Contextual decision making and action enforcing applications in wireless networks and IoT using SDN as a platform

by

Maryam Karimi

Master, Shiraz University of Technology, 2015

Submitted to the Graduate Faculty of

the Computing and Information Science in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2022

UNIVERSITY OF PITTSBURGH

SCHOOL OF COMPUTING AND INFORMATION

This dissertation was presented

by

Maryam Karimi

It was defended on

April 1, 2022

and approved by

Dr. Prashant Krishnamurthy, School of Computing and Information

Dr. David Tipper, School of Computing and Information

Dr. Martin Weiss, School of Computing and Information

Dr. Mai Abdelhakim, Department of Electrical and Computing Engineering

Dissertation Director: Dr. Prashant Krishnamurthy, School of Computing and Information

Contextual decision making and action enforcing applications in wireless networks and IoT using SDN as a platform

Maryam Karimi, PhD

University of Pittsburgh, 2022

With the rise of the Internet of Things (IoT), new challenges and opportunities have emerged, providing the potential to improve living standards, higher efficiencies and lower costs. A variety of wearable devices and sophisticated, yet not very expensive connected devices are now able to gather different kinds of information (mostly) on their premises and enable people to make decisions or control actions. The complexity of such decisions need to be balanced with performance tradeoffs. Different applications may need different levels of service that require different communication, storage, and service costs. A suitable architecture that can employ data driven decisions for performance improvements comprises of software defined networks (SDN) and software defined perimeter (SDP). SDN and SDP are suitable platforms due to their ability to have a view of the network traffic for implementing the services that require gathering information from the context and performing contextual decision making while balancing performance tradeoffs, managing specific parameters, and enforcing actions based on them. This dissertation examines three case studies that look at such performance tradeoffs.

In the first case study, we explore the use of historical data in WiFi networks to create a classification QoS decision tree that predicts the maximum delay due to specific traffic situations with specific context parameters and makes rapid decisions possible to manage wireless network resources considering quality requirements. We use OpenFlow network access and gathering necessary context in wireless networks. The tradeoffs here involve balancing traffic flows from WiFi access points to meet latency constraints of end devices.

In the second case study, we investigate contextual integrity verification in IoT where we look at "levels" of integrity verification. A variety of IoT devices may be required to outsource sensed or generated data to multiple heterogeneous cloud servers. We posit that it is the Data Owner's responsibility to verify whether the stored data remain unchanged. How-

ever, the "level" of this verification may be different under different contexts. We propose four typically disparate methods of integrity verification and consider the "toll" in terms of time, storage and communication to decide on a suitable data integrity verification process. We adapt the notion of contextual privacy to extract important parameters that determine the right level of integrity to be applied to data blocks. Such contextual information can be extracted in a SDN while security context information and a secure infrastructure for authentication and communication is possible through a secure architecture with the integration of SDP and SDN.

Finally, in the third study, we investigate "levels" of reliability of contextual data along with the use of Bayesian Regression to enhance decision making and determine how critical each context variable is and how it would affect choosing appropriate parameters for action to perform. Again, SDN and SDP can provide the contextual information needed to maintain the various reliability levels.

Table of Contents

Acronyms

ADI	Ambit of Data Integrity
ADR	Ambit of Data Reliability
AH	SDP Accepting Host
AP	Access Point
BDD	Binary Decision Diagram
BER	Bit Error Rate
BF	Bloom Filter
BW	Bandwidth
CBF	Concatenated Bloom Filter
CTRL	Controller
CTRL	SDP Controller
d-CFF	d-Cover-Free Family
DDoS	Distributed Denial of Service
DoS	Denial of Service
DP	Differential Privacy
DPDP	Dynamic Provable Data Possession
ERM	Empirical Risk Minimization
FEC	Forward Error Correction
FN	False Negative probability
FP	False Positive probability
GBF	Generalized Bloom Filter
GW	Gateway
HAIL	High Availability and Integrity Layer
HCI	Human Computer Interaction
HBS	Hash based sampling
HAIL	High availability and integrity layer
IH	SDP Initiating Host
IoT	Internet of Things

