the datagram is forwarded to the control plane for further directions. [11] Although they are contained within a single device, control and data plane are separated on a hardware level, with each plane possessing its own processing units. The switch or router data plane uses the fast application-specific integrated circuit (ASIC) or switching fabric to process and forward known data packets at cable speed. The control intelligence is implemented in a general purpose CPU to compute the necessary action for an unknown or special packet. [15]

Security Measures

Such a decentralised autonomous system has to receive sufficient protection from abuse and malicious intent. However, specialised security measures for each individual network element might have a tremendous impact on overall performance and network costs. Administrators therefore rely on various appliances between demarcation points or in front of important assets to harden their network. This section presents common techniques and designs to secure a network on the basis of the information provided in *Information Security, The Complete Reference* [16]

A company network is typically divided into different logical zones. The internal section of an organisation forms the intranet. Any information exchanged, stored, or published here is designated as private and protected from the public domain. If companies desire to exchange services and information with other businesses without exposing their private network, an extranet can be established to connect two sites over the internet. Although an extranet is based on mutual trust, it adds new attack opportunities and must be independently secured. In many cases, organisations have assets which can or need to be accessed by the public, including web or email services. Since the open exposure makes these servers a likely target of an attack, they are contained in a separate area, the demilitarised zone (DMZ). The DMZ resides between two security domains and can be seen as the outer ward of a company's network.

A simple technique to enforce this separation and to add security to a network is to enhance the tool set of routers and switches. To prevent ARP spoofing, devices can be configured only to react to certain pre-defined MAC addresses. Similarly, the virtual local area network (VLAN) capability of a switch can limit the amount of potential threats and aids in the division of the network into varying levels of security. If an administrator desires to block specific data or user packets from travelling through his network, routers and switches are equipped with Access Control Lists (ACLs). ACLs specify a white- or blacklist for particular TCP/UDP ports or IP addresses and defend either the hosts or, in case of edge or border routers, a whole network chunk from attacks. Likewise, routers and switches write logs of any network activity and provide access via the management plane of the device. Mechanisms to monitor this information are either a custom command line interface (CLI) of the vendor, the standardised Simple Network Management Protocol (SNMP) or the emerging, XML-based configuration access NETCONF [17]. The connection is generally secured using TLS or the Secure Shell (SSH), but may occasionally be established over unsafe remote command lines such as Telnet. Administrators may use these tools to identify unusual behaviour in the network or to configure an affected router. The management connection may be performed *out-of-band*. An out-of-band communication is either physically or virtually separated from the network and does not travel on the same channel as conventional data traffic. Although more expensive than in-band access, the operator retains full control over the network even when normal links are overburdened and is able to reconfigure the individual devices unhindered.

However, a dedicated intruder can easily overcome these features. In order to adequately defend a conventional network, operators utilise specialised devices between nodes, commonly referred to as middleboxes. A common and essential protective tool is the firewall. Although the ACL of a router can filter IP and port addresses, it is not able to remember the state of a connection or react dynamically. Operators have to manually configure the individual devices to specify addresses and can only react to threats. Stateful firewalls extend the functionality of ACLs and screen or block undesired activities. Packets that do not match the typical procedure of a specified communication (e.g. an unusual TCP handshake) are kept from accessing the network. Modern firewall versions are even capable of blocking unspecified behaviour in the traffic of trusted applications or detect reconnaissance attempts such as the scan of open server ports.

Often integrated into firewalls is Network Address Translation (NAT), which converts global into local IP addresses in order to save IP4 address space. Although originally not intended to function as security appliance, NAT hides the structure of the internal network and protects hosts from targeted attacks. As the storing capabilities of routers and switches are typically limited, firewalls additionally provide an extended logging and auditing capability for the management plane. They are often placed between network bottlenecks or gateways to observe a maximum amount of traffic, but further defence lines can also protect highly sensitive hosts or network areas.

Firewalls are sufficient to block specific network applications or traffic in most cases, but they do not have any means to detect suspicious activity. This defensive network feature is implemented in a intrusion detection system (IDS). An IDS has a similar functionality as a simple anti-virus program. It detects and reports threats based on familiar malicious packet signatures or unusual network behaviour. A common version is the network-based intrusion detection system. Similar to firewalls, it is placed at strategically advantageous points and monitors incoming and outgoing traffic.

The reactive approach however faces a significant problem. When intrusions are detected, countermeasures might already be too late. Intrusion Prevention Systems (IPS) therefore extend the functionality to not only report threats to the administrative station but also try to diminish or repel a detected attack as early as possible. Over the course of time the functionality of firewalls and the IPS has converged significantly, as intrusion reaction has become more crucial. In many cases threat solutions are delivered as Unified Threat Management (UTM), a combined set of firewalls, IDS/IPS, and additional measures. A new type of mechanism developed for UTM is the Deep Packet Inspection (DPI), in which the payload of selected traffic is examined for virus or spam patterns and filtered accordingly. In many cases traffic has to pass a sequence of all the previous static security devices before it is permitted to enter the internal networks. Nevertheless, the utmost important part of a security framework is the administrator himself. He is required to be aware of any ongoings or incidents in the network and has to recognise false positive or negative alerts. To be able to fulfil these requirements, administrators rely on security information and event management (SIEM) systems. Any relevant device reports incidents, real time data, and logs to the SIEM, which simplifies the data to a high degree and presents it in a human-readable way. Since a SIEM is generally not an automated tool and provides an operational interface, it can be considered the extended hand of the administrator and part of the network management plane.

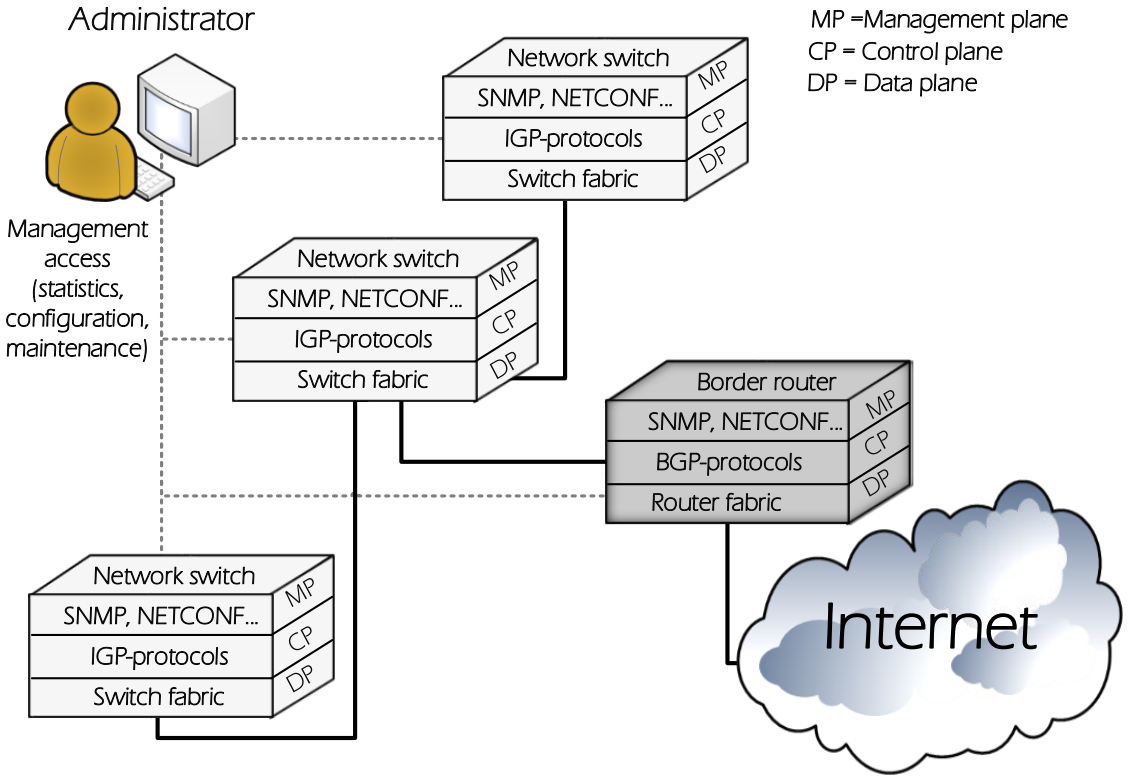

Figure 2.1: Distribution of logical planes in legacy networks.

Constructing a Network Blueprint

Figure 2.1 visualises the distribution of the logical planes in the network. Every switch and router contains its own data and control plane, formed by the various network protocols and the forwarding CPU. Each device provides management functionality and statistics to a central administration station. The routing policies, VLAN, and access control of every switch and router in the network have to be manually set up and configured by the administrators using the management protocols. Figure 2.2 represents an instance of a generic company network with added security mechanisms. For each important demarcation point or sensitive host a new defensive devices or threat management appliance has to be placed. The intranet is protected from the extranet and internet by a second line of firewalls and intrusion detection systems. A router with an ACL protects the DMZ from undesired access while the open web services are protected by a third firewall. While sensitive applications and data might receive special protection, the internal communication of the company network is not universally monitored. All devices have to reconfigured by the administrator individually and events are reported to a central management system in the intranet.

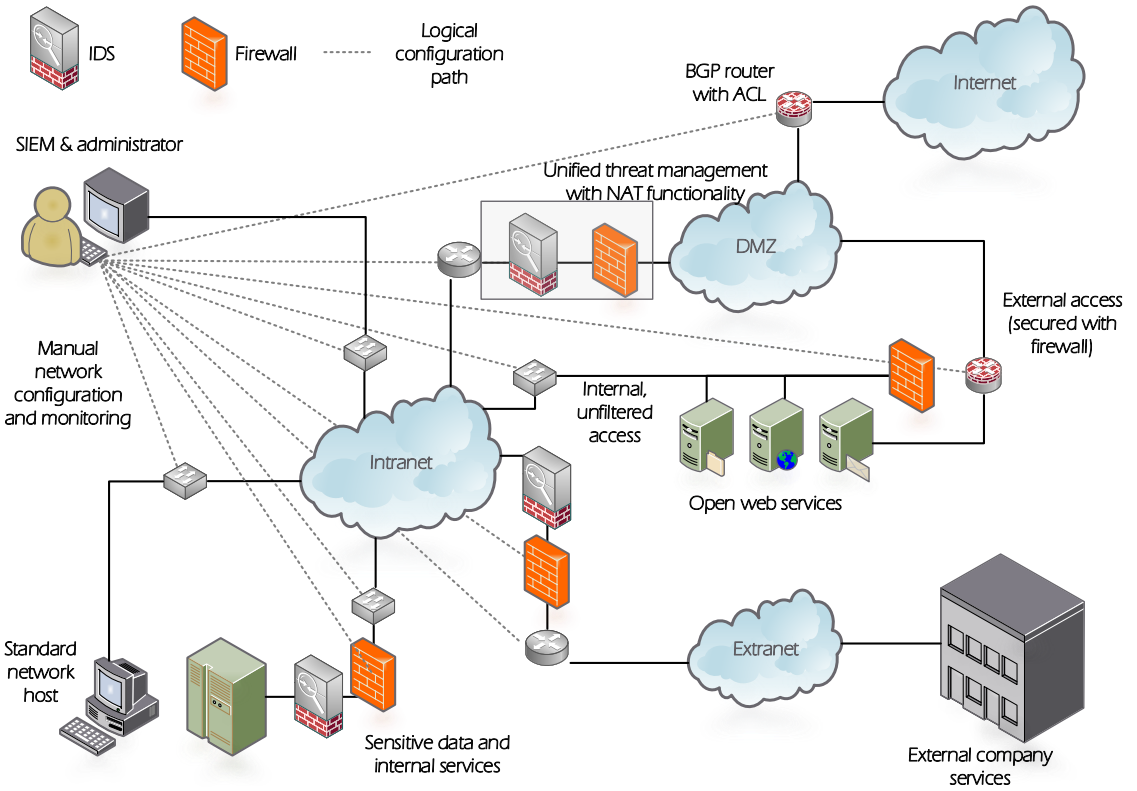

Figure 2.2: A sample architecture of a modern network.

2.2 The Motivation for Software-Defined Networking

Software-Defined Networking plans to abandon the previous network design, which centred around reliability. Instead of devices maintaining their own control and routing, the responsibility falls to external controllers. The result is a fully separated data and control plane, but a high level of dependency. Considering the success of the initial very robust design, a departure seems questionable. Indeed, the SDN movement has been controversial.
However, a significant portion of the networking community deems a change necessary and advocates for the shift to SDN. A large list of companies, including Microsoft, Ebay, Goldman Sachs or Yahoo, are investigating SDN solutions. [18] Google has already come forward to announce they implemented a fully functional data centre using the SDN paradigm. [19] An other example of the increasing interest is the company Nicira, founded by a developer of the popular SDN implementation OpenFlow. In 2012 the startup has been bought by major cloud software vendor VMWare for an unprecedented sum of over 1.2 billion dollars [20].
Several reasons have driven this rise in popularity. A large influence has been the desire to simplify general networking. New requirements and challenges such as the increased traffic, security, and the need for quality of service (QoS) have introduced a troubling amount of complexity. [21] Relying on device-specific protocols and vertical integration has continuously increased the difficulty to manage a network. As devices have to be configured individually, misconfiguration errors are a common occurrence. [22], [23] The integration of middleboxes to compensate for the lack of inherent functionality further complicates the infrastructure. [24] Researchers claim that the time spent on maintenance and additional equipment has given a rise to operational and capital costs while proprietary and incompatible devices impede deployment of new technologies. A common belief is that the underlying infrastructure and incompatibility stifle flexibility and innovation in the network. [21], [25], [26]
In order to simplify networking and to mitigate the aforementioned problems, researchers have called for abstraction of the discipline. Similar to the field of programming, the underlying mechanisms should be hidden and the functionality generalised. The control of the network is no longer distributed in protocols but decoupled from the single devices. A specialised entity collects and maintains knowledge of the entire network composition. The data plane of the network is abstracted and thus the control plane is agnostic of any technical details and mechanisms of the underlying layer. Devices of the control plane are solely responsible for communicating network goals and policies of the management plane. Performing these commands is only done by the devices of the forwarding layer. The individual layers are required to provide application programming interfaces (APIs) for flexibility and programmability. This separation of concerns presumably simplifies the architecture of the network and facilitates abstraction into manageable pieces. [27] Overall, the new network architecture aims to represent the design of a conventional modern computer. The applications cooperate with an operating system which translates the abstract commands it receives for the underlying hardware processors to execute. In theory, an introduction of programmable and abstracted networks with a logically centralised control would abolish a majority of protocols and middleboxes. With the equipment being programmable, most expensive and vendor-specific solutions are expected to become obsolete. Instead of having to purchase middleboxes, operators would be able to implement their own control and security solutions by developing and deploying custom programs on general purpose servers. A major hope is that operational costs would decrease significantly due to simple control of the network and the reduction of overall proprietary devices. [2], [12]

2.3 Past and Present of SDN

Programmable networks are not a recent emergence. In fact, the concept has been steadily progressing and developing for over 20 years. Feeling limited in their ability to experiment and overwhelmed by the growing complexity of network architecture, researchers were looking for novel systems which might enable innovation. [2] The first major endeavour in this area has been the active networking project [28]. Whereas a network is normally agnostic of the packet content they receive, active networks were not. Switches could be configured out-of-band to react differently to user data. Introducing extended data plane functionality, users could even send *capsule* packets which were able to directly alter the behaviour of a network node. By and large, active networks only suggested programmability of the network, they did not advocate for a pragmatic approach at simplifying network management. In the end the project did not manage to achieve widespread adoption, partially due to a lack of necessity, initial deployment complexity and heavy security implications. [2], [29]

Several years later, in the first half of the 2000s, research focus shifted to entangling the data and control plane. This separation has been seen as one of the central points in the simplification of network design. A pioneer in this area is the Forwarding and Control Element Separation (ForCES) [30]. ForCES classified the network components into two distinct types. Any unit that solely forwards and filters traffic is designated as forwarding element. The instructions how to process packets are provided by a control element. The elements are connected via a standardised open interface, which is considered to be a core feature of the ForCES protocol. The motivation is to facilitate vendor interoperability and increase the scalability of the network. A diverse set of multiple control elements could utilise the non-proprietary access point of the hardware-based data plane. ForCES did not emphasise logically centralised control, but is complemented by parallel projects such as the Path Computing Element (PCE) [31] and the Routing Control Platform (RCP) [32]. The PCE introduced a central computation node for carrier networks while the RCP was intended to unify the distributed BGP path computation from a central position. Both concepts ease the management and coordination of inter-domain traffic for internet service providers. [2] Although ForCES is still under active development and provides backwards compatibility, it has not been successful in deployment. Major vendors had little incentive to abolish their proprietary API and a lack of clear language abstraction limited the range of functionality. [2]

In an effort to rethink network structure, clean-slate projects surfaced, drawing inspiration from the preceding innovations and suggesting an entirely new perspective. With past experience and current knowledge in mind, researchers approached network design from the ground up. An influential formulation of the new way of thinking has been the 4D architecture [11]. 4D centres around 3 key principles which depart from the decentralised and low-level perspective. It encourages a network-wide view and global objectives enforced by routing elements directly controlling the entire domain. The network itself is divided into the 4 "D" planes. A **D**iscovery Plane estimates the capabilities of the network and provides the information to the **D**ecision Plane. In the spirit of the new paradigm the decision plane regulates the network and configures the devices of the **D**ata Plane, which processes and forwards packets. Providing a logical and separate channel, the **D**issemination Plane is the point of connection between the two planes, comparable to the open interface in ForCES. 4D is not a practical solution but a proposal for a new approach and provided incentive to redesign the network. Building on the design principles established by the clean-slate

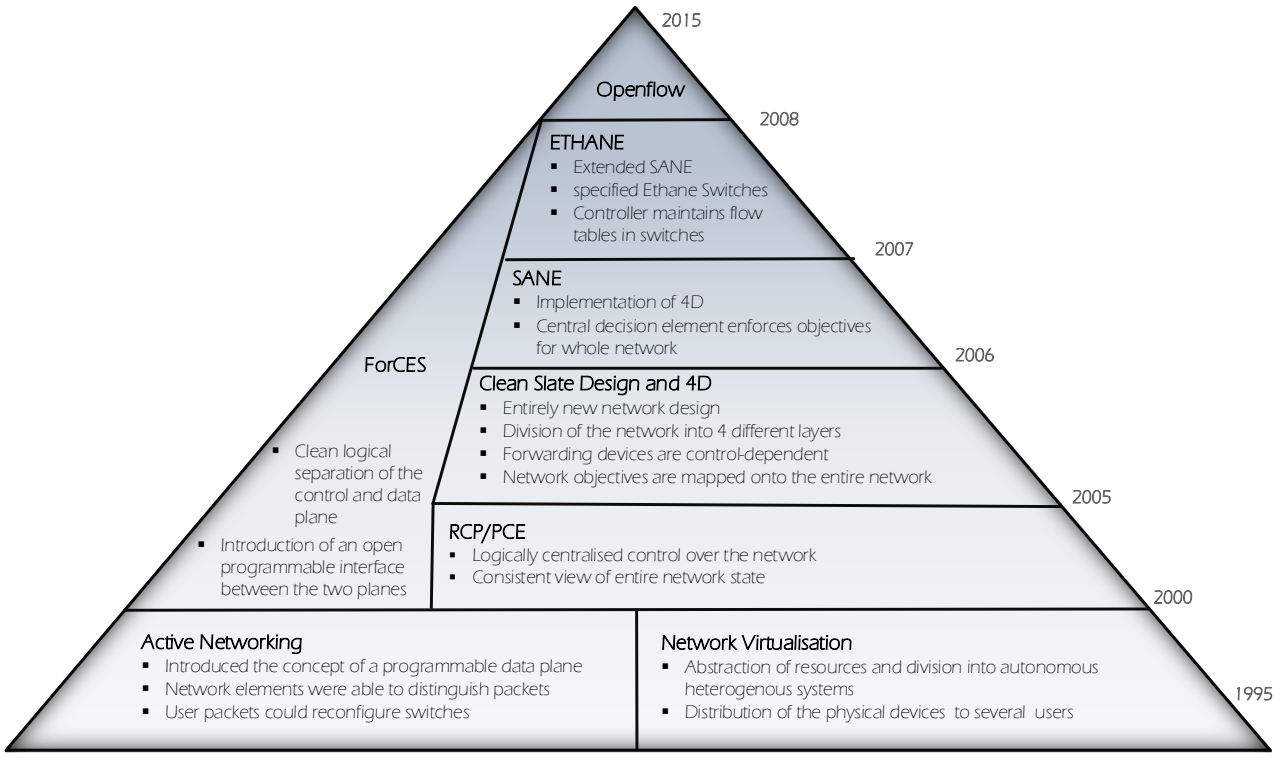

Figure 2.3: Timeline of programmable networks on the road to SDN.

project, a Stanford team developed SANE [33]. Intended to be a feasible implementation of the concept in enterprise networks, the framework implements a single protection layer containing servers which enforce global security and access policies. ETHANE [34] extended the work of SANE and introduced a centralised controller and specialised ETHANE switches. The controller communicates with the switches and updates their flow tables according to an administrator policy. In contrast to routing tables, a flow-based system does not route based on destination of IP and MAC addresses but rather matches the header fields against pre-existent table entries and forwards the packet out on the specified port. As the protocol does not operate based route calculations, the view of the devices is reduced to their own structure. From the view of the controller a continuous traffic flow travelling over generic network nodes is created. ETHANE was successfully deployed in the Stanford computer science department and paved the way for the next step in Software-Defined Networking, known as OpenFlow [3]. The research and building stones of the past years eventually culminated in the publication of this protocol, which was heavily inspired by ETHANE.

Similar to preceding solutions OpenFlow was initially designed to catalyse innovation and to lower costs in campus networks. However, due to its backwards compatibility and simple deployability it managed to breach into company and data centre networks. The success of OpenFlow in the industry was solidified with the formation of the Open Networking Foundation (ONF) in 2011. As of 2015 the organisation has more than 150 industry members and receives endorsement by network equipment vendors such as Cisco, Dell, Brocade and HP. [18] The term Software-Defined Networking itself became closely associated with the

project [5], [12], but despite being a major component in the network, OpenFlow is not synonymous with Software-Defined Networking. Several instantiations, including ForCES and PCE, are feasible. [35] To address growing confusion and to prevent spread of misinformation, various standard committees and surveys have published definition and taxonomy guides. [12], [35]–[37]
The general consensus as to what Software-Defined Networking is, or tries to be, is summarised in the following four principles:

- **Separation of data and control plane**
  In any version of a SDN the planes must be logically separated and connected via an interface. The control aspect is removed from forwarding devices and delegated to an external entity.

- **Network programmability**
  The provision of open APIs is a core aspect and eponymous for the paradigm. Software and scripts should be able to access, configure, and modify network elements with ease.

- **Network abstraction**
  The view of the network is virtualised for any elements of a higher hierarchy. Services and applications are aware of the state of the whole network, but physical attributes and resources are irrelevant for configurations and computations.

- **Logically centralised control**
  All forwarding devices of a domain are linked to a controlling entity and are subject to its enacted policies.

Despite the large variety of implementations falling under this classification (see Figure 2.3), OpenFlow is among the most adopted and influential SDN solutions. [29], [38] While there may be vulnerabilities specific to the protocol, many threats arise due to the inherent structure of SDN and can be applied to similar communication and management protocols of the data and control plane. OpenFlow will thus serve as point of reference when discussing modern software-defined networks, their architecture, and security. [2], [4]

2.4 OpenFlow: A General Overview

This chapter specifies the most recent technical details of the OpenFlow protocol and presents an overview of popular controller designs. The specific sections explain the individual layers, the mechanics of an OpenFlow switch and controller, and define the structure of a general OpenFlow network.

2.4.1 Standardisation Work

OpenFlow is steadily maintained and enhanced by the Open Networking Foundation. The group regularly publishes technical specifications and recommendations for SDN deployment and architectures to propagate their vision of a software-defined network and to promote open standards during development. The size and influence of the ONF ensure that OpenFlow effectively acts as standard for the southbound interface (i.e. the connection between the forwarding and control elements). In the wake of OpenFlow's success the Internet Research Task Force (IRTF) has also has chartered a SDN Research Group (SDNRG) and tries to complement standard publications with informational RFC reports. Less prolific standard organisations (SDOs) involved in SDN and OpenFlow include the ITU-T, ETSI, IEEE, and IETF. [39] While the data plane has been largely unified under OpenFlow, northbound standardisation has yet to reach a common consensus. A large variety of controller and control platforms exists, tailored to any specific need or demands. [12] As a result, the ONF has launched the Northbound Interfaces working group "to define and subsequently standardise various SDN Controller Northbound API Interfaces (NBIs)" [40]. To the author's knowledge, the ONF effort remains the only SDO invested in unifying the northbound interface to this day. Since its inception in October 2012 the working group has not published any further documents. Possible reasons for the hesitation might involve concerns of a premature commitment to a prevalent design. A flawed publication could stifle innovation or fade into irrelevancy in favour of superior solutions. As there is no clearly dominating concept in sight, an extensive standard encompassing multiple APIs is also highly probable. [41]

Nevertheless, the *ONF Architecture & Framework* group has presented an abstraction of the minimal capabilities of SDN controllers and the northbound interface. [36] To narrow the breadth of SDN, the ONF suggestions draw the baseline of a plain network and its inherent security. Key standards utilised in this chapter are the *OpenFlow Switch Specification 1.5.0* [42] and *SDN Architecture 1.0* [36].

2.4.2 OpenFlow Infrastructure

Figure 2.4 depicts the general structure and idea of an OpenFlow network. Multiple switches are connected to one OpenFlow-capable controller via the *OF-Switch protocol* [42]. The switches are specialised OpenFlow devices utilising flow tables curated by the controller. They are designed as simple forwarding units and constitute the data or infrastructure plane of the network. Legacy protocol functionality such as OSPF is relocated from the physical layer to higher planes. The controllers, which are also capable of collaborating over a west- or eastbound interface [12], comprise the control plane. The northbound interface connects the control and the application plane. OpenFlow applications define detailed policies to enact by the controllers below and configure the data plane by proxy. SDN applications may in fact be seen as controllers of controllers. The highest authority in the network, the manage-

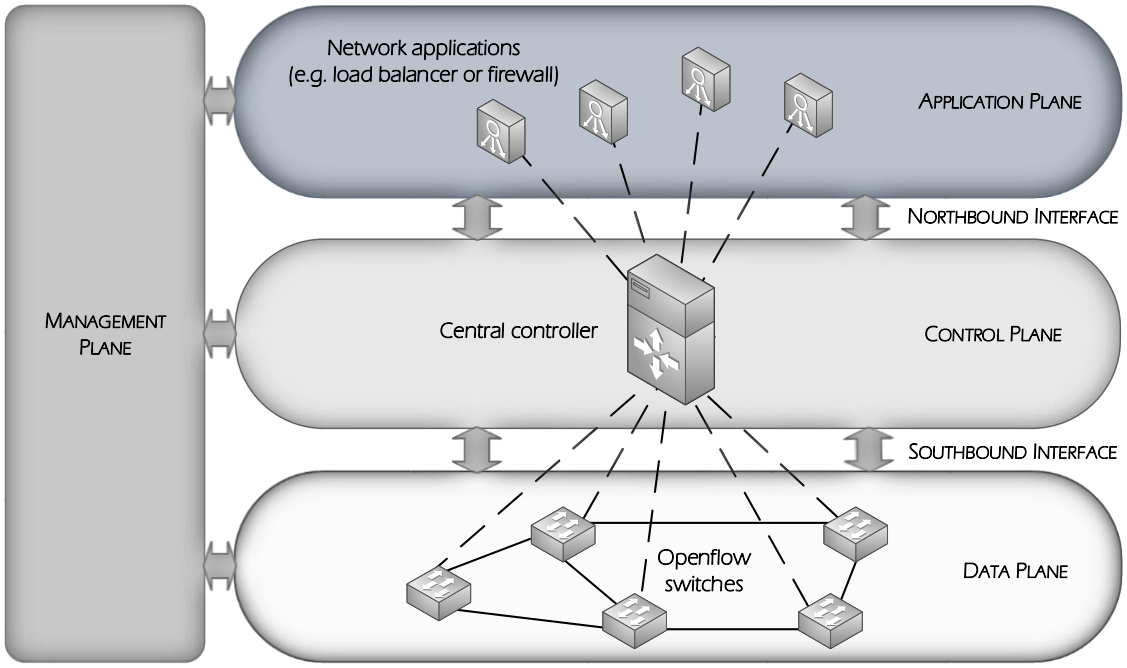

Figure 2.4: In a software-defined network, the various logical planes are strictly separated. The controller is the central element of the network, while the management has logical access to all devices.

ment plane, is vertically integrated across all layers for debugging and installing purposes. In the fashion of classic networks, management may consist of a central administration system or, as it is referred to in the ONF architecture recommendation, a generic operations support system (OSS). The management includes tasks such as dividing the network into separate domains for operators or clients, network monitoring or adherence to negotiated service level agreements. The second distinct protocol, *OF-Config* [43], denotes a direct control and management interface for OpenFlow switches which is implemented using the NETCONF [17] protocol. It is complementary to the OF-Switch specification and aims to specify manual configuration access of OpenFlow switches. All network elements contains a resource database (RDB), also known as Network Information Base (NIB), and a client agent for resource abstraction and execution of higher level commands. Control is exposed to other network devices. Every level has a virtualised view on lower and provides abstraction to higher tiers. This structure provides the possibility to recursively stack control planes and theoretically assign different trust domains. The recursive hierarchical model is designed to enhance security and scalability of the network. The aforementioned planes are inspected in detail in the subsequent section.

2.4.3 Data Plane

In OpenFlow the data plane denotes any device capable of processing and forwarding user traffic. The protocol is designed to adapt to new concepts with no restrictions imposed on choice of transport technology such as Ethernet and IP. Technically, the OpenFlow standard only specifies the functionality of forwarding elements and implements an API consisting of instruction primitives for higher levels. Control plane elements can utilise the language provided by OpenFlow. As a southbound interface, it establishes a connection between the two planes but does not constrain freedom in data, control, and application design. Nevertheless, the switch specification and its fundamental requirements paint a clear picture of the OpenFlow data plane structure. For simplicity the ONF refers to any unit that contains a data path and controller connection as OpenFlow switch, as layer two and three routing functionality of the OSI model is moved to the control plane. OpenFlow-compliant switches can be implemented physically or virtually. To accelerate deployment, hybrid solutions containing both classic routing intelligence and the independent OpenFlow logic are a possibility. However, in all variants the basic switch structure remains the same.

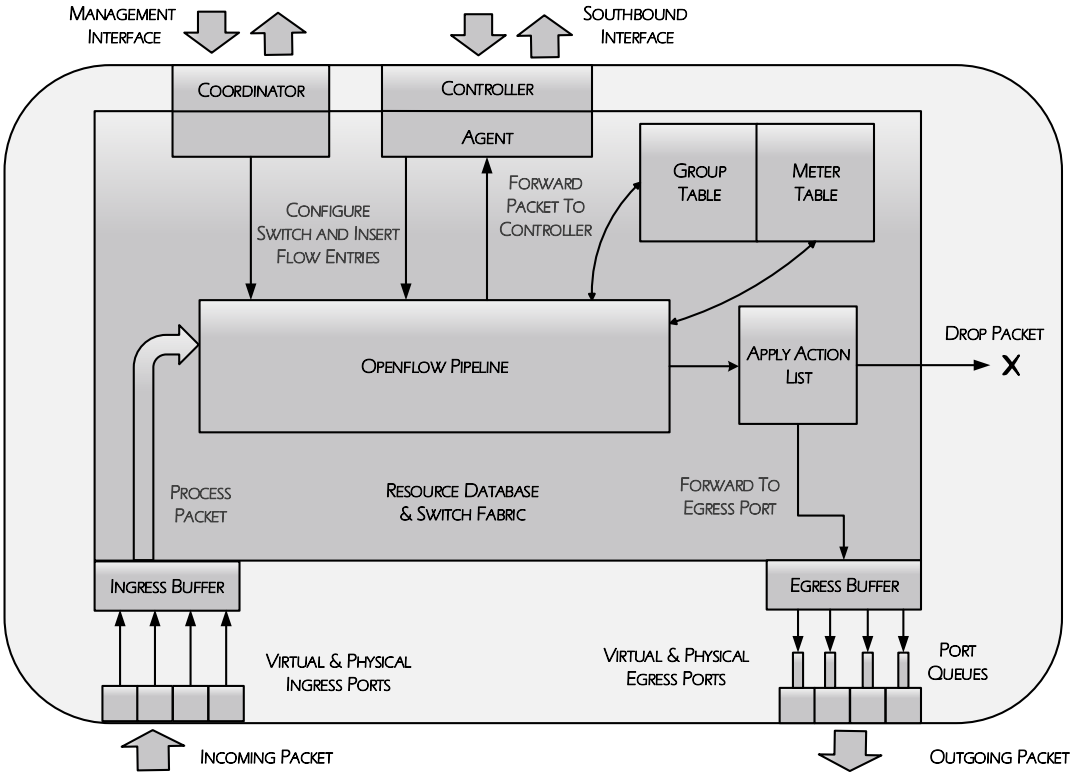

Figure 2.5: Internal mechanics of an OpenFlow switch.

