UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

SECURE MONITORING FOR A SECURE SMART GRID

Rodrigo Barão de Medeiros

Dissertação orientada pelo Prof. Doutor Fernando Manuel Valente Ramos
e co-orientada pelo Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Arquitectura, Sistemas e Redes de Computadores

2015

Acknowledgments

I am very grateful to many for the support, insights, and influence that have led to this completed work.

To my colleagues and administrators in the Informatics Department here at The University of Lisbon: thank you for your support and flexibility as I completed my coursework, and then researched and wrote this thesis. I couldn't have done it without you behind me.

To my instructors in this work: thank you for helping to form my thinking throughout the program as I approached this capstone research project; and thank you especially to Dr. Fernando V. Ramos and Dr. Nuno Neves, for your time and willingness to guide me, for your knowledge, and for your valuable insights that helped me fulfil this work.

To my friend and colleague Ricardo Fonseca, thank you for your knowledge as well for the full support on the way of this work.

To my classmates in the program, Radu and Andre: I truly believe that learning happens in community, and your presence in the classroom with me as we attempted to progress was essential. Thank you.

To my colleagues in this masters, Bernardo, Miguel, Tiago, Faneca, thank you for the great and productive discussions and for the amazing time we spent together.

To my family: I wouldn't be the scholar I am today without your encouragement; thank you for giving me the confidence to pursue what fulfils me and for always making my work feel worthwhile.

And, to Soraia: thank you for being my reason for sitting down at the computer and cranking this work out.

Funding This work was partially supported by the EC through project FP7 SEGRID (607109), by national funds of Fundação para a Ciência e a Tecnologia (FCT) through project UID/CEC/00408/2013 (LaSIGE).

Dedico este Mestrado à minha familia, e a ti.

Resumo

O crescimento das redes e a necessidade de responder à procura exigida pelo maior número de aplicações e a concomitante utilização de dados, levam a que a monitorização desempenhe um papel fundamental não só para os sistemas de redes actuais mas também na resposta a um crescimento futuro. O sistema de monitorização é uma componente crucial numa rede, suportando muitas funções essenciais como engenharia de tráfego, detecção de anomalias e diagnóstico de desempenho. Um dos requisitos principais para estas soluções avançadas na gestão da rede é a necessidade de precisão na monitorização. Contudo, as técnicas tradicionais de monitorização não estão preparadas para responder a estas necessidades. Um exemplo disso é o SNMP, o protocolo de gestão e monitorização da rede mais usado. O SNMP permite que pedidos sejam feitos aos switches para obter contadores por porto e por interface, e obter estatísticas gerais dos nós da rede. O problema é que muitos dos *switches* estão limitados aos contadores que agregam o tráfego para todo o *switch* e para as suas interfaces. Por consequência o SNMP não permite obter estatísticas a uma granularidade maior, por *flow*, um requisito de muitas aplicações modernas, além de ter problemas de escalabilidade. Protocolos mais recentes, como o Netflow, resolvem o problema de escalabilidade mas as técnicas de amostragem utilizadas trazem consigo outras limitações.

As redes definidas por software (Software Defined Networks) têm sido propostas como solução para alguns destes problemas. Numa SDN, o plano de controlo é separado do plano de encaminhamento, centralizando-se a lógica de controlo da rede num controlador que corre num (*cluster* de) servidor(es). Para tal ser possível, é necessário adicionar-se uma camada de comunicação entre o controlador e os dispositivos, algo feito tradicionalmente através do protocolo OpenFlow. Este protocolo de comunicação permite ao controlador acesso remoto para gestão das tabelas de encaminhamentos dos dispositivos de rede. Este desacoplamento possibilita a centralização lógica do controlo, oferecendo ao controlador uma visão global da rede. Com este novo paradigma de redes surgiu um conjunto avançado de primitivas de monitorização mais sofisticadas, que respondem aos requisitos impostos pelas redes de hoje. Os *switches* OpenFlow mantêm estatísticas de tráfego que podem ser recolhidas pelo controlador SDN a pedido. O controlador pode ainda injectar pacotes na rede, tornando assim possível empregar técnicas de monitorização activa e passiva.

Apesar da sua importância como elemento fundamental da infra-estrutura SDN, nenhuma das soluções até agora propostas inclui a segurança como requisito, o que as torna vulneráveis a um conjunto extenso de ataques, inclusive pouco sofisticados. Acreditamos que tais primitivas devem ser resilientes de raíz, assegurando que as medições permaneçam correctas mesmo sob ataque.

Esta tese centraliza-se na inclusão da segurança na construção de novas ferramentas de monitorização da rede. Para demonstrar tal necessidade, e justificar a preocupação, realizamos uma avaliação das limitações das técnicas de monitorização comuns demonstrando experimentalmente que elas são vulneráveis a ataques. Para tal, utilizamos o OpenNetMon, um monitor SDN *open-source*, como alvo principal dos ataques. Apesar de termos usado o OpenNetMon, alguns ataques são mais genéricos, sendo portanto extensíveis a outras plataformas SDN de monitorização.

O foco dos ataques foi sobre as duas métricas de uso mais comum para operação e gestão da rede: atraso na rede e taxa de transmissão. Estes ataques foram realizados numa plataforma física e numa virtual. Para os testes na plataforma virtual foi utilizado o emulador Mininet. Para os testes físicos criámos uma *testbed* composta por *switches* em hardware da Pica8, com suporte Openflow, e múltiplas maquinas (para o controlador SDN e para os múltiplos *hosts*).

Finalmente, discutimos o impacto que estes ataques podem ter em sistemas críticos. Mais precisamente, usámos uma Smart Grid como estudo de caso. As Smart Grids distinguem-se dos sistemas eléctricos actuais pela sua capacidade muito mais sofisticada na monitorização e controlo da rede. Dado a Smart Grid ser um sistema crítico, discutimos algumas soluções de monitorização segura para este tipo de infraestrutura.

Palavras-chave: Rede, Monitorização, Ataques, Smart Grid, Segurança em SDN

Abstract

Monitoring plays a fundamental role in current network deployments, supporting diverse activities such as traffic engineering, anomaly detection, and performance diagnosis. The Software Defined Networks - a new paradigm in networking - has become an enabler for precise monitoring. In SDN the control plane is separated from the forwarding plane, leading to the logical centralization of the network control in a controller that runs in a (cluster of) server(s). For this purpose, a layer of communication is added between the controller and devices, something traditionally done through the OpenFlow protocol. This communication protocol allows the controller to have remote access to the forwarding tables of network switches. With the advent of SDN an array of advanced monitoring primitives has emerged, exploring the centralized vantage point offered by the controller. Such primitives should be resilient from the ground-up, ensuring a correct view under attack. In this work we intend to demonstrate that security should be a first class citizen when building SDN network monitoring frameworks. To justify this need, we perform a threat assessment on common monitoring techniques and demonstrate experimentally that they are vulnerable to attacks, including relatively unsophisticated ones. This indicates that further work is needed in this area and, with that aim, we include an initial discussion on possible solutions for secure monitoring. We discuss the impact of these attacks on physical systems, more precisely we use a Smart Grid as a study case. Smart Grids differ from the traditional electric system by having an intelligent monitoring capability and network control. As a Smart Grid is a critical system, we discuss several solutions to make the monitoring system secure.

Keywords: Network, Monitoring, Attacks, Smart Grid, SDN security

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, huge demands are placed on data networks due to the large abundance of high bandwidth usage applications. The challenge is substantial, as current networks are difficult to expand and hard to manage. To meet the level of service requirements, network operators have traditionally opted for overprovisioning. Unsatisfied with such inefficient solution, companies like Google [18] and Microsoft [17] are leveraging on the SDN paradigm [21] to increase the efficiency of their networks, achieving significant cost savings. One of the key requirements for these advanced management solutions is the need for accurate network monitoring. Google's B4 [18], for instance, requires traffic demand to be continuously measured, in order for its traffic engineering application to enforce the necessary bandwidth limits at the edge. For similar purposes, Microsoft's SWAN [17] system includes network agents that collect and report information about traffic at a flow-level granularity.

Today's electrical grid, alike data networks, seems unfit to deal with the increasing demands. The complex challenges faced by this infrastructure are driving the evolution of smart grid technologies. These smart grids have as core enabler an advanced communication network, in which traffic engineering and monitoring are essential.

These use cases are paradigmatic examples of the need for a measurement infrastructure that is agile to cope with the dynamics of networks and its traffic requirements. They are also demonstrative of the finer monitoring granularity these new wide-area and data center applications require [23]. Unfortunately, traditional monitoring techniques are not fit for this challenge. SNMP, the most common protocol for network monitoring, is too coarse-grained. NetFlow is more fine-grained, allowing measurement at the flow level, and scales better by using sampling approaches. The drawback is that this solution is ex-

pensive by requiring dedicated hardware and specialized algorithms. In addition, sampled information may lead to inaccuracies – for instance, Netflow has been shown to be insufficient for anomaly detection [24]. More recent work [22] proposes to instrument switches with hash-based primitives to increase measurement accuracy, but these methods require hardware modifications that may not be available in regular switches soon.

The fine-grain visibility of network traffic offered by SDN is seen as an enabler for the development of sophisticated network monitoring techniques, which fulfill the requirements of today's network environments. The interfaces offered by OpenFlow-based switches are an important facilitator for this task. OpenFlow switches maintain traffic statistics that can be collected by the SDN controller by simple querying. The controller can also inject packets into the network, making it possible to employ both active and passive monitoring techniques.

With the take-up of SDN by the industry [31], the networking community has indeed started to explore the use of this technology for advanced network monitoring. Several tools rooted in SDN principles have been proposed recently (see Chapter 2). These include frameworks to reduce monitoring latency (e.g., Planck [32]) and to improve SDN-based monitoring (e.g., OpenSketch [41]), and also network monitors such as OpenNetMon [37] and SLAM [39].

Despite their importance as a core element of the infrastructure, none of the network monitoring solutions proposed thus far considers security in their design. They are therefore vulnerable to an array of attacks, including relatively unsophisticated ones.

1.1 From the Power Grid to a Smart Grid

The power grid is a centralized unidirectional system of electric power transmission, electricity distribution, and demand-driven control. The current architecture of the power grid poses problems by not allowing a significant portion of our energy needs to be generated through renewable sources. Having been designed for predictable power sources such as coal, natural gas, and nuclear power plants, it is not able to accommodate the high penetration of intermittent sources, such as renewable, without losing stability. The maximum amount of intermittent sources that can be utilized is estimated to be about 20% to 25% of the total demand using established control methods [14]. This poses a fundamental challenge to the integration and penetration of renewable sources in the future. Furthermore, the distribution system is designed for one-way power flow (from central power plants

to distributed loads). The introduction of a large number of distributed sources, such as photovoltaic cells on residential roofs, is not easily manageable and increases stability liabilities in the operation of the grid.

In order to respond to these needs, developments are required on intelligent monitoring and network control, to make the electrical grid cope with the challenge. The resulting energy system is usually known as Smart Grid. A smart grid can repair itself, ensures a consistent and premium-quality power supply that resists to power leakages and can be operated more efficiently. This grid enables the decentralization of power generation, allowing the individual user to generate on-site power by employing any appropriate method. The main enabler for a smart grid is the inclusion of advanced communication networks in the electric network. In this context, a strong enabler is Software Defined Networks with their advanced monitoring platforms.

1.2 From a conventional network to a Software Defined Network

Smart grids bring modern communication technology to electrical networks, but traditional networks are not without its problems. In current networks the control and forwarding planes are bundled together in the network devices, as seen in the top of Figure 1.1. This makes management difficult, with each device having to be manually configured and managed using low-level tools. The wide variety of protocols leads to very complex networks, a complexity that could complicate the management of the smart grid.

An interesting solution to this problem are software defined networks (SDN). SDN separates both planes as shown in the bottom part of Figure 1.1, placing control plane functionality in a logically centralized SDN controller. The SDN controller is a (cluster of) server(s), running software that monitors and controls the network behaviour. This allows network administrators to manage network devices (routers or switches) using software that runs as an application on the SDN controller. SDN enables networks to become programmable [31]. To enable this separation the main method used for communication between these layers is Openflow [26] [1].

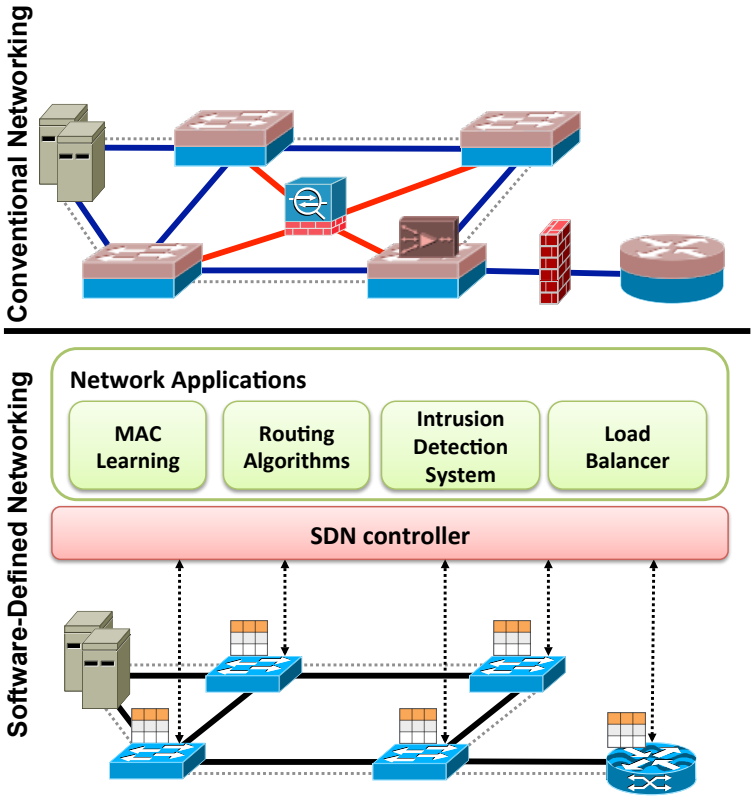

Figure 1.1: Traditional networking versus Software-Defined Networking (SDN). With SDN, management becomes simpler and middle boxes services can be delivered as SDN controller applications [21].

1.3 Motivation

Today's networks can be overwhelming in their complexity. The challenge of knowing the complexity and structure of a network and being able to accommodate the information on how all the individual elements are performing at a certain time (with accurate and correct measurements)is huge, but is also a key factor in maintaining the performance and integrity of the network as a whole.