IP	Inetrnet Protocol
IP-ECC	Integrity Protected Error Correcting Code
KDEs	Kernel density estimate
L2	Layer 2
LSH	Locally Sensitive Hash
LTE	Long-Term Evolution
MBR	Minimum bandwidth regenerating
MDS	Maximum Distance Separable
ML	Machine Learning
MSR	Minimum storage regenerating
OG or OGW	Owner's Gateway
ONF	Open Network Foundation
OVS	Open Virtual Switch
PCA	Principal Component Analysis
PDP	Provable Data Possession
POR	Proof of Data Retrievability
PPT	Probabilistic Polynomial Time
QoS	Quality of Service
RACE	Repeated Array of Count Estimators
RS	Reed-Solomon codes
RS	Random sampling
SDN	Software Defined Network
SDP	Software Defined Perimeter
SG	Service's Gateway
SINR	signal-to-interference ratio
SLA	Service-Level Agreement
SMR	State Machine Replication
SOM	Self-Organizing Map
SPA	Single Packet Authentication
SKA	Sparse kernel approximation

SVM	Support Vector Machine
SW	Switch
TBF	Temporary Simple Bloom Filter
TLS	Transport Layer Security
UGW	User's Gateway
VM	Virtual Machine
VPN	Virtual Private Network

List of Tables

List of Figures

Preface

Throughout my childhood, I loved the night sky and I never liked sleeping and thought of it as a waste of time while there is still much to be learned, enjoyed, and discovered far far away from us. I spent many hours poring through books and watching documentaries. I volunteered for all scientific opportunities and competitions in school and at university, I worked on extracurricular projects and competitions with my teammates in the summer. Now, I'm burning yet another candle at 6 am to finish the last part of my dissertation. This isn't the end of my journey. I just learned how to perform scientific research and now I have to put it to work. It is my plan to continue learning, growing, and discovering.

It has been a pleasure working with, learning from, and receiving the support, time, effort and guidance from people who have helped me get here. I can't adequately express my gratitude for meeting them and having the chance to work with them. First and foremost, I would like to express my gratitude for the excellent support and guidance provided by my advisor Prof. Prashant Krishnamurthy during this process. Prof. David Tipper, Prof. Martin Weiss, and Dr. Mai Abdelhakim were excellent to work with and provided great feedback, I would like to deeply appreciate their time and support. In addition, I would like to thank Dr. James Joshi, Dr.Rosta Farzan, Dr. Paul Munro, Dr. Yuru Lin, Dr. Hassan Karimi and all the other wonderful professors and instructors with whom I had the pleasure of working and learning.

My friends and family have also been a significant help to me throughout these years. I would like to thank my husband, Omid, for all the love and support that helped me get through difficulties. I would like to thank my parents, without their support this entire process would not be possible. Also, I would like to thank my colleagues Alekhya and Stephenie for making school an enjoyable environment.

1.0 Introduction

There are many situations in wireless networks such as Wi-Fi and in the Internet of Things (IoT) where decision making can be used to *improve performance depending on the context*. A huge variety of smart applications can be classified as "decision making and action enforcing applications". Such a class of applications may utilize historical information and (or small) data and employ machine learning, Bayesian inference, data mining, artificial intelligence and other techniques to build a decision maker model. Then the applications can use the context information as an input to this model to detect a situation or predict one, and decide on an action to be enforced considering the requirements of the detected situation. Such applications require an architecture that gathers (i.e., in a network of devices) context information and provides the flexibility to enforce the proper action (again on devices in the network). Software Defined Networks (SDN) and Software Defined Perimeter (SDP) together form a suitable secure platform for implementing services that require gathering information from the context and performing contextual decision making while managing specific parameters and enforcing actions based on them. We chose SDN because it is able to have a global view of a (local) network and the flexibility it affords. We chose SDP due to its unique security features and its architecture compatibility with SDN (explained in details in Chapter 2). We then use this architecture in three different scenarios to show examples of its applicability with proofs of concept.