OpenFlow Switches

The OpenFlow switch can be decomposed into several logical components:
Physical and logical ports, the OpenFlow pipeline, and the OpenFlow control communication channel (see Figure 2.5).

OpenFlow Ports

An OpenFlow switch must implement three port types. Physical and logical ports form the standard variant. The third type is the reserved port. In contrast to physical ports, which rest on the hardware interface of the device, logical ports are unbounded. They can, however, be mapped to a physical port and used for network services. From a switch perspective, physical and logical ports are not distinguished. Reserved ports are a special virtual type influencing the forwarding behaviour of the switch. Switches can write several types of reserved ports into the header of an OpenFlow datagram and differentiate between ingress and egress ports of the pipeline. There are several noteworthy port types in an OpenFlow switch. The CONTROLLER port indicates that a packet is to be forwarded to the controller or, if CONTROLLER is written as ingress port, originates from a controller. ALL is a generic flooding feature relevant for all OpenFlow ports of the switch, FLOOD signifies the same behaviour for all legacy ports in hybrid devices. TABLE is only available as egress port and is used by the controller to denote that a packet should be queued and processed by the standard flow table pipeline.

OpenFlow embraces traditional port queueing. Queues are attached to logical and physical output ports and are identified by special identification numbers. Packets can be sorted into different queue types and assigned to a port to facilitate QoS measures or algorithms.

OpenFlow Pipeline

The heart of every OpenFlow switching device is the OpenFlow pipeline, a sequence of one or more flow tables. Any table contains a list of flow entries consisting of a set of fields (see Figure 2.6). The flow tables and their entries are stored in Ternary Content Addressable Memory (TCAM). TCAM provides fast access but only low data storing capacities, which is further reduced due to the increased size of the forwarding tables. [12] The header fields of incoming packets are compared to entries in the match field. Properties may include MAC and IP source and destination address or special metadata inserted by previous flow table actions. The match field is designed to be expandable and contains any comparable packet element. It is possible to wildcard irrelevant fields in order to provide extensive matching. The counter field tracks statistics about each logical component of the device and monitors packet quantity or the age of installed flow tables. The packet is passed down the processing pipeline until a match is found in a table. If the packet matches multiple times in one flow table, only the entry with the highest priority is selected. However, multiple actions can be executed for different tables in the pipeline. A packet without a corresponding entry is either dropped or processed by the table-miss instruction, if such an entry exists. In reactive approaches a table-miss instruction may forward the packet as PACKET_IN message to the controller for further advice. The controller either drops the packet or installs a suitable flow entry in the switch. The switch may truncate the packet and buffer the payload until the controller responds in order to reduce the overall load on the channel. The timeline field of a flow entry specifies its time-to-live in order to avoid superfluous entries and to prevent an overflow of the limited flow table memory. Cookies are simple identifiers inserted by

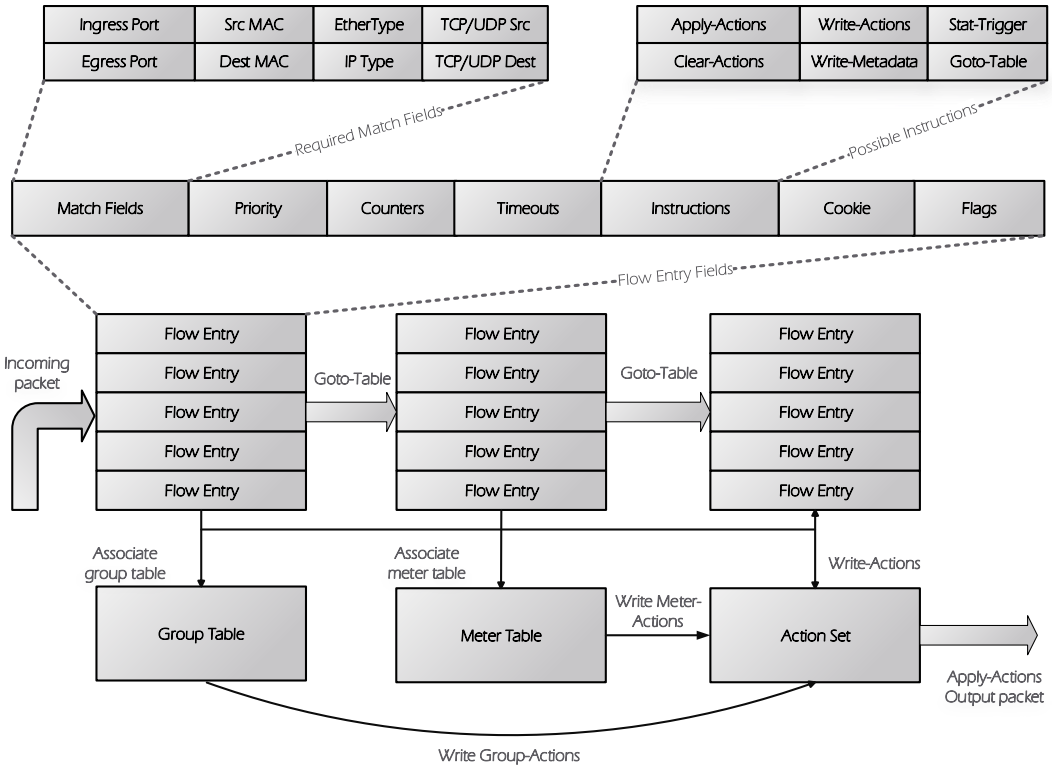

Figure 2.6: The OpenFlow processing pipeline.

the controller to foster performance and are not relevant for processing. Flags can be set to require special handling such as notifying the controller when a flow table is modified. Once a packet match has occurred, the switched updates the statistics in the counters group and performs the associated instructions. Instructions either call or modify an *action set*, write metadata, notify the controller of statistics or jump to the next flow table for additional instructions. Every packet is associated with an initially empty action set and is subsequently filled during processing of the flow table entries. If the packet reaches the end of the pipeline or an flow entry is instructed, the aggregated list of actions is executed. Actions are an essential part in the programmability of OpenFlow switches. A variety of actions exists and the list is continuously extended. Noteworthy examples are listed in Table 2.1.

To simplify the forwarding process and to reduce the space wasted by specific flow entries *group tables* were introduced in the v1.2 specification. Flow entry instructions can point a packet to a group containing pre-defined *action buckets*, which are then added to the action set of the individual packet. Another recent feature of OpenFlow switches are *meter bands* to enable traffic shaping and QoS. A meter band measures different types of packet rate at a specified port and is able to limit data rate or burst size of a particular flow to regulate traffic in an efficient manner.

Action	Description
`Drop`	Drops the packet.
`Output port_id`	Forwards the packet to any of the three port types.
`Group group_id`	Associates the packet with a certain output group.
`Set-Queue queue_id`	Chooses the queue at a selected port.
`Meter meter_id`	Associates the packet with a certain meter.
`Set-Field field_type value`	Modifies header fields of the packet for maximum granularity and customisation.

Table 2.1: Selection of valid actions in an OpenFlow flow entry.

Southbound Interface

The OpenFlow protocol itself represents the southbound interface of the network, a standardised and mutual communication method between the switch and the controller. Over the OpenFlow channel interface the remote controller configures the listed tables and supplies the switch with intelligence. It is possible to encrypt the channel with TLS, however due to performance considerations, TCP is also an option. The protocol supports three basic message types: *controller-to-switch*, *asynchronous* or *symmetric*. The controller-to-switch messages are initialised by the controller and contain a broad range of possible applications. Via a controller-to-switch message, the controller can request the identity, state, settings, and statistics of a switch or write the flow, meter or group tables. In the opposite direction, switches are able to send asynchronous messages to the controller. Any packet with CONTROLLER as egress port is sent as PACKET_IN message for further instructions. These notifications are generally reciprocated with a flow modification message or discarded. Modifications the switch performed autonomously, such as the deletion of a flow entry, are also broadcast. Additionally, when a link or controller goes down, a port state changes or miscellaneous network events occur, other controllers are informed immediately. Either side can send symmetric messages, which encompass *Hello*, *Error*, *Echo* and *Experimenter*. Hello messages initialise the OpenFlow connection between two devices. Echo verifies that the connection is still active, while Error generally reports a failed request. Experimenter is a message type reserved for future use and custom useful extensions to the OpenFlow protocol. To address concerns of availability, a switch can connect to multiple controllers which can take on the master, slave or equal role. Only master and equal controllers have full configuration access. The slave role may only read statistics. If one controller is promoted to master, all remaining controllers are demoted to slaves. In equal mode every controller has full rights. Additionally, a switch can establish auxiliary connections to bolster the main connection and increase bandwidth and reliability. The main connection must be implemented over TCP or TLS. Auxiliary connections may also use UDP and are associated with a main link using a specific ID. While UDP communication is restricted, it is possible to send PACKET_IN messages or read information about the switch configuration. If the main link fails, any auxiliary connections are also terminated.

The switch specification does not detail how multiple controllers coordinate their actions or how the topology is synchronised. If connection to the controller is lost, the switch instantly enters fail secure or standalone mode, depending on device support or configuration. In *fail secure mode* the switch continues to operate using the flow table information it has re-

tained and drops any unknown packets designated for the control channel. In *fail standalone mode* the (hybrid) switch reverts to native switching behaviour, if available, and resumes OpenFlow functionality after the connection is restored. Controller designs and details are not specified in the technical publications of the ONF. The thesis nevertheless attempts to sketch functionality and designs based on popular solutions and recommendations.

2.4.4 Control Plane

The control plane is the central intelligence in SDN and an essential aspect in the success and deployment of the software-defined network. It is represented by its control composition and devices. As development of the software and elements is not bound by OpenFlow or restricted by standards, a large landscape of implementations has evolved. Each version attempts to succeed in its vision of a network and fulfils different aspects seen necessary by its developers. Moreover, companies are continuously devising their own proprietary controllers adapted to proprietary interfaces. This thesis focuses on the open OpenFlow interface and thus on open-source OpenFlow controllers. To abstract the variety of controllers and their internal composition the chapter consults the *ONF Architecture Recommendation* [36] and selects several noteworthy control projects.

OpenFlow Controllers

Controllers can be divided into two distinct types. The first type is a physically and logically centralised controller with high individual performance linked to multiple switches. To guarantee fault tolerance and resilience, distributed controllers employing many-to-many connections have been introduced. [44] In order to secure the consistency in a switch, the ONF recommends that controllers are either strictly synchronised or manage disjunct slices of the network while exchanging information horizontally. [42] Controllers either operate in proactive or reactive mode. In the former mode, the controller pre-installs flows based on the known topology, which saves flow latency and reduces congestion created by controller requests. In reactive mode the controller waits until a switch inquires about the flow for a particular packet to take the network load and QoS conditions into account.

Even though a broad range of controllers exists, a few particularly influential implementations have emerged. NOX [45], made available to the research community by the Nicira startup has been the first open-source controller designed for OpenFlow. The controller utilises a relatively basic and centralised approach and two versions exist. NOX is the classic version written in C++ and dedicated to maximum performance. POX is the less complex but slower alternative written in Python. Development on these controllers has effectively been discontinued since 2012. [46] Nevertheless, a fundamental amount of research and security solutions have been performed on the basis of these versions.

Heavily backed by industry are the OpenDaylight [47] and Floodlight [48] projects. OpenDaylight is an open-source controller and a collaborative effort by several ONF members and The Linux Foundation. It is designed to be compatible with more southbound interfaces than OpenFlow (e.g. the PCE protocol or BGP configuration) and is regarded as one of the candidates with the highest potential for northbound API standardisation. [49] Floodlight has been forked from the academic Beacon controller and is maintained by Big Switch Networks. It is highly modular, supports multiprocessor parallelism and contains several already integrated application components such as path computation and device management.

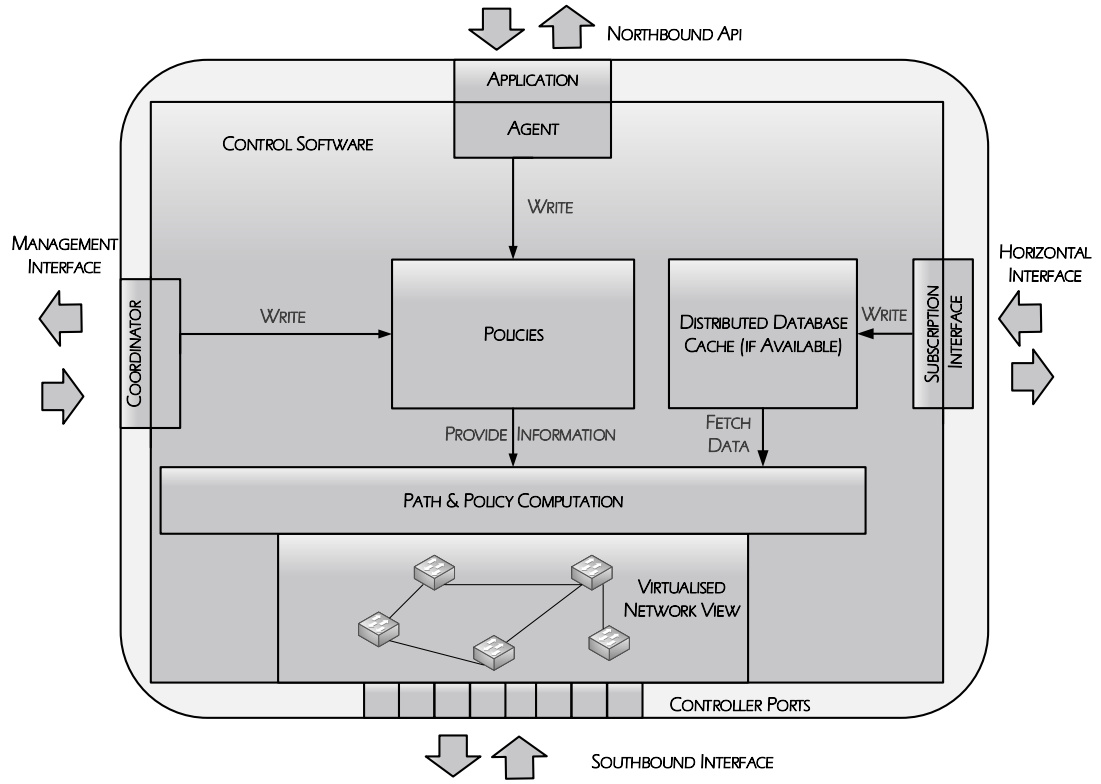

Figure 2.7: Internal mechanics of an OpenFlow controller.

A recently published and noteworthy controller is ONOS [50], which is specifically designed to achieve a carrier-grade control plane based on availability, scalability and performance. It is a natively distributed and logically centralised controller. Although multiple controllers are present, the data plane switches only connect to a single logical device.

Khondoker et al., 2014 [51] developed a decision making template to recommend a controller as objectively as possible. They selected RYU [52] in competition with OpenDaylight, Floodlight, Trema, and POX as the best candidate, based on criteria such as modularity, interface support, and maturity. Floodlight and OpenDaylight were placed second and third respectively. RYU is a composable framework based on Python and well documented and defined. It supports several southbound interfaces as well as the OF-Switch wire protocol. Floodlight, NOX/POX, RYU are centralised control architectures, while OpenDaylight and ONOS also offer distributed functionality via clustering.

The general SDN controller can be dissected into four different logical interfaces it needs to maintain (see Figure 2.7).

The first interface is the data-control plane interface (D-CPI), more commonly referred to as southbound interface. Due to the OpenFlow virtualisation of the network, the controller views switches as a connected list of active data paths and network states. The controller connects to the agent of the underlying switch, accesses the resource database information and enters the data into its own RDB. The resources the controller has collected from connected switches and network elements are again virtualised or abstracted for the view of

the management and northbound interface applications. All selected controllers support the OF-Switch protocol or additionally the OF-Config specification as their D-CPI. Although not all controller software implements the latest publication OF-Switch v1.5, it can be assumed that it will eventually be adopted.

The east-/westbound interface is implemented in distributed controller systems to guarantee consistency and the quick exchange of data. [12] Conventional OF-Switch controllers are only indirectly informed about adjacent peers by asynchronous switch messages and thus generally do not utilise a horizontal interface. However, due to scalability and performance concerns, controllers eventually need to be efficiently distributed. [44] Neither NOX nor Floodlight nor RYU seem to offer horizontal communication functionality and therefore no east-/westbound interface.

Several controller architectures have emerged which implement native and logically centralised distribution. The NOX extension Onix [53] is the most prominent example, with controllers connecting to a distributed data store and exchanging data over imports and exports. Building on Onix, OpenDaylight as well as ONOS support clustered controllers which share a distributed data cache and network information base (NIB), on which any participating controller can subscribe and write. [47], [50] As opposed to direct, inter-controller communication, all three approaches utilise common open-source distribution services to synchronise and maintain this data. The second approach is the placement of an orchestration layer atop or below of the individual controllers to enable communication via a proxy. Examples are Hyperflow [54], an additional layer to coordinate multiple controllers, and FlowVisor [55], a sub-layer to slice out autonomous network parts for each individual controller to manage.

The last interface is the northbound or Application-Control Plane Interface (A-CPI), which links the controller to SDN applications and top level devices. It is a highly contested territory as application requirements differ vastly. Similar to the switch providing information to the controller, the controller provides data to the network applications over the northbound connection. There is no clear access method defined for this interface and many controllers (e.g. OpenDaylight, Floodlight, RYU or ONOS) expose multiple interfaces. A common pattern is the use of a Representational State Transfer (REST) API for remote applications and a custom interface for direct and more extensive module integration based on the development language (e.g. OpenDaylight exposes a REST interface for web applications and a Java framework for "native applications"[47]). As the northbound interface is not standardised and interpretation may vary, applications and similar features are not portable and generally restricted to one controller type.

Similar to agents, all controllers are required to provide a coordinator functionality for the management interface. The access is commonly provided over a secure shell and a CLI in combination with the northbound interface of the controller (e.g. REST). Often, a graphical user interface (GUI) is supported to improve configuration procedures and usability. In addition to the APIs provided by controllers, several programming languages have emerged to foster the SDN paradigm of extracting simplicity. They move away from the lower level of OpenFlow to higher abstractions to significantly ease the programming of network applications. Again, SDN languages can be divided into language tiers. Lower languages such as Nettle are able to program a network controller, higher levels such as Frenetic are designed to develop applications in cooperation with an existing controller. [12]

2.4.5 Application and Management Plane

Although the control plane is often referred to as the brain of the network, the application plane fits this description more accurately. The SDN applications compute and develop the policies and instructions which the controller installs on the forwarding layer. However, as the field of potential use cases is highly diverse, no generic functionality can be extracted. Applications generally have entire control over the network slice they are connected to with the controller having to synchronise conflicting policies and actions. To avoid decision and operational conflicts, applications are often packaged with the controller, or consolidated as a single entity with internal conflict resolution. [56]

The existing network applications can be grouped into five categories: traffic engineering, mobility and wireless, measurement and monitoring, security and dependability, and data centre networking. [12] The fields measurement and security are particularly relevant for the scope of this thesis, as a large amount of security functionalities may be outsourced to the application plane.

The management plane is connected to all three planes and continuously queries state, performance, and network events. Contrary to the application layer the management plane is often represented by a single administrative station keeping vigil over the network. [36] The administrator performs manual tasks which controller and applications are not technically capable of or authorised to enact. Installing new network elements or software, dividing and ration resources and identifying issues are part of the responsibility. In general, any Operations, Administrations, and Management (OAM) [57] functionality is employed in the management plane and the switches but can potentially be extended to SDN applications. An example of a useful OpenFlow tool an administrator might use is the dpctl [58] command, which requests flow information and is able to manually insert flow tables into a switch. The OF-Config specification additionally defines management functions to directly configure an OpenFlow switch. As it is common practice it can be assumed that any management connectivity is performed over a SSH terminal or authorised connection.[16] Note that in SDN the administrative station is assumed to have full configuration access and authority over every single element in the network.

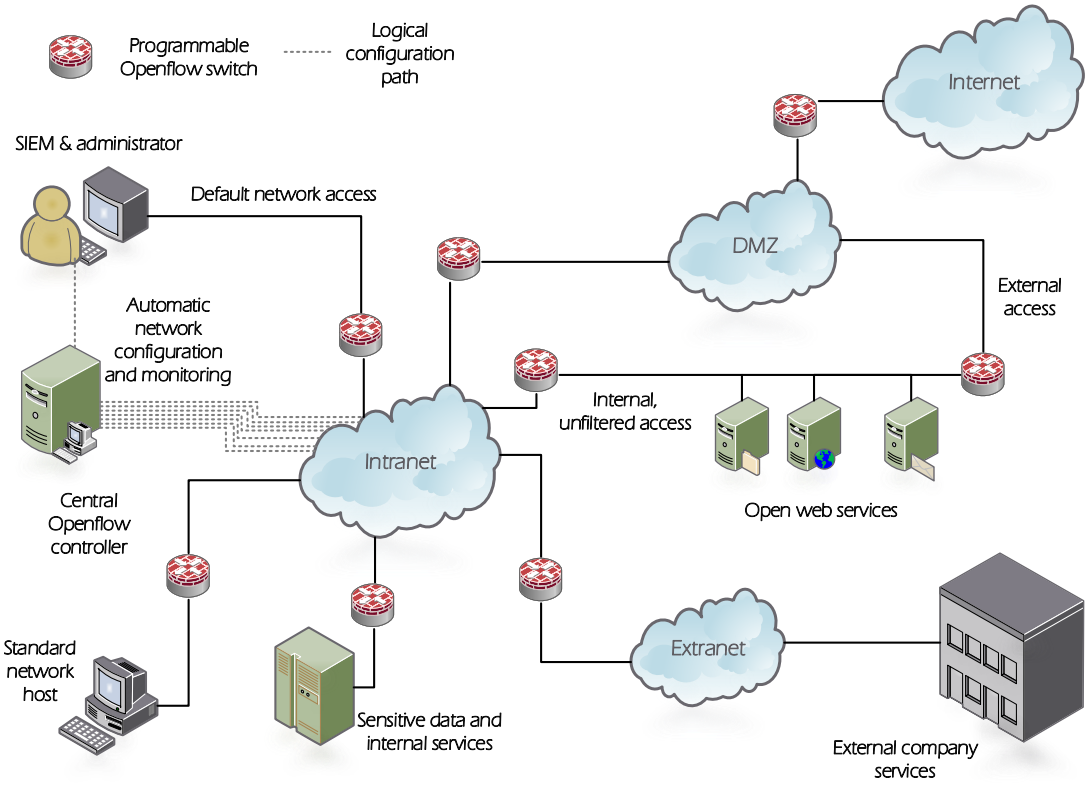

Figure 2.8: The OpenFlow network.

2.4.6 Constructing a SDN Blueprint

Knowing the technical details, a software-defined network and its individual components can be constructed. Several virtual and physical switches are connected to a single controller over the OpenFlow southbound interface. Instead of manually configuring the devices of the network, the administrator formulates a general network aim or develops applications on the controller, which enact an optimal network configuration. The firewall, access control, and VLAN channels are distributed over the OpenFlow switches and are automatically adjusted by the controller, which is aware of every single switch state in the network. A central management system connects to all devices of the network for individual configuration, or may instead fetch all information from a management application installed on the central controller. It should be noted that one goal of SDN is to supersede static middleboxes placed at demarcation points using flexible security applications or generic server distributed in the network. However, generic SDN does not employ any intrusion detection or load-balancing mechanisms and the necessary interoperability of OpenFlow controllers with middleboxes is not yet clearly defined in this thesis. Therefore, specialised appliances are not included in this network base design. Over the baseline technical details of SDN presented in this chapter, the security of typical software-defined networks can be examined and a technical data flow diagram can be devised.

3 Security Threats

Following the public discourse, SDN appears as a technology which is able to solve all the nuisances and problems of legacy networks. However, the academic invention has been largely tried and tested in safe and small environments and not in actual company or data centre networks. So far it has not been of interest for malicious intruders and hackers and obvious problems might have not been exposed yet. The question remains, whether SDN is capable of overcoming the security, reliability, and scalability issues which have been newly introduced or inherited from conventional networks. As SDN gained traction, more and more security assessments and tests have been performed on the new paradigm. Several comprehensive surveys have already commented on its state of security. Kreutz et al., 2014 [12] and Jarraya, Madi, and Debbabi, 2014 [4] dedicate a chapter to security deficiencies and research in their surveys. Scott-Hayward, O'Callaghan,and Sezer, 2013 [59] focus on security and address challenges and opportunities in their survey of SDN. There are several attempts to specifically analyse the OpenFlow protocol. [60]–[62] Using STRIDE and attack trees, Klöti, 2014 [63] utilises a similar approach as this thesis, but does not extend the scope beyond the OpenFlow v1.0 protocol and OpenFlow switches and only models three of the six threats. In the fashion of these surveys, this chapter tries to thoroughly examine the security issues of a standard SDN and compiles the contemporary and latest literature while supplementing personal consideration. The STRIDE methodology is applied to the network constructed in Chapter 2.4 and the individual threat aspects are summarised and visualised using attack vector charts.

3.1 General Adversary and Threat Methodologies

Threat models are a popular method to roughly assess the security of a system or the overall company. A variety of models exists. The most relevant approach for this thesis is the software-centric model, which abstracts individual assets and components of the system with the purpose is to highlight potential misconfiguration or malfunctioning parts.

Many of the secure design approaches in information technology are grounded in the eight secure design principles proposed by Saltzer and Schroeder in 1975 [64] (see Table 3.1). The Microsoft Security Development Lifecycle (SDL) includes these principles in their thread modelling step and utilises the STRIDE methodology. Although SDN is not completely software-based, the SDL approach is also applicable to the overall design of the network and its technical components. The book *Thread Modeling: Designing for Security* [65] serves as the foundation for the description of STRIDE in the next chapter.

Principle	Description
Economy of mechanism	Keep the design as simple as possible to avoid unnecessary mistakes.
Fail-safe defaults	Only allow access by special permission instead of blocking intruders.
Complete Mediation	Check any object at any time for unauthorised access.
Open design	Overall design should not be obscured and depend on ignorance of the attacker.
Separation of privilege.	In order to be able to perform an operations in a system at least two conditions should to be met.
Least privilege	System elements are only given as much access as they require to function.
Least common mechanism	The system should avoid shared resources or, if they are necessary, treat them with special care.
Psychological acceptability	Security measures and policies must be simple, reasonable, and realisable to minimise mistakes and gain widespread adoption.

Table 3.1: The eight design principles as defined by Saltzer and Schroeder. [64]

3.1.1 STRIDE

STRIDE visualises and decomposes a system in order to infer inconsistencies or potential deficiencies. To lower complexity and to gain an overview the product is divided into individual sensitive components and modelled into a data flow diagram (DFD) (see Table 3.2). The term itself is an acronym of six basic security threats: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. These threats violate several properties, which are seen as necessary to establish a secure system:
Spoofing infringes on the authentication property and allows attackers to conceal or fake their identity to gain access to information or cloud their attack. Potential targets, which can be spoofed in a system, are interactors or system processes. A lack of proper authentication enables the spoofing of network elements and might result in man-in-the-middle (MITM) attacks. Due to a mutual dependency, a single STRIDE threat might trigger or induce further attack possibilities. Consequently, MITM attacks are associated with several vulnerabilities in the network.
Tampering is the second threat and affects the integrity of the data and systems, as system processes, data flows or any data stores might be modified. A change in data integrity or system components must be instantly identifiable or visible and reported. Otherwise an attacker might stealthily gain financial advantage, damage sensitive enterprise property or cause legal repercussions.
Repudiation is a closely related problem. Interactors and processes must not be able to deny any actions in the system or access to restricted components or areas. Any action has to be accurately tracked. Additionally, the possibility to tamper with logs, recordings, and documents must be eliminated to guarantee absolute responsibility.
Information disclosure is a significant vulnerability and affects the confidentiality of the system, a fundamental aspect of IT Security. If data flows, stores or processes accidentally

Component	Description	Graphical representation
Data Flow	Flow of information.	One-sided arrow →
Data Store	Sensitive data linked to data flows.	Parallel lines =
Process	Algorithm, code, or machine.	Circle ◯
Multi Process	Complex entity.	Double-lined circle ◎
Interactors	System end points and users.	Rectangle □
Trust Boundary	Different levels of trust and security.	Dotted line - - -

Table 3.2: Components of a typical data flow diagram.

expose sensitive information during standard operations the user or company might suffer severe repercussions. Eavesdropping on the right data streams or communication channels provides malicious users with a range of possibilities.
Denial of service reduces the availability rate of the system and prevents customers and users from accessing data or services. While interactors generally can not be overloaded or shut down, other technical components very well might be. Abusing the capabilities of the system is one of the most simple and popular methods to threaten or damage an enemy financially. Ensuring the perpetual availability is thus of the utmost concern for network operators and companies.
Elevation of privilege is the last principle and violates proper authorisation. If an attacker is able to pose as higher authority to grant himself more capabilities, he might gain access to large sections of the system, bypass guard mechanisms, and proceed undetected and undisturbed. This threat emerges from lack of access control mechanisms, differentiation of user authority or successful Spoofing.

After having modelled the DFD, each component is evaluated and analysed for its robustness or vulnerabilities based on these aspects. STRIDE can be extended with *Attack Trees* to trace the path and behaviour of a potential attackers. These models are a popular method to reflect on overall security of the systems architecture and evaluate the feasibility of various attacks.

3.1.2 Attack Trees

According to Bruce Schneier, attack trees are defined as "a formal, methodical way of describing the security of systems based on varying attacks. [7] Attack trees are not strictly defined and may be customised at will to adapt to the problem at hand. In general the tree represents the viewpoint of an attacker trying to achieve a certain objective, in this case one of the STRIDE aspects. The root node is the goal of the attacker, while the leafs of the node form sub-goals. The tree is decomposed into elementary or complex operations and requirements. Complex operations may be deconstructed into a more detailed attack tree. A path to a node is achieved by a junction of AND or OR gates. Either one or all of multiple actions are required to rise up in the tree hierarchy. Once all possible attack paths have been found and established, the feasibility of the attacks is assessed. Actions which are very costly, unlikely or averted by the base design are marked, more detailed description might also be applied. The remaining paths are open security vulnerabilities, which have to be prevented or addressed. Figure 3.1 shows a small, simplified model of a Denial of Service attack on a network router. Rectangular forms signify elementary actions, while

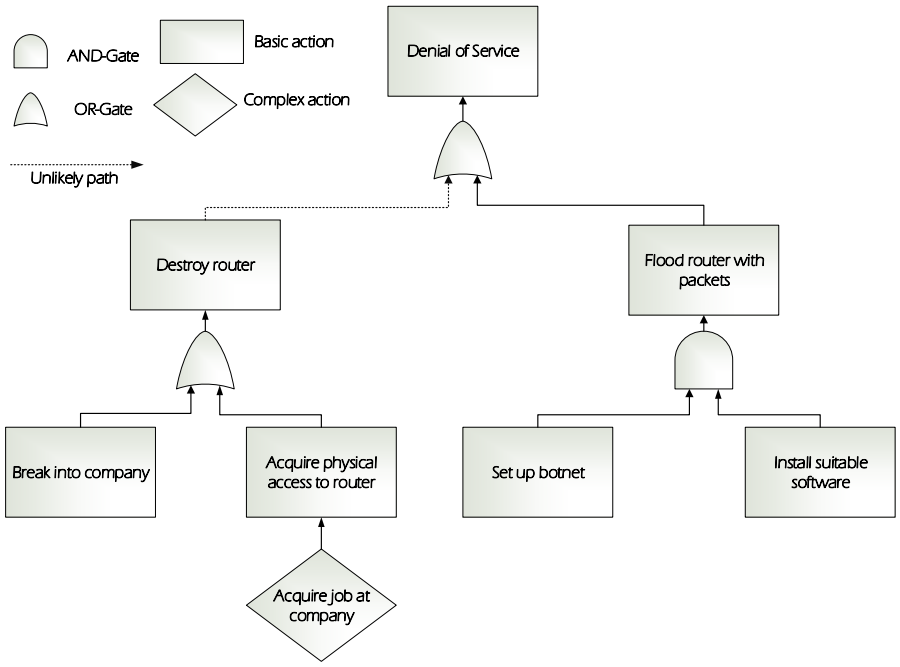

Figure 3.1: An attack tree showing methods to incapacitate the main router of a network.

diamond shapes are complex achievements. Dotted lines highlight a path which is unlikely or sufficiently mitigated.