Monitoring plays a fundamental role in current network deployments, to face this challenge. Software Defined Networks are an exception to this rule. The problem to this rule is that monitoring solutions are not secure. This problem is particularly acute in critical systems as Smart Grids. This thesis addresses this problem, taking the Smart Grid as a use case.

1.4 Contributions

The main contributions of our work are:

1. We demonstrate a series of vulnerabilities in SDN-based monitoring solutions.
2. We evaluate all the attacks using *both* an emulated and a physical platform.
3. Since this work was developed in the context of a critical infrastructure – the smart grid –, we discuss several solutions to secure the network monitor of a smart grid.

We used OpenNetMon [37] as target for the attacks we demonstrate in this thesis. We have chosen this platform for two main reasons. First, because it monitors several metrics at per-flow granularity – throughput, packet loss, and delay – whereas other proposals focus on a single metric. Second, because it is available open-source. Although we have used OpenNetMon, several attacks are generic and will succeed against recent proposals. For instance, recently proposed SLAM [39] is an example of a system that uses the same techniques as OpenNetMon for measuring particular network metrics.

1.5 Planning

In the beginning of this work we have devised the following work plan, with six phases:

1. Study of the state of the art regarding smart grids and SDN.
2. Definition of the methodology and evaluation plan. This include setting up Open vSwitch in a smart grid aggregator (or a feasible alternative) and controlling it using an SDN controller (e.g., by testing a simple firewall application).
3. Devise algorithms to prevent Denial of Service attacks to the smart grid.
4. Implementation of the algorithms as an application in the SDN controller.
5. System evaluation.
6. Writing the thesis.

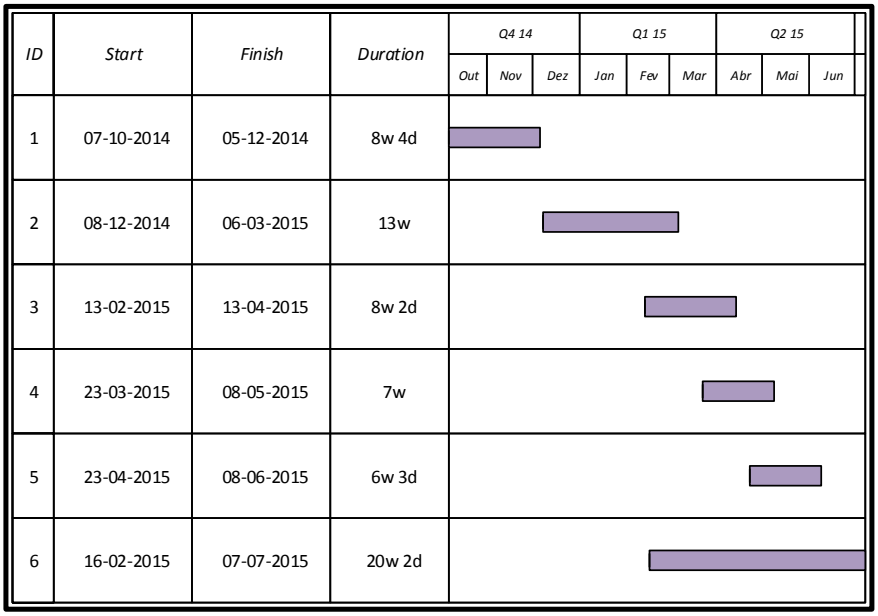

Figure 1.2: Initial work plan.

Our workplan has changed from the initial idea. The work we present in this thesis is part of an European project - SEGRID: security for smart grids - and a redefinition of the FCUL tasks in this project led to a significant change. Namely, FCUL was unable to buy the smart grid aggregator because these are only sold to electrical companies. Without this component, our original idea could not be implemented.

As such, we decided to focus our attention in the fundamental component that is part of the resilient communications infrastructure that FCUL will develop during the course of this project: a secure monitoring system.

As such, We performed a threat assessment on common monitoring techniques. We devised software plugins to perform the attacks, implemented them and experimentally demonstrated the attacks. Finally, after an evaluation of existing solutions, we devised some more secure and accurate techniques to overcome the noticed vulnerabilities.

1.6 Document structure

This dissertation is structured in the following way:

- **Chapter 2:** Scientific background and related work.
- **Chapter 3:** This chapter discusses the relevance of precise measurements and de-

scribes the attack model. This is followed by a description of the implementation for performing the attacks. Finally, we demonstrate the attacks that cause the monitoring platform to report invalid measurement data. We concentrate on the two metrics most used in practice today: path delay and link throughput.

- **Chapter 4:** This chapter presents a discussion on how to secure an SDN-based monitoring platform.
- **Chapter 5:** Presentation of the conclusions of the work.

Chapter 2

Background and related work

This chapter presents background on the subject under study. We address SDN, OpenFlow and network monitoring with an emphasis on SDN-based monitoring frameworks. Finally, we describe some useful networking tools - Mininet, Ettercap and Scapy - that were used in this work.

2.1 Software Defined Networking

Software-Defined Networking (SDN) is an emerging networking paradigm [21]. This paradigm aims to change the limitations of the current network infrastructure. The fundamental property of an SDN is the separation of the control and the data plane. Network control is logically centralized in a controller (SDN-controller) that has a global network view and control on the behaviour of the network devices (Figure 2.1). Logical centralization does not imply that the control plane is centralized. It should be physically distributed, to achieve responsiveness, reliability, and scalability goals. SDN allows network administrators to flexibly manage network devices (routers or switches) using software running on servers.

The main advantage of SDN is allowing programming the network through software applications development, using high level abstractions offered by the controller. Network-state oriented applications (e.g., routing and load-balancing) use these abstractions to achieve the desired network behaviour without needing knowledge of the detailed physical configuration. It is the controller responsibility to install the network application logic in the switches, thus relieving the programmer from concerns regarding low level details.

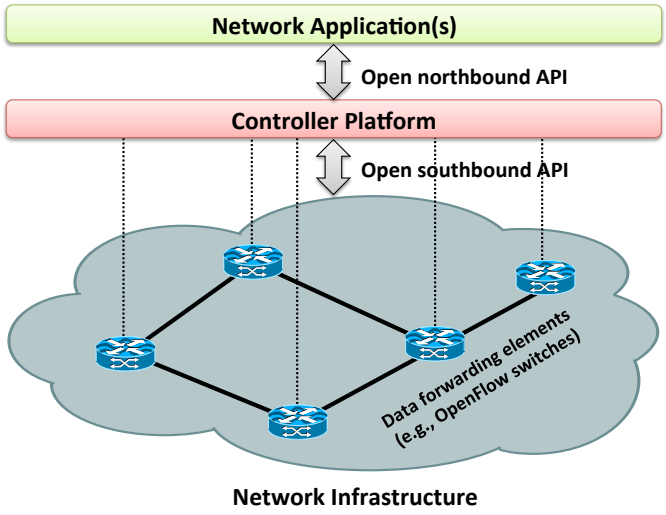

Figure 2.1: Simplified view of an SDN architecture [21].

2.1.1 OpenFlow

OpenFlow is an open standard supporting the communication interface between the control and forwarding planes of an SDN architecture. The main idea of OpenFlow is to give access to and facilitate manipulation of the forwarding plane of network devices and as such is the main enabler of an SDN. It provides an open interface to control how data packets are forwarded through the network, and a set of management abstractions used to control topology changes and packet filtering. The OpenFlow protocol specifies a set of instructions that can be used by an external application to program the forwarding plane of network devices.

OpenFlow consists of three parts (Figure 2.2):

1. Flow Tables installed on switches. The switch is informed how to process network flows by means of an action associated with each flow entry.

2. A Controller, which uses the OpenFlow protocol to communicate with switches to impose policies on flows. The OpenFlow protocol provides an open and standard way for the controller to communicate with switches and allows entries in the flow table to be defined externally.

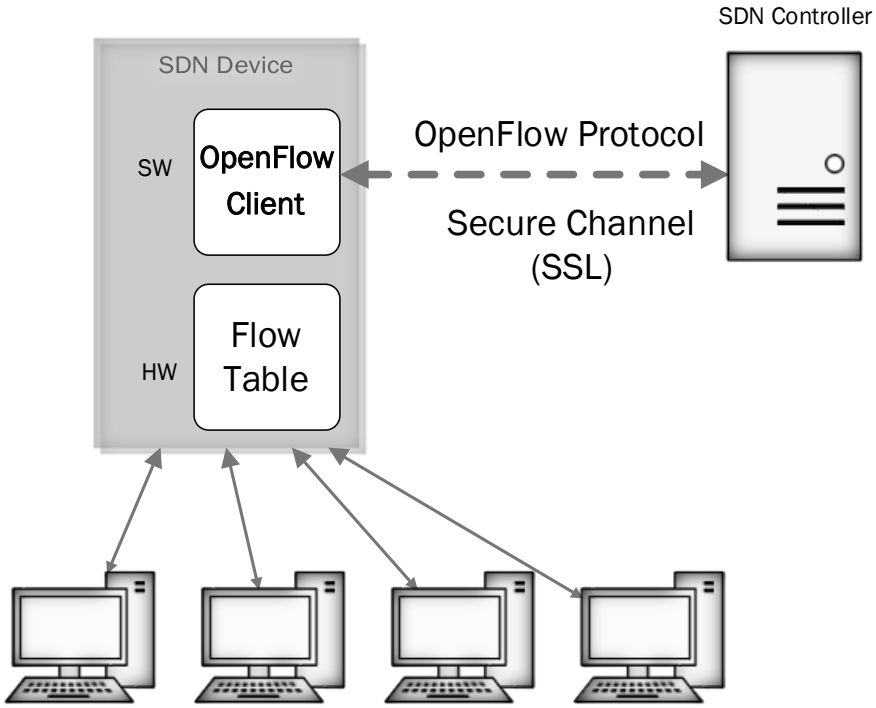

Figure 2.2: OpenFlow-enabled SDN devices.

3. A channel that connects the remote controller (remote control process) to switches and allows communication between them. The SSL protocol may be used to securely send commands and packets from the controller to switches using the OpenFlow protocol. Alternatively, this channel can also be based on TCP.

Priority	Ingress Port	MAC Src	MAC Dest	Proto	Vlan ID	IP Src	IP Dest	Source Port	Dest Port	Actions
10000	*	*	*	TCP	*	*	10.1.1.34/32	*	*	Forward to Port 1
1000	3	*	*	*	*	*	10.1.1.0/24	*	*	Forward to Port 2
100	*	*	*	*	2500	*	*	*	*	Send to Controller
0	*	*	*	*	*	*	*	*	*	OF Normal

Table 2.1: Simplified example of a flow table in OpenFlow switches.

Table 2.1 shows a simple example of an OpenFlow table present in OpenFlow switches. This table has the function to associate specific traffic to a (some) specific action(s). It is used by the controller to define the forwarding rules for each packet. A flow does not have to use strictly the packet headers as its match fields, since it can match a flow per *Inport*. Different priorities are defined to set the match order of a flow table.

When a packet arrives to a switch, and it does not match with any of the existing

flows, it is sent to the controller (or it is dropped). This action depends on the mode of operation of the SDN, which can be either reactive or proactive. In reactive mode the non-matching packets are forwarded to the controller. In response, the controller installs the respective flow rules, avoiding packets with the same header to be forwarded to the controller. Next in proactive mode, if the switch receives a non-matching packet it will drop that packet. In this mode the controller install the flows proactively, restricting the network traffic to the desired configurations. This can be seen as a drawback since it does not allow new traffic in the network but, on the other end, it may became secure as it may avoid DoS attacks and prevents undesired traffic to travel through the network. For more details on OpenFlow, please see [1].

2.1.2 SDN controllers

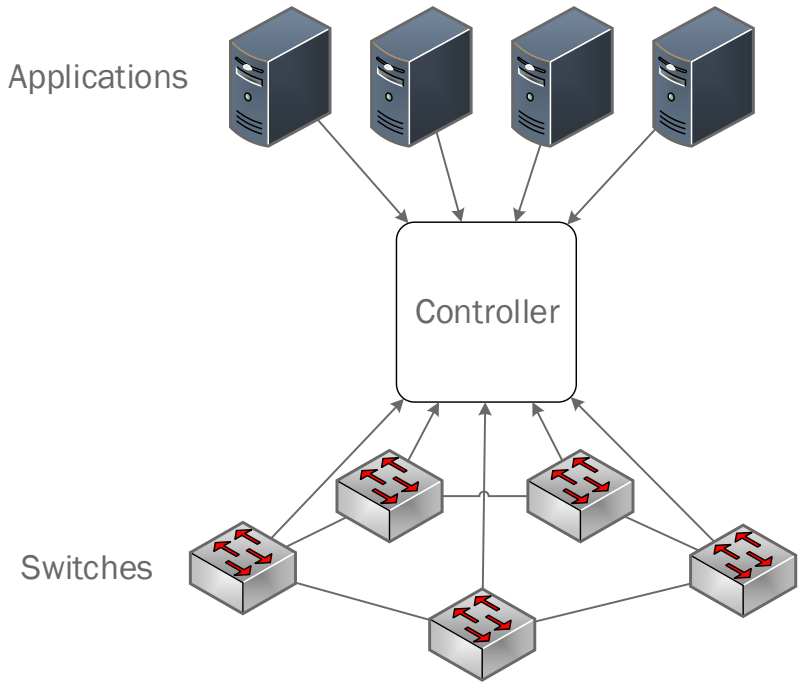

Figure 2.3: SDN basic architecture.

Since network control is moved to a logically centralized controller (Figure 2.3), this device is the core component of Software Defined Networks. It is located strategically in the SDN network, relaying information to the switches/routers “below” and the applications and business logic “above”. This controller platform typically contains a collection of “pluggable” modules that can perform different network tasks. Some of the basic tasks include control on flow tables, keeping an inventory of what devices are within the net-

work and the capabilities of each, and gathering network statistics. To make possible performing network-state abstractions and forwarding, a few SDN controllers have been proposed, including: NOX [15] [2], POX [2], Beacon [12], and FloodLight [3].

2.1.2.1 NOX

NOX [15] [2] was the first OpenFlow controller, so a significant amount of network applications have been implemented on top of NOX. NOX allows developers to choose whether to build network applications with a developer-friendly language (Python), or high performance applications (using C++). NOX made three major contributions: a centralized programming interface, a centralized policy based on a network filtering module, and explicit support for middleboxes. NOX supports a switch control abstraction where OpenFlow is the prevailing protocol. The NOX model is event-based allowing a programmer to write an application by programming event handlers for the controller. On the other hand, NOX requires programmers to know and understand the semantics of low-level OpenFlow commands.