- In Chapter 3, we consider using historical data to reduce latency for certain flows. We build a classification tree that can be used to make decisions toward handling packet flows. Such a classification tree can be embedded into an SDN controller to manage resources in a WiFi network.
- In Chapter 4, we consider the problem of efficient integrity verification (with healthcare IoT devices as an example, although this approach may be used for other IoT applications). We develop mechanisms that tradeoff between latency and quality of integrity verification and communication costs, storage, and computation. Again, such tradeoffs can be implemented using an SDN/SDP architecture.

- In Chapter 5, we consider decision making for reliably storing data for IoT applications using appropriate combinations of coding and redundancy. Such decisions can be implemented using an SDN/SDP architecture.

We discuss these in more detail below and in the respective chapters.

1.1 SDN and SDP in Brief

Software Defined Networking (SDN) is a promising solution for next generation networks (including 5G wireless) [156]. SDNs provide fine grained network management by splitting the data plane from the control plane and push the control plane off of the switches and routers to the controller software. The programmability of the control plane results in enhancing flexibility by enabling rapid ability to applying new policies (for routing, management, security, etc.) or improving *performance* - an objective in this dissertation - or adding features and services and improving system-wide intelligence [134, 118]. A generic SDN's architecture is shown in Figure 1. An SDN switch's routing table is called the flow table. When a switch receives a new flow, it compares the flow against its' flow table. Each entry in the flow table consists of two parts: header and action (forward, drop, modify, ...). If the flow header matches the table, the switch applies the proper action based on the flow table; otherwise, it sends the flow to the controller. The controller uses the global view of the network and creates a new rule based on policies; then, it installs the new rule in the appropriate switches [87]. IoT, cloud computing, and data centers are among applications that are using SDNs [139].

The control center in the SDN gathers information about the network and device status and stores a global view of the network. Some SDN controllers such as POX [6] also provide messengers that can exchange information with the installed agent on switches to gather further required information such as power, packet rate, packet size, etc. Such information can be utilized to solve many challenges (e.g., interference management in wireless networks, sensor resource management in sensor networks, etc.), improving performance and reducing costs and overall leading to more efficient and in some cases innovative automatic decision

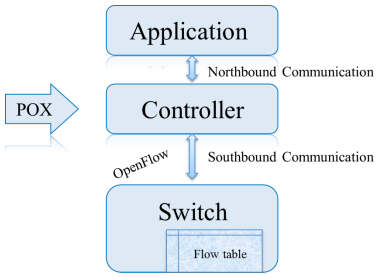

Figure 1: SDN simple architecture [86]

making solutions. SDN also provides flexibility both by providing programmability and by installing sets of match and action – and in some cases statistical – rules in switches. These rules enable enforcing the decided actions.

SDNs however, have many security issues. The SDN central controller is a single point of failure and is vulnerable to attacks from malicious network elements. In addition to the controller, applications, switches (including access points and gateways), host and servers and other elements in an SDN network are vulnerable to attacks such as credential harvesting, taking over the device, man-in-the middle, denial of service (DoS) and many other failures and attacks, as shown in Figure 2, and examined in detail in Chapter 2. These attacks can be mitigated by providing security services such as authentication, confidentiality, access control, identification, availability and integrity. A Software Defined Perimeter (SDP) can meet most of these requirements. It identifies and authenticates all network elements before allowing them to connect to the network and secures their communications using protocols such as Virtual Private Network (VPN) and Transport Layer Security (TLS). We suggest integrating SDN and SDP controllers into a single controller, and to integrate SDN and SDP switches for our purposes. The architecture is depicted in Figure 3. As can be seen, first the SDN-SDP controller authenticates switches and then SDN-SDP switches (or gateways) forward hosts and server authentication to the controller; any other request from host and servers before authentication will be dropped which protects the controller and other elements from malicious elements.