3.1.3 General Threat Modelling Methodologies

STRIDE is not the only risk assessment methodology. Three common types of approaches exist: Asset-centric, attacker-centric, and software-centric. Asset-centric focusses on sensitive data, information or hardware, which needs to be protected. For each asset, the damaging methods and the overall impact of a loss are evaluated. STRIDE can be extended with DREAD (Damage, Reproducibility, Exploitability, Affected Users, Discoverability) to gauge the potential chance or extent of an attack. Similarly, the Common Vulnerability Scoring System (CVSS) [66], an industry standard to rate and assess the potential of a security vulnerability, can be utilised to prioritise and identify security issues for individual assets. However, these models are not part of the scope of the thesis as they classify security priority and company loss, which is not relevant for theoretical frameworks, such as the model established in the previous chapter. The attack-centric method humanises, categorises attackers into several personalities (e.g the "Script Kiddie" or "Agent") and estimates their capabilities in the current system. According to these capabilities, prevention methods are developed and applied. This approach is less concerned about the technical intricacies of the relevant system and more about the damage potential and risk of varyingly professional attackers. The software-centric approach is the most suitable methodology, as it disassembles the system into single interworking mechanisms, actors, dependencies, and trust boundaries. The software-defined network can be interpreted as a large operating system structure and thus the software-centric model can aptly analyse and highlight potential

vulnerabilities. Instead of DFDs, the software architecture can also be decomposed into UML diagrams, state-charts or Petri nets. Ariss, Wu, and Xu, 2011 [67] presented an approach to integrate attack trees into state charts to visualise the insecure state of a system. Nevertheless, these approaches are likely more suitable for software systems with processes and varying system states and not fluid, interaction-based structures as in SDN. The thesis *OpenFlow - A Security Analysis* [63] discusses further methodologies for threat modelling and presents extensions to already existing models.

3.2 STRIDE Assessment

The decomposition of a system in STRIDE is a necessary measure to accurately understand the potential deficiencies of the product and graphical analysis of a data flow diagram is a useful methodology to inspect an SDN architecture. Based on the technical depictions of Chapter 2 the DFD symbolises the flow and storage of information in a software-defined network.

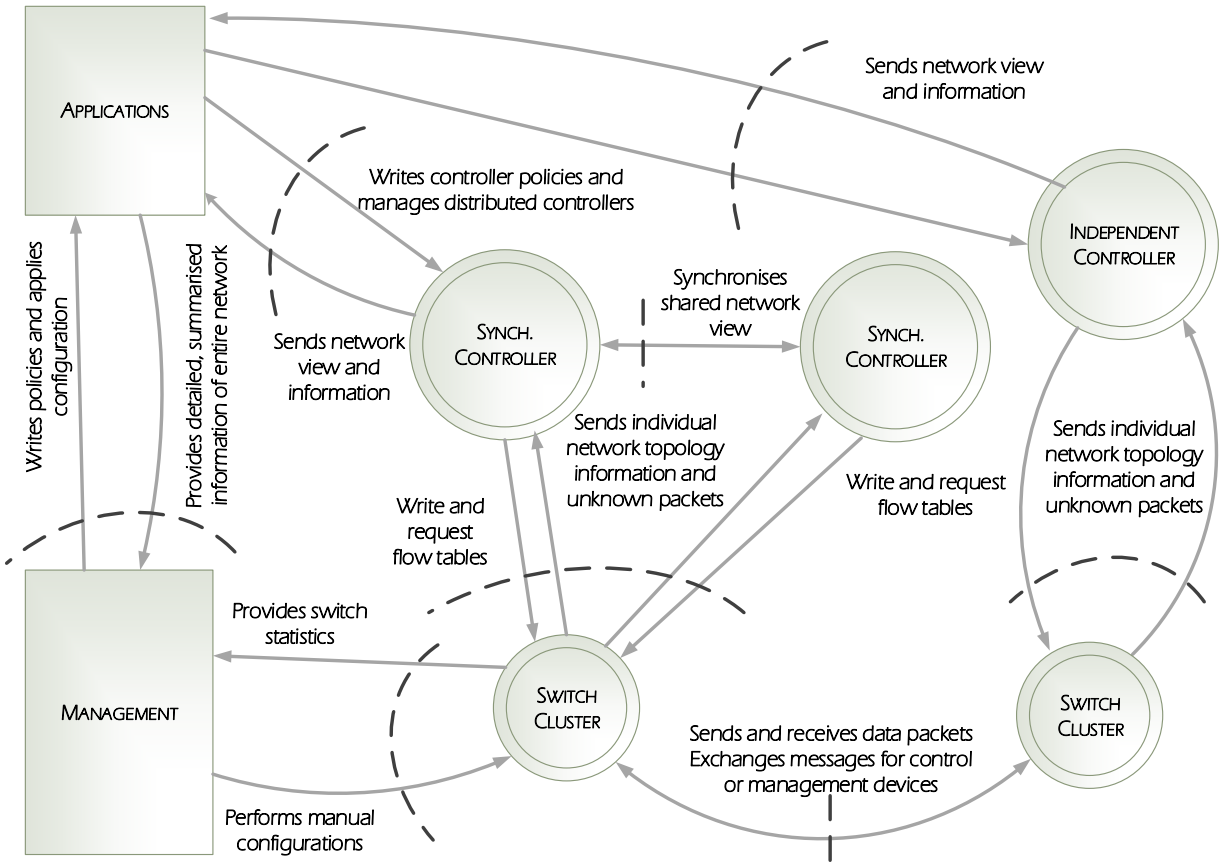

Figure 3.2: Data flow of a generic OpenFlow setup employing two synchronised and one distributed independent controller.

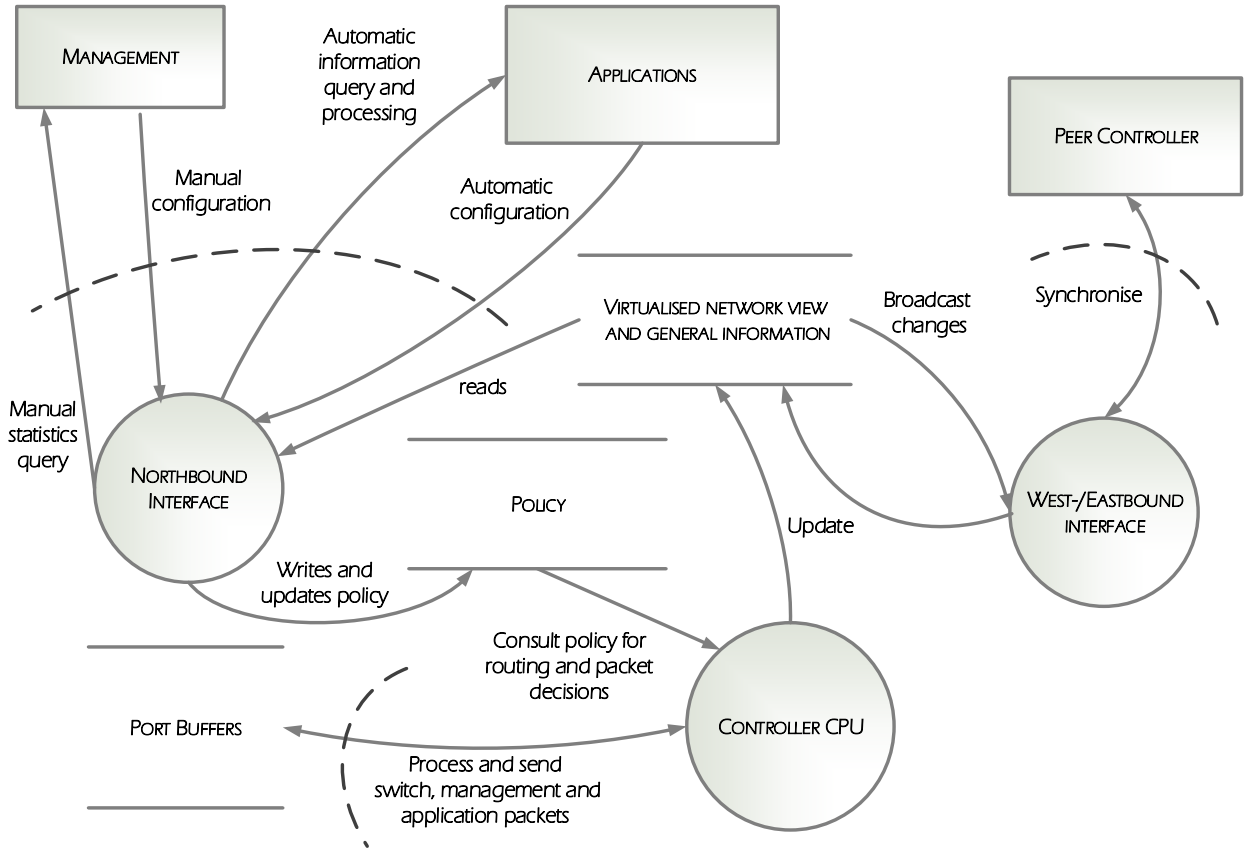

Figure 3.3: Data flow of a SDN controller.

3.2.1 Data Flow Analysis

Four main bidirectional data streams are present in SDN. The management is logically connected to all devices, while the three remaining components are hierarchically layered. Compromising any of the main units impacts the entire network and therefore each entity possesses their own trust boundaries. It is mandatory for a secure network to properly authenticate any and all communication of the four main components. Section 2.4 discussed the eventual necessity of distributed controllers for the sake of reliability and scalability. The method of communication is not clearly defined in the ONF specifications, but it can be assumed that controllers are either orchestrated by applications or communicate natively over the horizontal interfaces. The devices either control disjunct slices of the network or are strictly synchronised. Management and applications are highly diverse and can not be sufficiently abstracted into DFD elements. They are regarded as external entities. Nevertheless, the upper layers offer a vantage point for an attacker and are most likely a major source of attacks. Applications are particularly vulnerable to software exploits and should not be fully trusted by the controller.

Switches and controllers are modelled into several smaller, logical parts, based on the details of Section 2.4. A controller is expected to run on a virtual server with a power-

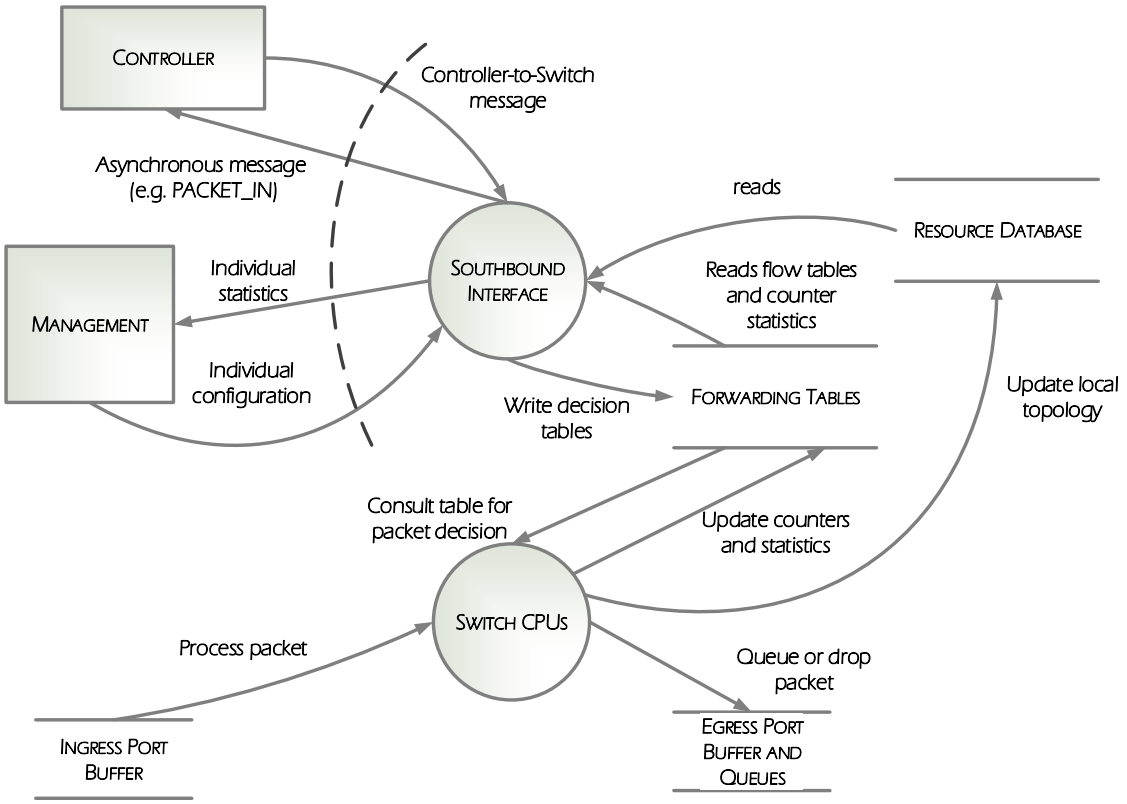

Figure 3.4: Data flow of a SDN switch.

ful computing unit and connects to the network via physical ports. All versions maintain the policy database, which is written manually by the management, or automatically by applications. The application and management interface are merged into the northbound interface, as they enact the same functionality from the view of the controller. For the sake of simplicity, routing and computing applications integrated into the controller software are not regarded as separated process. They operate within the trust domain of the controller and may consume the same system resources. If the internal routing and application intelligence is compromised, so is the controller. Natively distributed controllers contain a shared cache database, which also influences topology and computing behaviour. Management and applications access topology information and statistics over the API of the controller. Asynchronous messages of switches or adjacent controllers form the main view of the network. Combining the view with the policy, the controller formulates and computes forwarding decisions of the applications and sends control commands to the switches.

A switch contains ingress and egress ports with attached queues or buffers, the southbound interface, and the tables. Meter, group, and forwarding tables are consolidated into the umbrella term *forwarding decision tables*. Although not further detailed in the DFD, meter tables and the capability to automatically limit bandwidth are possibly relevant for DoS mitigation, while the individual statistics saved in the counters and forwarding tables might

be subject to Tampering. Switches generally provide individual link discovery information and general local statistics to higher hierarchical entities. The ONF does not mandate that the communication with the control plane takes place over out-of-band connections. Management and control messages might be sent over the standard data network, the only requirement is a TCP/IP connectivity. An in-band communication of configuration messages is a possibility. [42] The entire DFD is divided into three sub-DFDs. Figure 3.2 represents the overall flow structure, Figure 3.3 the internal flow of a controller and Figure 3.4 the switch. There are two STRIDE variants to the conventional method. STRIDE-per-element applies every aspect to every component of the DFD. However, for complex systems such as a SDN isolating every component does not provide sufficient insight into vulnerable dependencies and is a redundant approach. The second variant focusses on interactions of the individual parts and how attackers might abuse these interactions. While this approach is far more detailed, the complexity of the diagram increases considerably. The chosen method in this thesis is a basic, general model of the data flow in the network and the view of an attacker trying to achieve individual goals. For every aspect, an attack tree is generated to abuse any of the components of the network, some of which are bilaterally dependent. Attacks which are not feasible or already prevented by the basic specifications and design choices will be marked as such.

3.2.2 Spoofing

Spoofing an element of the network is frequently the first step of a larger attack. Whether an attacker successfully mimics, compromises or implants a device is negligible as long as the goal of trust or inconspicuousness is achieved. Conventional ARP Poisoning is assumed to be weakened due to the ARP-agnostic nature of OpenFlow switches and the abolishment of automatic switch broadcasts. Controllers nonetheless are still vulnerable to spoofed addresses of internal hosts. [68] In this thesis and in the context of SDN, a Spoofing attempt is further denoted as deceiving other devices to be a legitimate member of the network (e.g. a switch, controller or application), which implies a lack of proper authentication. Spoofing is often a gateway for other STRIDE threats and is considered to be a base threat in the following sections.

Four main components are susceptible to Spoofing in a software-defined network. The switch, the controller, the applications, and the management.

The internal, affected mechanics of the switch and controller are the interfaces and their respective authentication procedures. It can be assumed that hijacking and spoofing the controllers of the network is the highest priority of an attacker. The controller cooperates with all logical planes and has control over the entire network. Without reliable certificates it is easily achievable to connect as a legitimate network element. The virtualisation of controllers and switches in a software-defined network simplifies the insertion of a malicious device by installing it on a compromised host. [69] The right MITM placement of a decoy switch potentially gives an intruder control over the entire remaining network slice and enables a broad range of attacks and insights. [60] In this case, Tampering also prohibits a further detection by administrative services as the switch can modify all OpenFlow notification and control messages.

Out-of-band communication is not required as it is cost-intensive and elaborate to implement. The potential absence of a separate channel renders the northbound interface openly accessible in the network and prone to a man-in-the-middle attack over a compromised

switch. [69] A controller that does not demand authentication can be misled to enact commands on behalf of the attacker. Depending on the implementation, natively distributed controllers could fall victim to a rogue version which has replaced a legitimate controller or registered as a new peer. If the access method of the administrator and applications is not an out-of-band communication, the entire controller could be spoofed for other network members, thus resulting in an absolute need for strong authentication mechanisms.

To counter the threat of Spoofing, every trust boundary in the main DFD has to be sufficiently secured. The OpenFlow protocol supports TLS as a relevant measure. [42] However, the authentication service is often not fully implemented, not activated by default in controller and switch implementations, or is simply disabled by network administrators due to performance concerns. [60] The focus on data centres, which are often regarded as "physically secure" [60], neglected strict authentication measures and recommendations. The last broad survey on TLS support in controllers and switches has been conducted in 2013 and highlighted a severe lack of TLS adoption. [60] The report states that most OpenFlow switch vendors do not support TLS. Coincidently, since 2011 the requirement of TLS support has been dropped in the OF-Switch specification and the link between switches and controllers has been renamed from *secure channel* to *controller channel*. TLS is still optional or requires manual implementation in various control softwares (e.g. Floodlight, NOX or OpenDaylight). Additionally, switches may not be authenticated since mutual authentication is not demanded. [60] The developer team of Floodlight currently has no plans to deploy native TLS support, since the "control plane is typically isolated from the data plane and in a secure environment" [70]. The protocol has to be manually installed and configured using additional extensions. It is to be expected that past criticism and rising awareness will eventually drive further TLS adoption. Since the release of the critical OpenFlow security assessment in 2013, the criticised controller architectures seem to have implemented at least rudimentary support for the TLS protocol. [51] Considering the fact that the protocol supports authentication, security-conscious administrators and companies should be able to choose acceptable solutions in the future. Nevertheless, a network employing TLS still faces challenges, as the protocol suffers from a constant stream of new bugs and exploits. In fact, OpenSSL is considered to be one of the most vulnerable software types due to common usage of unsupported versions. [71] The ONF does not specify or mandate a particular TLS version. One of the dangers is backwards-compatibility and fall-back mechanisms to insecure, older version, most recently demonstrated by the POODLE attack. [72] Furthermore, UDP messages with a static data path ID [42] can freely be sent over a auxiliary connection. As it is possible to guess the ID due to low randomisation, spoofed UDP configuration messages may be injected into the control channel. The OpenFlow protocol supports VLAN flow matching, which also introduces a virtually separate control and management channel and might prevent the spoofing of a controller or switch and MITM attacks. This dynamic and automatic configuration reduces potential misconfiguration in VLAN setups. Additionally, flows can be configured to drop control and management traffic from questionable sources. However, by compromising a switch, VLAN hopping, or modification of suitable flow tags, the attacker might gain access to these channels. These problems are not exclusive to SDN, but the centralised structure extends the potential impact of a vulnerability significantly. While IPSec Tunnel Mode between switches and controllers is theoretically supported to guarantee authenticity and confidentiality in the network [42], adoption by vendors and developers is not observable, most likely due to lack of demand or interest.

The northbound interface of controllers is problematic in SDN and highly vulnerable to

Spoofing. Many network operating systems rely on unencrypted HTTP basic authentication with weak or no passwords at all. HTTPS is not enabled by default. [73] Controllers utilise common software frameworks and APIs, which, if not properly maintained by code reviews, might introduce a broad range of known and unknown vulnerabilities. [12] Frequently, multiple southbound and northbound APIs are supported, each exposing a different set of weak points. A DEFCON demonstration in 2014 [73] showed the possibilities of exploiting the widespread Floodlight and OpenDaylight software. The speaker accesses the interface of the controllers as switch and authenticates himself as trusted. The spoofed switch gains control over the entire network by sending application messages to the controller and blocking out any detection devices or administrators from repairing the network.

Code hardening, separate channels, and rigid authentication for any access point are the only native measures to counter an breach in the general controller design. It is necessary to detect suspicious activity early to prevent a hijacking of the northbound interface. As the management interface of switches is accessed over SSH by default, the connection can be considered secure. The OF-Config publication [43], which includes a specification of management access, demands a SSH or TLS connection to the switch. The STRIDE assessment by Brandt et al., 2014 [62] also infers an inherent resilience of the management connection against Spoofing. Klöti, 2014 [63] presumes that hijacking the management interface is only possible via more extensive attack methods. Whether via HTTPS, SSH or an application proxy, the management connection to the controller depends on the choice of the developer. Nonetheless, previously discussed considerations regarding any type of authentication measure also apply.

The OF-Switch specification does not further define the method of synchronisation between controllers. Strictly adhering to the definition of multiple controllers in the specification, a rogue distributed controller could demote all legitimate controllers to the slave role and enact the intruder's policies. Switches in listener mode are particularly vulnerable to this attack, if they accept any incoming connection attempt. Distributed controllers, which are coordinated over the application plane, are susceptible to the security issues of SDN applications and the northbound interface. If the controller possesses a horizontal interface, it is necessary to validate the new peers. Neither Onix [53] nor OpenDaylight [74] nor OpenDaylight seem to define or implement a suitable protocol for verification and allow new controllers to instantly subscribe to information and write data. However, these controller softwares may use third-party distributed databases such as Apache ZooKeeper, which employs TLS authentication.

All these attacks are likely performed from a hijacked unit of the intra- or extranet. Either remote or physical access to a host are a requirement before an attacker is able to advance. The preparatory work is not exclusive to SDN, yet an integral part of Spoofing. Soft- or hardware exploits, social engineering, malware or unmonitored remote access are common tactics. The *Trustwave 2012 Report* [75] identifies remote access applications as the most widely used intrusion. Infecting clients is the second-most frequent method to compromise a legitimate device. A new emerging risk is the trend of Bring-Your-Own-Device (BYOD) in companies. [75] Employees may bring their own laptops, smartphones, or tablets and connect these to the intranet. An attacker can take advantage of naive users and gain simple access to the externally secured network by installing malicious software on the BYOD device host and waiting until the user connects at his workplace.

Figure 3.5 shows the attack tree developed for this STRIDE threat for each affected component. As soon as an attacker has gained access to the network either physically or

remotely, he has various options to deceive switches or controllers. It is unlikely that an attacker gains physical access to the network and a physical breach can not be prevented by network devices. It is presumed that available authentication is properly implemented by responsible operators and that an intruder can not send switch or command messages. Nevertheless, security exploits do exist on various levels and are of more variety than in legacy networks.

Comparison to Traditional Architecture

Although an attacker must use conventional methods to establish himself in the network, his succeeding capabilities are considerably extended. SDN introduces the controller and applications, two new imitable targets in the network. The new logical components are capable of exerting a great amount of power over the entire network and are therefore a prime target. The programmability and programming interfaces potentially harbour a multitude of new security holes and virtualisation of physical network devices, such as switches and the controller, lowers the barrier for imitation. Traditional authentication protocols exist as countermeasure. However, as highlighted in the security threat reports, they might not be sufficient to protect controllers and switches from abuse. The increased impact makes Spoofing is a sizeable threat in SDN, more so than in legacy networks. Even assuming all trust boundaries are secured in the network, Spoofing is still possible. It is thus considered a base vulnerability for further threats. Security-conscious network operators need to address this threat with special care and caution and deploy mechanism to automatically detect spoofed connections and devices based on suspicious behaviour. A new possibility of SDN is the deployment of sophisticated applications, which are able to detect and neutralise Spoofing in the entire network. A necessary tool to avert the overall danger of this STRIDE threat.

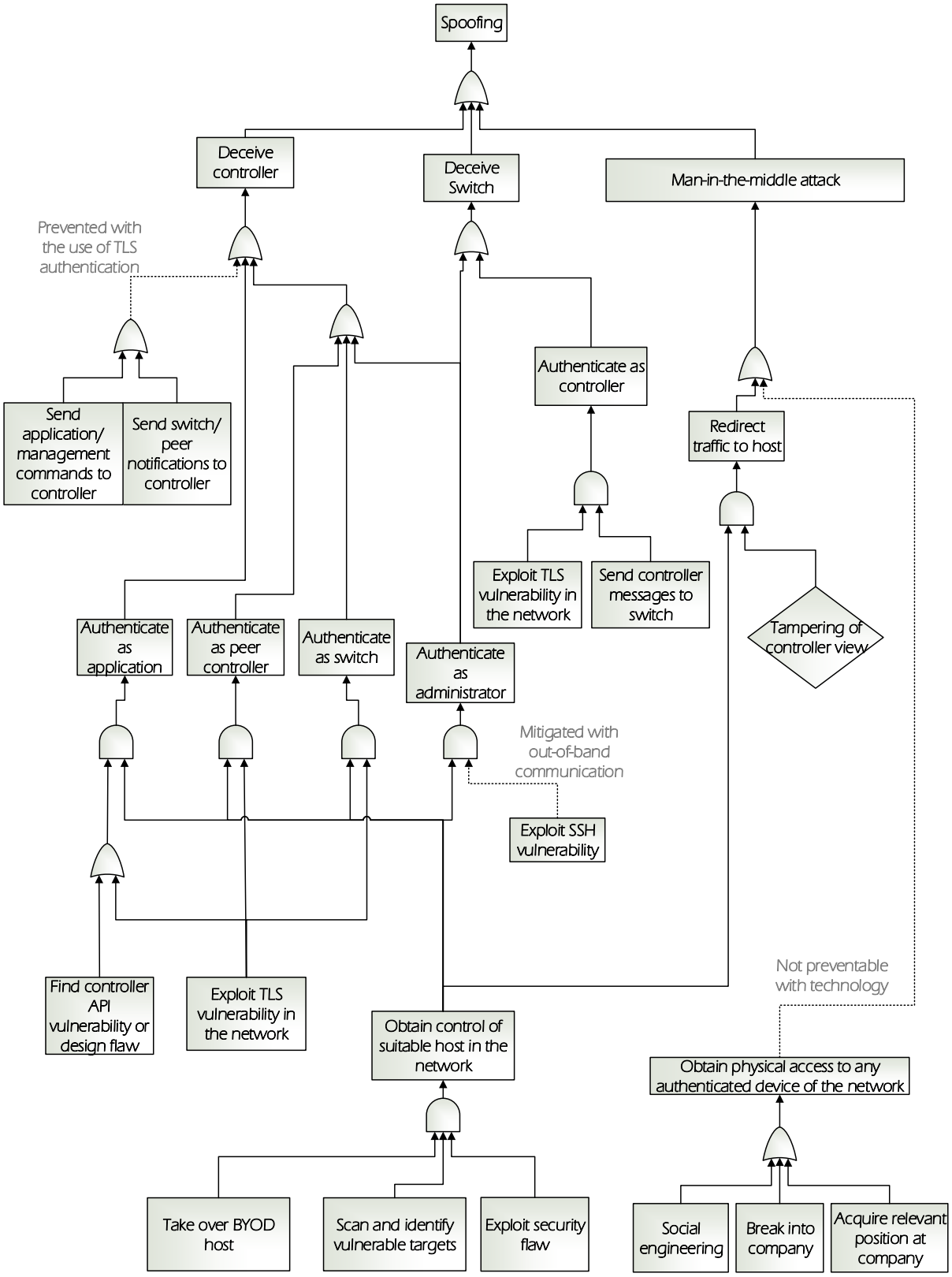

Figure 3.5: Attack Tree showing several methods to achieve Spoofing in SDN.

3.2.3 Tampering

Tampering grants an untrusted person a multitude of advantages. The successful modification of data can lead to the next step of an attack or the corruption of important information. A common example of the hazardous implications of the second STRIDE threat are bank transactions. Without knowledge of the involved parties, a man-in-the-middle can modify the specified transfer sum and redirect an arbitrary amount of money to his personal account.

SDN increases the amount of dependent systems and the centralisation of intelligence, which makes it particularly vulnerable to falsified information. Spoofing and Tampering are often intertwined and mutually dependent. The act of forging data flows and topology deceives other members of the network while improper authentication provides the ability to manipulate data stores and flows. Several of the following attacks require successful Spoofing or enable further Spoofing attacks. All data flows, stores, and processes of the system are in danger of carrying misleading information. On a network scale, an attacker could intercept any of the main data streams and manipulate asynchronous, controller-to-switch or management messages. An underestimated threat are man-in-middle attacks, as the network is often considered to be physically secure. [60] The interception is performed by physically interrupting the connection or spoofing a device of the network and redirecting traffic. A strategically placed switch or host could rewrite OpenFlow control and topology information. Any subsequent switches receive tampered controller-to-switch messages while the controller is wrongly informed about network state as asynchronous report messages are retained and modified. [60]

A switch contains several vulnerable assets. The resource database and the decision tables are of special interest for an attacker. It is unlikely that the internal data flows of the device are tampered with. If a person has access to the physical components of the switch and is capable of restructuring them to the point of successful, undetected Tampering, then his capacities exceed the normal threat considerations.

Forged control messages can rewrite flow, group, and meter tables. Manipulated decision tables reroute traffic to a black hole or to a different user, essentially generating Denial of Service (see Section 3.2.6) and Information Disclosure (see Section 3.2.5). A consideration is the remapping of virtual ports to a new physical port. However, any changes to port states are reported to the controller and most likely detected unless the controller has been hijacked. [42] If the management or control channel is compromised, port remapping might already be a superfluous action. The second data store is the resource database, which contains local topology of the switch. The switch forwards information about his neighbours to provide a virtualised overview of the network. If an attacker is able to connect a new adjacent device, the deceived switch might inform the controller about non-existent or malicious neighbours as legitimate network nodes. The controller erroneously takes the new entities into consideration and recomputes traffic over the malformed path. [60]

The last potential target is the southbound interface itself. Albeit a sophisticated approach, the attacker could install malicious or malfunctioning firmware on the switch and reform incoming management and control messages. Although this method is unlikely for hardware switches and out of the scope of the protocol itself, virtual switches could be severely affected. It might be a necessity to verify flow correctness and isolate malfunctioning switches.

Controllers are exposed to similar dangers as the OpenFlow switch. A major difference is the computational structure, as the controller generally runs on the operating system

of a server. Vulnerabilities of the OS consequently become the vulnerabilities of the controller. An infected server can manipulate the network view to hide a breach or overwrite the policy database schema, as control distributions do not seem to restrict writing access to the database, verify the integrity of stored data or secure sensitive information. Controller servers might openly expose a debug configuration interface which an attacker can utilise to stealthily reconfigure the device. [73] If the controller maintains a log in the resource database, it might be overwritten to prevent detection and non-repudiation. Similarly to the firmware manipulation of switches, malware could be inserted into the operating software or affect the controller applications. Successfully spoofing network elements and deceiving the controller is a prime motivation for Tampering, as authenticated management, applications or peer controllers are able to alter the policy while rogue switches and peer controllers can send falsified reports to distort the network view. [69]

A further problem is the lack of Byzantine fault tolerance. Byzantine faults may emerge, if disjunct and distributed systems need to reach agreement. If multiple logically centralised controllers are present in the network, they are required to communicate and vote on a consistent decision for the data plane. A malicious controller could potentially inject inconsistent decisions or modify the distributed database in order to alter the packet flow. [76]

In an argument for a controller security enforcement kernel, Porras et al., 2012 [77] discuss policy tampering performed by malicious and buggy applications. A demonstrative example is a method to circumvent a functioning firewall, which is possible due to dynamic nature of OpenFlow routing tables. A rogue or defect application commands controllers to install a forwarding rule which modifies the destination IP address of particular source addresses. The attacker then pretends to send packets to a legitimate address of the network, thus bypassing the firewall. Once the packet has moved past the wall, the misled switches change its destination address according to the malicious but legitimate flow entries and the attacker is able to target a protected host. The attack demonstration highlights the requirement of application control and policy consistency in order to minimise security backdoors.
(see Figure 3.6)

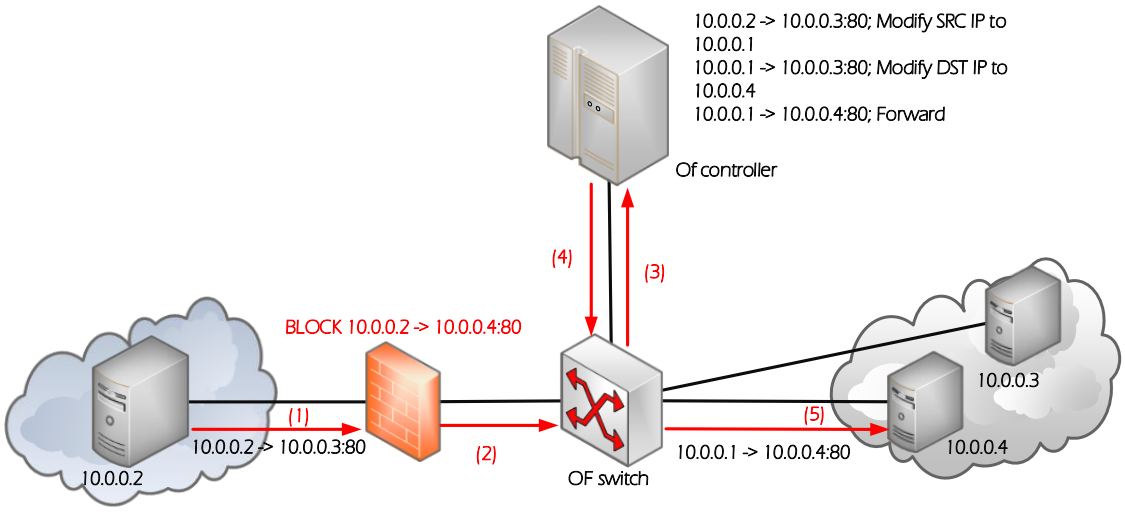

Figure 3.6: As demonstrated by Porras et al an attacker is able to bypass the firewall due to the inconsistencies in the policy of the controller (Figure inspired by [77]).