2.1.2.2 Beacon

Beacon [12] is a fast, cross-platform, modular, Java-based OpenFlow controller that supports both event-based and threaded operation. Its applications are implemented as bundles, an abstraction of the OSGi framework [4]. These code bundles that can be started, stopped, refreshed, installed at runtime, without interrupting other non-dependent bundles. Unlike previous controllers that provided users and developers with either compilation-time modularity or start-time modularity, Beacon thus allows for run-time modularity.

2.1.2.3 Floodlight

Floodlight [3] is an open, free, Apache-licensed Java-based OpenFlow controller. Forked from Beacon, the Floodlight controller consists of a set of modules. Each module provides a service to the other modules and to the control logic application through either a simple Java API or a REST API. Unlike Beacon, OSGI support was removed for deployment and performance reasons.

2.2 Network Monitoring

Network environments are growing more complex, but the need to provide a certain level of service stands. The exponential grow in data, the increasing number of devices, between other factors, are the cause for the existing pressure placed on systems, networks and applications. A large slice of the work done nowadays by TI specialized teams is regular monitoring of systems and applications, granting a correct state to the organizations and optimizing the resource usage. The monitoring systems thus represent a critical centerpiece of productivity.

The Simple Network Management Protocol (SNMP) [10] is the most commonly used protocol for network monitoring, but it is too coarse-grained. Among others, SNMP can be used to request per-interface port-counters and overall node statistics from a switch. Monitoring using SNMP is achieved by repeatedly polling the switch, which can cause switch performance degradation due to CPU overhead.

NetFlow [13] is more fine-grained, allowing measurements at the flow level, and scales better by using sampling approaches. It collects samples of network traffic and estimates overall flow statistics based on these samples. NetFlow assumes the collected packets to be representative of all traffic passing through the collector which may not be true. It uses a 1-out-of-n random sampling, meaning it only stores every n-th packet. Another drawback is that this solution is expensive by requiring dedicated hardware and specialized algorithms. Recent work [22] proposes to instrument switches with hash-based primitives to increase measurement accuracy, but these methods require hardware modifications that may not be available in regular switches soon.

SDN increases the flexibility of monitoring by offering a programmatic interface for fine-grained measurement. OpenFlow-based SDN switches expose a high level interface to the controller for per-flow and aggregate statistics collection. Network applications can use this interface to monitor network status and create a global view of the network. OpenFlow provides both pull- and push-based measurement techniques. The most typical statistics collection mechanism is pull-based. Switches maintain per-port and per-flow rule counters that track the number of packets and bytes handled by each port and each flow rule, respectively. The controller can periodically query the switches about these statistics. The polling frequency determines the accuracy and overhead of the whole process. The push-based approach is based on `packet-in` and/or `flow-remove` OpenFlow messages. When a packet does not match any rule in the switch, the switch sends a `packet-in` message containing the packet header (and optionally part of the payload)

to the controller. When the SDN is configured to work in reactive mode this allows monitoring of all *new* flows. An alternative push-based approach is for the controller to request a switch to send a flow expiration message (`flow-remove`) whenever a flow expires.

Monitoring can also be done actively. On the control plane, the controller may periodically perform measurements of the network status. One example is through the transmission of probe packets to be forwarded by switches back to the controller. This implies injection of new traffic onto a network.

2.3 SDN-based monitoring frameworks

An already significative number of recent works have leveraged on SDN techniques to propose advanced network monitoring tools and mechanisms. A good amount of research has focused on the tradeoff between measurement accuracy and overhead/cost. As fine-grained monitoring is very challenging due to resource constraints (e.g., number of TCAM entries), several solutions have explored alternative hybrid designs.

iSTAMP [25] dynamically partitions the TCAM entries to allow both fine-grained and coarse-grained measurements. One partition aggregates measurements (helping the system scale), and another is used to provide accurate per-flow measurements for the most informative flows. iSTAMP then processes these aggregate and per-flow measurements to effectively estimate network flows using a variety of optimization techniques. An algorithm is proposed to intelligently select the most informative traffic flows.

Zhang [43] proposes an anomaly detector based on SDN principles (mainly the centralized view of the network) that instructs a flow statistics collection module to provide fine-grained measurement data in case it is anticipating an attack, or to collect coarse-grained data otherwise.

Payless [11] provides a RESTful API for flow statistics collection at different aggregation levels. Payless presents techniques to compute throughput, both active and passive. It uses `packet-in` and `flow-remove` to get statistics passively, but it is adaptive since it uses an adaptive statistics collection algorithm that delivers highly accurate information in real-time without incurring significant network overhead. For instance `flow-statistics-request` messages are sent to the switches, but just if a flow is inserted for a longer time than a defined parameter. PayLess provides an abstract view of the network and a uniform way to request statistics about the resources. PayLess itself is developed as a collection of pluggable components. Interaction between these

components are abstracted by well-defined interfaces. Hence, one can develop custom components and plug into the PayLess framework.

DREAM [28] is a dynamic resource allocation measurement framework that balances a user specified level of accuracy and resource usage for measurement tasks. Resources are dynamically deployed to achieve the desired level of accuracy. It ensures high accuracy for tasks, while taking network-wide resource constraints as well as traffic and task dynamics into account.

Conventional SDN techniques introduce delays that may be prohibitively high for particular network management tasks. To tackle this challenge, OpenSample [34] is a low-latency, sampling-based network measurement platform that leverages sFlow packet sampling to provide near–real-time measurements of both network load and individual flows. In order to deal with the sampling they use a collector. The rate of samples produced by sFlow is not constant, it is equal to the packet rate on the port divided by the sampling rate. To overcome this limitation, the authors exploit the fact that each TCP packet carries a sequence number indicating the specific byte range the packet carries. Fortunately, when sFlow samples the header of TCP packets, this header also includes the TCP sequence numbers. So if two distinct packets from a given TCP flow are sampled, they can compute an accurate measure of the flow's average rate during the sampling window by subtracting the two sequence numbers and dividing by the time between the samples. Exploiting TCP information drastically increases estimation accuracy for any given sampling rate. This TCP-aware sFlow analysis is the key innovation OpenSample incorporates compared to prior sFlow monitoring frameworks.

Planck [32] is a network measurement architecture that goes one step further, extracting network information an order of magnitude faster than alternative approaches (including OpenSample). Planck leverages on the port mirroring capability that is present in commodity switches. Multiple ports are mirrored to the mirroring port that is then connected to a collector to aggregate information and send the required statistics to the SDN controller.

Other proposals aim to advance the measurement abstractions present in SDN-based systems. OpenSketch [41] proposes a flexible measurement API that, complementary to OpenFlow, separates the measurement control plane from the data plane. The objective is to provide flexibility for network measurement, and for that purpose it allows the operator to reprogram its measurement tasks. It adds a reconfigurable measurement logic to switches and exposes an interface to program it (in the same way OpenFlow allows for programming forwarding behavior). The main problem is the fact that it is based on a

clean-slate redesign of portions of the switch hardware. It does entails the need to upgrade network nodes and to standardize this new protocol.

HONE [35] extends the scope of traffic management to the end-host networking stack, allowing joint host-network traffic management. It presents a uniform stack for a diverse collection of measurements in SDN-based systems.

2.4 SDN-based monitors

One of the first SDN-based monitors was OpenTM [36], an OpenFlow-based solution that estimates a traffic matrix by keeping track of statistics for each flow. OpenTM queries switches on regular intervals and stores the information needed to create the traffic matrix.

More recently, OpenNetMon [37] was proposed as an open-source tool to measure throughput, packet loss, and delay. To measure delay, this platform installs rules in the entry and exit switches of a path to generate notifications (e.g., `packet-in` messages) to be sent to the controller. It then sends probes that match these monitoring rules, and from the received messages it infers path latency. In addition, OpenNetMon adapts the probing frequency to the current estimated link utilization. In order to measure throughput, OpenNetMon uses an active approach and requests Per-flow statistics from the OpenFlow switches.

We selected OpenNetMon [37] as target for the attacks. Not only because it monitors three metrics at per-flow granularity – throughput, packet loss, and delay – whereas other proposals focus on a single metric, but also, because it is available open-source. We have used OpenNetMon, but several attacks are generic and will succeed against recent proposals.

SLAM [39] is a recent example of the generality of the techniques used by these platforms. It is another SDN-based framework dedicated to latency monitoring. The main method used is very similar to the one proposed by OpenNetMon for delay monitoring. SLAM improves over OpenNetMon by suggesting techniques to increase measurement accuracy. It also considers passive approaches that do not require the injection of "probes" but instead use unmatched application packets that are sent back to the controller for delay measurement.

Contrary to the previous approaches, FlowSense [40] is a passive technique used to estimate network performance. In FlowSense `packet-in` and `flow-remove` mes-

sages are used to estimate per flow link utilization. The communication overhead is low, but the estimation is not as accurate as with the active approaches.

The novelty of this work we present in this thesis arises from the fact that none of the related research in this topic (including all work presented in this and the previous sections) has yet addressed security.

2.4.1 A more detailed view on OpenNetMon

OpenNetMon [1] is an SDN application that runs on top of the Pox [2] controller and performs adaptive monitoring of network path delay, flow throughput, and packet loss. It relies on a Pox service for link discovery, which provides knowledge about the network topology (e.g., switches and links between switches). OpenNetMon itself implements a learning service that discovers hosts and the switch ports they are connected to, along with a forwarding service that installs flow entries along the shortest paths between known source and destination hosts.

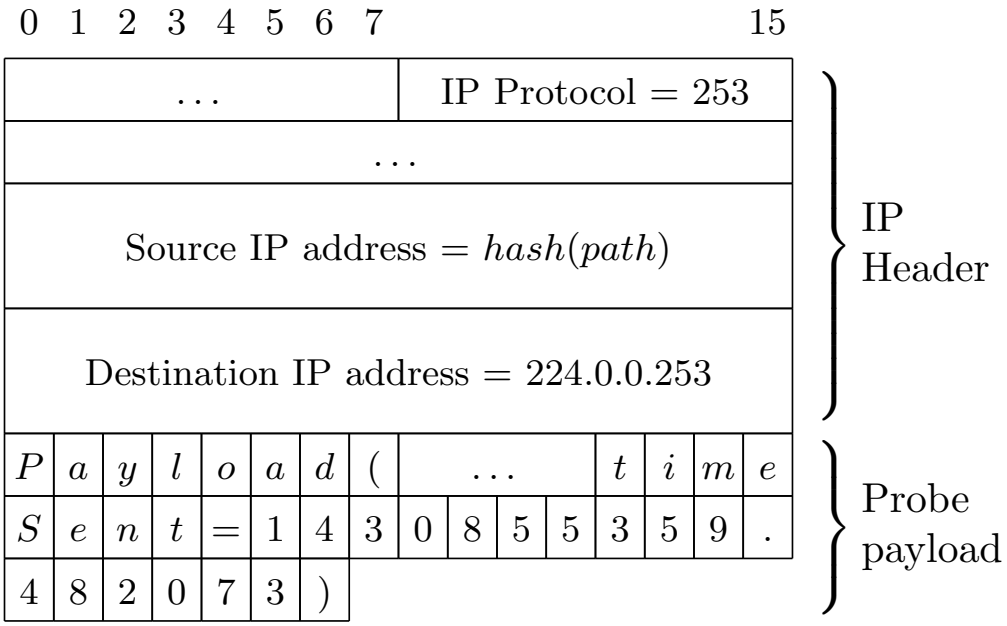

Figure 2.4: Structure of a probe packet.

When a data packet[2] arrives at one of the switches and there is no match with the installed flow rules, it is transmitted to the controller (as a *Packet-In*) where it is processed

[1] OpenNetMon is an open source SDN based monitoring solution. Available at: https://github.com/TUDelftNAS/SDN-OpenNetMon

[2] For our purposes, we consider a data packet to be any Ethernet frame that is neither a LLDP or an ARP packet, nor a probing packet.

by OpenNetMon. OpenNetMon learns the packet source MAC (Media Access Control) address and maps it to the switch ingress port. In order to decide where to forward an unmatched packet (as a *Packet-Out*), OpenNetMon looks at its destination MAC address (MAC_{dst}) and performs one of two actions: 1) if MAC_{dst} has not been learned yet or is a multicast address, it floods the packet through all ports of every connected switch *except* those ports that are part of inter-switch links to avoid loops; 2) if MAC_{dst} and its respective port are known, OpenNetMon calculates the shortest inter-switch path that connects the entry and exit switches and installs a flow rule on every switch of the path. Then, it forwards the packet directly to the known port of the exit switch.

For the purpose of network monitoring, when installing a new data flow OpenNetMon adds additional monitoring rules on the switches along the path. Path delay is measured periodically using an active probing strategy. A special probe packet 2.4 is transmitted to the ingress switch, which is then forwarded along the flow path until it reaches the egress switch where it is returned back to the controller by means of a `packet-in` message (using the a priori installed monitoring rules). This is the same technique used by other monitoring platforms, such as SLAM [39]. Flow throughput is obtained differently, using a query approach. The controller periodically requests counter values from the last switch of a given flow path. With each query, the controller receives the amount of bytes received and the duration of each flow, enabling it to calculate the effective throughput.

The frequency of measurement (in terms of probe transmission and query rate) varies accordingly to the combined throughput of all switches. The higher the throughput, the more frequent the values are collected.

2.5 Networking tools

In this work we used a few networking tools, which allowed us to create an emulated testbed, and perform the attacks on the monitoring system.

2.5.1 Mininet

Mininet [5] [16] is a realistic network emulator. Although Mininet virtualizations are created in software, they behave as the real hardware and have enabled the migration to hardware using unmodified code. In order to be able to emulate a full network it uses

lightweight virtualization, which implies that the whole network can run in the same system. As a virtual platform it has advantages and disadvantages:

- It is fast to start a network as it uses container-based virtualization. Running a platform in a single system is convenient, but on the downside it can introduce resource limitations.
- It allows the creation of custom topologies to meet specific needs and it is possible to customize packet forwarding, since Mininet switches are programmable using the OpenFlow protocol. As disadvantage, it is based on a single Linux kernel for all virtual hosts, and as such it is impossible to run software for other OSes.
- As it is an emulator, Mininet runs real programs on the hosts, just as in a hardware platform, and so it makes it easy to port from the virtual environment to a physical deployment. But as Mininet is not a simulator, and it does not have a strong notion of time, which means that timing measurements may be skewed by external factors, so care needs to be taken.