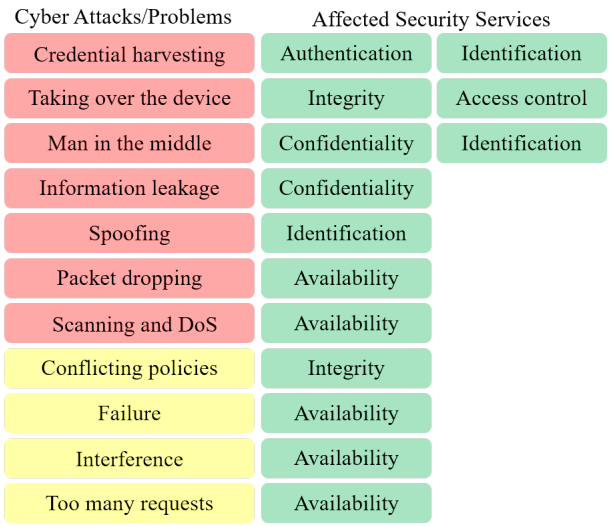

Figure 2: Attacks (red box) and Malfunctions (yellow boxes)

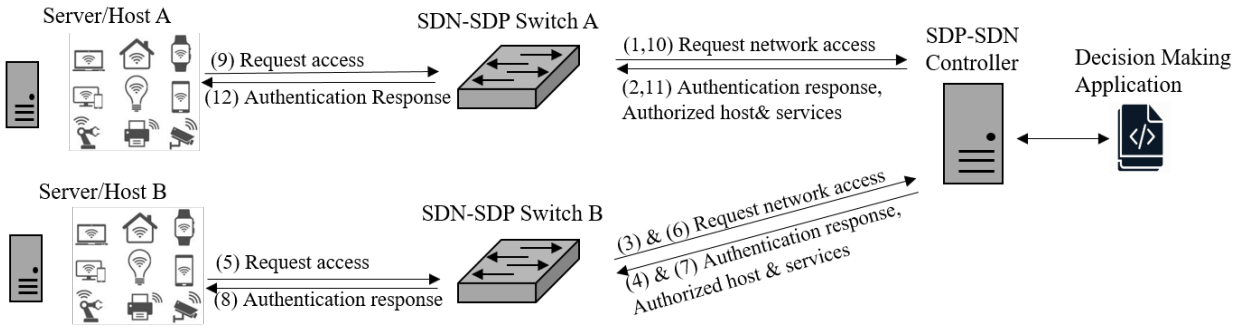

Figure 3: SDN-SDP architecture

1.2 Overview of Dissertation

Using the SDN-SDP framework as the underlying platform, the purpose of this dissertation is to provide examples of how the framework can support *performance improvements and tradeoffs* in wireless and IoT applications. The examples are in-depth - and consider resource management, data integrity verification and data reliability applications. We use this architecture in three different scenarios to show possible applications with proofs of concept. In each case, we evaluate example tradeoffs through analysis and modeling with real or synthetic data.

1.2.1 Interference and Delay Tradeoffs

In the first case study, we investigate "Interference Management" in a multi-variable environment and consider maintaining latency and throughput in wireless networks as the application (Chapter 3). WiFi networks often seek to reduce interference through network planning, macroscopic self-organization (e.g., channel switching), or network management. We explore the use of historical data to create a classification QoS decision tree that predicts the maximum delay due to specific traffic situations with specific context parameters. Such a decision tree can be embedded in an SDN controller where the decision tree makes possible rapid decisions to manage wireless network resources based on the predicted maximum delay and the quality requirements based on a service level agreement or other criteria provided by a network administrator. OpenFlow cannot directly provide the necessary contextual information for managing wireless networks – so we use an agent on each AP for adjusting the network access. We explore the possibility of updating the tree using feedback from hosts. Our results show that such trees are effective in managing the network and decreasing maximum packet delay.

1.2.2 Contextual Data Integrity

The second application that we investigate is contextual integrity verification in IoT (Chapter 4). The Internet of Things (IoT), comprising of sensors and actuators, is increas-

ingly changing human life by helping to solve new challenges that can lead to a better quality of life, higher efficiencies through less waste, and reduced costs. A variety of wearable devices ("things" in the Internet of Things - IoT) and sophisticated, yet not very expensive connected devices gather different kinds of data that is generated (mostly) on their premises about the environment, the human body, social activities, etc., and enable people to make decisions or control actions. The complexity of such decisions, especially where performance tradeoffs are possible, needs a suitable architecture that can employ data driven decisions for performance improvements.