Applications and management are regarded as end interactors of the system. Their structure is not subject to detailed analysis, but it can be assumed that they are transitively affected by vulnerabilities of the controller, i.e. falsified network view and policies.

Solutions to many of the addressed issues are the use of TLS, reliable authentication, and proper review of potential open network services. [60], [69] However, Section 3.2.2 has demonstrated methods of an attacker to achieve access despite proper authentication measures. The OpenFlow design choice removes intelligence from the switches. They do not authenticate and verify their peers and thus relay the threat of an ARP and LLDP-spoofing attack or malicious connection vulnerability to the controller. A practical approach to implement this theory and to poison network visibility and topology is demonstrated by Hong et al., 2015 [78]. According to their research, the attack was successfully performed on several common controller implementations (Floodlight, OpenDaylight, NOX), despite proper TLS authentication of the control channel. The team presents *Host Location Hijacking* and *Link Fabrication* as two new attack methods in an OpenFlow network. After acquisition of compromised hosts, bogus LLDP packets were sent into the network to deceive the link discovery protocol of the controller. The mechanisms to identify genuine packets are circumvented by obtaining and mimicking packets or inspection of the open source code. The first attack tricks the controller into believing a network host has relocated. The second injects non-existent or detour paths into the network. Both attacks were performed from a generic host in the intranet. These methods largely replace ARP-poisoning in OpenFlow and are only possible due to weak topology management and packet verification on the controller side. [78] Furthermore, Klöti, 2014 [63] identifies three possible Tampering attacks on the switch. Two are based on the manipulation of flow aggregation and the redirection of traffic by either sending out the packet on another port or modifying the packet header fields. The third attack is a manipulation of the flow entry counters. He assesses that switches could send packets to cause a buffer-overflow in the counters field of the flow entry. However, Klöti deems this threat not feasibly in the scale of current networks. He concludes that Tampering has to be mitigated by implementing proper controller intelligence and the authentication of every single switch in the network. Antikainen, Aura and Särelä, 2014 assess that if a controller does not authenticate every single switch of the network, topology spoofing and traffic diversion remains an issue. [69] The attack tree of Figure 3.7 summarises the Tampering attack. Many possibilities emerge from successful Spoofing or a compromised server. However, a modification of the central network topology can be performed without need for authentication or the hijacking of a switch.

Comparison to Traditional Architecture

Tampering displays a similar threat pattern as Spoofing. The average access risk is not exacerbated if authentication measures are properly implemented and the network is physically secure. However, the application and control plane reveal several new entry points for an attack. The modification of central information has a significantly larger impact on the network. Routing intelligence is not distributed and switches are dependent on a single entity maintaining the view of the network. If this database is affected, the whole network is compromised. A controller has to correctly identify corrupt and conflicting information in the same fashion as it has to notice and detect Spoofing attempts. Many practitioners agree that the responsibility of security and consistency shifts to the control platform, which has to verify switch topology and application reports. [79], [80]

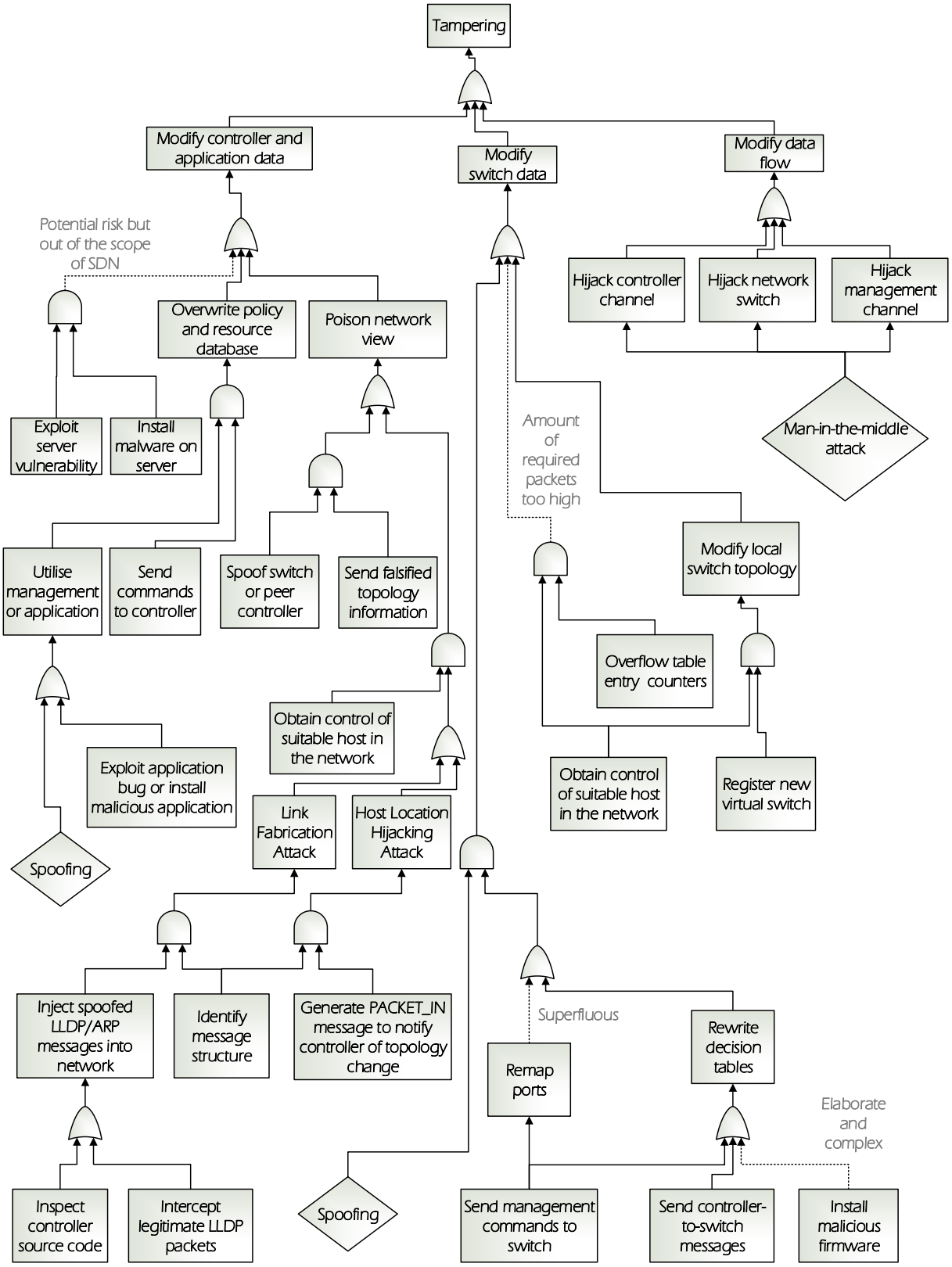

Figure 3.7: Attack Tree showing several methods to achieve Tampering in SDN.

3.2.4 Repudiation

It is desirable for administrators to trace back any actions or events in the system to a single entity for either legal or debugging purposes. Simultaneously, users prefer to be reaffirmed of authentic communication and the arrival of their messages at the expected destination. Non-Repudiation assures accountability and transparency of interactors in the system. The concept itself overlaps with authenticity and resilience to Tampering to a great extent, but is often an ancillary requirement to promise users a trustworthy system. From a technical perspective it is not considered feasible to absolutely prevent Repudiation, as computing systems might be compromised or authentication tokens illegally acquired. [81] Nevertheless, measures to guarantee a degree of liability for system components and users exist. This thesis defines non-repudiation as the capability to accurately trace back any kind action to a single device or host of the network and as a result identify and incriminate malicious or compromised devices. Legal considerations or host based signatures and encryption are not inspected, as these are out of the scope of an OpenFlow network. OpenFlow supports IPSec and host authentication is in theory available to secure the end-to-end user communication. In the network the control plane is notably vulnerable to Repudiation. The data plane lacks intelligence, is not autonomous and thus does not perform any deniable actions.

Nonetheless, data plane devices should report all modifications and internal data changes to higher logical planes. The OpenFlow southbound protocol [42] does indeed specify automated messages and reports all changes of the flow tables, port states or controllers in a switch. This is expected, as forwarding knowledge to the control plane is a fundamental part of the OpenFlow paradigm. To the authors knowledge however, asynchronous messages do not detail who has performed or commanded a particular modification. This poses a problem with multiple controllers or management connections. Even if all the connections are authenticated it does not seem possible to pin the action down to a single actor.

Controllers possess knowledge of the entire network and its hosts and switches in the topology and virtualisation database. The central intelligence provides great potential to remedy the Repudiation threat and monitor the behaviour of network interactors. [82] Still, native OpenFlow does not deploy sufficient forensic mechanics [82] and controllers face several issues regarding identification of accountability. Controller implementations maintain a log for debugging purposes, but do not seem to discern application input from the northbound interface. [83] Porras et al., 2015 [56] specifically inspected the Floodlight architecture and deem the lack of application accountability problematic. The controller does not distinguish policy operations of multiple applications or identifies conflicts. If applications are designed as monolithic block and commands are forwarded to the controller uniformly, internal conflicts and misbehaving applications are not detected. Overall, the southbound as well as the northbound interface do not provide the ability to track actions of higher level devices.

A second security consideration is tampering of log files maintained in the resource database. An infected controller or server might simply overwrite suspicious log entries, if data is not continuously sent to the OSS or SIEM of the network. It is also possible to configure switch to rewrite certain MAC and IP source addresses to cloud the exact origin of malware or Distributed Denial of Service (DDoS) attempts. Network monitoring tools or basic intrusion detection mechanisms might simply be blocked from certain data paths and thus do not report suspicious activity. If controller modifications are not reported instantly to central management regardless of configuration, covert activity could stay undetected for extended periods of time.

Comparison to Traditional Architecture

The Repudiation threat in SDN does not differ significantly from legacy networks as the major cryptographic protocols TLS and IPSec are theoretically supported as countermeasures. In fact, the centralised overview amplifies the potential to trace covert communication activities and rogue devices. [82], [84] Repudiation issues in OpenFlow largely stem from Tampering or general implementation negligence, since the authentication measures are supported but not advocated. Nevertheless, introducing unique IDs for controllers and applications and informing other network members of the actions of peer devices are compulsory considerations. The activities of malfunctioning controllers and applications have to be meticulously documented and monitored to provide a minimal capability to correctly track down or find suspicious behaviour. Figure 3.8 demonstrates the overall possibilities to deny actions or hide activity.

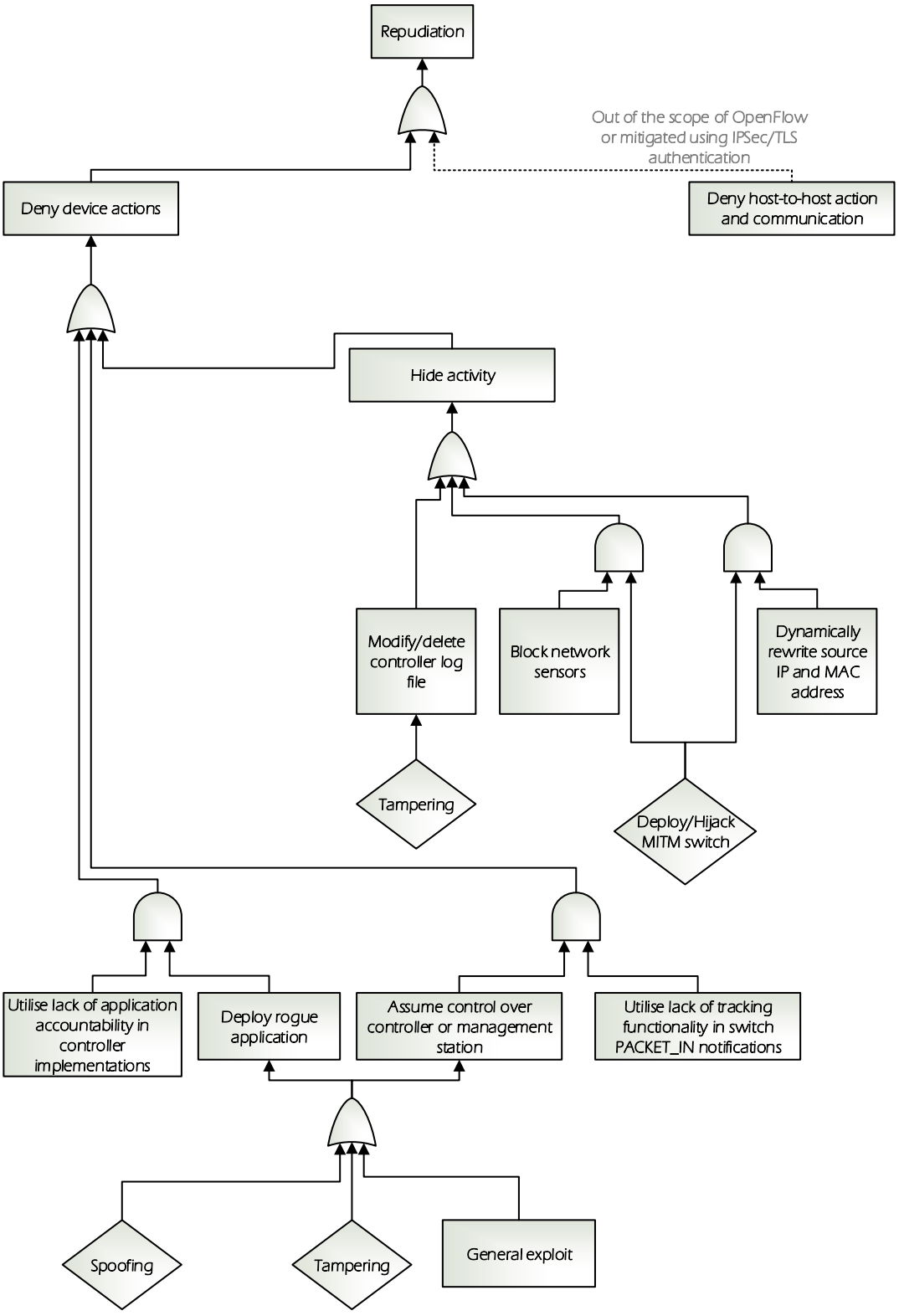

Figure 3.8: Attack Tree showing several methods to achieve Repudiation in SDN.

3.2.5 Information Disclosure

Two distinct types of Information Disclosure are applicable to software-defined networks. On the one hand, user traffic can be monitored, recorded, and crucial data subsequently acquired, on the other hand, reconnaissance and the identification of potential targets are preparation steps for a larger attack, most commonly DoS. Concerning data interception, OpenFlow suffers from the same vulnerabilities as traditional networks. An attacker can sniff passwords and confidential data, if a man-in-the-middle device has been installed and the traffic is not encrypted. Network administrators attenuate the problem by implementing authentication and encryption protocols. OpenFlow supports TLS and IPSec and thus employs a comparable amount of security to state-of-the-art networks. Further concerns regarding authenticity an the prevention of Spoofing have been addressed in Section 3.2.2.

A potential new problem is the extraction of information on the controller host, as controller distributions offer multiple XML-based services which are potentially vulnerable to fault parsing attacks. An example is the NETCONF management service deployed on controller architectures such as OpenDaylight [47] or RYU [52] and the OF-Config protocol [43] in general. Security exploits in OpenDaylight, which provide access to sensitive files or credentials, are already existent and documented. [85] However, NETCONF [17] and similar configuration access methods generally demand a SSH or TLS connection, which provides a sufficient protection, if and only if default passwords are changed.

The OpenFlow protocol itself exposes several new reconnaissance possibilities in the network.

By default switches and controllers listen on the TCP port 6653 for new connections [42] and might be accessible from external sources in order to provide home office functionality or remote site access. The openly accessible port presents a new possibility for attackers to scan for OpenFlow networks, as manual scripts or a port scan quickly identify the active OpenFlow services. Even if port scans are unavailable due to security measures it is still possible to fingerprint for OpenFlow network by measuring different response time in packets. The high fluctuation depending on the selected header fields suggests the existence of a control channel. [86] After the attacker has found suitable a target, he can distinguish switches and controllers by inspecting handshake messages. A *OFPT_FEATURES_REQUEST* answer to an OpenFlow HELLO message denotes communication with a controller. [42] These attacks can be prevented if border switches drop any packets targeting the OpenFlow ports. However, it might be necessary to enable open exterior access to controllers to perform remote operations. Isolating the network with VLANs is a possible measure, but does not guarantee full security. Certificates are therefore a requirement to ensure a minimal degree of security in respect to outside connections.

Once an attacker has compromised a host that is connected to an authenticated switch in the network, several methods to gain information about the devices and the network topology present themselves. Switches expose network data in their RDB and forwarding decision tables, controllers provide information about the entire network topology, access control lists, and general policy. An attacker then obtains these resources by abusing faulty authentication and querying the devices. A basic management, application or controller request prompts an insightful information response. Furthermore, flow table instructions simplify the mirroring of traffic to specific hosts as one single, remotely installed entry suffices as configuration. [61]

Even if switches and controllers are mutually authenticated and Spoofing is prevented,

network data can be collected using a *side-channel timing attack*. To determine which packet headers generate new flow rules, the attacker sends packets with different modified header fields and awaits a response. A statistically significant difference in round-trip time ascertains, which packets are sent to the controller by the switch and create a new flow entry in the table. Klöti, 2014 [63] as well Shin and Gu, 2013 [86] succeeded in implementing an experimental test scenario of this reconnaissance method. Both projects measured and compared response times of sample packets and were able to determine whether a flow rule existed in a particular switch. Shin and Gu additionally presented a SDN scanner tool, capable of identifying OpenFlow networks without using port scans and assessing flow rule conditions. An important step in the preparation phase for a DoS attack on switches or controllers.

Comparison to Traditional Architecture

New network components introduce new possibilities to collect data. The agile and programmable nature of an OpenFlow network facilitates Information Disclosure as single devices can be quickly reconfigured to direct traffic over detour sniffing paths. The variance in response time helps attackers to map parts of the network without needing to access any device. In SDN multiple elements maintain information about the entire network in flow tables and virtualisation databases. This information may be revealed using remote queries or by gaining access to a server. While user data is protected with TLS and IPSec, basic SDN does not provide adequate methods to hide information about the overall network structure. The attack tree in Figure 3.9 highlights the three basic aspects of Information Disclosure in SDN and methods to retrieve information from the data flows or devices.

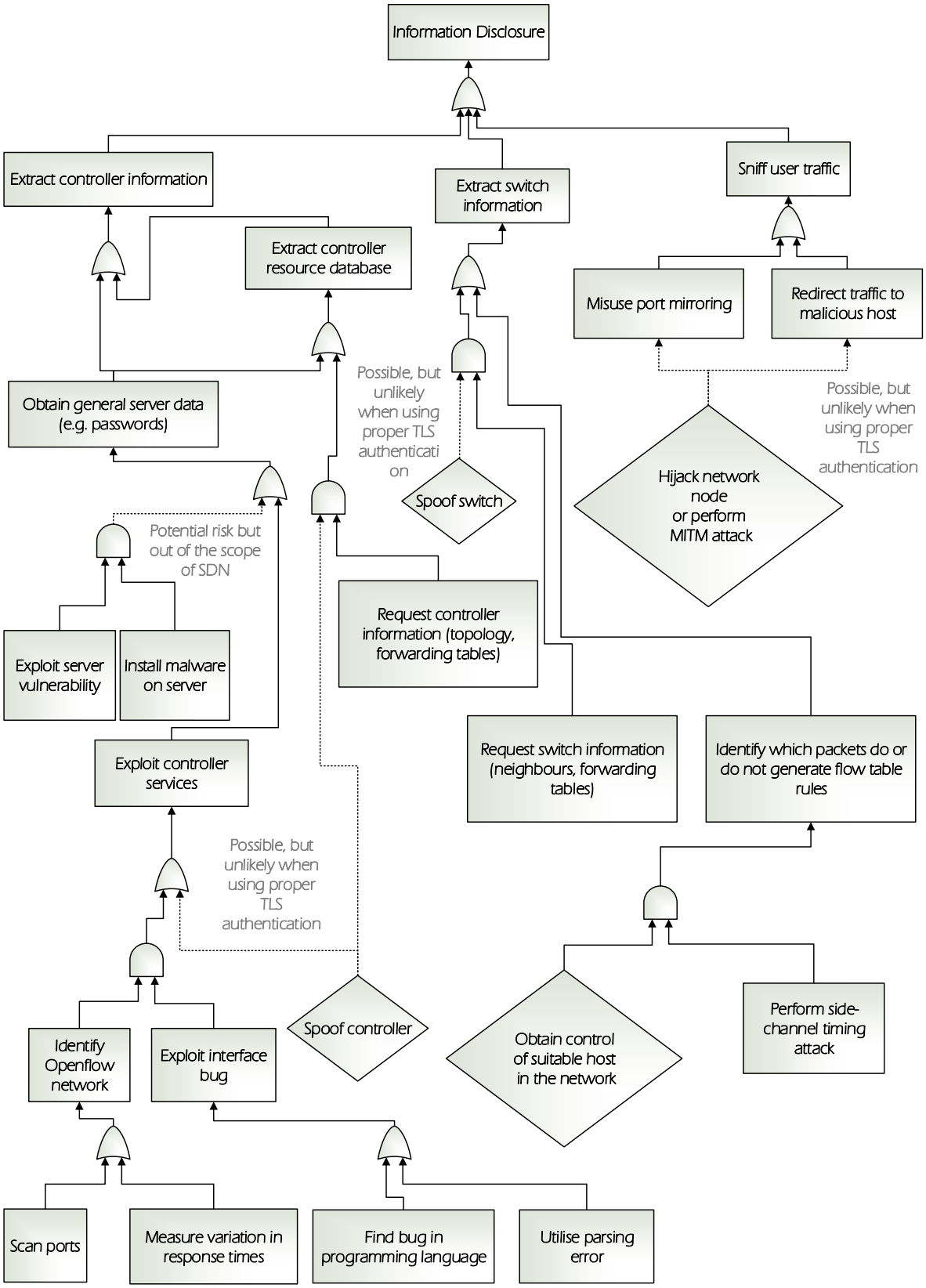

Figure 3.9: Attack Tree showing several methods to achieve Information Disclosure in SDN.

3.2.6 Denial of Service

The centralisation of network elements suffers from the constant danger of creating a single-point-of-failure. Software-Defined-Networking proposes the central controller, an opportunity as well as an Achilles' heel in the network. The ONF itself acknowledges that the controller "emerges as a potential single point of attack and failure that must be protected from threats" [87]. However, the controller is not the only blind spot in SDN. Denial of Service can be considered the most prevalent threat in SDN. All of the STRIDE aspects play a part in the success of DoS, as the attack surface is broad and a variety of targets is available.

The first component to expose several weaknesses are switches. Using conventional DoS methods an attacker might simply flood the ingress ports and interfaces of the switch. It is unclear if the flow tables of OpenFlow switches produce more processing load than routing tables in traditional devices. Flow table matching, table lookup, and application of actions generate a significant computing overhead, which could lead to a slower processing of larger amounts of data. Case studies reveal that current switch CPUs may be a limiting factor in the reactive approach, as they are not able to handle a high installation frequency of flow table entries while simultaneously sending PACKET_IN messages. [61], [88]

If the CPU is sufficiently fast, ingress and egress interfaces and their buffers are the limiting hardware factor. It can be assumed that every interface and physical port utilises a dedicated buffer and shares additional dynamic memory with its neighbours. If one ingress interface is flooded, the remaining fabric (including the management access) is still operational. The output buffer is most likely not affected by flooding as traffic has to pass ingress ports and the CPU. Over-subscription of switch ports leads to the possibility that traffic from multiple interfaces is traverses one particular egress port and renders the outlet unusable for any remaining flows. However, determining and acquiring this informations requires intimate knowledge about the network and is generally infeasible. These considerations do not differ significantly from traditional network switches. However, the change from routing to flow tables and the new controller-to-switch communication channel introduce new options. An attacker can send any kind of packet to the connected switch, assuming he has gained access to one or multiple hosts in the network. Under the condition that the controller operates in reactive mode and allows PACKET_IN messages, the malicious host is able to send diligently crafted packets to the switch without being authenticated.

Information Disclosure (see Section 3.2.5) provides the opportunity to determine which packets actually generate flow rules. The flow table is then flooded with malformed table entries. Kuznier et al., 2015 [89] have determined that large flow entries severely impact switching performance in some vendor devices. If the flow table is constantly filled within the expire date, the switch is effectively incapacitated. OF-Switch [42] specifies that, instead of replacing old entries, the switch replies with an ERROR message to the controller. Automatic flow entry eviction is a possibility, but only optional and not a required function. As TCAM generally utilises low space for the sake of performance, it is achievable reach the size limit in short time. Switches are assumed to support around 8K to 32K flow table entries [12], in reality the typical maximum seems to average around 1500 elements [88], which is not sufficient for larger networks. Some switches allow as little as 250 CAM entries. [61] False instructions are continuously forced into the flow tables and hinder the installation of legitimate entries, thus achieving a Denial of Service for particular network flows. To address the general problem of limited entry size, switch vendors are already increasing the

minimum available size. Furthermore, flows may be aggregated to save space. However, these approaches may be futile, if the attacker is aware of the flow entry configuration.

The controller-to-switch channel itself is vulnerable to various DDoS attacks. Zombie hosts generate unknown packets with bogus header fields which switches then forward to the control unit for advice. The sheer amount of possible header field combinations allows for effective burdening of the channel. While the computing power of the server requires large amounts of traffic and these packets might eventually be dropped by the controller, it is a possibility to slow the processing power significantly enough to cause an overload in control operations. In addition authentication is not required for this attack and the implementation of a computing-heavy TLS encryption could amplify the effect.

Resource attacks have been studied and demonstrated extensively in SDN security research. [60], [61], [63], [86], [90] Controllers which have been thoroughly examined are POX, Floodlight, ONOS, and OpenDaylight, each of them being vulnerable to abuse. Many of the proof of concept demonstrations have been performed in the virtual network environment Mininet [91], which is freely available for testing and research purposes. Although not a real production network, it is able to represent and simulate the architecture and behaviour of real devices and can be used to highlight general design flaws. Klöti, 2014 [63] tested an attack on a POX controller in Mininet. The malicious host minimally altered the header fields and successfully filled the flow tables of the target switch. The controller did not attempt to remove any superfluous entries, although it received error replies and performance degraded substantially. Inspecting the log file of the default Mininet controller, a new and previously unconsidered vulnerability was found. Old log entries are not purged and the file grows indefinitely. The behaviour of a controller without any available storage space is undefined and could lead to potential issues, most likely device failure. A thorough examination of writing functionalities to avoid unexpected system failure is therefore necessary in a code review. A similar problem has been identified by Romaõ et al., 2013 [61] after examining the popular virtual switch software Open vSwitch. The switch does not implement a hard limit for flow table entries. As soon as the program exceeded the reserved space in the machine the switch process was terminated to avoid further waste of memory. The network chunk connected to the switch is effectively taken out of service. Tri and Kim, 2015 [90] measured the impact of a resource consumption attack on flow tables and OpenDaylight in detail. The results displayed an expected heavy drop in performance as exhausted switches forwarded packets through the controller. The controller also did not address full tables and continued to its attempt to install new entries.

An alternative to these resource consumption attacks is cutting off the controller-to-switch communication channel. After losing connection, the switch proceeds to revert to *fail secure mode* or *fail standalone mode*, heavily limited in its switching capability. [42] The network may still be operational, but switches in fail secure mode are essentially frozen in forwarding behaviour. Hybrid switches in fail standalone mode have to rely on conventional routing protocols, which can be considered a violation of the SDN paradigm. Other than the injection of TCP FIN packets, it is possible to interrupt the communication by spoofing the target switch and exchanging OFTP_HELLO messages. The controller perceives a change in topology and cancels the connection with the original switch. After the real, benign switch attempts to reconnect and the false connection is cut, the malicious script registers a new device and thus continuously interrupts the connection. [92] Shalimov et al., 2013 [93] studied various controller implementations and found that malformed messages can cause interruptions or program failure. OpenFlow headers in particular are responsible for discon-

nects or malfunction in architectures such as NOX, POX, Floodlight, and OpenDaylight's predecessor Beacon. Malformed packets are not dropped and processed by the controllers, thus exposing a potential vulnerability. RYU, however, passed the tests. A similar bug has been published by the ONOS project. The ONOS controller [50] throws an exception and drops the connection to the switch after inspecting invalid or truncated switch packets. [94] The frequent occurrence of fault-prone packet processing in various controller designs exposes a common potential vulnerability in the control plane, which could lead to simple and harmful attacks and is not sufficiently address in either the OpenFlow protocol or the major controller distributions.

Lastly, a research report of Dover Networks [95] identifies an exploit in the Floodlight controller, which does not purge old switch records. The attacker registers a new switch using the conventional OpenFlow handshake. After a successful connection an unsolicited OF_FEATURE_REPLY with a modified switch ID is sent. The controller replies with an error message but stores the false ID. The connection is cancelled and the process repeated until the switch table and network topology database is sufficiently filled. The lack of memory renders the controller unable to maintain the switch tables. The device ceases being operational and drops active connections, resulting in a dysfunctional network.

The aforementioned attacks are deemed to be unavailable if TLS is active. It depends on the implementation and configuration whether an infected host can send packets with a malformed OpenFlow header to controllers over an authenticated switch. Assuming the switch does not drop the bad packet and relays it to the control channel over a PACKET_IN message, the controller could be broken despite the proper use of authentication. Another possibility is malicious payload to compromise the controller. The OpenFlow protocol implements a switch buffer which truncates packet content when transmitting an unknown packet to the control plane. However, the buffering of packets in the data plane is an optional feature in the OpenFlow protocol and not supported by all switch vendors. Deep-packet-inspection (e.g. for QoS or security purposes) could expose the controller to malware or risk software failure.

While switches and the channels are a viable access method, the hourglass design of SDN and the controller bottleneck grant the most potential for DoS attacks. The erratic behaviour of controllers demonstrate that the software is a new liability in the network. An attacker can utilise device failure in response to innocuous actions or packets without having to resort to more sophisticated methods. If the source code of controllers is not properly reviewed, the central intelligence is in constant danger of accidental or intentional failure. Assuming the control channel is not separated and out-of-band, an attacker can use several conventional methods to overload the controller. The resource database of the controller server can be exhausted with TYP SYN floods, preventing legitimate management or application access. Depending on the underlying hardware, a simple flooding attack on the control channel causes congestion on the network interface card and port buffers, effectively disrupting any type of traffic. From a northbound perspective, applications which reside on the controller server and thus in the same trust boundary pose additional danger. Shin et al., 2014 [79] demonstrated that crashing applications also crash the controller. To signify a general controller design problem, they tested their theory on three different controllers. Floodlight, NOX, and Beacon all terminated due to application errors. Furthermore, if not restricted, malicious or defective applications might consume valuable system resources of the controller, as they operate in the same process or memory space as the control software. [79]

Spoofing (see Section 3.2.2) and Tampering (see Section 3.2.3) provide the unique ability

to cut out large slices of the network, redirect traffic or shut down any functionality. Switches can immediately be reconfigured to drop user or administrator traffic. It is even probable to create a flexible botnet in a software-defined network. Abusing the controller and flow tables, traffic can instantly be redirected to target an external entity or network.

Even without application or management access the virtualised view of the deceived controller might be tampered. Discovery protocols and host tracking provide a false view of the network and mislead the controller into redirecting traffic into null routes. An example has been demonstrated by Hong et al., 2015 [78], who utilised the *Link Fabrication Attack* to trick the spanning tree protocol of controllers into blocking incorrect ports. Users which were connected to these ports were subsequently denied entry to the network.

Overall, TLS does not suffice in preventing DoS attacks. To limit, control, and direct traffic, OpenFlow implements forwarding and meter tables. These mechanisms are essential addressing traffic based Denial of Service. An effective prevention requires a functional control plane and channel. While OpenFlow implements auxiliary connections to support the channel, they are potentially unreliable as they depend on the main link. If the main connection is interrupted (e.g. via a TCP FIN packet), all auxiliary connections are dropped. Distributed controllers are a necessary measure to harden a network, but introduce complexity and increase the risk of misconfiguration and asynchronism. OpenFlow switches support multiple controllers and fallback mechanisms to ensure reliability. However, automated DoS response by multiple controllers is not scope of the OpenFlow protocol or addressed in distributed implementations. The responsibility to detect and address anomalous behaviour is delegated to the applications and is not part of the controller or data plane.