2.5.2 Ettercap

Ettercap [6] is a free and open source security tool that allows man-in-the-middle attacks in LAN environments. It supports active and passive dissection of protocols (including encrypted / encapsulated) making it possible to listen to interfaces, intercept, change and even inject new packages. Ettercap allows running filters and plugins on the packets that pass through the interfaces. The language for programming filters and plugins in Ettercap is C.

Initially we implemented all the attacks in Ettercap, but we noticed that it inserted a high delay on the modified packets, which is a problem since the attacks should be precise. As alternative, we opted for Scapy, which not only performs much faster that Ettercap, but is actually simpler to use.

2.5.3 Scapy

Scapy [7] is a powerful interactive packet manipulation framework in Python that enables packet decoding, creation and modification. It supports a wide range of network protocols. Injected packets are created from scratch or can be modifications of received

packets. Scapy can also handle and manipulate wireless communication packets, so there is not the restriction to work on wire.

After setting the context, in the next chapter we present attacks on SDN-based network monitors, in order to demonstrate the serious security vulnerabilities of these platforms.

Chapter 3

Network monitoring under attack

Network monitoring plays a fundamental role in many management tasks, as it collects and provides data about the current state of the network. Several network management applications act upon this information, namely: traffic engineering may reroute flows after an overload is observed in a link, to ensure that a previously agreed QoS level is maintained; a link is perceived as malfunctioning by a network diagnosis component after showing a high loss rate, and therefore is disconnected; a traffic shaper may throttle some flows if the associated counters indicate that they are consuming more than their share of the available bandwidth. These examples give evidence that, by manipulating the measurement data, an adversary can (indirectly) influence management decisions and consequently the behaviour of the network.

Based on this, we envision many attack scenarios where an adversary may gain some profit. Taking Eve as an example, and being eavesdropping the main goal, she could make several routes look overloaded (or loss) to force traffic to be forwarded over particular paths (or switches) she observes. She could also manipulate the contents of some packets if the associated flow is redirected to a previously compromised router. A black hole could remain undetected by silently dropping packets while ensuring the corresponding counters are maliciously updated. Alternatively, by falsely reporting some routers as lightly loaded, she could create an excess of packets to be directed towards a region of the network, with the consequence of packet drops (e.g., due to overflowed port queues).

Of course, in practice it will be challenging for an adversary to obtain the control needed to perform meaningful attacks, since measurement data is collected at different points. Moreover, the correlation of monitoring data can trigger an anomaly detector to indicate an attack is under way. However, even a localized attack can cause signif-

icant damage, and a stealthy attacker may remain unnoticed. We, therefore, argue that monitoring solutions need to be resilient from the ground-up to avoid being the target of compromising attacks.

In a network system, if routing decisions depend on the output of monitoring functions, and if they especially depend on the *correctness* of that output, then from a security point of view, the monitoring subsystem must be resilient to attacks. We posit that an adversary with few capabilities can successfully affect monitoring results and thus gain some measure of control over the network routing algorithms. The main idea is to report incorrect monitoring values to the controller, hence affecting possible reconfigurations of routing parameters on the data plane.

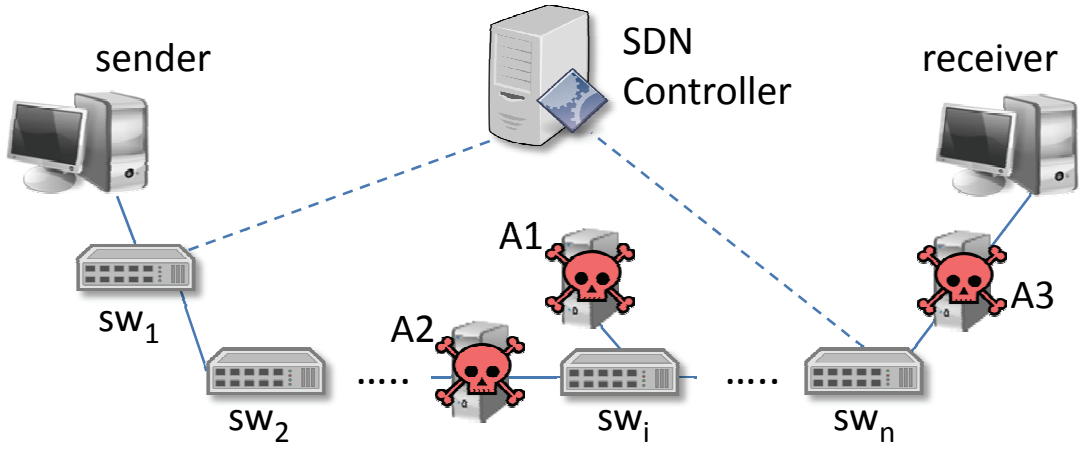

Figure 3.1: Overview of network and attackers.

3.1 Threat model

We consider an adversary that wants to modify the measurement data retrieved on a network path between an ingress switch (sw_1) and egress switch (sw_n), as depicted in Figure 3.1. We assume both switches support an SDN protocol (e.g., OpenFlow) and are therefore controlled remotely by an SDN controller. To form the end-to-end path these switches are connected by an arbitrary number of other components (e.g., routers and switches), that may or may not be SDN-capable. The monitoring application runs in the SDN controller, obtaining periodically from the SDN switches the counter values associated with the flows.

A strong adversary can create a direct intrusion on a switch or the controller itself, and thus gain full control of the component, which simplifies the attack on the monitoring

subsystem. Another strong attack could directly interfere with control plane communications, but if we assume communication links between the controller and the switches are secure, and more importantly if they guarantee at least data integrity, then a weak adversary will not be able to use that avenue to attack.

We limit the adversary actions to the data plane, meaning that the links between the controller and the switches are considered secure (e.g., using SSL). Additionally, we assume the switches involved in the measurements (e.g., sw_1 and sw_n) operate accordingly to their specification, i.e., they are not compromised by the adversary. This means that adversaries with further abilities may be able to expand the impact of our attacks.

Our adversary model includes attackers at multiple levels of sophistication, each one increasingly more powerful than the previous:

- `A1`: the adversary can insert arbitrary traffic in the target network, namely along the path we want to monitor;
- `A2`: the adversary can observe, insert, and modify the packets transmitted along the path;
- `A3`: this adversary is used together with `A2`, and she has the additional capability of dropping packets after sw_n.

In summary, we assume the attack surface to be solely the data plane of an SDN network. The attack vectors are the links between each pair of connected switches and the links between switches and connected hosts. Figure 3.1 illustrates these vectors.

3.2 Implementation

As stated, three types of adversary are considered in our attack model, with the ability of injecting, eavesdropping, modifying or dropping packets. In order to perform the attacker role, we implemented plugins for Scapy, a fast packet manipulation framework in Python.

We have implemented several plugins during the course of our studies, but we will only focus on the ones that led to successful attacks. Each plugin was developed to a particular adversary, each with its different capabilities. The attacks are focused on the two most relevant metrics: path delay and link throughput. Here we present the generic implementation of the attacks. In Section 3.3 we go further in detail when needed.

3.2.1 `A1` plugins

The A1 adversary represented in Figure 3.2 has the sole capability of injecting arbitrary traffic into the network.

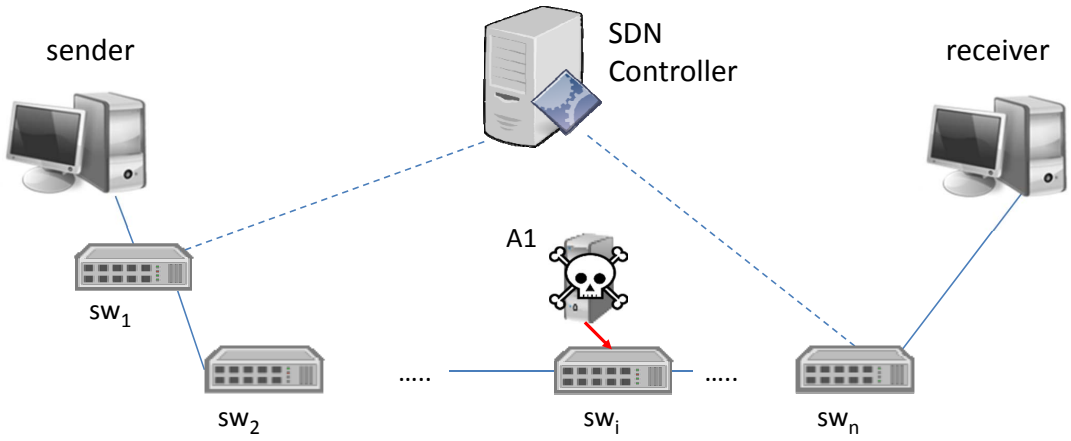

Figure 3.2: Adversary `A1` injecting fake traffic.

3.2.1.1 Delay attack

This plugin injects packets similar to the probe used by the controller to actively measure delay between switches, but using a different (fake) timestamp in the payload (recall Figure 2.4). The major issue of this plugin is the need to guess the packet format (recall that this adversary does not have the capability to eavesdrop, but only inject packets). This is made easier if the controller is open source. With the capacity of eavesdropping the adversary A1 could easily create packets to inject, based on the network traffic he would observe.

3.2.1.2 Throughput attack

This plugin injects traffic that match a specific flow in order to artificially increase the throughput on that flow. Similar to the delay attack, the difficulty for the attacker is knowing the format of the flow in the network. With the capability to eavesdrop, it would be easy: it would merely be needed to inject duplicate packets.

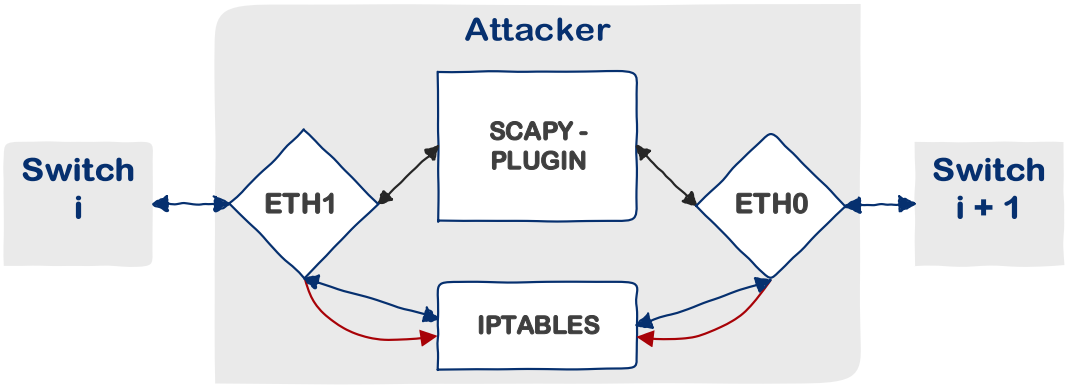

Figure 3.3: Scapy plugin architecture.

3.2.2 `A2` and `A3` plugins

The A2 adversary represented in Figure 3.4 has higher capabilities, which allow him to inject, eavesdrop, modify or drop packets. This adversary needs two network interfaces and a bridge between them. As seen in Figure 3.3, in Scapy the packets that arrive at the interfaces of the attacker are sniffed, modified as needed by the plugin, and dropped by the IPTABLES (they are injected by Scapy in the correct interface avoiding duplicates).

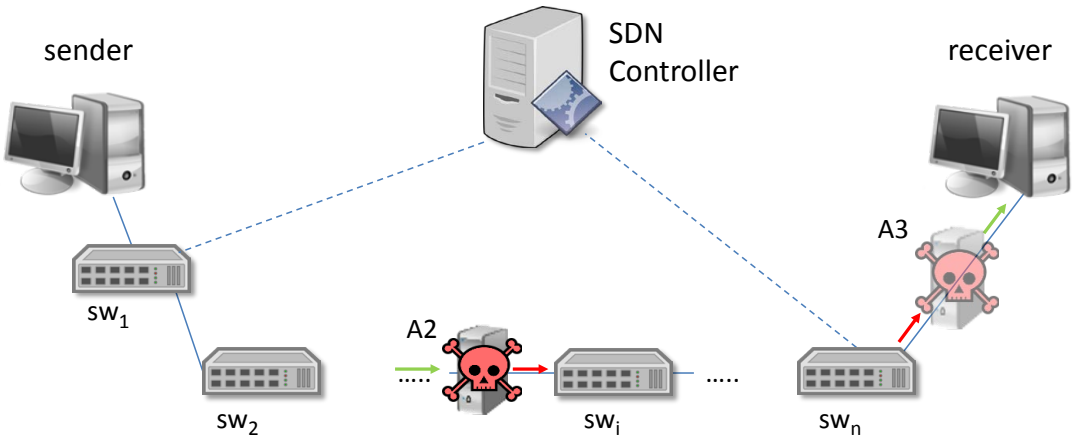

Figure 3.4: Adversary `A2` and (optional) `A3` modifying traffic.

3.2.2.1 Delay attack

This plugin is the most powerful, allowing the modification of delay of the probe packets. Special care is needed if this plugin is used to decrease the delay measured by the controller: the fake timestamp needs to be calculated in such a way to guarantee that the

overall delay computed by the monitor does not have a negative value (as this would make the attack easy to spot). In our implementation, the inserted timestamp is based on the already existing one, plus a minimum calculated delay based on the passing traffic.

3.2.2.2 Throughput attack

This plugin has the same packet injection capability as adversary `A1`. It also allows to catch a packet and modify its fields, usually the source MAC address since it is not checked in most end-to-end applications. This modifications applies to each packet that goes through the attacker. A different MAC is used for each one, assuring each packet does not match in any of the switch flows. This modification causes the controller to install new flows per each modified packet.

As we can see in Figure 3.4, optionally a second attacker exists, removing the modifications made by adversary A2 so that the receiver is completely oblivious to them. Adversary A3 employs a similar plugin as A2. This plugin has the option to modify or replace the correct state of the packet.

3.3 Attacking the monitor

This section presents a series of attacks that expose vulnerabilities of SDN-based monitoring platforms. We focus our attention in attacks that affect the measurement of path delay and throughput. We have chosen the recently proposed SDN-based monitoring application OpenNetMon [37], which is available open-source, to demonstrate the feasibility of the attacks. Most approaches used by this application are generic and have in fact been employed by other monitoring tools, such as SLAM [39].