Comparison to Traditional Architecture

SDN magnifies the risk of Denial of Service in the network to a great extent. Network elements drop independence for the sake of agility and ease of configuration, but if the central intelligence fails, the network breaks down. The programmability and software-centric approach introduces new attack vectors and error potential, leading to failures or outages. Manipulation of the network map results in intentional configuration errors and traffic black holes. Furthermore, the multitude of possibilities to damage the system, which is also demonstrated by the variety of paths in Figure 3.10, broadens the attack surface. Nevertheless, SDN also provides several opportunities to dynamically mitigate DoS attacks. Applications can isolate zombie hosts, if they are identified in due time. Traffic can quickly be rerouted to avoid congestions and switch meter bands limit incoming data rate, resulting in dynamic and agile protection of sensitive network areas. Constant network monitoring facilitates fast identification of anomalous behaviour. These possibilities, however, are based on the assumption that the controller is operational or utilises the necessary protective tools. OpenFlow does not include these capabilities by default. Intelligent applications and distributed controllers have to be deployed in SDN in order to guarantee reliable attack defence.

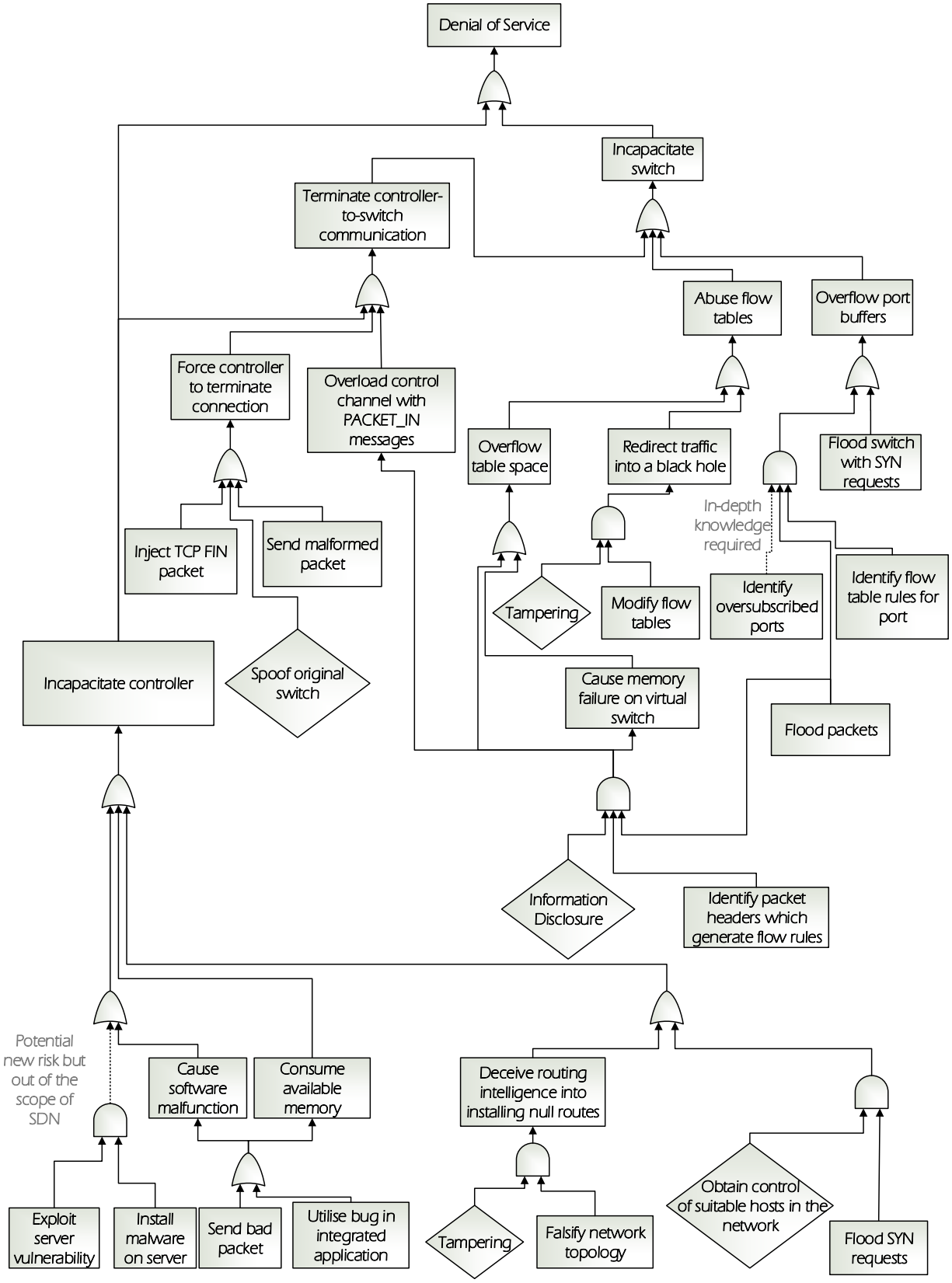

Figure 3.10: Attack Tree showing several methods to achieve Denial of Service in SDN.

3.2.7 Elevation of Privilege

The least privilege imperative ensures that users, processes, and interactors of a system do not perform any more operations than they are permitted to do. If a process of a system gains higher authorisation, i.e. elevates his privilege, the whole system is compromised. The threat affects the OpenFlow network in two distinct ways (see Figure 3.11). The basic path is the intrusion into discrete trust zones. If an attacker manages to bypass the basic controller and switch authentication mechanisms, he has obtained a higher level of authorisation in the network. The methods to gain trust from the individual OpenFlow devices are covered in Section 3.2.2 and 3.2.3 respectively.

New threat considerations emerge with the adoption of service concepts such as Network as a Service (NaaS), which postulate virtualisation of network architecture and sharing of physical resources. [96] If SDN grows and eventually achieves carrier grade deployability, controller and network resources are sold by service provides to external customers. Additionally, companies may provide services of their network to untrusted remote affiliates or subsidiaries over a extranet. Each network user requires a different set of visibility and traffic quality to control their service over a disjunct or shared section. One argument for SDN is the simplification of network division and allocation over control and management platforms. However, the sharing of resources between various users introduces new threats and problems. A common suggestion of network separation in SDN is the FlowVisor [55] application, which acts as a proxy between control and data plane and assigns controllers to strictly separate network sectors. Every controller has only knowledge of its designated network and is not able to insert flows into external switches. In theory, multiple controllers are connected to FlowVisor, each managing a different virtual section of potentially overlapping devices. FlowVisor rewrites asynchronous and controller-to-switch messages according to the defined network topology and forwards the commands or reports to the relevant devices. Although FlowVisor is a single implementation of network separation and is generally used for research purposes, it represents a general design principle and its vulnerabilities. Assuming one attacker has taken over one of the controllers, it might be possible to deceive FlowVisor in the same fashion as switches are spoofed in OpenFlow. A spoofed controller can assume control over the network of the original device and insert any flow entry into the switches. FlowVisor is theoretically hidden from controller view in the network, but the proxy might introduce a measurable delay and FlowVisor activity could be identified by measuring differing response times. [63] An other notional vulnerability in FlowVisor is the proper verification of actions. It is possible to insert rules into switches, which rewrite the VLAN ID and other fields of the packet and manipulate flows to inject traffic into other isolated networks. [97] Furthermore, Yu, Quian, and Quian, 2014 [98] have identified problematic behaviour regarding shared flow table spaces in a switch. It is possible to override flow tables entries of other controllers by inserting more specific, higher priority rules into the flow table, thus taking control over the shared switch. These vulnerabilities have to be addressed before it is even possible to deploy SDN in multi-user networks. A second issue of supplying the controller as a service is the application and agent access control in the northbound interface of shared controllers. [56] Customers and subordinate network operators deploy personal applications on provisioned controllers to satisfy their network needs. However, controller implementations do not seem to differentiate application priority and authority. [56], [77], [83] Applications are neither restricted in their configuration scope, nor are they limited in access based on authority.

A client application can override administrative decisions and entries, assuming the controller has access to the respective network slice.

Comparison to Traditional Architecture

The maturity of SDN poses a problem in identifying potential risks in shared service networks. Dominant enterprise applications or deployments have not yet emerged to evaluate concrete design decisions. While Google did install a large-scale SDN datacenter [19], they avoided the problem of conflicting applications using single application blocks and internal conflict resolution. [56] Additionally, no actual mechanisms to share controller resources between different network users or entities are available yet. FlowVisor is a popular solution to divide the network into various security or control domains, but already exposes a multitude of design flaws. Overall, authorisation and proper permission distribution is a crucial cornerstone in the deployment of large scale software-defined networks, which should be considered in the development.

3.3 Threat Summary

SDN exposes several security flaws, which might hinder adoption or cause reservations in network operators. The fact that a native TLS implementation is not a priority and optional is a sizeable issue. Allowing controller developers as well as switch vendors to neglect authentication measures is a dangerous decision in a central and dynamic design such as SDN. The thesis exposed and evaluated a variety of attacks, which are possible if any trust-boundary is not appropriately authenticated. An attacker can take over the network with ease by simply gaining access to a host of the network. Even if TLS is implemented and all switches and controllers are mutually authenticated, several issues are apparent. Data integrity gains significance, as a single device maintains the network view and state as opposed to the distributed knowledge in the traditional network. Malformed and false information impacts the network more than previously considered. Similarly, new Information Disclosure and network mapping possibilities are enabled by probing switches or simply querying information from controllers. Traceability and unique identification of network elements such as controllers and applications are optional and impede the swift identification of malicious activity. There are no measures established to implement distinct trust levels and access control in switches or controllers. Most notably and significantly grown, however, is the risk of DoS attacks. The resilience of the network depends on a single device, which has to be appropriately secured. The controller is dependent on information from applications and switches while switches are entirely dependent on the controller. These dependencies introduce a myriad of new attack vectors. Distributed controllers and multiple connections can alleviate the risk but are optional and insufficiently specified. A dominated distributed controller design has not emerged yet or is properly defined in standards or by SDOs.

In summary, the OpenFlow protocol and basic general design of SDN can not be considered secure by default, particularly due to the lack of mandatory authentication and the the fragility of the control plane. If network administrators deployed a generic software-defined network as established by current standard literature, they would expose their network to a multitude of new threats and dangers. A risk which is not acceptable in modern, large-scale company networks. It is therefore necessary to develop a standardised secure design for SDN operators and to mandate basic security implementations and applications.

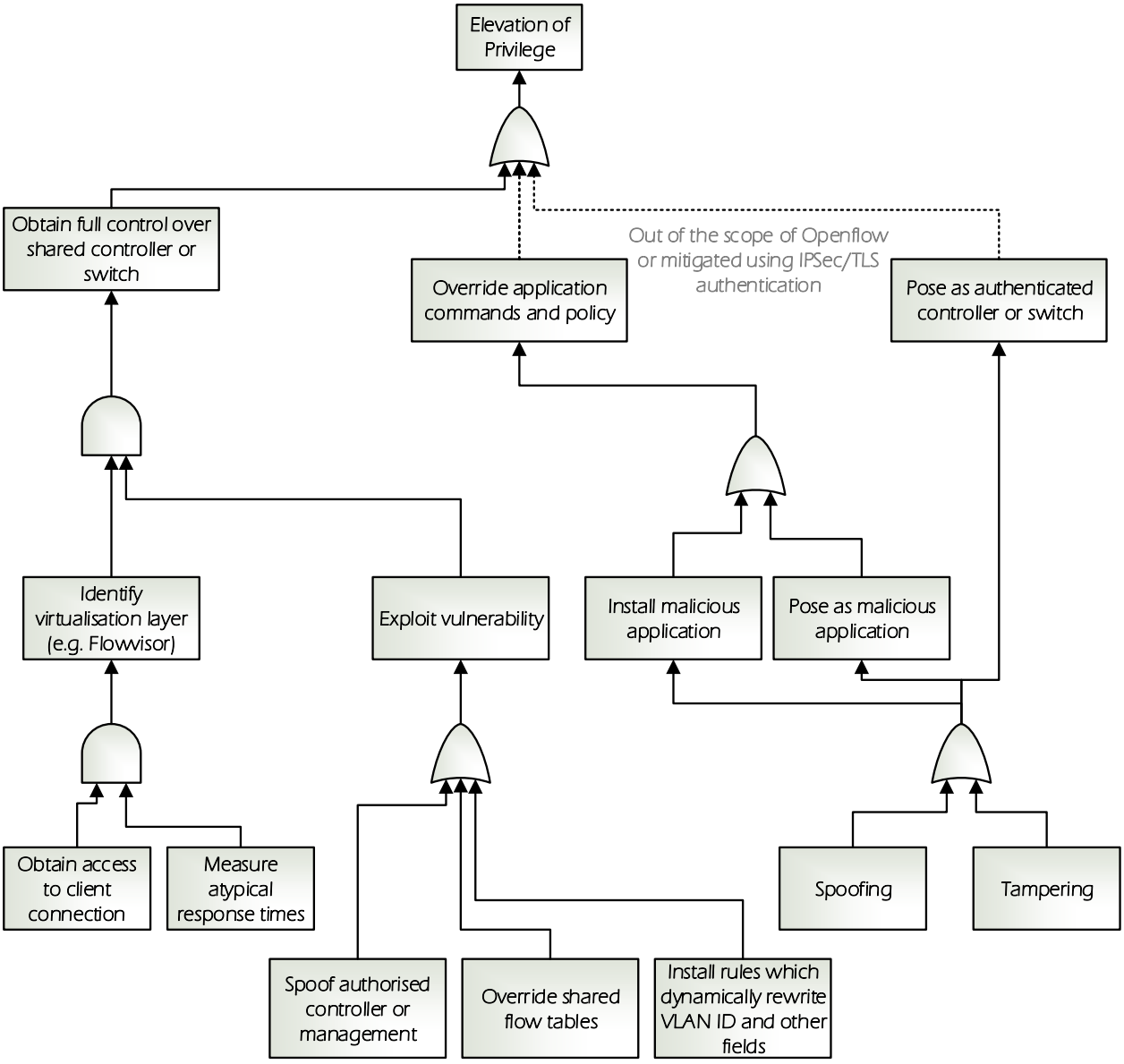

Figure 3.11: Attack Tree showing several methods to achieve Elevation of Privilege in SDN.

4 Threat Mitigation and Security Opportunities

The breadth of security problems in SDN raises concerns and poses the question whether any countermeasures are available at present. As evidenced in the preceding chapter, it is mandatory to extend the generic OpenFlow network to guarantee sufficient reliability. In the wake of the rising popularity of SDN and a recent shift of focus to security, researchers have started to propose a multitude of single solutions to secure and even enhance sections of the software-defined network. Nevertheless, a ubiquitous and dynamic security architecture design for SDN is an eventual necessity to satisfy industry needs and gain acceptance. Two noteworthy abstract architecture proposals and requirement definitions have been published so far. Kreutz et al., 2013 [8] developed a threat vector concept and argue for a "secure and dependable" general SDN design to cover the ascertained threat vectors. As of April 2015, during the writing of this thesis, an ONF committee has audited the *OpenFlow Switch Specification Ver. 1.3.4.* and published their own secure design and requirements, which are likely to be incorporated into the future standard. [99] In context of the set of problems identified in Chapter 3, the thesis examines the two approaches and complements the concepts with individual and suitable solutions. The feasibility and robustness are inspected and the remaining open STRIDE threats are highlighted. Additionally, a selection of noteworthy concepts, which do not match any of the previous aspects, showcases security opportunities. For any of the principles new vulnerabilities or insufficient requirements are demonstrated. It should be noted that potential implications on overall network performance and latency are not taken into consideration, as the scope of the thesis does not exceed the security and reliability aspect. The research and design solutions investigated in this chapter build the foundation of the secure SDN blueprint of Chapter 5.

4.1 A Secure and Dependable Data Plane

In response to criticism and confusion concerning the OpenFlow protocol, the ONF has chartered the ONF Security Project. The mission statement is to evaluate the current protocol, to define security requirements, and to amend the OpenFlow standard with the new definitions. In April 2015, the committee has published a first risk assessment of the *OpenFlow Switch Specification Ver. 1.3.4.* The researchers identify concrete weaknesses and failures in the protocol and attempt to solve the issues with specific solutions and requirements. The ONF intended the publicised security analysis to be complementary to a conventional threat analysis. Indeed, the identified STRIDE threats of Chapter 3 and the concrete discussion of the ONF recommendation share criticism and identify common security faults. Thus, the specified requirements are inspected and the solutions are mapped against the constructed STRIDE attack tree paths in order to assess the extent of the solution. Table 4.1 summarises the proposed requirements for the OpenFlow protocol v1.3.4. Each of the eight security principles is discussed and compared to suitable STRIDE threats.

Clearly Define Security Dependencies and Trust Boundaries	REQ 4.1.1	Mutual authentication between SDN components must be supported.
	REQ 4.1.1	The protocol must provide access control solution to limit the available set of operations.
	REQ 4.1.2	During connection negotiation the components should agree on a minimum of security associations.
	REQ 4.1.3	The security of the protocol should provide and define integrity protection of exchanged packets.
Assure Robust Identity	REQ 4.2.1	Any SDN entitiy must have a verifiable unique ID.
	REQ 4.2.2	Generation, distribution, and revocation of IDs should be considered in the OpenFlow protocol.
Build Security Based on Open Standards	REQ 4.3.1	Existing security solutions must be applied first.
	REQ 4.3.2	The protocol should explicitly advice against deprecated algorithms and standards.
	REQ 4.3.3	Handling of malformed packets by SDN devices has to be specified.
Protect the Security Information Triad	REQ 4.4.1	Security solutions should span over multiple OSI layers.
	REQ 4.4.2	The protocol should provide mechanisms to limit potential DoS threats.
	REQ 4.4.3	The OpenFlow protocol must be freely extensible in order to be able to address future attack types.
	REQ 4.4.4	All protocol messages should be equipped with integrity check data.
	REQ 4.4.5	Any possibilities for amplification attacks in the protocol should be considered and patched.
	REQ 4.4.6	In general, any new security design should be inspected for the potential introduction of new issues.
Protect Operational Reference Data	REQ 4.5.1	All policies, keys or certificates should be stored securely to preserve the integrity and confidentiality of the data.
Make Systems Secure by Default	REQ 4.6.1	Default security behaviour should be specified for all major configuration and deployment scenarios.
	REQ 4.6.2	Mandatory cryptographic algorithms must be specified.
Provide Accountability and Traceability	REQ 4.7.1	All critical events should be logged and reported to affected entities.
	REQ 4.7.2	All logging information must be protected from Tampering.
	REQ 4.7.3	OpenFlow counters and device status are to be continuously monitored in order to detect suspicious activity.
Properties of Manageable Security Controls	REQ 4.8.1	The security mechanisms should support multiple security algorithms in order to satisfy personal preferences.
	REQ 4.8.2	All protocol mechanisms should be extensible to adapt to new solutions or challenges.
	REQ 4.8.3	Key and credential distribution in SDN devices should be automated to simplify security management.

Table 4.1: The OpenFlow requirements developed by the ONF Security Project. [99]

4.1.1 Clearly Define Security Dependencies and Trust Boundaries

The first security principle demands a precise clarification of trust-boundaries between the single elements and a corresponding adaptation of security mechanisms. All elements running the protocol should support mutual authentication and have to be able to agree on minimal security associations and mechanisms. Integrity protection algorithms (e.g. SHA-3) should be attached to any protocol exchange message to verify its legitimacy. Proper and mandatory authentication and integrity measures solve a large section of Tampering and Spoofing. The ability of attackers to inject messages into the network, tamper switch, or controller data and reconfigure the devices is limited to a great extent. The ONF assumes that the use of mutual authentication impedes fingerprinting and identification of controller servers as demonstrated in [73]. However, a timing attack on switches is still a possibility, as it is unrelated to the authentication of switches and controllers.

4.1.2 Assure Robust Identity

The second principle corrects the problem of Repudiation and proper traceability. According to the security research team, any SDN element must possess an ID which uniquely identifies the current device owner or performer of operations, and thus greatly simplifies the tracking of malicious devices. However, it does not seem that the researchers demand unique IDs for controllers or applications and do not address traceability in the case that a device is malicious but operated by a allegedly legitimate owner. Repudiation is therefore not fully mitigated. Another proposal to assure trustworthiness protects the datapath ID of auxiliary connection from being imitated, a problem which has been discussed in Chapter 3. In order to prevent the injection of harmful UDP packets, the ID of the auxiliary connection should be padded to 96 bits and randomised.

4.1.3 Build Security based on Open Standards

If the third principle is applied, insecure or immature security measures are not used, ensuring a better base security and utilisation of the most robust and tested algorithms in the network. The problem of deprecated authenticity mechanisms or fall-back vulnerabilities is consequently addressed at best effort. The team also requires a clearly defined and strictly enforced procedure to handle corrupted packets for all entities using the OpenFlow protocol, including the control plane. As demonstrated in Section 3.2.6 malformed packets denote a substantial threat for controllers or switches in SDN. The implementation of a clear development guide in the OpenFlow protocol could mitigate this Denial of Service attack vector.

4.1.4 Protect the Information Security Triad

The fourth principle draws attention to the fundamental "information security triad" *Confidentiality, Integrity, and Availability* and suggests a constant review process and identification of any design choices in the standard which might foster Denial of Service or enable new STRIDE threats. Additionally, the protocol has to be able to incorporate the latest security solutions or mechanisms in order to be future-proof and flexible. Concrete suggestions include the automatic notification of other controllers as soon as a controller has requested a master role change or the use of unique keys or certificates for any active connection. While

these requirements do not explicitly mitigate a STRIDE threat, they propose a reviewing process for the entire specification which could reduce the risk of future DoS, Spoofing or Tampering.

4.1.5 Protection Operational Reference Data

The fifth aspect concerns the issue of exposure of important data in SDN devices and the subsequent Elevation of Privilege. All policies, keys, certificates, or similar security relevant information should be protected with integrity checking and optional encryption mechanisms to prevent or, at the very least, slow the effective Information Disclosure (a measure similar to the encryption and protection of password data). If relevant OpenFlow assets are checked for integrity or encrypted, the second layer of security might prevent an easy extraction of network information and privilege escalation on vulnerable servers.

4.1.6 Make Systems Secure by Default

Designing technology to be secure by default is a well known Saltzer and Schroeder design principle [64] and applicable to software-defined networks. The ONF task force demands secure default deployments, regardless of configuration or network design. The requirement is held vague, but examples such as cryptographic algorithms, preconfigured access control lists, and support for mandatory security mechanisms as standard features are suggested. The deficiency of information regarding the use of TLS [42] and the handling of connection interruption are cited as a point of criticism for the OF-Switch specification and require clarification. It can be assumed that a TLS-by-default configuration is implied in a secure by default network. Similar to the need to clearly define trust boundaries, a secure by default protocol effectively solves a significant portion of the first two STRIDE threats, Spoofing and Tampering.

4.1.7 Protect Accountability and Traceability

The penultimate aspect is the provision of accountability and traceability of the individual interactors and processes of the system, a trait which is closely linked to Repudiation and forensic analysis. Any SDN device must maintain logs and, in case of a neighbouring network event, immediately inform all connected devices or the SIEM. Cryptographic algorithms ensure confidentiality and integrity and protect the logs files from harmful influence. If an attacker attempts to spoof the ID of a main or auxiliary connection, the system should be notified. Similarly, all controllers connected to a switch must be aware of role changes and external flow table modifications, unless they explicitly unsubscribed from receiving notifications. This extended logging and monitoring functionality averts the ability to deny or hide device actions and provides forensic tools in order to prepare for future attacks.

4.1.8 Properties of Manageable Security Controls

Lastly, a manageable security control is recommended which focusses on automatic key distribution, free choice of security mechanisms, and customisability of security systems. The ONF committee criticises the lack of proper controller policy conflict resolution and insufficient description of key distribution and management. Since pre-shared key mechanisms are not mentioned in the protocol, the research teams assumes that only certificates are available

as authentication mechanism and criticises this lack of flexibility. The aspect itself does not target a particular STRIDE threat but the amendment of the protocol may ease security configuration and deployment in large networks.

4.2 The Secure ONF Network

The recommendations of the ONF committee sketch the first approach to a secure programmable network. All switches and controllers are mutually authenticated and secured using the latest TLS standard. Certificates and keys are distributed securely and automatically in the network and every device maintains several keys for each connection. It is deemed impossible to inject falsified, malformed or spoofed messages into the network as the protocol messages are checked for integrity and authenticity. If multiple controllers are used, every controller is authenticated and a potential rule conflict of equal-role devices is avoided. All events are instantly logged and reported and are secured over integrity mechanisms, which prevent direct topology or data tampering. The approach tackles a large amount of problems in current SDN and prevents ambiguous definitions. However, there are still several security issues present, even when using a secure OpenFlow protocol. The ONF team does not propose out-of-band control channels and the overall secure by default design is held vague. It is desirable to separate management and the automated control messages of the network from conventional data traffic. If a network link is incapacitated, either inadvertently or intentionally, control messages to automatically remedy the problem might not pass through and the incident management capability is limited. Secondly, while several issues of multiple controllers are addressed, the lack of clearly specified coordination and load-balancing functionality is not. It is not precisely defined how a switch distributes control or PACKET_IN messages among the connected equal controllers. It can be assumed that, if conventional controllers (e.g. NOX or the OpenFlow reference controller) are used, the switch will send the majority of asynchronous messages to a single device or to all connected devices equally. In both cases a resource consumption attack is feasible. Assuming the main control channel is overloaded, the switch does not change to a less occupied controller as the exhausted connection is still active. Furthermore, the constant replication of OpenFlow coordination messages might burden the switch CPU and generate significant traffic overhead in the overall network. A stricter definition of load-balancing and distribution functionality, as it is the case with auxiliary connections, might be required. Thirdly, the exhaustion of switch flow tables is still possible despite authentication and integrity checking algorithms. Modified headers sent by a malicious host are presumable seen as legitimate and passed on to the controller. The ONF security evaluation of the 1.3.4 version of the protocol does not address flow table exhaustion. Since the release of OF-Switch 1.4 (outside the scope of the ONF security document) the protocol specifies an automatic eviction mechanism based on flow table priority, a measure which could successfully secure important flows. Still, the feature is optional and not a required mechanisms in OF-switches. Due to the flow table exhaustion attacks demonstrated in Section 3.2.6 and in order to be secure-by-design, the eviction mechanism must be a required feature. Additionally, Information Disclosure based on response times is still predictable and not prevented. It is unreasonable to demand proactive flow installation to prevent timing attacks and randomisation could introduce unacceptable delays in the network. [63] The OpenFlow protocol itself can not prevent the reveal of flow table entries in a switch, a design flaw which has to be solved by external applications.

Threat	Unique to SDN	STRIDE Threat
Forged or faked traffic flows.	No	ST
Attacks on vulnerabilities on switches.	No	SID
Attacks on control plane communications.	Yes	STID
Attacks on vulnerabilities in controllers.	Yes	TID
Lack of mechanisms to ensure trust between the controller and management applications.	Yes	SE
Attacks on and vulnerabilities in administrative stations.	No	SE
Lack of trusted resources for forensics and remediation.	No	R

Table 4.2: The seven threat vectors and their corresponding STRIDE threats. [8]

Lastly, the ONF team does not extend their security demands beyond the OpenFlow protocol. Inter-controller and application interaction is not covered. There are no measures to protect the control plane, to identify and isolate attackers, to automatically adapt and restore network state, or to mitigate controller failure. The identified STRIDE attacks related to higher planes still apply, as the ONF is only concerned with the communication protocol and the data plane. A network based on a secure OpenFlow protocol can not be considered secure-by-design or default. Consequently, the thesis leverages the secure design proposal by Kreutz et al., 2013 [8], which defines custom security requirements for the entire SDN paradigm, and attempts to introduce full network security based on concepts and practical design approaches.

4.3 A Secure and Dependable Control Plane

In order to draw attention to security deficiencies in SDN, Kreutz et al., 2013 [8] define seven default threats, which they assume to be prevalent in a software-defined network. The research team hypothesises that the new standard changes the nature of network security requirements. They designate centralisation and programmability as "attractive honeypots for malicious users and a source of headaches for less prepared network operators" [8]. Overall, seven potential attack vectors are found. Three of which are entirely new in SDN, whereas the impact of the remaining four weaknesses is amplified. Each threat targets a different architectural component and broadens the overall attack surface of the network. As opposed to STRIDE, the vectors do not indicate goals of an attacker but highlight potential weak spots. Nevertheless, it is possible to associate the threat vectors with the respective **S**poofing, **T**ampering, **R**epudiation, **I**nformation Disclosure, **D**enial of Service or **E**levation of Privilege aspect. Table 4.2 summarises the defined vectors and maps appropriate STRIDE elements.

To mitigate these deficiencies the researchers investigate nine potential solutions and propose a personal vision of a secure and dependable software-defined network (see Table 4.3). Although the concept is designed to be implemented without the need of additional security, middleboxes or dedicates servers are considered beneficial. It is likely that middleboxes have to cooperate with the controller to reduce overall load on the device and enhance the security functionality in the network. The team did not introduce specific practical solutions for any

Solution	Description	STRIDE
Replication	Provide control and application plane redundancy.	D
Diversity	Implement diverse controller/application solutions.	TD
Self-Healing	Automatically isolate or replace faulty devices.	TD
Dynamic Switch Association	Implement flexible switches and control message verification.	SD
Trust between Controllers & Devices	Implement switch authentication and fault detection.	SRI
Trust between Controllers & Apps	Implement application authentication and fault detection.	SRI
Security Domains	Isolate control software from the OS.	E
Secure Components	Install secure network components.	T
Fast and Reliable Update & Patching	Introduce dynamic and automatic data and control plane security updates.	TID

Table 4.3: The proposed solutions and their respective STRIDE aspects. [8]

of the principles and only sketches a design. Since the proposal of the concept in 2013, a wide range of potential solutions has been published or used the publication as basis for an application or controller framework. This thesis inspects the approach of the research efforts and attempts to incorporate practical, existent implementations to fulfil the postulations. The *OpenFlow Switch Specification Ver. 1.3.5* [100] has already been updated to incorporate selected requirements of the ONF security assessment *Principles and Practices for Securing Software-Defined Networks* [99]. In order to adapt to the fast-paced changes of the specification, the evaluation is fully taken into consideration and forms the new OpenFlow protocol basis when assessing the necessity of further security solutions or workarounds. To evaluate the current security of SDN the feasibility and the availability of the individual mechanisms are summarised.

4.3.1 Replication

Any large system dependent on a single entity can not be considered reliable. As a consequence, the first proposed requirement is the replication of any essential network asset and the introduction of fall-back mechanisms.

The controller needs to be duplicated to assure fast failure recovery and resilience in SDN. Although switches and physical connections have to be secured as well, available measures such as link aggregation or detour paths are conventional and not a new challenge in SDN. The control channel itself is secured using supplementary connections. A concern is the fact that the main link is another single point of failure. If an attacker successfully injects a TCP FIN packet to interrupt the main communication, all previously established auxiliary connections are terminated. The switch is not able to automatically select a new main link. The problem of connection interruption is addressed in the ONF security publication, but a remedy for the failure of a main link is not suggested.

The focus of availability and resilience is on the distribution of controllers, a central problem in SDN. Not only due to security but also due to scalability and performance concerns.

	Physically Centralised	**Physically Distributed**
Logically Centralised	Standard single controllers (e.g. NOX, Floodlight)	Clusters with shared database (e.g. Onix, OpenDaylight), orchestration layer (Hyperflow)
Logically Distributed	Separation layer (FlowVisor), Synchronised cluster for each domain (e.g. Onix, OpenDaylight)	Network partitioning (e.g. DISCO, Kandoo)

Table 4.4: Controller types.

Consequently, a vast amount of research about controller and application designs has been published to this day. As discussed in Section 2.4 and Section 4.2, the ONF definition of multiple controllers is incomplete. Switches are able to maintain several connections with controllers and provide a rudimentary Replication functionality. However, topology information has to be maintained for every single device and load-balancing mechanisms are not suggested. Large-scale networks might impose a heavy burden on every single connected controller. Controllers are therefore extended to function as natively distributed devices, which are capable of managing various sections of the network over shared databases. Complex controllers either operate as cluster incorporating a horizontal communication interface or are orchestrated by a proxy layer or SDN application. Generally, four types of distributed control systems are present, depending on the logical and physical distribution of the device (see Table 4.4). For the purpose of Replication, the physically distributed, but logically centralised approach with fallback-mechanisms is the most relevant philosophy. It is necessary that the controller architecture implements the master/slave concept of the OpenFlow protocol properly and provides minimal fault tolerance.