3.3.1 Testing environment

We created two separate testing environments that share the simplified setup of Figure 3.1. As represented in Figure 3.5 the main modification was that only two switches were utilized, sw_{entry} and sw_{exit}. The three attackers were connected to sw_{exit}. In addition, attacker `A2` is connected to switch sw_{entry} and attacker `A3` is connected to the receiver.

The first environment is based on the Mininet [16] emulator, allowing an evaluation based on software switches (namely, Open vSwitch [29]). The second environment is

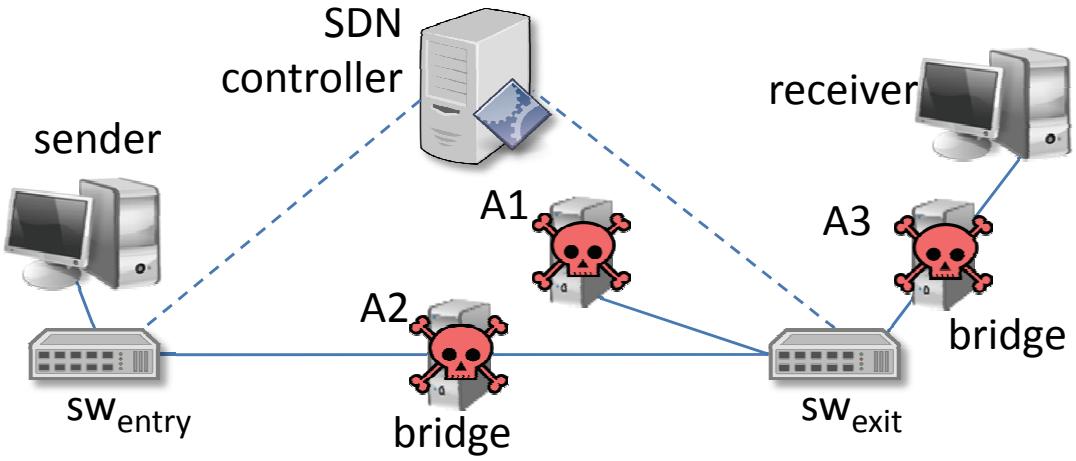

Figure 3.5: Testbed used in the experiments.

a real testbed platform with physical OpenFlow switches and separate machines for the controller, hosts and attackers. In both environments, we used Pox (version "eel") because OpenNetMon is an application that runs on this controller. The communications between the controller and the switches were made using OpenFlow v1.0 (the version supported by Pox).

For the Mininet environment, we run the network and nodes as separate processes on a Dell R320 server. The experiments were performed with Mininet v2.2.1 and Open vSwitch v2.3.1. In the testbed environment, we deployed two Pica8 P3297 switches with 4K TCAM entries[1] and 1 Gbps links. The controller, hosts and attackers were separate Dell machines. Attackers $A2$ and $A3$ were implemented in dual homed machines and they used Ethernet bridging to forward the traffic between network boards. With bridging, the adversary gains the capability to intercept, modify and drop packets while staying "transparent" to the rest of the network, i.e., there is no advertising of MAC/IP addresses. We used iptables together with Scapy v2.2.0 [7] to eavesdrop and inject arbitrary packets.

3.3.2 Attacking delay measurements

Measuring inter-switch link delays is not an easy task because it is often hard to define arbitrary starting and ending points in the network. SDNs help overcome this problem: due to the separation of the control and data planes, it is possible by design to reach any switch in the network from a single central point (the controller). This allows for

[1] In Pica8 switches, two TCAM entries are required to process one flow by default.

an accurate representation of inter-switch links when starting a new delay measurement. Estimating the total path delay of a given flow in an OpenFlow network, for example, comes down to the controller tracking a packet that traverses, in order, the set of switches composing the flow path.

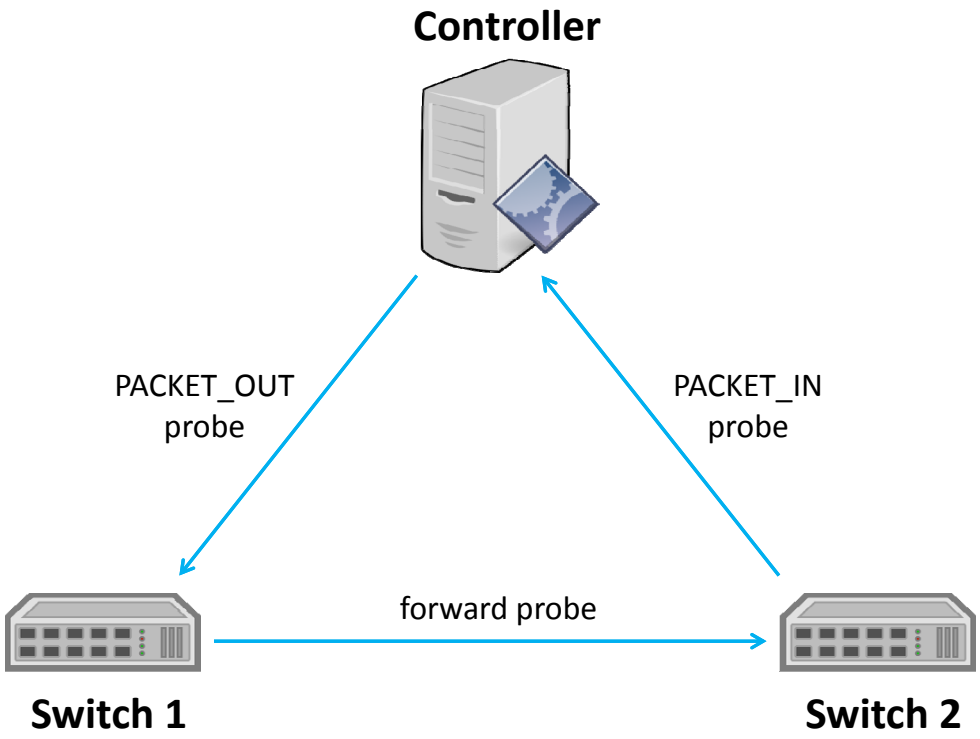

Figure 3.6: Delay measurement.

Active monitoring is the predominant approach for measuring the path delay in SDN-based monitoring solutions. The controller installs forwarding rules in the switches, and then inserts a probe in the data plane directed back to itself, while keeping track of the departure and arrival instants. The delay is calculated by subtracting the two instants. To improve the accuracy of the measurement, the controller might also get an estimate of transmission time between itself and the entry/exit switches and perform the appropriate correction on the delay. OpenNetMon operates exactly in this manner, storing the departure timestamp inside the probe (as seen in Figure 2.4). In this manner, when the probe returns, OpenNetMon gets a new timestamp and uses it to determine the flow path delay. This solution is relatively simple and recovers easily from packet drops – a lost probe is a single measurement sample that goes missing.

In this attack, the adversary objective is to force an incorrect calculation of the path delay, either by increasing or decreasing the calculated time interval. For this purpose, our

attacker either inserts a fake probe or modifies the original. We consider two scenarios, presented in increasing order by attacker strength.

3.3.2.1 Injection

Adversary `A1` generates and sends a malicious probe to one of the switches, causing an erroneous delay to be calculated on its arrival at the controller. To forge the probe, she fills the header fields exactly as OpenNetMon: (a) in the Ethernet header, the MAC addresses are the MACs of the sender and receiver hosts; (b) in the IP header, the source is an hash of the identifiers of all switches in the path. This hash is employed by the monitoring rules (pre-installed by OpenNetMon) to decide how the probe should be forwarded.

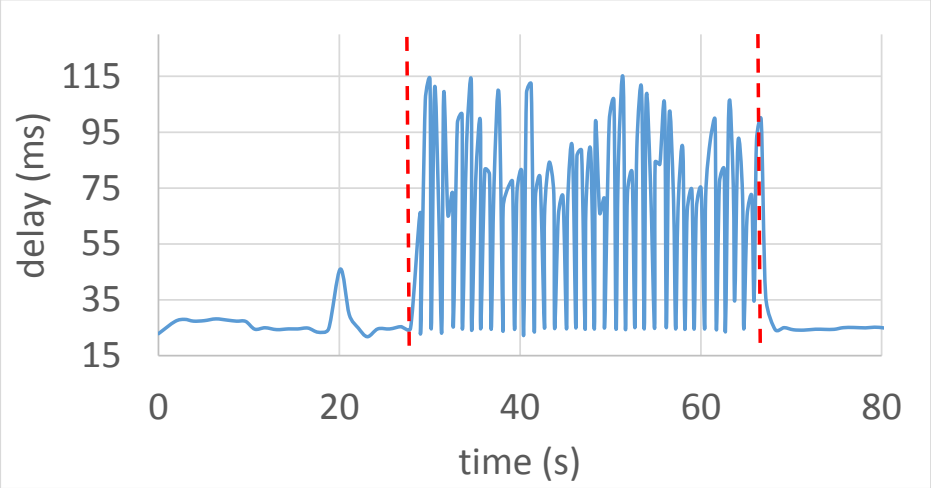

Figure 3.7: Increasing delay with probe injection (Mininet).

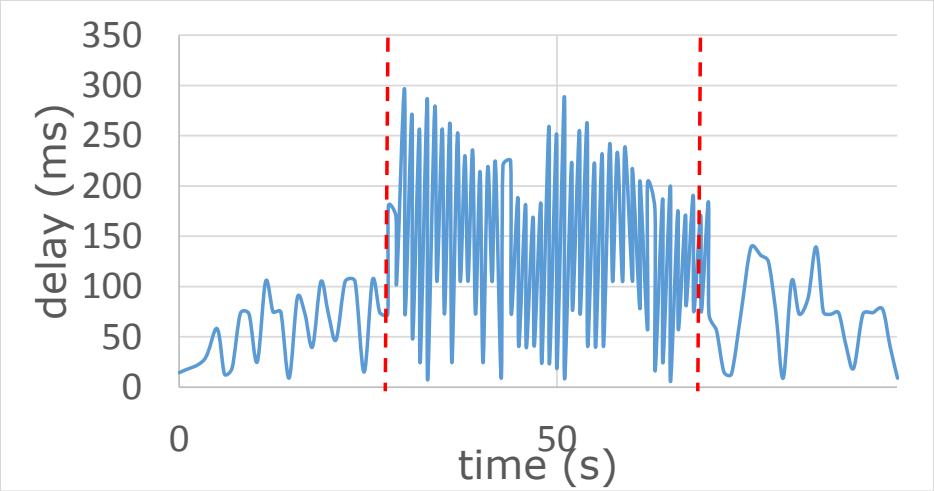

Figure 3.8: Increasing delay with probe injection (Testbed).

The packet payload contains a timestamp of the departure time. Therefore, by increasing or decreasing this value the adversary can easily fake a smaller or larger delay, respectively. In order to produce meaningful timestamps, an estimate of the delay between the attacker machine and the controller needs to be obtained. This estimate can be computed by trial and error or by resorting to more sophisticated methods.

The attack can be made more effective if the adversary is able to observe the path (as `A2`). In this case, when the attacker detects a probe she immediately sends the forged probe. Since the monitor accepts all probes as faithful, the effect is that the correct measurement will be considered only momentarily until the incorrect value is received.

Figures 3.7 and 3.8 illustrates the result of this attack (in the emulation setting and in the testbed, respectively). In all figures the attack was being perpetrated in the period between the dashed vertical lines. Attacker `A1` inserts spoofed probe packets, making OpenNetMon incorrectly report higher delay values.

3.3.2.2 Eavesdropping and Injection

Adversary `A2` eavesdrops and modifies, in real-time, the timestamp of probe packets. She starts the attack by capturing the probe and then forging a new probe that is a copy of the collected one with an adjusted timestamp. The original probe is dropped.

An attacker such as $A2$, reinforced with dropping and modification capabilities can attain a more effective attack. This is perceptible in Figures 3.9 and 3.10, which show an attack where the delay is increased. Figures 3.11 and 3.12 illustrate a fake decrease.

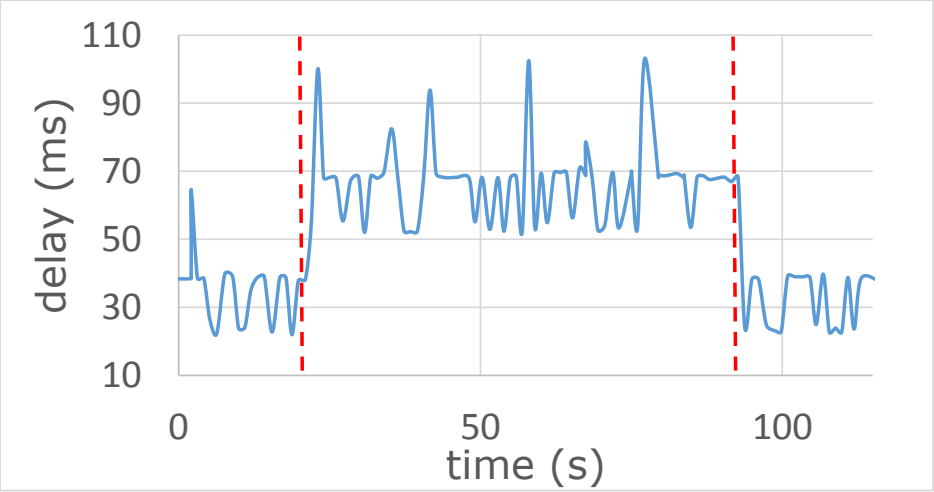

Figure 3.9: Increasing delay with probe modification (Mininet).

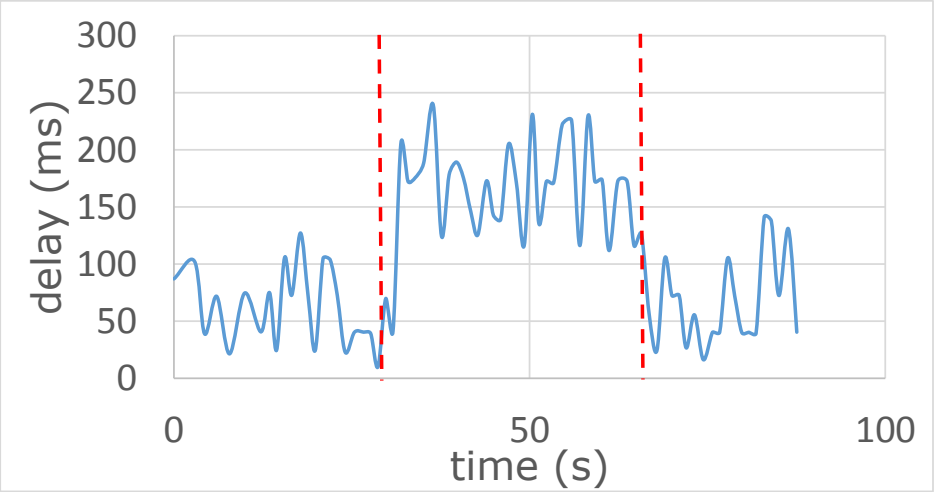

Figure 3.10: Increasing delay with probe modification (Testbed).