The first noteworthy and historically relevant example of inherent control distribution is the Onix controller. [53] Onix can be deployed on one or several server clusters, with each server running single or multiple instances of the controller. If one Onix device fails, neighbouring controllers automatically assume control over the unmanaged sections of the network. Distribution of the NIB and verification of consistency is delegated to the applications. Onix is a closed source controller and does not provide insight into its internal mechanics. It is unclear how the OpenFlow master/slave concept or validation of peer controllers is implemented. Further details about the highly influential Onix controller have not been published and are likely to remain undisclosed, as Google deploys the system in its B4 data centres. [19] Nonetheless, the controller serves as a suitable demonstration of a reliable distributed deployment of OpenFlow controllers. Major projects, which are open-source and support distribution techniques, are OpenDayLight [49] and ONOS [50]. The network operation systems are well documented and consist of a similar architecture. Both controllers run on a cluster of servers for high availability, use a distributed data store with optimal TLS authentication mechanisms, and employ leader election, i.e. the master/slave concept of the OpenFlow protocol. [50] Still, as demonstrated in Section 3.2.6, the controllers suffer from various vulnerabilities which might lead to device failure. Using software and parsing exploits it is possible to incapacitate the controllers. Consequently, it might be feasible for an attacker to take down every single backup instance in the network one after another by abusing a specialised exploit. The ONF addresses the vulnerability of the OpenFlow protocol regarding the handling of malformed packets in their security publication. If a strict

procedure is defined, the risk of malformed packets could be solved, but software faults are out of the scope of the protocol. While DoS flooding and resource consumption attacks are mitigated in the distributed controller design, resilience to software faults remains an issue. Additionally, the new peer controllers might introduce new attack possibilities (see Section 3.2.3). All of the three discussed controller software delegate the responsibility to verify topology consistency and malicious devices to the application plane are thus still vulnerable to Tampering or Spoofing, if authentication is bypassed.

Several smaller proposals are currently in the development or concept stage and are not readily deployable, but nevertheless offer insight into valuable reliability concepts.

DISCO [101] is a fully distributed controller deployed on top of Floodlight, which assumes control over a domain of switches and computes end-to-end paths by exchanging network information messages. The controller is designed to be adaptable to any underlying network design and offers the capability to connect various network domains. In case of controller failure, neighbouring controllers are informed and recompute their domain traffic path accordingly, similar to the behaviour of legacy routing protocols. DISCO does not provide replication or fault tolerance mechanisms and the affected domain is effectively taken out of service. DISCO, while it does reduce load on controllers by partitioning the network into single domains, is therefore only a partial solution to avert DoS and guarantee robustness.

The Kandoo framework [102] proposes hierarchical layering to reduce the load on the controller. A single centralised controller regulates several sub-controllers, which only possess knowledge about their assigned domain. The segmentation of the control responsibility provides robustness against heavy control channel load and reduces the impact on a network under attack. Kandoo faces a similar problem as DISCO, as it does not seem to offer replication and failure recovery mechanisms. While it is theoretically possible to assign the main controller as slave listener in cooperation with various sub-controllers to provide a recovery measure, the single point of failure simply shifts to the top device. Tootoonchian et al., 2014 [103] identified a problem in current distributed controller designs. Onix and similar architectures do not address high load on single controllers, which is also identified as a security negligence in the OpenFlow protocol. To approach the problem of load-balancing the team presents Elasticon, a distributed controller which dynamically shares the network load among a set of controllers. Using the elastic approach a DoS attack might be significantly weakened when sharing the workload among several devices. The controller design is heavily centred around the master/slave concept of the OpenFlow protocol. As additional devices are connected to the switch, sufficient fault tolerance of the control plane is assured. A drawback of the design is the horizontal TCP connectivity of the controllers and the resulting lack of authentication. The controllers maintain a distributed data store over the TCP connection, which a malicious device is able to manipulate. Secondly, Elasticon is proposed as autonomous but highly specialised controller. An approach to integrate the design into existing, popular controller structures is desirable to prevent a fragmentation of controller architectures. As ONOS is partially inspired by Elasticon, future integration of the load-balancing feature is a possibility.

Applications which coordinate standard controllers are another method to achieve distribution. Hyperflow [54] is an example of a flexible distribution platform. The application is installed on every controller and connects to a distributed data store via a publish/subscribe mechanism. Upon connection the controllers periodically advertise their existence and store or fetch information in the database. If a controller ceases to advertise itself, the Hyperflow application assumes device failure and reconnects the switches to the next available

Solution	Description	Fault Tolerance - Project Status
Onix	Distributed and functional control systems, which coordinate actions over a horizontal interface and share network information in a distributed database server.	Strong - Deployed, but undisclosed.
OpenDaylight		Strong - Actively maintained controller.
ONOS		Strong - Actively maintained controller.
DISCO	One controller pro network domain, which exchanges messages with peer controllers over a horizontal interface.	Weak - Proof of concept.
Kandoo	Local controllers coordinated by a single top controller.	Weak - Proof of concept.
Elasticon	Adaptive distributed controller framework, which dynamically removes or adds controllers based on network load factor.	Strong - Proof of concept.
Hyperflow	Integration of a coordination application on top of standard controllers.	Strong - Proof of concept

Table 4.5: Projects to achieve Replication and fault-tolerance by distributing controllers in a software-defined network.

controller. The research team implemented the application on a NOX controller with only minor changes. It can be assumed that the application is flexible enough to be implemented on other controller builds. Again, the proposal addresses failure recovery but not the authenticity of the connected devices.
Overall, a crucial aspect of SDN Denial of Service, the failing of the control plane, can be solved by deploying multiple synchronised controllers in the network. Practical and available controller solutions are the open-source projects ONOS and OpenDaylight. Additionally, applications such as Hyperflow can coordinate and secure multiple controllers for added reliability. Overloading of the control channel can be solved using the Elasticon load-balancing approach. Two aspects of DoS remain unsolved thus far. Software exploit attacks on the entire control plane are still feasible due to homogeneous controllers and an unauthenticated horizontal communication introduces the risk of topology falsification and blackhole routing. A secure design must verify inter-controller communication or automatically overrule the decisions of single rogue devices.
Table 4.5 shows an overview of the relevant proposals, their security properties, and overall deployability.

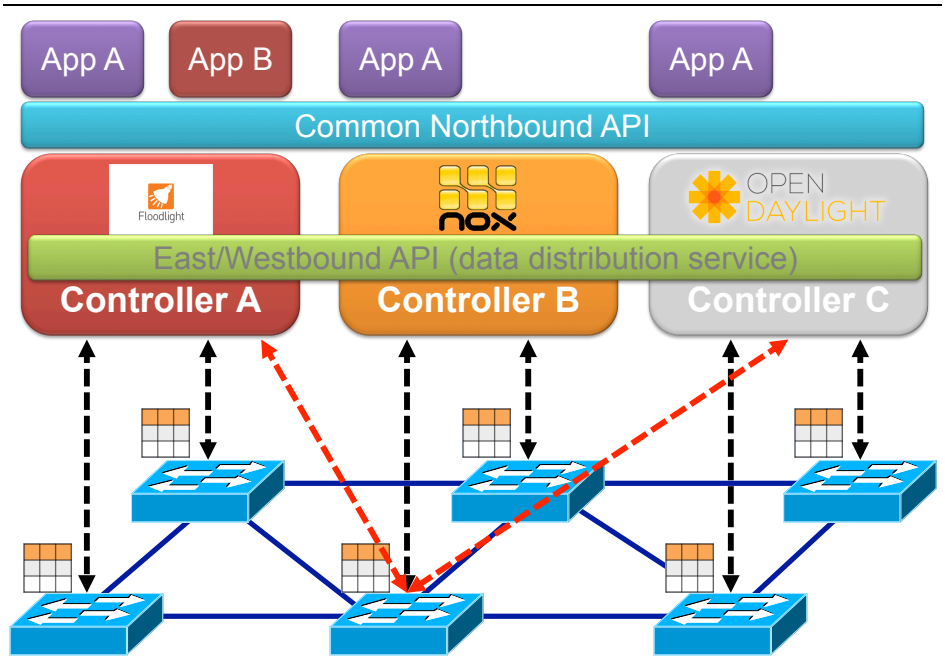

Figure 4.1: Securing the networking with Replication and Diversity as envisioned by Kreutz et al., 2013 (Figure devised by [8]).

4.3.2 Diversity

A major incentive for the introduction of SDN is to relinquish vendor dependency and the adoption of open standards. Networks consisting of a wide-ranging set of hardware and software are less vulnerable to specific development or device faults. [8] Operators could leverage the vast choice of software and operating systems to implement an optimal network configuration. The focus on open-source solutions in Software-Defined Networking demonstrates great potential to deploy a wide range of specialised and diverse solutions to increase robustness and security in the network. However, apart from the ONF southbound interface specification effort, the amount of attempts to promote interoperability is marginal. Controller developers only address distribution of the same device and do not cooperate to achieve a horizontal interoperability. An example is DISCO [101], which does emphasise the support of diverse network architectures and supports inter-domain communication, but distributes the same controller type in the network. Different infrastructures might require specialised controller designs for performance as well as security features. The lack of northbound interface standardisation hinders the development of flexible and ubiquitous hypervisor applications for any controller type. Major network equipment vendors are on the move to develop proprietary and inflexible controller solutions to retain network dominance. [104], [105] If the development of network applications and controllers diverges further, SDN might face a challenge of dependability and vendor lock-down reminiscent of legacy networks.

Few attempts and possibilities to facilitate controller diversity in the network exist. In an IETF draft researchers propose the Software-Defined Networking interface (SDNi) [106] as a message and synchronising protocol. The purpose of the protocol is to enable inter-domain

communication between different controllers and to implement a true east-/westbound interface which does not rely on distributed databases. The informational draft has been published in 2012, but has expired since then. No further developments regarding SDNi are apparent and it seems that the proposal has been abandoned. The cause is unclear, possible reasons are high development complexity or lack of demand or motivation.

OpenDaylight and ONOS were selected as examples of viable distributed controllers in Section 4.3.1, yet they do not support Diversity. OpenDaylight implements SDNi as a horizontal interface since the Helium release [47], but communication is only available between OpenDaylight controllers thus far. ONOS has plans to implement an ONOS east-west interface, but details are not available yet. [107] It is likely, that coordination falls under the responsibility of the SDN applications. With the northbound interface not being standardised, the next viable solution is the insertion of a proxy plane between the southbound interfaces of the data and control plane.

The previously discussed FlowVisor [55], which separates the network into disjunct slices for controllers to manage, is a viable and practical approach. Although the presentation of the software was performed with homogeneous NOX controllers, FlowVisor is theoretically agnostic of the controller design due to the standardised OpenFlow communication protocol. It might be feasible to implement diverse controllers as proposed in the threat vector approach (see Figure 4.1). Using distributed controllers for a single FlowVisor network slice it is also possible to integrate further DoS robustness and the Replication aspect. FlowVisor, however, is not able to operate multiple controller types on the same network slice without introducing rule conflicts.

The hypervisor proposal CoVisor [108] addresses this problem and specifically focusses on the deployment of different controller platforms and applications in the network in order to provide administrators with the ability to select the best network applications and functions. Using detailed access control the modification capabilities of the controllers are limited to their dedicated purposes. Additionally, administrators can customise the control and topology view of the controller to limit the power of malicious or faulty devices. The project has been published very recently in May 2015. Albeit an feasible concept, a full assessment of its practicality is not possible yet.

If Diversity is present in the network the impact factor of individual vulnerabilities is significantly reduced. Diversity specifically addresses the Denial of Service threat by reducing the overall availability of switch and controller bugs to an attacker. The amount of security holes and the resulting Tampering risk in the software might increase with different controller types, but a software failure will not affect all network components. If a switch or controller is incapacitated due to an attack, an immune device could theoretically take over the load in the network and prevent Denial of Service. Additionally, Elevation of Privilege is limited as the concurrent devices are restricted to only perform certain actions, as proposed in CoVisor. It is achievable to utilise the FlowVisor mechanism to construct an architecture as described in this chapter. However, the lack of current practical implementations impedes a proper evaluation of the concept. Overall, negligence of controller interoperability and northbound interface standardisation poses a potential future problem, which is currently circumvented with the use of application hypervisors.

It might be a matter of debate, if a diverse system is a necessity when designing for security or if a homogeneous approach is not sufficient. In risk assessments such as CVSS [66] homogeneous systems increase the severity of the environmental risk factor. On the other hand high diversity might violate the Saltzer & Schroeder [64] principle of keeping the system

simple. Many different controllers and switches in the network increase the complexity and overall fault probability that SDN attempted to solve. A counterargument is the choice of "best of breed applications" [108] to achieve maximum security and reliability. In summary, Diversity is currently neglected in SDN and is not mandatory for security, but provides additional reliability, which could work in favour of the network. Nevertheless it is unlikely that this principle can be fulfilled, as controller interoperability requires the same standardisation process as the southbound interface. At this moment true diversity is not practical.

4.3.3 Self-Healing Mechanisms

Self-Healing Mechanisms are the third and most complex building stone of the secure design. The research team does not clearly define concrete measures or solutions and resorts to the statement that "to explore diversity in the recovery process, strengthening the defence against attacks targeting specific vulnerabilities in a system" [8] is a necessary action. It can be assumed that Self-Healing Mechanisms heavily rely on the previously defined security aspects Replication and Diversity. Backup and watchdog controllers should be able to isolate and replace compromised devices or divert traffic attacks to robust stations. In the original design principle the scope of Self-Healing in the network is not clearly defined. Therefore thesis extends the definition to applications or controllers, which are capable of replacing, repairing, and verifying network data or can recognise and mitigate DoS attacks. Self-Healing Mechanisms attempt to treat the Denial of Service and Tampering threats of the STRIDE assessment. The discussed solutions are automatic reactive approaches and only apply to already detected failures and inconsistencies in the system. They do not focus on Spoofing and the identification of suspicious behaviour.

Replacing Failed Components

The first aspect of failure recovery and Self-Healing is the automatic treatment of device outage in the data plane of the network, e.g. after the discovery of failed switches or links an attacker has incapacitated. Controllers constantly maintain a complete view of the network and are able to deploy the recovery mechanisms of traditional routing and reliability protocols in an improved and efficient manner. However, due to the fact that switches have to await controller commands before being able to forward on a backup path, OpenFlow faces the challenge of achieving a carrier-grade 50 ms recovery time. The controller has to be able to notify all affected switches within the recovery range, placing a heavy load on the device. [109]

One practical method to solve this problem in large networks is the proactive installation of fail-over protection entries in the switches. The flow is linked to a specialised group table, which supports the ability to match packet flows based on current associated port status. Once a direct link goes down, the switch instantly resorts to the less prioritised, albeit matching, backup path and provides instant recovery without any need for additional computation. Although the path protection scheme proposed by Sharma et al., 2012 [109] is a viable method to assure quick recovery, it might burden the already very limited TCAM address space of switches. Secondly, as the matching is performed based on port status, OpenFlow switches might not recognise or be notified of any downstream failures and forward traffic into a dead-end until the control plane has recomputed the topology. Adrichem et al., 2014 [110] build on the proactive protection and introduce *crankback routing*. In crankback routing, the dead-end switches simply return the packet until a switch, which is connected

to a backup path is reached or the controller has updated the detour path. Packet loss is prevented, but the limited TCAM table size remains an unsolved issue.

The controller communication channel is sufficiently secured, if the Replication property is fulfilled. In case of controller failure, the affected switch is able to select a backup controller as new master or is instructed by the remaining equal controllers. Once the main link has failed, auxiliary connections are vulnerable due to the lack of a backup connection.

A new consideration is the verification and repair of malformed flow tables or topologies in the network. As deploying switch intelligence would contradict the OpenFlow and general SDN design, switches do not have any means to check the installed entries for Tampering excluding the technique of sending of resource database information to responsible controllers. While the ONF security publication does demand authenticity and integrity verification for any protocol message, hijacked devices can still inject malicious data into controllers or switches.

A concept to assure automatic correctness is the implementation of a democratic configuration process of applications and controllers. If multiple devices are present, only majority decisions are performed, eliminating the possibility of an attacker who is capable of installing single malicious or nonsensical configurations. To address the issue of Byzantine faults in distributed control systems (see Section 3.2.3). Li et al., 2014 [76] developed a *Byzantine fault algorithm* to override single malicious controller decisions. The controllers form a cluster and are abstracted into a single big controller. In case of PACKET_IN message or a network event, the new configuration is computed by every controller and the votes are collected. The majority choice is selected to be deployed on the switch, essentially overriding any single attacker decision and preventing Tampering of the switch flow tables. The researchers presented the algorithm and several successful use-cases but did not publish a practical and modular solution. The proposal thus far is purely theoretical and still in development.

An other democratic approach on a higher-level is Fleet [111], a fault-tolerant distributed controller design to identify and correct malicious administrator configurations. As soon as a network event occurs, one of the internal distributed controllers of Fleet can propose a reconfiguration based on a previously agreed proposal sequence. The remaining controllers sign the configuration with a cryptographic sequence to verify their vote. If the majority agrees, the policy is enacted in the network. If not, the next controller proposes a modification, which is voted upon. The Fleet controller design focusses on the possibility of malicious administrators but can be reduced to the controller itself based on the voting behaviour. A malfunctioning or compromised controller is overruled by majority vote and at the same time it is not possible to add more malicious controllers due to previously distributed controller keys. Nevertheless, similar to the algorithm developed by Li et al. [76], Fleet has not been tested in real network environments or is published as a well documented, practical solution. A new controller introduces further control plane fragmentation, but the concept could potentially be integrated intro prevalent distributed controllers (e.g. OpenDaylight, ONOS) to add extended security and Tampering resilience.

Solving the Denial of Service

The largest field of study in OpenFlow is the mitigation of Denial of Service, since the attack can be seen as the most severe threat in the network. It is preferable to implement features capable of autonomous limiting or blocking of packet floods to provide a network "healing" mechanism without any administrator interference. While the SDN concept introduces mul-

Problem	Solution	Description
Sub-50ms Failure Recovery	Proactive installation of table entries.	Traffic engineering application of the controller installs failover entries and routes on the switches, which are instantly activated as soon as a port goes down.
Malicious controllers and Byzantine faults	Redundancy and democratic policy decisions.	In case of relevant events, multiple authenticated controllers vote on a decision in order to override single malicious or failing devices.
Flooding attacks on the control channel	AVANT-GUARD, Lineswitch, conventional IDS.	Dedicated protective switches cooperating with the controller to repel and filter TCP SYN attacks on the control plane.
Flooding in the general network	Cooperation with monitoring and intrusion detection systems.	Monitoring and IDS inform the controller about suspicious events, which then reconfigures the network accordingly.
	Detection of anomalies based on PACKET_IN messages.	The controller identifies network anomalies based on the switch packets it has received and blocks the responsible flows.

Table 4.6: Approaches to repair failed network components or to avoid service failure in the network.

tiple DoS risks, most notably the *control plane saturation* and *resource consumption* attacks (see Section 3.2.6), it also provides many opportunities to quickly stifle flooding. The first and foremost priority of DoS mitigation in SDN is to assure an undisturbed control plane communication process in order to retain control over the network.

To protect the controller from packet flooding the concrete solution AVANT-GUARD [112] is proposed, which implements additional security features in the switches and thus the data plane. Shin et al., 2014 [112] develop the new *connection migration module* in OpenFlow switches, which effectively acts as proxy to shield the control plane from TCP SYN floods. The switch stores SYN cookies, drops unfinished handshakes, and forwards legitimate connection attempts to the controller, acting as protective wall for vulnerable network servers. Secondly, if certain predefined conditions are measured or fulfilled, the switch utilises a newly implemented *actuating triggers* module and actives a flow table which handles the new network event and drops, redirects, or limits traffic. DoS attacks are prevented and handled without any manual interference. A bonus feature is the hampering of port scans, as the AVANT-GUARD switch responds to every TCP SYN packet. A malicious scan identifies all possible ports as opened, effectively clouding any existent active ports for an attacker. Simultaneously, the scan is reported to the SIEM or controller. Nevertheless, there are several flaws in the protective SDN switch approach of AVANT-GUARD. The framework itself is elaborate and costly to implement, since virtual or actual hardware switches have to be modified to incorporate the module. Similarly, the OpenFlow standard has to be extended to support new messages and controllers must be modified accordingly. The proxy layer is only able to prevent TCP requests, not ICMP or UDP flooding. Furthermore, Ambrosi et al., 2015 [113] highlight the switch itself as significant weakness and demonstrate new attack

methods. It is possible to disturb any TCP communication that is passing the switch by establishing legitimate TCP connections and filling the proxy buffer of the device. Any new clients are then rendered unable to establish a connection over the targeted switch.

It can be argued, that these attacks are easily identifiable by intrusion detection systems as it denotes unusual behaviour. Nevertheless, the team has been able to implement their theory and exhausted the AVANT-GUARD switch in short time. As an extension they propose Lineswitch, a switch modification which distinguishes packets based on IP address and proxies packets only until the first successful TCP handshake. All subsequent TCP SYN packets are forwarded to the server but selectively proxied by the AVANT-GUARD mechanism based on a defined probability value. If an attacker establishes a legitimate connection beforehand and starts a subsequent SYN flood the attempt is eventually discovered after inspection and the whole IP is automatically temporarily blacklisted by the switch. Consequently, the attacker has to use real IPs in order to be able to respond to SYN-ACK packets, but is blocked as soon as a flooding attempt has started. Although an extension to AVANT-GUARD, Lineswitch faces the same practicality issues and relies on active modification of OpenFlow switches and controllers. Both of the aforementioned designs are able to protect the control plane and particular sensitive servers similar to an IDS or firewall, but do not provide automated mechanisms to regain control over the entire network during an attack. Additional solutions, which automatically recognise and prevent malicious traffic based on local anomaly detection, have been published.

Mihai-Gabriel & Victor-Valeriu, 2014 [114], rely on external intelligence and developed a SDN intrusion detection system. They utilised sFlow, a high-speed, sampling-based monitoring interface supported by several switch types and vendors. The OpenFlow switches export sample traffic to a separate virtual machine, which computes a risk threshold. If the machine identifies a risk, it sends a command to the controller to either block or divert traffic to the machine for traffic inspection purposes. The team has only developed a proof of concept and plans to develop a native Floodlight application to replace the virtual separate machine. Gkounis, Kotronis & Xenofontas, 2014 [115] are investigating a solution to prevent *crossfire attacks*, a recent attack technique which accumulates undetectable and legitimate low-bandwidth flows to flood and overload essential network links.

Lastly, OpenDaylight incorporates the optional DDoS application Defense4All [47] as a viable and practical solution to protect the network. The application cooperates with the OpenDaylight controller and separated *attack mitigation systems* or scrubbing centres. After installation, the system installs flow entries on specified switches to continuously monitor traffic and learns normal network behaviour from a central position. Once an attack is identified, the controller is instructed to redirect flows into the scrubbing or attack mitigation centres to cleanse the traffic. This model presents a potential path to leverage the central position of the SDN controller and simultaneously provide thorough protection against DoS in the network.

All of these designs, however, rely on external dedicated machines to support their techniques. It is desirable to install native detection and mitigation applications in the controllers in order to reduce capital and operational cost of the network and to prevent traffic flow amplification. A selected attempt to incorporate native IDS applications and to protect cloud provider networks in the controller is DaMask, developed by Wang et al., 2014 [116]. The Floodlight application inspects unknown packets, which the controller has received, and identifies potential anomalies using the open-source IDS software Snort as a basis. If an attack is detected, the system installs FORWARD, DROP or MODIFY table entries on the

affected switches, depending on a pre-defined administrator policy.

While the integration of IDS network applications in the controller does not require additional stations or monitors, a possible trade-off is the lack of monitoring of default network flows and the substantial load of the applications on controller latency and performance. To address the problematic latency impact of traffic monitoring, Giotis et al., 2012 [117] integrated sFlow into the controller server. The experimental tests showed a significant decrease in CPU and performance impact when using sFlow and demonstrate the possibility of a performant monitoring and automatic intrusion reaction system coupled into the control plane.

To the author's knowledge, the inspected proposals do not utilise the OpenFlow meter bands, which were added in the OpenFlow Switch Specification 1.3 in April 2012. A potential reason is the lack of adoption of newer OpenFlow protocols by controller developers. The flow rate measurement and limiting tools can provide significant DoS prevention capabilities to reduce traffic originating from certain network sections without cutting off the complete communication.

SDN is able to leverage the potential of Self-Healing Mechanisms in the network. The centralised view and control facilitates quick reconfiguration and rectification of network failures, assuming they are detected. Several solutions to prevent distributed Denial of Service are already proposed and the control plane can be protected using special switches or applications. Similarly, verification mechanisms to prevent injection of malicious controller messages can be used. Overall, Self-Healing Mechanisms is a viable design principle and a promising aspect of software-defined networks.

4.3.4 Dynamic Device Association

The fourth security mechanism centres around the principle of Dynamic Device Association. According to the research team, switches should be able to freely associate with multiple controllers in order to be able to tolerate faults and failures and to increase the overall robustness of the network. This approach, however, is largely covered with the introduction of Replication and Self-Healing Mechanisms and is thus a redundant proposal. A new aspect is the suggestion to provide switches with programmable multi-purposes CPUs or specialised attachable boxes. The purpose is to induce switch intelligence, capable of detecting malicious controllers and automatic control channel load-balancing. The team deems generic data plane intelligence helpful in the provision of security and fault-tolerance. [8]

The introduction of low-level configurable switch intelligence is a point of debate, as it could bring back complexity and security issues into the software-defined network. Reminiscent of the active networking project [28], programmable switches could be subject to new attack vectors, misconfiguration or software faults. Switching devices might need to be configured individually to function properly, leading to a step back in the OpenFlow paradigm. Additionally, the development of programmable switches or compatible dedicated devices has to rely on vendor acceptance, a hurdle, which several network designs failed to overcome in the past [2] (see Section 2.3).

On the other hand, an emerging belief is that "a 'smart controller, dumb switch' model is too naïve" [118]. Switches might need to be customised in order to adapt to the new model and be capable of providing self-reports of the topology, black holes or critical nodes in the network. [119] Indeed, a considerable amount of measures and switch designs that address automated load-balancing and selective routing on the data plane has been published [12].

The previous Section 4.3.3 discussed AVANT-GUARD and Lineswitch, two designs which require custom switch modification. They focus on security measures inherent to the switch, but only target DoS prevention and control plane protection. To the authors knowledge, no extensive projects, in which the data plane can detect malicious controllers or switches in the network, have been proposed so far.

The question remains, if data plane programmability does not actually introduce more security and reliability difficulty than it solves. Dumb devices are an easy target, but are less vulnerable to the configuration and software faults of complex systems. Large sections of this design proposal are covered by preceding mechanisms and overall effectiveness is yet unclear. The dependence on vendor support and adoption is high and no solutions are present yet. Consequently, this principle might be superfluous at this moment or in the future and is not considered in the final design of this thesis.

4.3.5 Trust Between Devices and Controllers

A fundamental requirement is to establish trust between controllers and switches in the network. The research team therefore proposes a consistent and reliable relationship between controllers and switches while the control plane should manage a list of authenticated devices in the network. It should be noted that the fifth design principle does not explicitly demand mutual authentication, a potential negligence in the team's design. If switches do not authenticate controllers, attackers are able to connect their own devices to perform operations.

Section 4.2 extensively covers trust management and authentication of the OpenFlow devices. If the OpenFlow protocol is updated according to the released security specifications, and strict mechanisms are predefined and mandatory, reliable authentication is expected to be present in any network design and does not need to be further addressed. This design principle therefore focuses on an aspect which would be mitigated if the OpenFlow protocol is secure-by-design.

However, Kreutz et al., 2013 [8] assume that controller-managed authentication of any device would stifle the flexibility of the network (an issue which could be solved with the automated certificate and key management suggestion of the ONF Security Project). Additionally, the possibility of an OpenFlow device of the network being hijacked and misused is always present. The researchers suggest intrusion and anomaly detection mechanisms to find and trace suspicious behaviour in the network. Switches and controllers maintain a trustworthiness threshold of any device. Once it has reached a limit, the device is automatically isolated, disregarding if it is authenticated or not. While it does not seem that a concrete implementation of this particular concept has emerged yet, a multitude of algorithms, applications, and controllers to detect and report anomalous behaviour in the network are being developed. The inherent design of SDN provides a great opportunity to realise network-scale intrusion detection and a remedy of the identified Spoofing threats. The Topoguard solution of Hong et al,. 2014 [78] demonstrated that it is possible to prevent the *Host Location Hijacking* and *Link Fabrication Attack* of the Tampering threat 3.2.3, if the controller is equipped with the basic algorithm to monitor inconsistent port state and the verification of message integrity and authenticity of LLDP or ARP response messages. Nonetheless, further available methods are not in the scope of this security mechanism. Several projects to deploy middlebox functionality in software-defined networks are discussed in Section 4.3.3 and 4.4.

4.3.6 Trust Between Application and Controller Software

In contrast to the southbound interface, applications and their variety of interfaces are not standardised. At the present time, no group is devoted to define strict requirements and security mechanisms for the northbound interface. The ONF security committee does not extend their scope beyond the southbound interface. However, due to the variety and configuration power in the network, applications are one of the most sensitive weak spots of software-defined networks. Security mechanisms to verify and track the legitimacy of applications in the network are a must. For this very reason, the fifth design principle demands a trust relationship between application and controller software.

Section 4.3.3 addressed the automatic resolution of application policy conflicts an attacker could induce. Nevertheless, limiting the command set of insecure or new applications by default or denying access based on lack of credentials is a preferable course of action. Since the controller is treated as the operating system of the network, introduction of role-based permission models and access control implemented in the controller itself are common suggestions. Ongoing and detailed projects to enforce security in controllers are Security-Enhanced-Floodlight (SE-Floodlight) [56] and the ONOS Security Mode [120].

SE-Floodlight (see Figure 4.2) is a controller extension, which provides a multitude of application and network security mechanisms. The developers specifically focus on application access control and rule conflicts. In SE-Floodlight the northbound API is secured using OpenSSL authentication. Any application integrated directly into the controller has to be signed by an administrator and packaged with credentials before they can be executed. All applications are divided into privilege levels, currently consisting of APP (default), SEC (security applications) and ADMIN. Depending on the role of the application, it is permitted to perform a particular set of rule modifications or requests. Operations, which are beyond the assigned privilege are rejected. If a rule-conflict arises due to commands stemming from different applications, SE-Floodlights selects the higher privilege operation, effectively providing privileged applications with the ability to override the flow rules of a lower tier. The registered applications are uniquely identifiable and all activity is logged for auditing and forensic purposes.

A very similar extension to Floodlight is OperationCheckpoint [83]. As opposed to a static and ubiquitous role-based functionality, the applications are granted individually selected configuration permits in order to keep flexibility in access and assure dynamic permission changes. Members of the research team that has proposed SE-Floodlight are working on the ONOS Security Mode. In comparison to SE-Floodlight, the functionality of the security mode is narrowed and only specifies access control. Nevertheless, applications are also grouped into *admin* and *non-admin* mode based on their authentication and have to request permission for various tasks beforehand. If the role does not match, the request is denied. Application IDs and activity logs are a native feature of ONOS and can most likely be utilised for forensic analysis.

The authentication and role-based approaches significantly reduce the threat of bad applications in SDN. As long as credentials are not compromised, it is impossible to force false policies into the network, preventing another aspect of Spoofing. Furthermore, the different access levels inhibit the risk of an Elevation of Privilege. Unique application identification and the possibility to trace malfunctioning controller applications address concerns in regard to Repudiation.

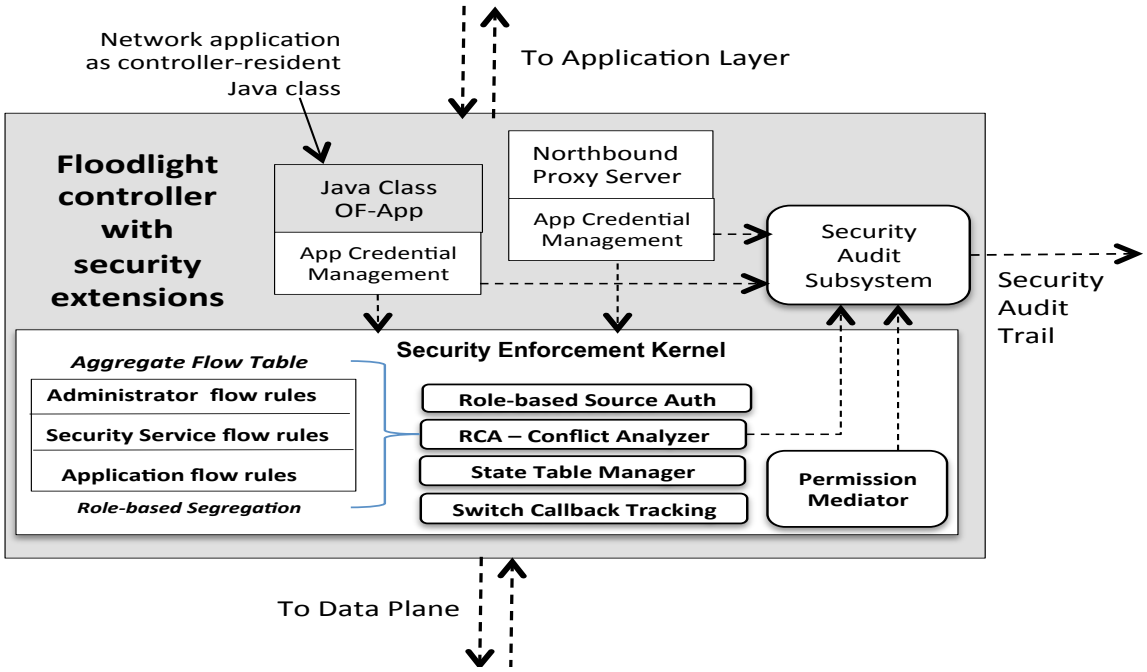

Figure 4.2: An overview of the architecture of the Security-Enhanced-Floodlight and its authentication mechanics (Figure devised by [56]).