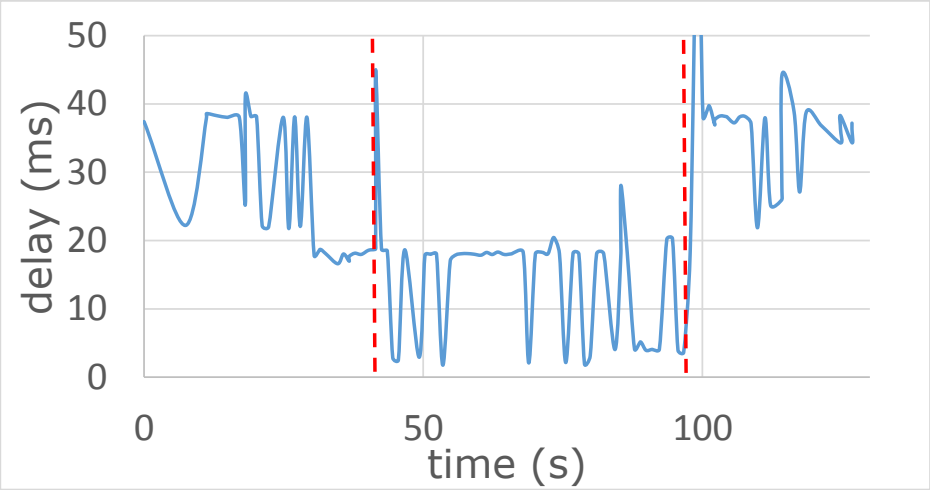

Figure 3.11: Decreasing delay with probe modification (Mininet).

3.3.3 Attacking throughput measurements

From the viewpoint of an application, a flow is the sequence of packets transmitted from the sender process to the receiver process. Inside the network a flow can therefore be identified based on the 5-tuple that consists of the source and destination IP addresses, the source and destination ports, and the transport protocol. In OpenFlow networks, the throughput can then be measured by extracting the counters associated with the rule that matches the flow.

OpenNetMon measures the throughput using this technique. It associates a counter to each flow rule that is inserted in a switch. Whenever there is match in a rule, its counter is incremented with the number of bytes of the packet. Throughput is then obtained by

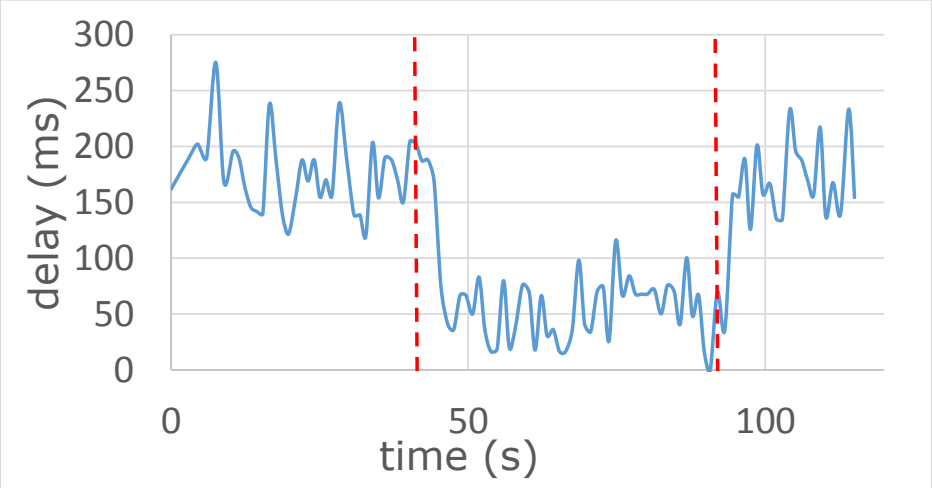

Figure 3.12: Decreasing delay with probe modification (Testbed).

the controller application by periodically querying the switch to obtain a reading of the counter. An adversary that wants to attack the throughput measurement of a specific flow simply needs to mislead the monitoring application by forcing a wrong count number, either by decreasing or increasing its value.

3.3.3.1 Decreasing throughput

The goal of the adversary is to hide from the monitor traffic passing through a switch, making it report a throughput that is lower than its real value. Since the switch counts the bytes that are actually received, the adversary cannot perform this attack by simply manipulating packet headers.

To attack OpenNetMon, the adversary `A2` takes advantage of an optimization performed by the monitor when it receives a `packet-in` message triggered by the arrival of an unmatched packet. In this case, OpenNetMon inserts, as usual, a flow rule in every switch on the path of the packet (from ingress to egress). Then, as an optimization it retransmits the incoming packet to the *nearest* switch to the destination as a *packet-out* message. Consequently, this packet does not go through the usual path and the counters of the switches are *not incremented*, i.e., they remain at zero.

The adversary can explore this optimization to decrease the measured throughput, while still allowing all packets to be delivered to the final receiver. This is attained by modifying the packets as they go through `A2` in such a way to prevent a match with the existing rules in the switch. Changing particular fields from the same flow (for instance, the Ethernet source address) triggers the switch to send a `packet-in` message to the

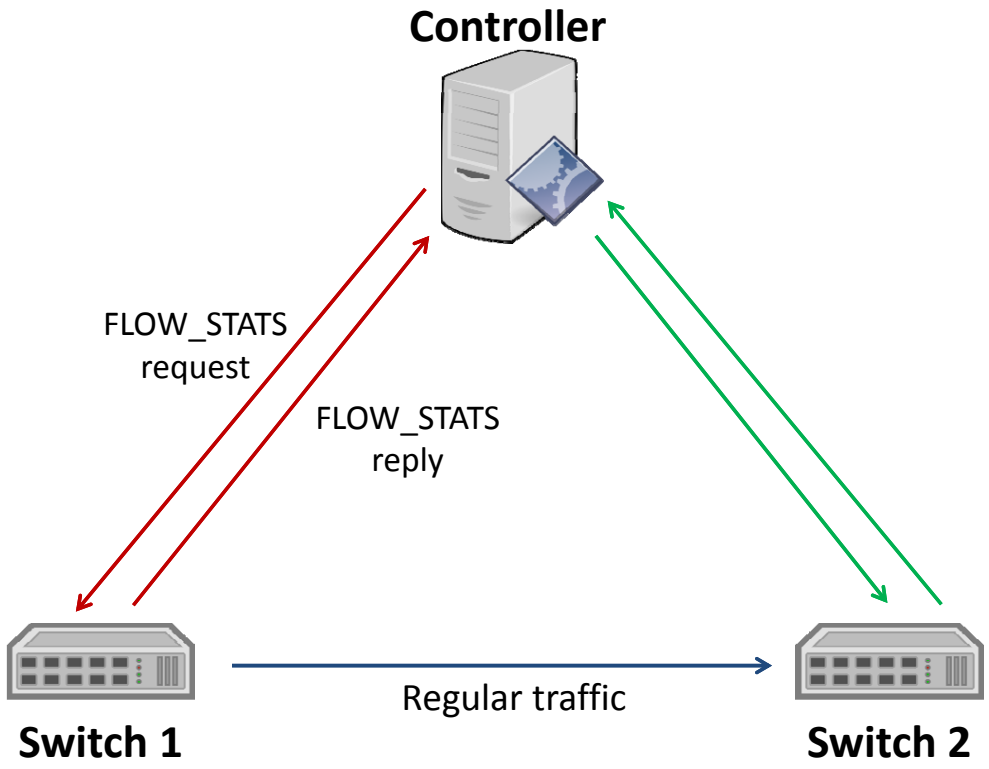

Figure 3.13: Basic throughput measurement.

controller but it does not change the normal forwarding of these packets. Every time such change is performed on a packet, the associated bytes are hence made oblivious to the monitor.

In this attack since the packet arrives to its destination modified, this may cause problems for particular applications. Notice, however, in our tests, the applications continued to work as expected, without showing any change of behaviour during the attack. A more sophisticated version of the attack includes adversary A3, located after the switch and before the destination host. Both run Scapy with a bridge between interfaces, but with different options. Modifications to the packets are made in the first attacker, and are withdrawn by the second attacker, which makes the packet return to its initial state. This attack works for any protocol because the modifications are only visible to the controller, which derives to the insertion of various flows on the switches, without increasing the throughput.

Figures 3.14 and 3.15 present the results of this attack. We ran `iperf` in the sender to start a 400kbps UDP flow to the receiver. When the attack begins, the throughput drops to zero, and it gradually returns to its normal value when the attack stops. Bringing the

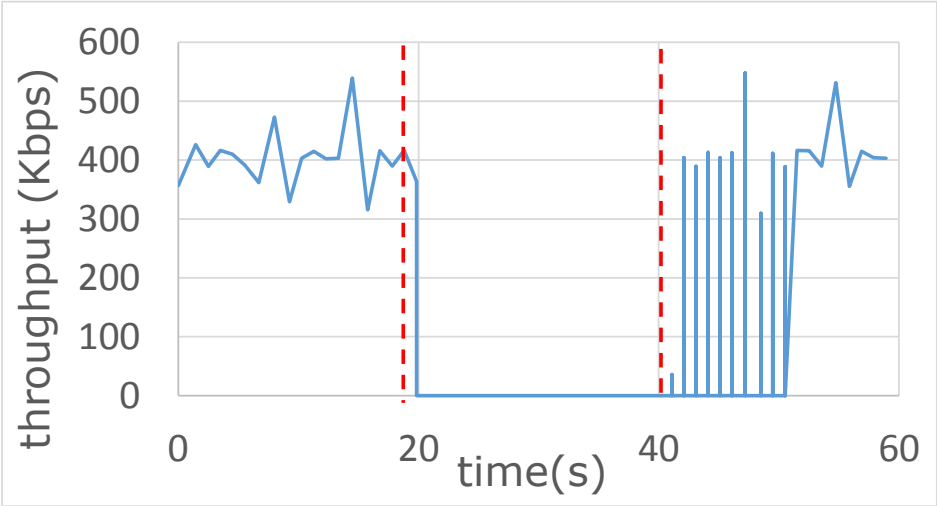

Figure 3.14: Decreased throughput (Mininet).

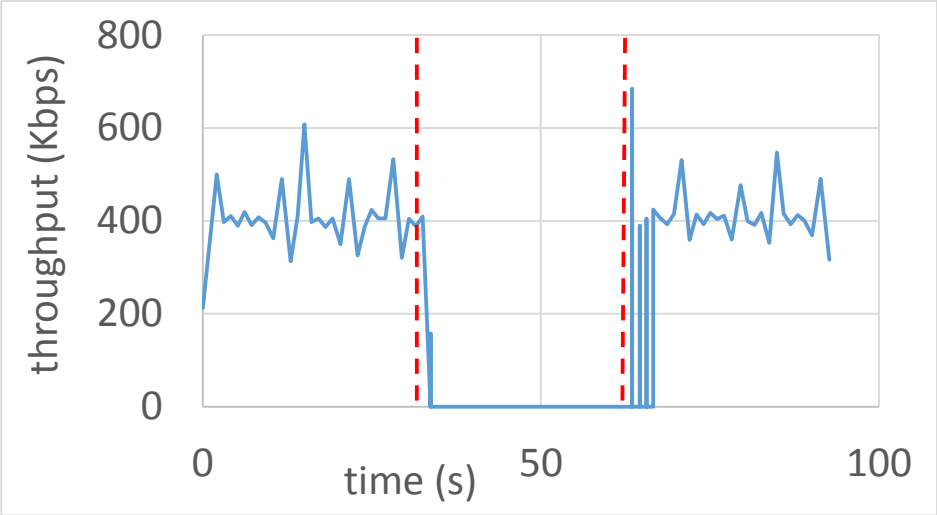

Figure 3.15: Decreased throughput (Testbed).

throughput to zero can have undesired effects. For example, in the smart grid scenario we are considering this would lead operators to erroneously believe that the monitored path had problems and unnecessarily would cause a maintenance team to be sent to the field.

Note that this attack may introduce additional per-packet end-to-end delay as it requires all packets to go through the controller. A more sophisticated version of it could tradeoff per-packet latency for an increase in throughput estimation. Instead of bringing throughput to zero we could let some packets follow its normal course without delay.

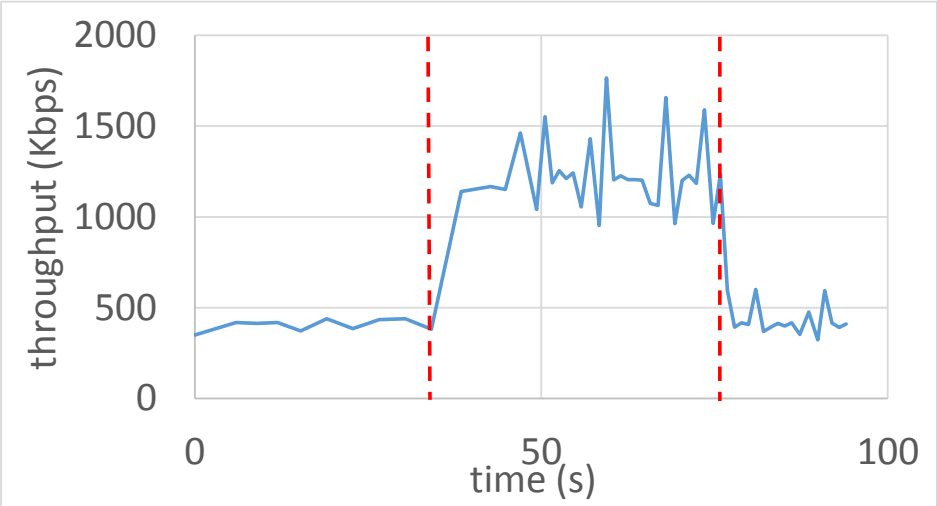

Figure 3.16: Increasing throughput (Testbed).

3.3.3.2 Increasing throughput

The goal of the adversary is to artificially inject traffic in a specific switch, making it report a throughput that is higher than the one from real traffic. The attack can be carried out by A2 by observing traffic and replaying packets that match an already existing flow. These packets will be dropped by the final receiver (they are considered duplicates). For the attack to be as stealthy as possible, the adversary can take advantage of `A3` to drop these packets. The result of this attack can be observed in Figure 3.16.