These research efforts, however, are limited to the development of a designated controller. If a dominating controller does not emerge, a flexible application layer or sandbox securing the controller kernel must be an achievable option. It does not seem that any projects are dedicated to authentication between the application and control plane. In 2013, PermOF [80] has been suggested as a platform which introduces an additional authentication and access layer on top of the control operation system. However, presumably because of a lack of interest or demand, development on the project has come to a halt.

Access control is a central issue in SDN, especially admission to the control platform. SDN emulates the structure and composition of a conventional operating system and exposes a similar weakness to custom applications. If role-based access, credential query, and authentication are applied to the control software, security could be vastly improved. Applications express as much power in the network as the control plane itself. As they could harbour software bugs, are prone to vulnerabilities and are installable by attackers, controllers have to utilise methods and measures to restrict permissions and the impact of configurations. The obtainable solutions proposed in this chapter focus on enhancing an strengthening the controller itself, an approach which is limited to the particular controller software and could restrict diversity. Nevertheless, a control operation system that is capable of restraining applications and utilising the secure OpenFlow protocol of the ONF also mitigates a significant amount of the security threats of conventional OpenFlow networks.

4.3.7 Security Domains

Security Domains are only tangentially covered in the ONF security evaluation, but a differentiation of authorisation levels in the Openflow protocol itself is postulated. [99] Although the Architecture Recommendation [36] discussed methods to restrict access and provide services to different user levels, detailed mechanics are not proposed and not part of the scope of the document. The OF-Protocol itself is currently not able to differentiate between various trust and authentication levels. In the data plane, separation of Security Domains is not possible and has to be handled by proxy applications such as FlowVisor or Hyperflow.

Two aspects can be classified as approaches to develop Security Domains. Kreutz et al., 2013 [8] focus on the control platform and the need to isolate the software from the underlying system. Failures of the controller should not affect the remaining programs on the server and vice versa. While team does not further specify the term "Security Domains", a second definition may be the partitioning of the network into various restricted domains. Control systems such as RADIUS and Kerberos could restrict access to the sensitive control plane services or limit the extent of an attack from a compromised host in the network.

Concerning the protection and isolation of the control software, the previous section addressed the risk of equal security levels of applications. However, applications are often coupled to the controlling operating system and risk operation shut-down as evidenced in Section 3.2.6. As a result, Shin et al., 2014 [79] developed Rosemary, a controller specialised in isolation and robustness. The team highlights the problem of poor resource separation and control of applications residing on current software designs. The proposed controller implements the design philosophies of SE-Floodlight and Fort-NOX of role-based access and application authentication. Applications are outsourced to external processes independent of the main control software and are thus unable to accidentally or intentionally terminate the controller. The memory consumption of applications is limited by a resource monitor, which restricts the CPU and memory usage based on a predefined threshold.

The controller design has been deployed and tested successfully. Nonetheless, the proposal contributes to the aforementioned problem of controller fragmentation. It would be recommendable to integrate the robust and independent architecture into well-maintained and dominating network control systems (e.g. OpenDaylight, ONOS, Floodlight). As several members of the research team that has developed Rosemary contribute to ONOS, advanced security developments are a possibility. A further and potentially controller-agnostic solution to protect the controller from application failures is LegoSDN [121]. The prototype integrates a hybrid of an application hypervisor and rollback-mechansism to isolate the control kernel from the custom applications, while enabling quick recovery to stable state in case of faulty configurations. Potentially failing and memory-exhaustive application processes are also separated from the sensitive controller.

Approaches such as Rosemary and LegoSDN heavily decrease the destructive capabilities of applications and thus Tampering and Denial of Service in the system. Nearly all dangers stemming from rogue applications and spoofed authentication are addressed in this design. Albeit a very robust architecture, performance could be a limiting factor.

The second significant aspect is the separation of network hosts and sections into Security Domains with restricted view and access. Accessing vital network services, particularly the OpenFlow configuration service, has to be limited to authenticated hosts and network partitions to reduce the range of attackable devices in the network. As discussed in Section 3.2.7 and 4.3.2 the network slicing of FlowVisor and similar approaches are not mature enough to

provide secure network restrictions yet. However, OpenFlow is able to leverage the 802.1X mechanisms and protocol for Authentication, Access, and Accounting (AAA). If 802.1X is not supported by OpenFlow devices, the controller could integrate the authentication protocol or communicate with a dedicated authentication server using standardised commodity switches. As the flow matching is capable of dropping packets based on port, EtherType and source addresses, it is possible to filter service access from certain network regions. Mattor & Ferraz [122] utilise this mechanism and augment the OpenFlow network with a RADIUS-based authentication to install fine-grade access control. The team develops the AuthFlow mechanism to match credentials and service access to flows and restricts admission based on individual host credentials and network location. Access to sensitive services and the control plane from insecure hosts or network section is effectively impeded. This aspect does not prevent a specific STRIDE threat, but it considerably diminishes the risk of infected conventional hosts or personal devices of the network.

In summary, the two interpretations of Security Domains are viable techniques to limit the open access to software-defined networks. Specific to SDN is the need to segregate the controller from applications and the remaining network services to establish a proper isolation of the operation and management planes. Extraction of controller application and general services processes coupled with tight access control reduces the risk of software failure and prevents several STRIDE threats of the control plane. On a larger scale, the adaptation and enhancement of conventional network access control, host verification, and service restriction in SDN is one viable way to fully secure important components of the network.

4.3.8 Secure Components

In order to retain confidentiality and data integrity, the internal components of the SDN devices need to be safeguarded individually. The research team suggests the use of a Trusted Computing Base, a separated and secured hardware and software section of system. The concept of Rosemary (see 4.3.7) fulfils this requirement as it separates the control software from applications and requires access credentials. Moreover, the ONF extensively covers the demand for data protection in their fifth principle, "Protect Operational Reference Data"[99]. However, only sensitive data such as policies, credentials, and service descriptions are listed. The ONF does not explicitly require a security mechanism for the NIB of the controller or the forwarding tables of the switch. A disclosure attack on these databases could reveal an ample amount of information about the network. As it can be assumed that TLS and SSH have to be used to acquire this data due to the ONF security requirements, a two-factor authentication measure might be superfluous. A second detriment is the negative impact of an encrypted lookup table on flow latency and performance.

Consequently, this security principle might be redundant in the overall scheme. The individual benefits are covered by similar suggestions or the effective and secure implementation of the OpenFlow protocol and the protocol itself can thus be considered a secure component of the system, if the requirements are fulfilled. Extending the principle, the control topology of the controller and the flow tables of switches may be considered parts of the network which have to be secured or securely assembled. Several tools to verify the network policy, topology and overall correctness of the network system are already available. The algorithms check and examine network configuration based on invariants and models and alert the administrator if violations occur while providing concrete examples. [12], [59]

As evidenced by the Denial of Service threat, TCAM table sizes are a significant and

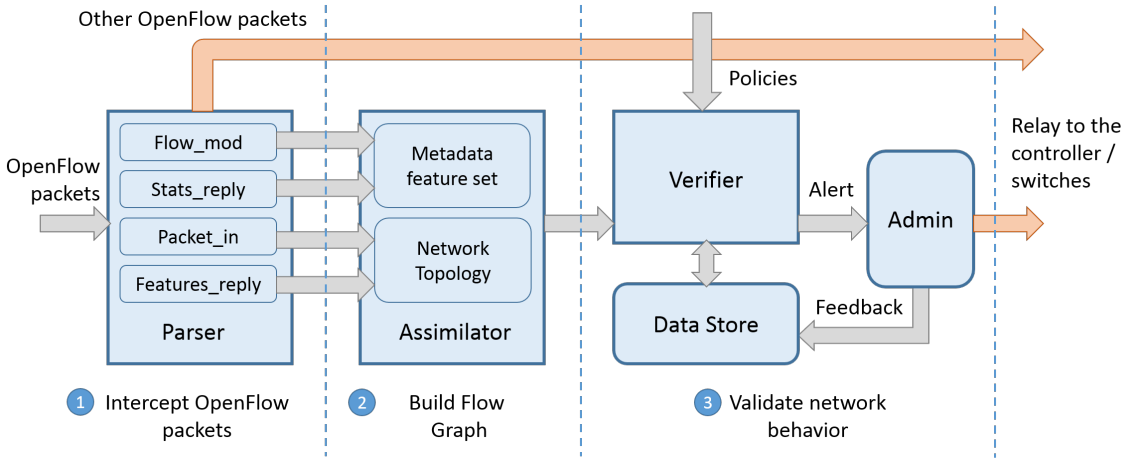

Figure 4.3: The detection mechanism and data flow of the SPHINX security layer (Figure devised by [123]).

exploitable issue in the current SDN design.

The application module SPHINX [123] is a comprehensive project to protect the controller and the table size of switches. The framework operates as a intrusion detection system and is able to detect a multitude of attack types, including TCAM exhaustion attacks, topology spoofing, blackhole routing, and general DoS traffic targeting the controller. The application is inserted as a proxy layer between the control and data plane and monitors incoming switch packets. Based on the metadata of the packets, a network graph and traffic database are constructed (see Figure 4.3). Any deviations in the trusted topology, attempts to exhaust the limit of table entries in switches, or excessive traffic patterns are instantly reported to the administrator. As it does not solve the raised alerts, SPHINX can not be considered a Self-Healing Mechanism. Nonetheless, its inherent and controller-agnostic intrusion detection functionality aptly illustrates a general concept to protect the integrity of data stores in SDN.

4.3.9 Fast and Reliable Software Update and Patching

The centralisation aspect of SDN faces further hurdles concerning deployment and maintenance procedures. The controller is a central element in the network. As soon as a security flaw or an attack opportunity in the control software is detected, the systems need to be updated and repaired. If it is necessary to shut down the controller for an update, the entire network is left in an insecure and restricted state. Similarly, migration of control servers or full updates could affect policy or security decisions or introduce incompatibilities with installed and developed applications.

Current controller updates rely on total replacement of the software. [124] The traffic and topology history and computed policy are deleted, leading to a clean-slate controller. If the subsequent policy and forwarding decisions are incorrect, security configurations might be reverted. Distributed control planes reduce the problem, as controller can be incrementally updated and the load is redirected to the functioning peer devices. However, architectural changes or differences in message handling might lead to inconsistencies or potential incom-

patibility issues. An update or migration of controllers should not affect the network in any way. As a consequence, the new controller has to possess the same applications configuration and network knowledge as the old, flawed device.

A proposal to address these update considerations is HotSwap [124], which modifies Flowivsor and inserts a proxy plane in the southbound interface. During the recording and replay phase, the proxy layer effectively mirrors selected network events and policy decisions of the old controller, which are then forwarded to the new device. The process continues until the network view and information of the new controller is consistent with the data of the older model and achieves a deployable state. As soon as correctness is verified, the older device is detached and the new controller assumes the main role. The update does not affect packet loss and latency and the network remains unaffected. As the mechanism builds on FlowVisor and the standardised OpenFlow protocol, it is adaptable to any controller type and does not require extensive customisation. HotSwap is a feasible mechanism to quickly and safely substitute deprecated control software, although the replication and mirroring might generate significant load on the CPU. However, it does not cover a comprehensive roll-back mechanism. Common security guidelines and best practices of OS and software development should apply to the control plane. It is necessary to validate the compliance and security of the entire new configuration. If miss-management occurs, roll-back and restore points need to be available. Lastly, the complete setup should be tested in capable and extensive virtual environments before actual deployment.

As a future potential solution Kreutz et al., 2013 propose ClearView [125], a project that is capable of automatically patching identified errors in the software and refusing malformed or dangerous patches. It has been developed for Windows x86 binaries, but its mechanism and design philosophy are helpful for securing and debugging the highly sensitive controller. While powerful and comprehensive, the application has to be rebuilt and customised by all controller developer teams individually. The concept itself, although preferable, seems currently infeasible.

Independent of control software and SDN protocol requirements, the availability problem of server shut-down in live networks may be aggravated in the central design. If the underlying OS has to be restarted, the controller is still unavailable despite proper patching flexibility. As control software commonly resides on Linux servers, Kernel Live Patching, which has been released for the Linux 4.0 kernel in April 2015 [126], may alleviate this problem by abolishing the need to reboot during updates.

The inherent properties and principles of SDN additionally enable the use of checkpoints in the system. Network topology is virtualised, flow tables can be queried, and policies are saved in the controller. The abstraction of the network facilitates mechanisms to save the complete state of the network in a database. If an attacker has damaged the network view or configuration, a saved checkpoint can quickly revert the network into a secure state.

LegoSDN, which has been presented in Section 4.3.7, provides a method to undo failures and configuration error but only targets the application plane. To extend the scope to troubleshooting of faults and misconfiguration to the entire network, NetRevert [127] has been published as a solution. The concept is similar to HotSwap and inserts a new plane, which intercepts OpenFlow messages and saves the controller state and its processes periodically. If a system failure occurs, the saved security points of the past are ready to be restored until a correct configuration has been reached.

The last aspect of secure and fast deployments is to establish a safe testing environment for new security updates and configurations. To fulfil this principle Mininet [91] may be

of service. It is fully customisable and is capable of simulating a real OpenFlow network environment and its switches. Mininet is limited in its scale due to space and resource costs on a single server, but can be extended and distributed with Maxinet [128] to support the emulation of large-scale data or regional networks.

The emphasis on software in SDN is a major new risk factor. As it is difficult to fully prevent exploitable bugs or faults in applications and controllers, an administrator should possess tools to quickly, securely, and reliably update the individual network elements. If mechanisms such as HotSwap, ClearView, or NetRevert are present and usable, the risk of Tampering or subsequent attacks could be significantly reduced.

4.3.10 Selected Additional Research Efforts

In the following section, several selected security mechanisms are presented. The solutions can not be categorised into a definite secure design principle and do not necessarily address a particular STRIDE threat, but nonetheless present a noteworthy approach or innovative concept to ensure security and enhance overall robustness in OpenFlow networks.

OpenFlow Random Host Mutation

A new way to block reconnaissance attempts is the OpenFlow Random Host Mutation (OF-RHM) algorithm [129]. Attackers and network worms frequently utilise port scans to identify the IP of vulnerable hosts in the network for future actions. In order to protect the network from the consequences of successful port scans, OF-RHM clouds the real IP of network hosts and maps the address against virtual, continuously changing IPs. The only device aware of the virtual addresses is the controller, which installs temporary forwarding rules to translate and hide the real address. If a legitimate host of the network attempts to communicate, the switch sends a PACKET_IN message to the controller, which assigns a virtual address for temporary communication. Only packets from an authorised source are constantly forwarded to real destinations. Even if an attacker succeeds in scanning a host with a virtual IP address, the IP is changed after a short period and thus voided. The team reports an invalidation rate of up to 99% of scans. OF-RHM can replace the security bonus of NAT middleboxes, which hide the real IP address of internal hosts. However, due to computational overhead and flow table limitations (every legitimate connection requires two forwarding table entries in a switch), the application might only be suitable for smaller networks.

Location-Based Authentication

Bifulco & Karame, 2014 [130] leverage the central design of SDN to develop the Network Proofs of Location (NPoL) protocol, a location based access control mechanism. If a new host connects to the network, its identity has to be verified by sending a signed proof of location message to the controller. The controller approves the key and maps the identity of the host to the IP and port it is connected to. A corresponding forwarding entry is installed on the switch and the host device is permitted to communicate in the network. If the location changes and the host relocates, the connection is blocked until it is confirmed again by a signed message. This mechanism effectively prevents internal IP spoofing of stationary hosts. Secondly, it is possible to map stationary devices interacting with the controller, e.g. an application or administration server, to a particular switch port in order

to prevent the spoofing of configuration messages in the network. If the location of an IP changes, the controller could theoretically warn the SIEM and alert an administrator of unusual behaviour. Although user authentication is guaranteed using AAA mechanisms such as RADIUS, the location-based approach can potentially provide a second layer of authentication for vital stationary servers or hosts without requiring additional devices or servers.

Stateful Firewalls and Flowguard

It is possible to implement a distributed, stateful, and scalable firewall in OpenFlow natively as demonstrated in a patent submitted by Google. [131] The central controller maintains information about the state of flows for every individual switch in a stateful firewall application. While the network is active, the controller monitors and collects information about particular flows and sends sample packets to the application. If the state of the connection violates a predefined policy or rule, the application informs the controller, which installs a flow entry to drop the corresponding traffic on the switch. If the state of the connection is legitimate, the controller may install a permanent forwarding flow entry. The goal of this solution is to reduce the performance impact of stateful firewalls and to monitor the entire network. However, the performance effect on the controller side is not considered in this patent proposal.

Although OpenFlow is capable of distributing the firewall in OpenFlow switches over the network, inconsistencies and policy violations may arise during the automatic configuration and deployments of rules in the network. As demonstrated by STRIDE Tampering (see Section 3.2.3), it is possible to circumvent firewalls by abusing switch flow table inconsistencies. Flowguard [132] extends the generic firewall mechanism of OpenFlow to detect and resolve any policy violation in the network. The application tracks every flow path of the network and identifies potential detour tracks bypassing the firewall. If a rule conflict is detected, the firewall patches the hole, while only minimally altering the remaining network flow. As SDN strives to supersede hardware security appliances, Flowguard and similar correctness models are necessary to validate the overall consistency of the specified policy.

FRESCO & OrchSec

An other security product of the research team that is dedicated to SDN security projects such as SE-Floodlight, AVANT-GUARD, and Rosemary is the FRESCO language. [133] FRESCO is intended as a development framework to accelerate and ease the deployment of security applications. The framework abstracts the imperative of security functions such as isolating, redirecting or forwarding traffic and exposes a simplified API for developers. Scripts in the FRESCO language are designed to report or handle network threats autonomously using the information of legacy or SDN monitoring devices and cooperate closely with the controller to enact the specified actions. To demonstrate the effectiveness of the application framework, the team developed several use cases. The first examples builds on custom-made modules written in the FRESCO scripting language. The script is able to identify port scans based on failed TCP connection attempts and automatically instructs the controller to redirect traffic to a honey pot. The second demonstration utilises legacy intrusion detection software (here: Bothunter) and reacts to its reports. As soon as Bothunter identifies an attack or malicious code and reports it to the controller, the FRESCO

script is triggered to activate an automated quarantine mechanisms on the affected host.

A similar concept is OrchSec [134], an architecture presented by the Fraunhofer Institute for Secure Information Technology. OrchSec is specifically designed to address several of the secure design principles of Kreutz et al., 2013 [8]. In order to realise the Diversity and Replication property, the research team has developed an orchestration application, which can be installed on any controller type and is able to coordinate multiple controllers. The orchestration layer exposes a northbound API and cooperates with network monitors (e.g. sFlow) to enhance the security functionality of the network. Generic security applications, which are controller agnostic and may utilise the network monitor and controller information, can be developed on top of the orchestration layer in order to provide a sophisticated intrusion detection functionality separated from the controller. In order to demonstrate the effectiveness of the approach, the team developed an application which automatically detects and prevents DNS amplification attacks based on controller messages and reported monitoring events.

While FRESCO and OrchSec do not implement any security applications and mechanism per se, they demonstrate a viable practice to simplify and enable the composition of extensive security applications. An approach that is possible due to the general layer abstraction of software-defined networks and a new opportunity and chance in SDN.

In addition to the research presented in the secure design principles, these efforts extend or enhance the security of OpenFlow. OF-RHM and NPoL demonstrate sophisticated, expanded methods to implement defence against attackers, Flowguard ensures a robust use of the distributed OpenFlow firewall and FRESCO and OrchSec ease deployment and interoperability of security applications for developers, a move which might facilitate the spread of open-source security in SDN.

4.4 Outlook on Software-Defined Middleboxes

The design principles of the research team do not specifically demand the use of extended security functionality, but recommend the use of added hardware as final suggestion [8]. Whether it is social engineering or the hijacking of BYOD devices, an attacker has many possibilities to find a way to access the internal network hosts. Although the controller may be able to detect attacks based on patterns in PACKET_IN messages as demonstrated in the preceding chapters, securing the entire network traffic and individual hosts poses a challenge. Redirecting all traffic through the controller itself consumes resources and stresses the control plane. The data plane on the other hand lacks the intelligence to detect and repel attacks, unless highly specialised modifications such as AVANT-GUARD (see Section 4.3.3) are implemented. Although firewalls are possible in the native OpenFlow design, the lack of inherent ability to perform intrusion detection or prevention, control-independent stateful firewalls, and deep-packet-inspection risks exposure to sophisticated attacks. Anomalous traffic and single packets must quickly be identified and eradicated to avert actual asset damage. As a consequence, at least one instance of middleboxes and UTM is yet still required to fully protect the software-defined networks. The SDN structure provides an opportunity to significantly amplify the reach and features of security appliances, as the controller can quickly enact or prepare reactive measures. A further advantage is the ability to construct security paths and redirect traffic to specialised inspection centres, thus avoiding the need to carefully consider the placement of middleboxes. [135] Depending on its properties and

tags, traffic could be carried dynamically through a chain of service boxes distributed in the network, instead of having to pass through a physically static demarcation point. [135]

This thesis demonstrated several methods where the controller can leverage the information of security software installed on generic servers, yet these approaches are experimental or still rely on the reports of traditional security hardware. Although a valid temporary solution for hybrid deployments, the use of classic middleboxes might violate the long-term intent of SDN to abolish proprietary and complicated devices in the network. [2]

An ongoing and promising project parallel to SDN is Network Function Virtualization (NFV) [136]. NFV aims to abolish the use of specialised hardware and attempts to integrate the middlebox functionality into commodity hardware in order to reduce costs and provide flexibility and customisability. The approach is complementary to SDN and both projects might eventually merge, as they intersect in principles and share common traits. Indeed, several solutions proposed in this thesis, e.g. the DDoS protection mechanisms utilising external intelligence of the Section Self-Healing Mechanisms (4.3.3), can be categorised into a type of NFV. In theory, a network utilising both SDN and NFV would instruct selected switches to reroute traffic to the NFV servers, which in turn perform deep-packet-inspection and intrusion detection and report events to the controller. The controller reacts according to the threat at hand and automatically reconfigures the network state. In the vision of NFV and SDN, the controller is able to create dynamic security paths by quickly modifying the flow of the switches to pass through inspection centres seated at any place in the network. As a result, the appliances can be freely moved in the infrastructure. The details of NFV, a research effort distinct from SDN, are not covered in this thesis, but offer insight into the future integration and the replacement of proprietary security hardware.

The following chapter summarises the research findings and solutions presented in this thesis and presents the final sketch of a secure SDN architecture. The design is matched against the general traditional architecture presented in Section 2.1 and compares drawbacks and advantages.

5 Summary and Evaluation

An integration of the security requirements and the secure design principles establishes a comprehensive security framework. If these concrete approaches are implemented as envisioned, the attack paths of the STRIDE threats may be sufficiently averted.
The security amendments of the ONF document **Principles & Practices for Securing Software-Defined Networks** ensure a reliable communication and secure data plane of the network. Authentication such as TLS enforces trustworthy communication and the guidelines for switches and controllers clearly build a minimal data plane baseline security.
The **Replication** principle ensures resistance to Denial of Service and software-faults, while building a basis for Diversity, Dynamic Switch Association, and Self-Healing Mechanisms.
Diversity, albeit an optional and possibly controversial principle, lowers the overall impact factor of a technical vulnerability and augments the protective capabilities of the overall system using the best possible applications.
Self-Healing Mechanisms enacted by the central intelligence are a promising feature of SDN and an inevitable necessity in the software-defined network. Utilising the Replication property, the various algorithms guarantee correctness of the network information base and flow table entries and verify the behaviour of controllers. If high security and control plane trustworthiness is required, the democratic approach and the automatic overriding of unusual decisions diminishes the influence of single rogue devices and prevents the abuse of Byzantine faults. As the controller is aware of the entire network state, fast-failure recovery is readily available over pre-installed backup flow entries, and reactive recomputing. Via the integration of detection applications, the controller itself is also capable of recognising Denial of Service attacks on the control plane and can install instant protective measures in the switches. Additionally, the controller can cooperate with specialised OpenFlow switch designs to shield the server from packet flooding.
The principle of **Dynamic Device Association** is closely linked to Self-Healing and Replication and covered by a rigorous specification of the ONF Switch protocol. However, providing the data plane with intelligence and verification mechanisms could violate the abstraction property of software-defined networks.
The trust principles are significant cornerstones of the secure network. **Trust between Devices and Controllers** reduces the threat of Tampering and Spoofing significantly and fosters the reliability of the network. The authenticity of single OpenFlow devices is largely dependent on the specifications of the OpenFlow protocol. If the rigour of the ONF Security Evaluation is applied and controllers are able to identify and isolate malfunctioning switches, Spoofing and Tampering in the data plane are effectively prevented. In the higher application and control plane, an advisable course of action is to apply operating system principles of access control, role-based authentication, and trust management. These measures limit the malicious extent of a single application and enable the categorisation into applications of varying authority. Furthermore, entities and clients may be held accountable for their actions.
Security Domains either protect and isolate the controller or separate the overall com-

pany into zones of varying trust. The controller is independent of its applications and their memory consumption and is thus unaffected by malicious or malfunctioning code. Furthermore, isolating the controller from the remaining network and only accepting connection from authenticated domains in the network limits the breadth of access possibilities for an attacker. The mechanics of AAA servers may be enhanced by cooperating with the central SDN controller.
Secure Components guarantee an optimal security base and increase the overall robustness of SDN. The internal components of a switch and controller should be checked for integrity and need to be resilient to overflows. Code reviews, protection, or cleaning algorithms, which warrant the dependability of the code and memory space, secure the network components from falling prey to simple Denial of Service and Elevation of Privilege abuse.
Fast and Reliable Software Update and Patching as a reactive approach to vulnerabilities protects the network from future attacks. The vulnerable controller has to be updated quickly without affecting the network and could either be swapped or segmented into independent exchangeable components. Similarly, the controller could deploy firmware updates on switches automatically from a central position in the network.
Lastly, the **Selected Additional Research Efforts** enhance the network security and complicate attack preparation such as reconnaissance or the infection of a suitable host while easing the deployment of security algorithms and applications in the network.

5.1 Constructing the Secure Software-Defined Network

Combining the recommendations of the Open Networking Foundation and the nine design principles with the implementation strategies presented in this thesis it is possible to sketch the design of a secure, large-scale software-defined network (see Figure 5.1).

The first and absolute prerequisite in this secure design is the use authenticity and integrity for any device communication in the network. Any and all communication between applications, controllers and switches is mutually authenticated while sensitive messages such as topology reports and modification messages checked for integrity. The database of the controller itself is signed to guarantee the use of intact server data. Optionally, the control channel may be deployed out-of-band either physically or virtually in VLAN configurations enacted by the controller. The requirement of authenticity is fundamental in SDN to ensure a minimal amount of security and protection of the control flow. To avoid dependency on a single device, at least two independent controllers should be deployed in the network, which may coordinate or take over neighbouring networks in case of a malfunction. For added security and to overrule malicious or defect controllers, every switch may connect to multiple logically centralised controllers. In this design, every domain should employ a minimum of three replicated controllers, which communicate directly or indirectly over a distributed network database, to minimise the threat of a compromised device. As diverse implementations are horizontally incompatible yet, all controllers conform to the same type. If different controller types are to be implemented in the network, proxy layers might support the distribution and interoperability of the devices, while also reducing the load on single controllers in the network. The control plane resides in a protected area, similar to a vital database in a conventional network. Only authenticated hosts, which are part of a physically and logically secured domain, are able to access and configure the servers. Any traffic which is not a switch PACKET_IN or OpenFlow communication message is filtered

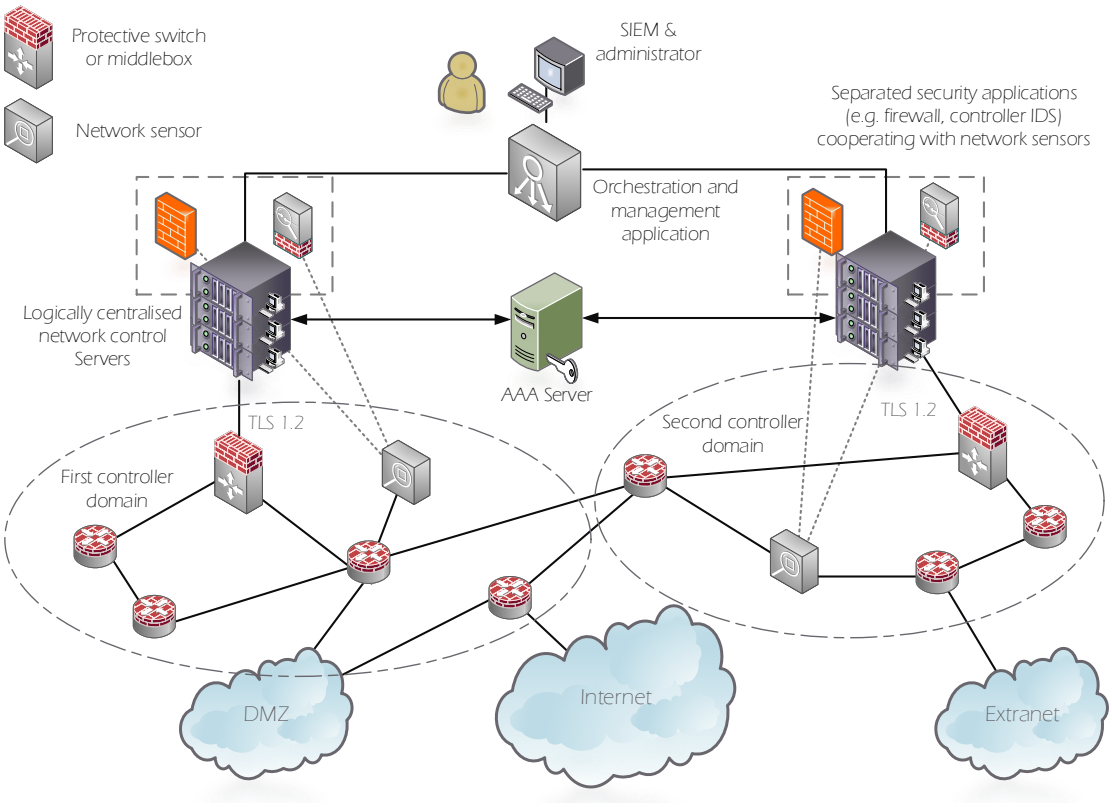

Figure 5.1: Sketch of a secure SDN design.

using the integrated flow table firewalls. Additionally, specialised DoS guard switches may shield the control centre from attacks. Albeit potentially costly, out-of-band access of management or security applications is recommended. Remote applications and hosts trying to access the server zone are verified based on location and identity using AAA servers and control algorithms. They are also limited in rights, network view and action scope. Security and latency-intensive applications may be packaged directly on the control server but are strictly executed in a separate process and memory space. Higher-privileged applications are able to override lower-tier decisions, with the administrator applications possessing complete configuration rights. Administrator applications report the state and log of all controllers and switches in the network and track actions of the single devices and applications. As middleboxes are yet irreplaceable, the control servers may be protected over intrusion detection or stateful firewall systems. Nevertheless, to quickly identify and resolve attacks in the entire network, switches can mirror traffic to selected inspection servers. The detection systems report the results to the controller, which swiftly reconfigures the network to isolate the affected sections. Additionally, the controller is capable of identifying suspicious network behaviour based on PACKET_IN patterns and reports any events and network anomalies to the management application or SIEM over the management interface.

In general, the network is blocked off from the extra- and internet and divided into varying protective zones by distributing a fine-grained firewall in the network and authenticating any new host in the network. However, firewalls might still be required, as the flow table space

of switches is limited and stateful firewalls are only functional via the control intelligence. Port scans could be thwarted using intrusion detection systems and virtual IP addresses to mask real internal communication. To guarantee longevity and the latest secure firmware, controller can be exchanged using the HotSwap methods or live-patched by replacing single modules.

5.2 Evaluation

After discovering security threats in the conventional SDN architecture, establishing basic secure design requirements and compiling necessary defence mechanisms, it is possible to assess the inherent and potential security of SDN in comparison to the standard network. Each of the STRIDE aspects is summarised in its current threat potential and the vulnerability of the data flow components is assessed.

5.2.1 Spoofing

Although new elements which may be spoofed are introduced, Spoofing is entirely preventable via authentication and access control mechanisms between every trust boundary. Considering the design postulations of the ONF Security Project, it can be assumed that a deployment-ready software-defined network will utilise TLS or similar authentication as basic countermeasure. Furthermore, SDN provides an opportunity in recognising and removing spoofed devices as the controller can identify irregularities in the network. The lack of application authenticity is a threat which is not considered by many developers, but functional countermeasures area already being developed by researchers. While attacks to spoof the host topology and redirect traffic are already present and published, they are preventable with the use of message integrity and various security applications installed in the control plane.

5.2.2 Tampering

Since the virtual network view and databases can be modified, Tampering is a greater risk in SDN. However, authentication and data integrity algorithms as demanded by the ONF [99] may protect the control traffic and data sufficiently. Additionally, democratic approaches in the control plane, as presented in the Section Self-Healing Mechanisms (see Section 4.3.3), may override the decisions of a mislead or malicious device. It is crucial to avoid dependence on a single control station in SDN, if security and network availability are valued. Lastly, harmful influence from applications and clients may be restricted using authorisation and role-based access control in the northbound interface while simultaneously isolating the controller from the underlying system and applications.

Tampering is a significant danger in environments such as SDN which relies on a virtual view and central database. A slight change substantially affects the network and this threat must be addressed accordingly. The mechanisms and design choices highlighted in this thesis are capable of shielding and verifying the database and policy consistency of SDN and can adequately defend the control and data plane from poisoned data.

5.2.3 Repudiation

This threat is not exacerbated in software-defined networks. Although, many new deniable actions of the autonomous applications and controllers emerge, an automatic and instant logging mechanism, unique identification of individual behaviour, and monitoring of control processes reduce the possibility to hide or deny malicious actions. Furthermore, the overview of the entire network state, traffic flow, and access control establishes a comprehensive tool set for attack forensics. These surveillance possibilities are an advantage of the new architecture.

5.2.4 Information Disclosure

Similar to Tampering, the risk of Information Disclosure gains new significance in SDN. The network relies on a central information base that stores ample data about topology, QoS policy, and forbidden areas and can be accessed over various interfaces and queries. Extracting this information, as demonstrated in Section 3.2.5, provides an attacker with substantial knowledge about assets, security appliances, and the location of sensitive data. However, if access to the control plane and channel is safely restricted, the sensitive information is moved out of reach. Relocating the controller into a secure and restricted zone is a core design choice. Access has to be limited to physical and authorised applications or administrative stations and the control channel should be secured using dedicated VLAN or out-of-band communication. These implementations are achievable and prevent the controlling software from unintentionally leaking information. Switches of the data plane may expose their flow tables over side-channel attacks, but the large amount of traffic needed to acquire this information is easily detectable by an integrated controller IDS (e.g. SPHINX). If the controller is secure and detached from the data network, it is also capable of reducing the transparency of the network by using dynamic proxy approaches such as OF-RHM (see Section 4.3.10).

In summary, Information Disclosure is preventable in the secure SDN design if the control channel and plane are appropriately guarded and entirely separated from the intranet and generic data traffic.

5.2.5 Denial of Service

Denial of Service is the main problem of SDN, as an attack on the control plane paralyses the entire network. The countermeasures presented in this thesis and the secure design, i.e. restricting the access to the controller, implementing a dedicated IDS, and building a protection ring, may shield the device but could be circumvented. Due to the focus on software and the control bottleneck, a multitude of possibilities arise to incapacitate a central switch or controller, ranging from simple flooding to the use of poison packets or malicious applications. The key to protection is isolation, replication of essential assets, and synchronisation, all of which assure fall-back guarantee and a sufficient degree of availability and reliability. The use of distributed data stores is problematic, as these may be susceptible to service failure or abuse. A currently prevalent problem is the limitation of the amount of flow table entries in OpenFlow hardware switches. SDN requires many specific entries on a single switch for fine-grained network management. Therefore, switches may operate on the brink of table exhaustion in large networks and thus become an easy target for attackers. Denial of Service as a threat is amplified in SDN and is still a ubiquitous risk in the secure design. Although it is possible to prevent most of the dangers with the methods

discussed in this thesis, these new measures might again introduce new vulnerabilities. The threat requires sophisticated protective measures and careful consideration in order to fully protect the sensitive controllers and the simple switches in the network and to guarantee high availability.

5.2.6 Elevation of Privilege

The last aspect is an entirely new factor, which has to be observed and studied when deploying the software-defined networks and component as a service. Due to the OS principle of SDN it is possible for unauthorised applications and clients to access the virtualised and shared network resources to enact configurations which they are not entitled to. Role- or permission based access control, verification, and separation mechanisms such as FlowVisor address these concerns. Nevertheless, breaking out of the virtualised, restricted box and traversing prohibited domains is a constant hazard when providing network services. Clients must not be trusted and have to be restricted and strictly monitored when accessing the public control plane. Research on SDN and NaaS is still in its infancy and solutions may risk neglect or miss security flaws during this development process. In order to prevent security leaks, the same rigour and meticulous process as in the development of current operation systems has to be applied to the controller and virtualisation deployment.

5.2.7 Concluding remarks

Summarising these thoughts, the secure SDN design is able to provide satisfactory security to meet the demands of a company network operator. Nonetheless, the security entirely depends on the choice of implementation and security mechanisms. The standard baseline OpenFlow network exposes many vulnerabilities and attack possibilities and is not a secure infrastructure. However, the current research and design concepts display extensive viable solutions and efforts to prevent these risks when adopting the new paradigm. It is mandatory for SDOs and control software developers to integrate security into the default deployment. Overall, despite all these security mechanisms, Denial of Service on the controller and Elevation of Privilege remain as prime threats and gain more significance when compared to conventional networks. New security software and solutions (e.g. a distributed database or misconfigured applications) might introduce new attack possibilities. As bugs or mistakes are always possible in development, the introduction of software and programmability also increases the intensity of the constant race between network offence and defence. Nevertheless, the benefits may outweigh the risks of the two threats, as centralisation and automation of administrative responsibilities lead to a quick and clean identification and mitigation of anomalies. New problems which have not been considered in this thesis are performance, latency and stability. Running many security solutions and applications, coordinating the distributed devices, and managing large network security paths could severely impact the controller and introduce unacceptable latency metrics, a risk which has to be calculated carefully.

6 Conclusions and Future Work

SDN and the vision of a future network architecture rapidly gain ground in both media and research, with the OpenFlow protocol representing a main component and fuel for this evolution. However, network operators are still sceptical, and their security concerns are warranted. This thesis has performed an extensive survey of security in software-defined networks and provided a concrete overview over general emerging problems and their solutions. The first chapter established the basic traditional architecture, its security framework and the division of the logical network into data, control and management plane. The motivation and history of SDN has been elucidated and the OpenFlow protocol has been identified as a main driver of SDN adoption by tracing the history of programmable networks. Subsequently, the thesis summarised and detailed the mechanisms of the SDN paradigm and the OpenFlow protocol in particular. Relevant controller designs were compiled and abstracted into design basics of SDN.

Chapter 3 utilised the technical framework of Chapter 2 and applied the STRIDE methodology of the Microsoft Security Development Lifecycle to the SDN architecture. Three data-flow-analyses of the general network, the controller and the switch were developed and examined. Based on these data-flow-diagrams, each of the STRIDE aspects has been summarised in an attack tree to visualise and demonstrate the security deficiencies of the software-defined network. For each threat, several attacks, security flaws and negligences in the protocol were found. Denial of Service and the lack of compulsory authentication emerged as the most crucial flaws of current SDN.

To solve the identified problems, the thesis presented two secure design approaches. It inspected the recently published security best practice guide of the ONF Security Project and presented the nine design principles proposed by Kreutz et al., 2013. While the requirement of the ONF Security Project may improve OpenFlow, a reliable protocol was found to be insufficient to ensure a secure by default SDN. Thus the nine design principles were evaluated critically and amended as necessary. The thesis compiled research and mapped security solutions to fulfil every design principle at best effort. Additionally, mechanisms and projects of interest were presented and an outlook on network function virtualisation and the use of generic middleboxes and network sensors in cooperation with SDN has been provided. On the foundation of the research findings and results of the preceding chapter, Chapter 5 constructed a template of a secure SDN design and compared its risks and opportunities to legacy networks. Denial of Service remains as a central point of concern while Elevation of Privilege on the controller emerges as new threat in the context of NaaS deployments. The secure design of SDN can adequately prevent and even improve defence against the dangers of Spoofing, Tampering, Repudiation, Information Disclosure and Denial of Service in the general network. Latency and performance of the network itself have not been considered in this thesis and might pose a limitation in the deployment of security. Overall, the concept of SDN itself may be a viable option for a company to supersede conventional architecture, if and only if security is given the appropriate amount of attention by all involved parties.

6.1 Future Work

The main findings of this thesis may be interesting for three future projects.

As the purpose of this thesis was to assess the state of security of SDN, it is desirable to test the full conditions and environment in deployment scenarios. A standard OpenFlow network could be installed over Mininet or real hardware to estimate the actual threat level of the encountered vulnerabilities and attacks. This approach is able to highlight the real extent of the identified security threats and could provide a more accurate and elaborate assessment. An approximation of the secure design, which has been constructed in this thesis, would be implemented to mitigate the vulnerabilities. The network is then examined based on the impact regarding performance and interoperability while evaluating the overall potential or deficiencies of the security mechanisms. The limitations of the measures will showcase upcoming design flaws or potential hurdles in the deployment of a high security software-defined network. As most security solutions are scattered and individual projects rely on proof-of-concept methods, which may or may not be feasible in the installation, a comprehensive security case study could potentially be of value.

Secondly, as a literature review of the integration of intrusion detection and comprehensive security solutions in cooperation with the control plane would have exceeded the scope of this thesis, an inspection of network function virtualisation and the cooperation with SDN based on security could be of further interest.

Furthermore, hybrid deployments and the emerging security risks in corresponding scenarios were not considered in this thesis. As many companies, research networks, and service providers will not install SDN in greenfield deployments and would have to migrate from conventional networks, an evaluation of possible hybrid security solutions and integration may provide better insight into the actual state of security in SDN.

List of Figures

List of Tables and Abbrevations

List of Common Abbreviations

AAA Authentication Authorization and Accounting

ACL Access Control List

API Application Programming Interface

BYOD Bring-Your-Own-Device

DDoS Distributed Denial of Service

DFD Data Flow Diagram

IaaS Infrastructure as a Service

IDS Intrusion Detection System

ONF Open Networking Foundation

SDN Software-Defined Networking

SDO Standard Organisation

SIEM Security Information and Event Management

TCAM Ternary Content-Addressable Memory

TLS Transport Layer Security

Bibliography

[1] P. Baran, "On Distributed Communications Networks," *IEEE Transactions on Communications Systems*, 1964.

[2] N. Feamster, J. Rexford, and E. Zegura, "The Road to SDN: An Intellectual History of Programmable Networks," *SIGCOMM Comput. Commun. Rev.*, 2014.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, 2008.

[4] Y. Jarraya, T. Madi, and M. Debbabi, "A Survey and a Layered Taxonomy of Software-Defined Networking," *IEEE Communications Surveys Tutorials*, 2014.

[5] K. Greene. (2009). MIT Technology Review 10: Software-defined networking, [Online]. Available: `http://www.technologyreview.com/article/412194/tr10-software-defined-networking` (visited on Jun. 24, 2015).

[6] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover Security Design Flaws Using The STRIDE Approach," *MSDN Magazine*, 2006. [Online]. Available: `http://web.archive.org/web/20150208212605/https://msdn.microsoft.com/en-us/magazine/cc163519.aspx` (visited on Feb. 8, 2015).

[7] B. Schneier, "Attack Trees," *Dobb's Journal*, 1999. [Online]. Available: `https://www.schneier.com/paper-attacktrees-ddj-ft.html` (visited on Jun. 24, 2015).

[8] D. Kreutz, F. M. Ramos, and P. Verissimo, "Towards Secure and Dependable Software-defined Networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'13, 2013.

[9] RAND Corporation. (2015). Paul Baran and the Origins of the Internet, [Online]. Available: `http://www.rand.org/about/history/baran.html` (visited on Jun. 24, 2015).

[10] Internet Hall of Fame. (2015). Inductee: Donald davies, [Online]. Available: `http://internethalloffame.org/inductees/donald-davies` (visited on Jun. 24, 2015).

[11] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A Clean Slate 4D Approach to Network Control and Management," *SIGCOMM Comput. Commun. Rev.*, 2005.

[12] D. Kreutz, F. M. V. Ramos, P. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A comprehensive survey," *CoRR*, 2014.

[13] J. Dong, *Network Protocols Handbook*, en, 4th ed. Javvin Technologies, 2007.

[14] *IEEE Standard for Local and Metropolitan Area Networks– Station and Media Access Control Connectivity Discovery*, 2009.

[15] N. Farrington, E. Rubow, and A. Vahdat, "Data Center Switch Architecture in the Age of Merchant Silicon," in *Proceedings of the 17th IEEE Symposium on High Performance Interconnects*, ser. HOTI'09, 2009.

[16] M. Rhodes-Ousley, *Information Security*, 2nd ed. Mcgraw-Hill Osborne Media, 2013.

[17] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, *Network Configuration Protocol (NETCONF)*, RFC 6241 (Proposed Standard), Internet Engineering Task Force, 2011.

[18] Open Networking Foundation. (2015). Member Listing, [Online]. Available: `https://www.opennetworking.org/our-members` (visited on Jun. 24, 2015).

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined Wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM'13, 2013.

[20] VMware. (2012). VMware to Acquire Nicira, [Online]. Available: `http://www.vmware.com/company/news/releases/vmw-nicira-07-23-12` (visited on Jun. 24, 2015).

[21] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet Impasse Through Virtualization," *Computer*, 2005.

[22] T. Benson, A. Akella, and D. Maltz, "Unraveling the Complexity of Network Management," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09, 2009.

[23] H. Kim, T. Benson, A. Akella, and N. Feamster, "The Evolution of Network Configuration: A Tale of Two Campuses," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC'11, 2011.

[24] J. Sherry and S. Ratnasamy, "A Survey of Enterprise Middlebox Deployments," EECS Department, University of California, Berkeley, Tech. Rep., 2012.

[25] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, 2013.

[26] T. Koponen, S. Shenker, H. Balakrishnan, N. Feamster, I. Ganichev, A. Ghodsi, P. B. Godfrey, N. McKeown, G. Parulkar, B. Raghavan, J. Rexford, S. Arianfar, and D. Kuptsov, "Architecting for Innovation," *SIGCOMM Comput. Commun. Rev.*, 2011.

[27] S. Shenker, M Casado, T. Koponen, N McKeown, *et al.*, "The Future of Networking, and the Past of Protocols," in *Proceedings of the 2011 Open Networking Summit*, Keynote, 2011.

[28] D. L. Tennenhouse and D. J. Wetherall, "Towards an Active Network Architecture," *SIGCOMM Comput. Commun. Rev.*, 1996.

[29] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, 2014.

[30] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, *Forwarding and Control Element Separation (ForCES) Protocol Specification*, RFC 5810 (Proposed Standard), Updated by RFCs 7121, 7391, Internet Engineering Task Force, 2010.

[31] J. Vasseur and J. L. Roux, *Path Computation Element (PCE) Communication Protocol (PCEP)*, RFC 5440 (Proposed Standard), Internet Engineering Task Force, 2009.

[32] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and Implementation of a Routing Control Platform," in *Proceedings of the 2^{nd} Conference on Symposium on Networked Systems Design & Implementation*, ser. NSDI'05, 2005.

[33] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A Protection Architecture for Enterprise Networks," in *Proceedings of the 15^{th} Conference on USENIX Security Symposium*, ser. USENIX-SS'06, 2006.

[34] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," in *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM'07, 2007.

[35] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, *Software-Defined Networking (SDN): Layers and Architecture Terminology*, RFC 7426 (Informational), Internet Engineering Task Force, 2015.

[36] *Technical Recommendation: SDN Architecture*, 1st ed., Open Networking Foundation, 2014.

[37] ITU-T Study Group 13, *Framework of software-defined networking*, ITU-T Y.3300 (Recommendation), 2014.

[38] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A survey," *IEEE Communications Surveys Tutorials*, 2014.

[39] IRTF. (2013). IRTF Software-Defined Networking Research Group (SDNRG), [Online]. Available: `http://irtf.org/sdnrg` (visited on Jun. 24, 2015).

[40] S. Raza and D. Lenrow. (2013). ONF Charter: Northbound Interfaces, [Online]. Available: `https://www.opennetworking.org/news-and-events/press-releases/1182` (visited on Jun. 24, 2015).

[41] J. Metzler. (2014). Where Do We Stand with SDN's Northbound Interface? [Online]. Available: `http://www.webtorials.com/content/2014/04/where-do-we-stand-with-sdns-northbound-interface.html` (visited on Jun. 24, 2015).

[42] *Technical Specification: OpenFlow Specification 1.5.0*, 0x06, Open Networking Foundation, 2015.

[43] *Technical Specification: OpenFlow Management and Configuration Protocol 1.2*, Open Networking Foundation, 2014.

[44] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically Centralized?: State Distribution Trade-offs in Software Defined Networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12, 2012.

[45] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M.-n. Casado, N. McKeown, and S. Shenker, "Nox: Towards an Operating System for Networks," *SIGCOMM Comput. Commun. Rev.*, 2008.

[46] (2012). NOX Repo, [Online]. Available: `https://github.com/noxrepo` (visited on Jun. 24, 2015).

[47] *OpenDayLight Developer Guide*, OpenDayLight, 2014.

[48] *User Documentation: The Floodlight Controller*, Project Floodlight, 2015. [Online]. Available: `https://floodlight.atlassian.net/wiki/display/floodlightcontroller/The+Controller` (visited on Jun. 24, 2015).

[49] R. G. Little. (2013). ONF to standardize northbound API for SDN applications? [Online]. Available: `http://searchsdn.techtarget.com/news/2240206604/ONF-to-standardize-northbound-API-for-SDN-applications` (visited on Jun. 24, 2015).

[50] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "Onos: Towards an Open, Distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'14, 2014.

[51] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou, "Feature-based Comparison of Software Defined Networking (SDN) Controllers," in *Proceedings of the International Conference on Computer Software and Applications*, ser. ICSCA 2014, 2014.

[52] RYU Project Team, *Ryu sdn framework: Using Openflow 1.3*, 1st ed., RYU, 2014.

[53] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A Distributed Control Platform for Large-scale Production Networks," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10, 2010.

[54] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for OpenFlow," in *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, ser. INM/WREN'10, 2010.

[55] R. Sherwood, G. Gibb, K. Yap, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A Network Virtualization Layer," OpenFlow Switch Consortium, Tech. Rep., 2009.

[56] P. Porras, S. Cheung, M. Fong, K. Skinner, and V. Yegneswaran, "Securing the Software-Defined Network Control Layer," in *Proceedings of the 2015 Network and Distributed System Security Symposium*, ser. NDSS'15, 2015.

[57] Y. Weingarten, N. Sprecher, E. Bellagamba, and T. Mizrahi, *An Overview of Operations, Administration, and Maintenance (OAM) Tools*, Informational, Internet Engineering Task Force, 2014.

[58] *Ovs-dpctl - administer Open vSwitch datapaths*, Ubuntu Manuals, 2008. [Online]. Available: `http://manpages.ubuntu.com/manpages/natty/en/man8/ovs-dpctl.8.html` (visited on Jun. 24, 2015).

[59] S. Scott-Hayward, G. O'Callaghan, and S. Sezer, "Sdn Security: A survey," in *Proceedings of Future Networks and Services*, ser. SDN4FNS, 2013.

[60] K. Benton, L. J. Camp, and C. Small, "OpenFlow Vulnerability Assessment," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'13, 2013.

[61] D. an Romão, N. van Dijkhuizen, S. Konstantaras, and G. Thessalonikefs, "SSN Project Report: Practical Security Analysis of Openflow," *SSN Project Report*, 2013.

[62] M. Brandt, R. Khondoker, R. Marx, and K. Bayarou, "Security analysis of software defined networking protocols - OpenFlow, OF-Config and OVSDB," in *Proceedings of the Fifth IEEE International Conference on Communications and Electronics*, ser. ICCE'14, 2014.

[63] R. Klöti, "Openflow: A security analysis," Master's thesis, Eidgenössische Technische Hochschule Zürich, 2013.

[64] J. Saltzer and M. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, 1975.

[65] A. Shostack, *Threat Modeling: DESIGNING for Security*, 1 edition. Wiley, 2014.

[66] P. Mell, K. Scarfone, and S. Romanosky, *A Complete Guide to the Common Vulnerability Scoring System Version 2.0*, NIST and Carnegie Mellon University, 2007.

[67] O. El Ariss, J. Wu, and D. Xu, "Towards an Enhanced Design Level Security: Integrating Attack Trees with Statecharts," in *Proceedings of the Fifth International Conference on Secure Software Integration and Reliability Improvement*, ser. SSIRI'11, 2011.

[68] R. Rietz, A. Brinner, and R. Cwalinsky, "Improving Network Security in Virtualized Environments with OpenFlow," in *Proceedings of the International Conference on Networked Systems*, ser. NETSYS, 2015.

[69] M. Antikainen, T. Aura, and M. Saerelae, "Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch," in *Secure IT Systems*, ser. Lecture Notes in Computer Science, vol. 8788, 2014.

[70] R. Izard. (2015). Does Floodlight Support SSL Connection, [Online]. Available: `https://groups.google.com/a/openflowhub.org/d/msg/floodlight-dev/2ZS0XHQPQzs/QVJwpSjzmgYJ` (visited on Jun. 24, 2015).

[71] Trustwave, "2014 Global Security Report," Trustwave Holdings, Tech. Rep., 2014.

[72] B. Möller, T. Duong, and K. Kotowicz, "This poodle bites: Exploiting The SSL 3.0 Fallback," Google, Tech. Rep., 2014.

[73] G. Pickett, "Abusing Software Defined Networks," in *Proceedings of the 22st DEF CON Conference*, ser. DEFCON 22, Whitepaper, 2014.

[74] *OpenDayLight Controller: MD-SAL:Architecture: Clustering*, OpenDaylight, 2015. [Online]. Available: `https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Architecture:Clustering` (visited on Jun. 24, 2015).

[75] Trustwave, "2012 Global Security Report," Trustwave Holdings, Tech. Rep., 2012.

[76] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers in Cloud," *IEEE Transactions on Cloud Computing*, 2014.

[77] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu, "A Security Enforcement Kernel for OpenFlow Networks," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12, 2012.

[78] S. Hong, L. Xu, H. Wang, and G. Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures," in *Proceedings of 2015 Annual Network and Distributed System Security Symposium*, ser. NDSS'15, 2015.

[79] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, and B. B. Kang, "Rosemary: A Robust, Secure, and High-performance Network Operating System," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'14, 2014.

[80] X. Wen, Y. Chen, C. Hu, C. Shi, and Y. Wang, "Towards a Secure Controller Platform for Openflow Applications," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'13, 2013.

[81] M. Roe, "Cryptography and evidence," University of Cambridge, Computer Laboratory, Tech. Rep., 2010.

[82] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, "Let SDN be your eyes: Secure forensics in data center networks," in *Proceedings of the 2014 NDSS Workshop on Security of Emerging Network Technologies*, ser. SENT14, 2014.

[83] S. Scott-Hayward, C. Kane, and S. Sezer, "Operationcheckpoint: Sdn application control," in *Proceedings of the 22nd International Conference on Network Protocols*, ser. ICNP 22, 2014.

[84] J. François and O. Festor, "Anomaly Traceback using Software Defined Networking," in *International Workshop on Information Forensics and Security*, ser. WIFS'14, 2014.

[85] *Security Advisories*, OpenDaylight, 2015. [Online]. Available: `https://wiki.opendaylight.org/view/Security_Advisories` (visited on Jun. 24, 2015).

[86] S. Shin and G. Gu, "Attacking Software-defined Networks: A First Feasibility Study," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'13, 2013.

[87] Open Networking Foundation, "SDN Security Considerations in the Data Center," *SDN Solution Briefs*, 2013.

[88] M. Kobayashi, S. Seetharaman, G. Parulkar, G. Appenzeller, J. Little, J. Van Reijendam, P. Weissmann, and N. Mckeown, "Maturing of OpenFlow and Software-defined Networking Through Deployments," *Comput. Netw.*, 2014.

[89] M. Kuzniar, P. Peresini, and D. Kostic, "What You Need to Know About SDN Flow Tables," *Lecture Notes in Computer Science (LNCS)*, 2015.

[90] H. Nguyen Tri and K. Kim, "Assessing the impact of resource attack in Software Defined Network," in *Proceedings of the International Conference on Information Networking*, ser. ICOIN, 2015.

[91] B. Lantz, B. Heller, and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-defined Networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, ser. Hotnets-IX, 2010.

[92] J. Dover, "A denial of service attack against the Open Floodlight SDN controller," Dover Networks LLC, Tech. Rep., 2013, Whitepaper.

[93] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced Study of SDN/OpenFlow Controllers," in *Proceedings of the 9th Central and Eastern European Software Engineering Conference in Russia*, ser. CEE-SECR'13, 2013.

[94] J. Hart, "CVE-2015-1166: Exceptions thrown while deserializing truncated or malformed packets," ONOS, Tech. Rep., 2015. [Online]. Available: `https://jira.onosproject.org/browse/ONOS-605` (visited on Jun. 24, 2015).

[95] J. Dover, "A switch table vulnerability in the Open Floodlight SDN controller," Dover Networks LLC, Tech. Rep., 2014, Whitepaper.

[96] Q. Duan, C. Wang, and X. Li, "End-to-End Service Delivery with QoS Guarantee in Software Defined Networks," *CoRR*, 2015.

[97] V. Costa and L. Costa, "Vulnerability Study of FlowVisor-based Virtualized Network Environments," in *Proceedings of the Second Workshop on Network Virtualization and Intelligence for the Future Internet*, ser. WNetVirt'13, 2013.

[98] W. You, K. Qian, X. He, Y. Qian, and L. Tao, "Towards Security in Virtualization of SDN," in *Proceedings of the International Conference on Computer Communications and Networks Security*, ser. ICCCNS'14, 2014.

[99] Project Security, "Principles & Practices for Securing Software-Defined Networks applied to OFv1.3.4 Ver 1.0," Open Networking Foundation, Tech. Rep., 2014.

[100] *Technical Specification: OpenFlow Specification 1.3.5*, 0x04, Open Networking Foundation, 2015.

[101] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain SDN controllers," in *Proceedings of the Network Operations and Management Symposium*, ser. NOMS'14, 2014.

[102] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12, 2012.

[103] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed sdn controller," in *Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS'14, 2014.

[104] E. S. Roberto Doriguzzi Corin, "Netide: Removing vendor lock-in in SDN," in *1st IEEE Conference on Network Softwarization*, ser. NetSoft'15, 2015.

[105] P. Hubbard. (2013). Bucking SDN controller interoperability will stifle innovation, [Online]. Available: `http://searchsdn.techtarget.com/opinion/Bucking-SDN-controller-interoperability-will-stifle-innovation` (visited on Jun. 24, 2015).

[106] T. Tsou, P. Aranda, H. Xie, R. Sidi, H. Yin, and D. Lopez. (2012). Sdni: A Message Exchange Protocol for Software Defined Networks (SDNs) across Multiple Domains, [Online]. Available: `https://tools.ietf.org/html/draft-yin-sdn-sdni-00` (visited on Jun. 24, 2015).

[107] P. Joshi. (2015). Roadmap 2015: Distributed Core, [Online]. Available: `https://wiki.onosproject.org/display/ONOS/Roadmap2015:DistributedCore` (visited on Jun. 24, 2015).

[108] X. Jin, J. Gossels, J. Rexford, and D. Walker, "CoVisor: A Compositional Hypervisor for Software-Defined Networks," in *12th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'15, 2015.

[109] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, "Openflow: Meeting carrier-grade recovery requirements," *Computer Communications*, 2013.

[110] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, "Fast Recovery in Software-Defined Networks," in *Proceedings of the Third European Workshop on Software Defined Networks*, ser. EWSDN, 2014.

[111] S. Matsumoto, S. Hitz, and A. Perrig, "Fleet: Defending SDNs from Malicious Administrators," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'14, 2014.

[112] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "Avant-guard: Scalable and Vigilant Switch Flow Management in Software-defined Networks," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS'13, 2013.

[113] M. Ambrosin, M. Conti, F. D. Gaspari, and R. Poovendran, "Lineswitch: Efficiently Managing Switch Flow in Software-Defined Networking while Effectively Tackling DoS Attacks," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS'15, 2015.

[114] I. Mihai-Gabriel and P. Victor-Valeriu, "Achieving DDoS resiliency in a software defined network by intelligent risk assessment based on neural networks and danger theory," in *Proceedings of the 15th International Symposium on Computational Intelligence and Informatics*, ser. CINTI 15, 2014.

[115] D. Gkounis, V. Kotronis, and X. Dimitropoulos, "Towards Defeating the Crossfire Attack using SDN," *CoRR*, 2014.

[116] B. Wang, Y. Zheng, W. Lou, and Y. Hou, "DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking," in *Proceedings of the 22nd International Conference on Network Protocols*, ser. ICNP 22, 2014.

[117] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments," *Comput. Netw.*, 2014.

[118] J. C. Mogul and P. Congdon, "Hey, You Darned Counters!: Get off My ASIC!" In *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12, 2012.

[119] L. Schiff, M. Borokhovich, and S. Schmid, "Reclaiming the Brain: Useful OpenFlow Functions in the Data Plane," in *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, ser. HotNets-13, 2014.

[120] P. Porras, S. Shin, M. Fong, and C. Yoon. (2015). Security-mode ONOS, ONOS, [Online]. Available: `https://wiki.onosproject.org/display/ONOS/Security-Mode+ONOS` (visited on Jun. 24, 2015).

[121] B. Chandrasekaran and T. Benson, "Tolerating SDN Application Failures with LegoSDN," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'14, 2014.

[122] D. M. F. Mattos, L. H. G. Ferraz, and D. O. C. M. Bandeira, "AuthFlow: Authentication and Access Control Mechanism for Software Defined Networking," Universidade Federal do Rio de Janeiro, Tech. Rep., 2014.

[123] M. Dhawan, R. Poddar, K. Mahajan, and V. Mann, "Sphinx: Detecting security attacks in software-defined networks," in *Proceedings of 2015 Annual Network and Distributed System Security Symposium*, ser. NDSS'15, 2015.

[124] L. Vanbever, J. Reich, T. Benson, N. Foster, and J. Rexford, "Hotswap: Correct and Efficient Controller Upgrades for Software-defined Networks," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'13, 2013.

[125] J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, W.-F. Wong, Y. Zibin, M. D. Ernst, and M. Rinard, "Automatically Patching Errors in Deployed Software," in *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*, ser. SOSP'09, 2009.

[126] J. Poimboeuf, "Live Kernel Patching Update," Red Hat Enterprise Linux, Tech. Rep., 2015. [Online]. Available: `http://rhelblog.redhat.com/2015/03/23/live-kernel-patching-update/` (visited on Jun. 24, 2015).

[127] Y. Zhang, N. Beheshti, and R. Manghirmalani, "Netrevert: Rollback Recovery in SDN," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'14, 2014.

[128] P. Wette, M. Draxler, A. Schwabe, F. Wallaschek, M. Zahraee, and H. Karl, "Maxinet: Distributed emulation of software-defined networks," in *Proceedings of the 13th Networking Conference*, ser. IFIP'14, 2014.

[129] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow Random Host Mutation: Transparent moving target defense using software defined networking," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, ser. HotSDN'12, 2012.

[130] R. Bifulco and G. Karame, "Towards a Richer Set of Services in Software-Defined Networks," in *Proceedings of the 2014 NDSS Workshop on Security of Emerging Network Technologies*, ser. SENT'14, 2014.

[131] A. Pani, "Scalable stateful firewall design in openflow based networks," pat. US8789135 B1, U.S. Classification 726/1, 726/25, 726/24, 726/11, 370/235, 726/23, 726/13, 726/12, 370/229; International Classification G06F17/00; Cooperative Classification H04L63/02, 2014.

[132] H. Hu, W. Han, G.-J. Ahn, and Z. Zhao, "Flowguard: Building Robust Firewalls for Software-defined Networks," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN'14, 2014.

[133] S. Shin, P. Porras, V. Yegneswaran, M. Fong, G. Gu, and M. Tyson, "Fresco: Modular Composable Security Services for Software-Defined Networks," in *Proceedings of the 20th Annual Network and Distributed System Security Symposium*, ser. NDSS'13, 2013.

[134] A. Zaalouk, R. Khondoker, R. Marx, and K. Bayarou, "Orchsec: An orchestrator-based architecture for enhancing network-security using Network Monitoring and SDN Control functions," in *Proceedings of the Network Operations and Management Symposium*, ser. NOMS'14, 2014.

[135] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying Middlebox Policy Enforcement Using SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM'13, 2013.

[136] European Telecommunications Standards Institute, "Network Functions Virtualisation," in *Proceedings of the SDN and Openflow World Congress 2012*, 2012.