This attack can also be performed in another way, by exploring the fact that some controllers/switches do not set the in_port match field on the flows installed. When this field is set it is not possible to send packets "backwards" from where they originated. Only A2 can perform this attack, since the packets it injects need to be dropped afterwards. The way to explore this vulnerability is to inject packets matching specific existing flows. These flows must have as action, forwarding the packet to the port from where adversary A2 injected the packets (they are sent "backwards"). This allows for A2 to drop all his injected packets afterwards. We tested this attack in Mininet with success (the result is similar to Figure 3.16). Note that the effectiveness of this attack depends on the controller flow installations, because some controllers set the in_port field in their flows to disable packets moving "backwards".

In this section we have demonstrated how SDN-based network monitors are vulnerable to attacks, even relatively unsophisticated ones. In the next chapter we discuss techniques to improve the security of network monitors, as increasing their resilience is

fundamental for their effective use in a critical infrastructure as the smart grid.

Chapter 4

Discussion: securing the network monitor

In the previous chapter we have shown potential ways to compromise the correct behaviour of SDN-based monitoring solutions. Some of the demonstrated attacks are relatively general, while others are more specific to OpenNetMon, but even these give evidence that a monitoring approach should take security aspects into consideration.

In a smart grid, secure and accurate monitoring is fundamental. This is because the monitor is a main enabler for the inclusion of advanced communication networks in the electric network. Since a smart grid is a critical system, the cost of wrong monitoring behaviour could be unbearable.

With this in mind, in the following sections we discuss some preliminary ideas on protecting SDN monitoring measurements, with the aim to make security a first class citizen in the design of these platforms. By such, the SDN monitoring service can be a strong enabler for a robust communication system for the smart grid.

4.1 Strategies to secure the network monitor

We consider three types of security guidelines that could contribute to improve the resilience of SDN-based security monitors.

Using traditional, well-proven security techniques. An example of where well-known security techniques could be employed is the protection of probe packets to avoid the attacks to delay measurements we demonstrated in Section 3.3.2. A possible solution

consists in protecting these packets from tampering. Conventional solutions based on authentication and integrity protection of packet contents (e.g., with a MAC) may prevent many classes of such attacks without sacrificing performance in a significant way.

Using SDN holistic view. SDN offers a global view of the network that could be explored by monitoring solutions. By correlating measurement information collected from different network locations it becomes possible to identify where and when attacks are occurring. A simple validation to avoid attacks on throughput measurements would be to check that the throughput at network ingress is the same at network egress for transit traffic.

Enhancements to switch design. Switch design could also be enhanced not only to improve the support for monitoring (as in OpenSketch [41]) but also to increase security. For example, certain packets could be securely timestamped as they leave or enter a switch by request of the controller. This would enable more precise delay measurement (e.g., PTP synchronization [8] could be explored) and ultimately raise the attacker effort.

4.2 Traditional security techniques

In this section we show how conventional techniques can be used to mitigate several types of attacks, namely path delay, throughput, and DoS attacks.

4.2.1 Path delay probing techniques

Active monitoring is the predominant strategy for measuring delay between network switches. The controller inserts monitoring traffic in the data plane directed back to itself, and keeps track of the departure and arrival moments of this traffic in order to calculate the path delay. This kind of monitoring is usually called probing.

Problem #1: The first attack is probe modification (Figure 4.1). In this attack an adversary (of type A2) eavesdrops the traffic flow between both switches, and modifies the passing probes. This modification affects the latency calculated by the controller. We performed this attack and explained it in further detail in Section 3.3.2.

Proposed solution: Programming the controller to add integrity protection to each transmitted probe packet in order to detect and discard incorrectly modified data. This can be achieved by inserting a MAC in the payload of each probe packet. This solution

requires cryptographic key derivation to be configured in the controller.

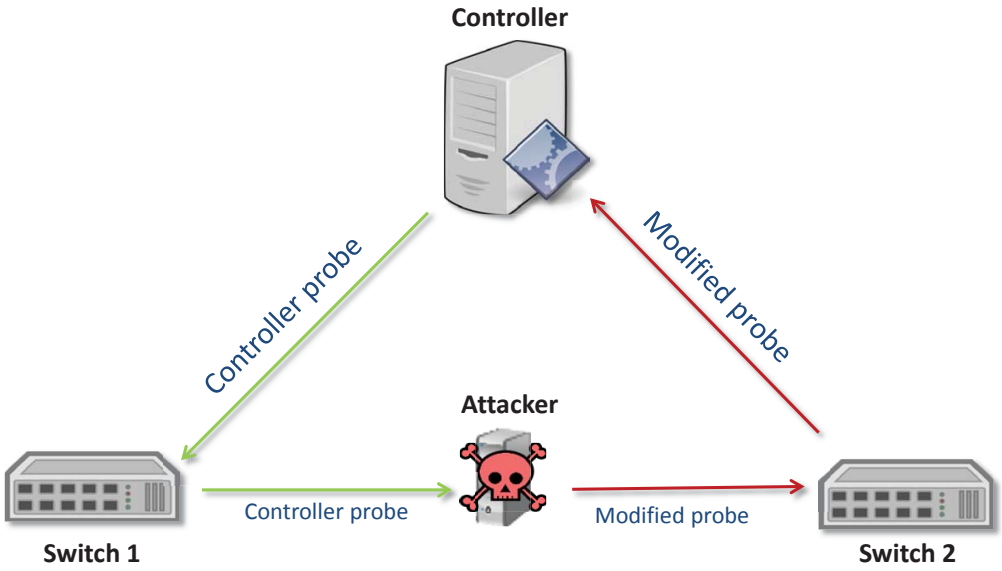

Figure 4.1: Attack by probe modification.

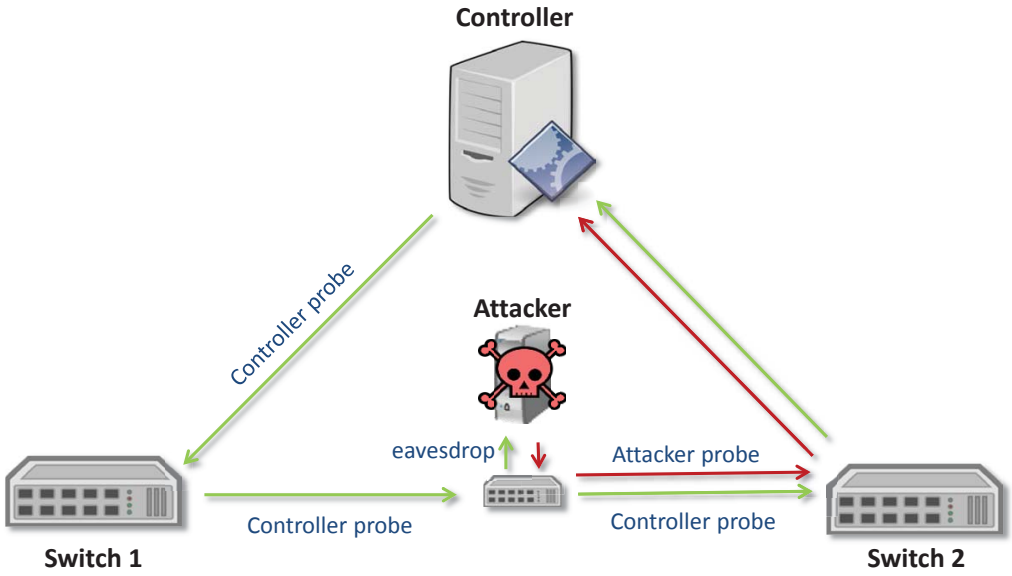

Figure 4.2: Attack by replicating probe.

Problem #2: A second attack is the replay of probe packets (Figure 4.2). In this attack an adversary (of type A2) eavesdrops the traffic flow between both switches. The adversary collects a probe and later retransmits it to the receiving switch (which then forwards it to the controller). The consequence of this attack is the controller mistakingly considering the received probe as a legitimate one and incorrectly calculating the latency.

Proposed solution: A solution is to program the controller to add a unique identifier to each transmitted probe in order to avoid repeated latency calculations. For this purpose a nonce could be inserted in the payload of each probe packet. This nonce should be modified in each probe.

4.2.2 DoS attacks

Problem: If the attacker sends many distinct packets to the switch, which redirects them to the controller, then the switch-to-controller link can become congested. A possible effect is that other legitimate traffic may be dropped (as seen in Figure 4.3).

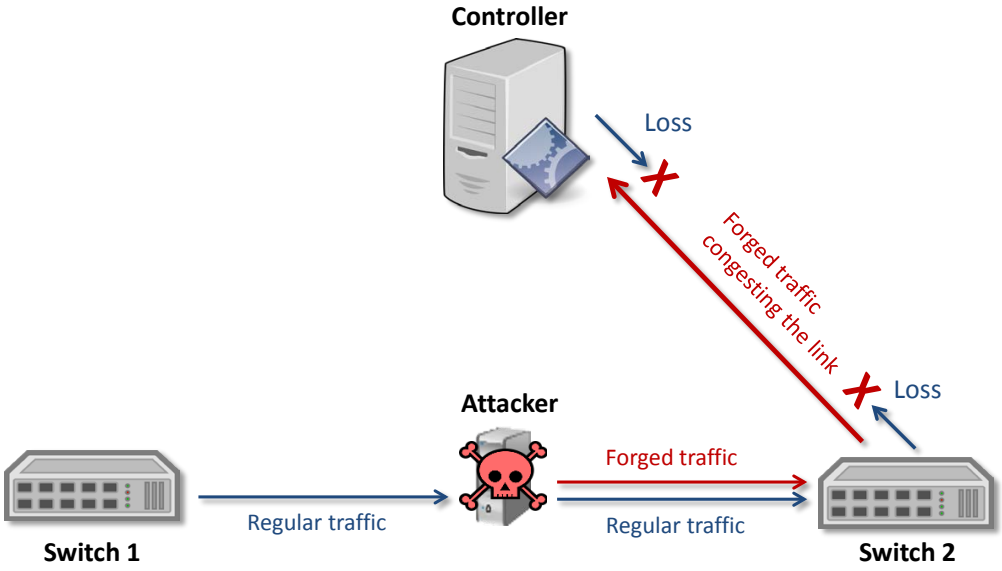

Figure 4.3: A2 adversary attacking the switch-to-controller link.

Proposed solution #1: One possible solution, but not ideal, is to install a meter in the probe-packet flow rule of every switch. This meter limits the rate of traffic sent to the controller. This solution reduces the traffic going to the controller, but the switch buffers still get filled and drop legitimate probes.

Proposed solution #2: To avoid this kind of attack, the best option is to configure all switch tables to drop all packets unmatched in the flow table, adopting a proactive approach. This way a controller installs previously the allowed flows on the switches and discards the rest.

4.3 Using SDN holistic view

Some kinds of attacks can be detected by correlating information retrieved from different network locations. As packets traverse different switches, information per traversed packet can be collected in each switch, and later transmitted to the controller for comparison. If the controller finds a discrepancy when comparing the data (e.g., switch counters or packet contents), that is evidence of a possible attack. Typically, the more complete the information received by the controller, the more effective is the protection and the more powerful the attacks that can be detected.

4.3.1 Correlating switch counters

The logical centralization of control offered by a SDN controller allows the correlation between counters from different switches to be used to detect attacks on throughput. Solutions similar to anomaly detection systems can be used, by comparing flow statistics of different switches to detect values that do not conform to an expected range.

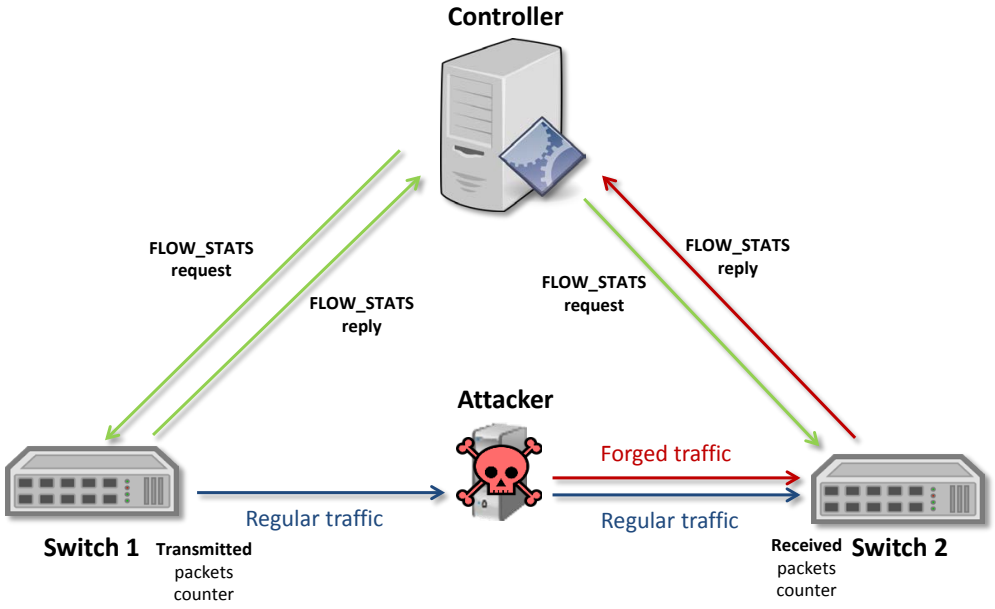

Figure 4.4: A2 adversary, attacking counters.

In throughput measuring, one easy step to avoid major problems would be to perform correct packet counting and not skip any packets (as OpenNetMon does). We explored this vulnerability and demonstrated it in further detail on Section 3.3.3. However, received throughput from a single switch is not a good indicator of link throughput if the

transmitted throughput from the link's entry switch does not match the former one (within a certain expected range).

Problem #1: Measuring link throughput by using byte/packet counters is vulnerable to attacks that affect the traffic between the two switches composing the link (Figure 4.4).

Proposed solution: Compare switch throughput from both switches at the endpoints of the link. When taking periodic "snapshots" of switch throughput from switch 1 and 2, it is expected that both values are within a certain range of each other. If the received throughput from switch 2 is higher than the transmitted throughput from switch 1, that is evidence of a malicious packet injection in the link.

4.3.2 Correlating sampled packets

The best monitoring approach would probably be to adopt a hybrid passive/active network monitoring. The idea is to use network traffic as probes. This can be achieved through packet sampling, as seen in Figure 4.5. The main issue is the need to sample the same packets on switch 1 and switch 2, and identify each packet. If the probe can be hidden in the normal forwarded traffic, we are not only solving latency measurements, but also providing an attack detector for statistics measurements. This happens because even packet modification would be detected.

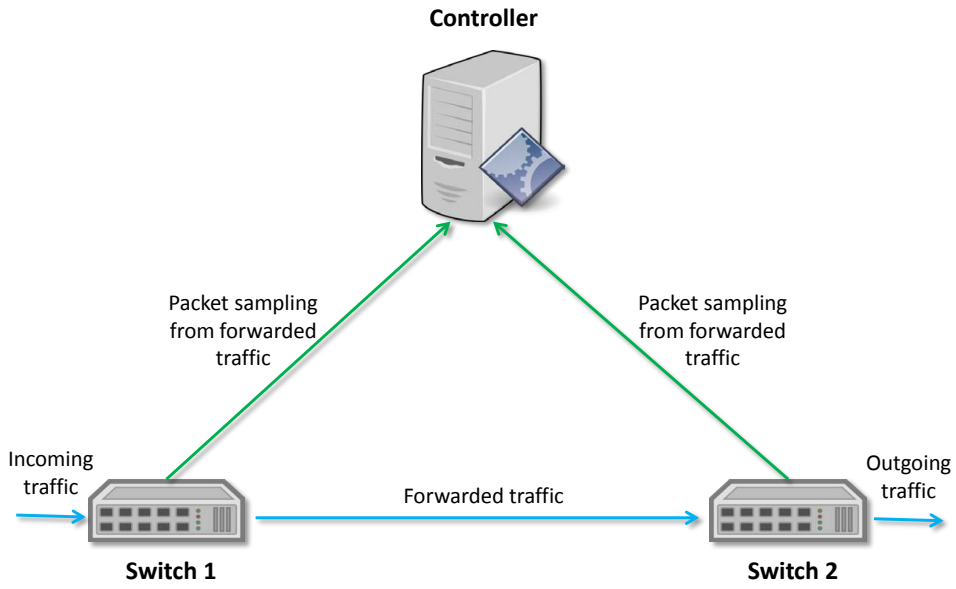

Figure 4.5: SDN network simple sampling.

Problem #1: A non-sophisticated man-in-the-middle attack (Figure 4.6), performed

by an A2 adversary, is what we call a probe XOR traffic delaying. In this attack the adversary delays all traffic but the probe packets. As a consequence, the delay measured for all data traversing that path will be lower than what it is in reality. Alternatively, if the adversary delays the probe packets instead, then the controller will calculate a delay that is higher than what it is in reality.

Proposed solution: This problem can be overcome by re-using normal network traffic for monitoring measurements (alongside the probe measurements), using a sampling approach.

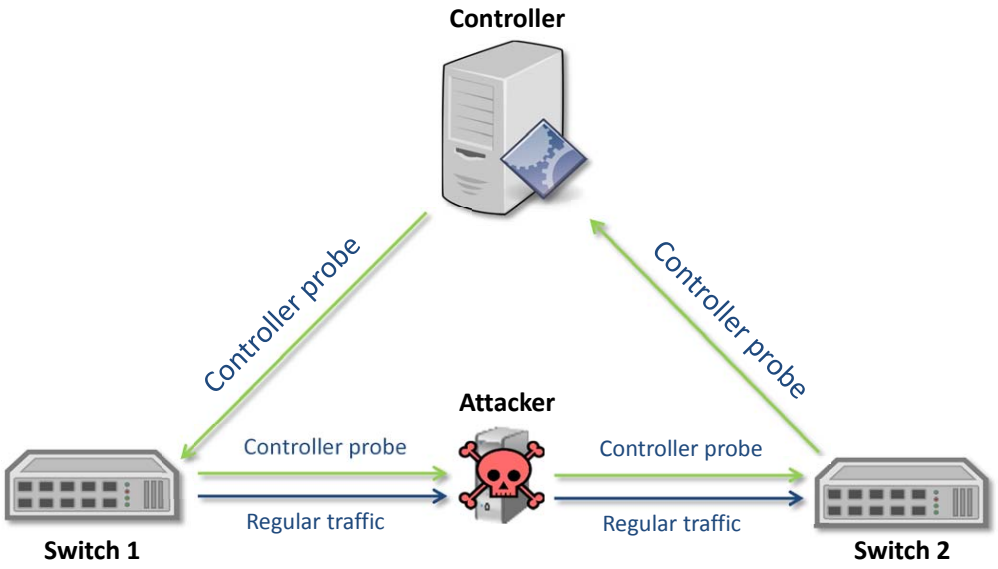

Figure 4.6: Man-in-the-middle attack.

In order to select which packets to use as samples, a sampling algorithm must be implemented in the switches. The idea is to transmit only a subset of traffic packets to the controller so that sampling does not incur too much load on the controller and its links to the switches. A simple strategy is to periodically request the switches to forward traffic to the controller during some specified time, i.e., once in a while a portion of the traffic forwarded by the switches is mirrored to the controller. The sampling periodicity and duration can be set to random values within certain ranges, in order to make it hard for the attacker to predict the sampling frequency. Additionally, the attacker must not be able to easily distinguish the sampled traffic from the non-sampled one, since otherwise she could attack the links only at times when sampling is disabled.

Proposed implementation: A possible solution is to use Multiprotocol Label Switching (MPLS) tags to identify traffic to be sampled. MPLS tags can be appended to or removed from any forwarded packets by the switches. Looking at Figure 4.5, the idea is for

switch 1 to append tags to incoming traffic and switch 2 to remove them from outgoing traffic. This way we allow packet identification to be non-interfering with other transport or routing protocols, while keeping it exclusive to each link. MPLS tags must always be used even with non-sampled traffic, in order to hide from the attacker which traffic is being sampled. MPLS tags must change at random times so that the sampling frequency is difficult to predict by the attacker.

Applying our idea, on switch 2, a flow rule that matches a specific MPLS tag is installed at random times with a random defined timeout, in which packets with that tag are mirrored to the controller. On switch 1, a similar flow rule that matches the same MPLS tag is installed immediately after the other one, in which packets are labeled with the specific tag and also mirrored to the controller.

The controller receives both traffic streams from switches 1 and 2 (which have the same MPLS tag) and compares them by value in order to detect packet corruptions or losses. Additionally, the controller also obtains timestamps whenever sampled packets are received, which allows an estimation of link delay and throughput (after taking into account switch-to-controller delays).

4.4 Enhancing switch design

The addition of packet authentication/integrity protection at switches would bring higher security. This addition ensures that network traffic modification does not go unnoticed. Forged packets are not accepted as valid, which also helps preventing DoS attacks in a proactive monitoring approach.

A couple of recent proposals that aim to increase the flexibility and programmability of SDN switches could be very useful for this purpose. The first such proposal was OpenSketch [41], a framework that adds reconfigurable measurement logic to switches and exposes an interface to program it. The data plane of OpenSketch switches use a three-state pipeline (hashing, classification and counting), enabling the support of many measurements tasks. In the control plane, OpenSketch provides a measurement library that configures this pipeline and allocates resources for the different measurement tasks. Such framework can be used not only to improve switch statistic measurements, but also to detect attacks on the network. In [41] the authors exemplify with a DDoS attack — by detecting if a host is contacting more than k unique destinations during a certain time interval.

The OpenSketch framework has recently been prototyped using the P4 language. P4 [9] allows the functionality of programmable switches to be not only specified by the controller but also changed in the switches. This allows programmers to decide how the forwarding plane processes packets without caring about implementation details. The P4 compiler transforms the imperative program into a control flow graph that can be mapped to different target switches. This capability grants the possibility of using a wide range of actions for specific packets. For instance, TTL fields can be decremented and tested, new tunnel headers can be added, and checksums can be computed. To increase the security of monitoring platforms authentication and integrity could be integrated in the switches using P4, by adding identifiers or by signing specific packets.

Chapter 5

Conclusion

In this dissertation we experimentally demonstrated how SDN-based monitoring platforms are vulnerable to attacks, including relatively trivial ones. Considering the increasing importance of these infrastructures in modern network deployments (such as Smart Grids), these vulnerabilities illustrate the need to consider security as a first class citizen in the design of future monitoring platforms.

We have demonstrated attacks that caused the monitoring components to report invalid measurement data. The attacks were concentrated on the two most used metrics in practice today: path delay and link throughput. The attacks were performed with different adversary capabilities.

We have also discussed some security guidelines and solutions that can be incorporated in such designs to mitigate specific classes of attacks. One of these solutions is to use traditional, well-proven security techniques. Another solution is to resort to the SDN holistic view, which allows correlating switch counters and sample packets. Finally, we can mitigate these specific attacks by enhancing switches design, adding packet authentication/integrity on the switches.

We hope this work will trigger the community to investigate on SDN-based network monitoring security flaws and – more importantly – to come up with effective solutions, as building a secure and accurate monitoring system is fundamental to critical systems as the Smart Grid.

Future work includes the implementation of a secure monitor that considers the previously mentioned techniques. Using SDN holistic view and investigating how to implement the security techniques on the network switches (e.g., using P4) are interesting avenues of work.

Bibliography

[1] https://www.opennetworking.org/sdn-resources/openflow.

[2] http://www.noxrepo.org/.

[3] http://www.projectfloodlight.org/floodlight/.

[4] https://www.osgi.org.

[5] http://www.mininet.org/.

[6] https://ettercap.github.io/ettercap/.

[7] http://www.secdev.org/projects/scapy/.

[8] IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. *IEC 61588:2009(E)*, pages C1–274, Feb 2009.

[9] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[10] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple network management protocol (snmp), 1990.

[11] S.R. Chowdhury, M.F. Bari, R. Ahmed, and R. Boutaba. Payless: A low cost network monitoring framework for software defined networks. In *IEEE NOMS*, 2014.

[12] David Erickson. The beacon openflow controller. In *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, HotSDN '13, pages 13–18, New York, NY, USA, 2013. ACM.

[13] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. *SIGCOMM Comput. Commun. Rev.*, 34(4):245–256, August 2004.

[14] H. Farhangi. The path of the smart grid. *Power and Energy Magazine, IEEE*, 8(1):18–28, January 2010.

[15] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: Towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[16] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. In *ACM CoNEXT*, 2012.

[17] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven WAN. In *ACM SIGCOMM*, 2013.

[18] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. B4: Experience with a globally-deployed software defined WAN. In *ACM SIGCOMM*, 2013.

[19] Vimalkumar Jeyakumar, Mohammad Alizadeh, Yilong Geng, Changhoon Kim, and David Mazières. Millions of little minions: Using packets for low latency network programming and visibility. In *ACM SIGCOMM*, New York, NY, USA, 2014. ACM.

[20] L. Kekely, J. Kucera, V. Pus, J. Korenek, and A.V. Vasilakos. Software defined monitoring of application protocols. *Computers, IEEE Transactions on*, PP(99):1–1, 2015.

[21] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-Defined Networking: A comprehensive survey. *Proc. of the IEEE*, 2015.

[22] Myungjin Lee, Nick Duffield, and Ramana Rao Kompella. Not all microseconds are equal: Fine-grained per-flow measurements with reference latency interpolation. In *ACM SIGCOMM*, 2010.

[23] Myungjin Lee, Nick Duffield, and Ramana Rao Kompella. MAPLE: A scalable architecture for maintaining packet latency measurements. In *ACM IMC*, 2012.

[24] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *ACM IMC*, 2006.

[25] M. Malboubi, Liyuan Wang, Chen-Nee Chuah, and P. Sharma. Intelligent sdn based traffic (de)aggregation and measurement paradigm (istamp). In *IEEE INFOCOM*, 2014.

[26] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[27] Masoud Moshref, Minlan Yu, and Ramesh Govindan. Resource/accuracy tradeoffs in software-defined measurement. HotSDN. ACM, 2013.

[28] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Dream: Dynamic resource allocation for software-defined measurement. In *ACM SIGCOMM*, 2014.

[29] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado. The design and implementation of open vswitch. In *USENIX NSDI*, 2015.

[30] James S. Plank, Micah Beck, Gerry Kingsley, and Kai Li. Libckpt: Transparent checkpointing under unix. In *Proceedings of the USENIX 1995 Technical Conference Proceedings*, TCON'95, pages 18–18, Berkeley, CA, USA, 1995. USENIX Association.

[31] Fernando M. V. Ramos, Diego Kreutz, and Paulo Verissimo. Software-defined networks: On the road to the softwarization of networking. *Cutter IT Journal*, 2015.

[32] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale monitoring and control for commodity networks. In *ACM SIGCOMM*, 2014.

[33] Sajad Shirali-Shahreza and Yashar Ganjali. FleXam: Flexible sampling extension for monitoring and security applications in openflow. In *ACM HotSDN*, HotSDN, 2013.

[34] Junho Suh, T.T. Kwon, C. Dixon, W. Felter, and J. Carter. OpenSample: A low-latency, sampling-based measurement platform for commodity SDN. In *IEEE ICDCS*, 2014.

[35] Peng Sun, Minlan Yu, MichaelJ. Freedman, Jennifer Rexford, and David Walker. Hone: Joint host-network traffic management in software-defined networks. *Journal of Network and Systems Management*, 23(2), 2015.

[36] Amin Tootoonchian, Monia Ghobadi, and Yashar Ganjali. Opentm: Traffic matrix estimator for openflow networks. In *PAM*, 2010.

[37] Niels L. M. van Adrichem, Christian Doerr, and Fernando A. Kuipers. OpenNetMon: Network monitoring in openflow software-defined networks. In *IEEE NOMS*, 2014.

[38] Han Wang, Ki Suh Lee, Erluo Li, Chiun Lin Lim, Ao Tang, and Hakim Weatherspoon. Timing is everything: Accurate, minimum overhead, available bandwidth estimation in high-speed wired networks. In *ACM IMC*, New York, NY, USA, 2014. ACM.

[39] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V. Mahdyastha. Software-defined latency monitoring in data center networks. In *PAM*, 2015.

[40] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. FlowSense: Monitoring network utilization with zero measurement cost. In *PAM*, 2013.

[41] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with OpenSketch. In *USENIX NSDI*, 2013.

[42] Ye Yu, Chen Qian, and Xin Li. Distributed and collaborative traffic monitoring in software defined networks. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 85–90, New York, NY, USA, 2014. ACM.

[43] Ying Zhang. An adaptive flow counting method for anomaly detection in sdn. In *ACM CoNEXT*, 2013.