On the Internet of Things (IoT), devices do not have the required computational power and storage capacity; and as a result, a variety of IoT devices may be required to outsource sensed or generated data to multiple heterogeneous cloud servers. We posit that it is a "Data Owner's" responsibility to verify whether the stored data remain unchanged when the owner or some trusted third party further requires accessing this data. However, the "level" of this verification may be different under different contexts based on the application need. We propose four methods of integrity verification (which we call the ambit of data integrity – ADI) that considers the "toll" in terms of time, storage and communication by enlisting typically disparate integrity approaches under a single orbit. The four methods includes the ones we designed, namely nested Bloom filter and hash tree, and we compare them with two previous approaches: provable data possession (PDP) and proof of data retrievability (POR) which are discussed in Chapter 4 in detail. The nested Bloom Filter and hash tree are most efficient when there is a third party (it can be a service provider to a data owner e.g., healthcare provider to a patient) retrieving the data and enabling the data owner to verify the data without actually having to send the data to the owner. PDP is useful when the data owner does not trust the cloud server and wants to check the integrity of data once in a while without having to download it. POR stores parity data and redundancy along with the message authentication code (MAC) of the data. POR is useful when the data is super critical and important and losing the data has catastrophic consequences; therefore, the owner not only wishes to check the integrity of data but also wants to be able to retrieve the data if any part is lost.

We adapt the notion of contextual integrity, previously used for assessing privacy grants,

to extract important parameters required to decide on a suitable data integrity verification process. In this scenario in addition to the contextual information that SDN provides, we need both security context information for decision making and also a secure infrastructure for authentication and communication; therefore, the proposed secure architecture using an integration of software defined perimeter (SDP) and software defined network (SDN) to perform authentication can gather each partition's context information for an SDN application to decide the proper integrity verification method that addresses the context requirements.

Here is a summary of how this contextual integrity is used in SDP-SDN platform. In SDP-SDN network, first all hosts, switches and servers are authenticated before communication, then the owner gateway(SDP-SDN switch) sends the data storing request to the controller, the controller sends the request along with the context information to the SDN-application to perform the decision making about the verification method that should be used. The application replies to the controller, the controller relays the response to the owner gateway and installs the route from owner gateway to the server gateway on corresponding switch and gateways. The agent installed on owner gateway uses the verification method decided by the application to add the required metadata (bloom filter, hash tree, PDP tag, POR tag and parity) and send the data and metadata to the cloud server to be stored.

To the best of our knowledge, this is the first time that the scope of integrity (or the data context) is used to determine the required layer of integrity verification in IoT.

1.2.3 Contextual Data Reliability

Finally, In the third scenario we used the SDN/ SDP framework to improve data reliability. We use Bayesian inference with specific datasets to improve the *performance* of decision making and automate the detection of how critical each context is and how the context would affect the probability of choosing a specific action to be enforced. The reliability method we suggest first summarizes the data using what is known as RACE sketch [39], then it hashes the data using a hash tree that is used to detect data "unavailability" and then we add an erasure code based on network coding to ensure data availability and retrievablity and to build any lost data packets. Bayesian learning may help the application detect which

contextual parameter value belongs to critical tasks and increase the probability of choosing a better specific level of summarizing and availability probability.

1.3 Organization of the Dissertation

In this proposal, in the following, Chapter 2 summarizes the literature in the Software Defined Network (SDN), SDN issues, literature review on provided solutions and then investigate Software Defined Perimeter (SDP) and provide a framework for integration of SDN and SDN which resolves SDN security problems, provide required information for contextual decision making and provide flexibility for action enforcing. Chapter 3 explains how SDN can help to gather contextual information required for managing wireless interference, use a decision tree; which is created using historical data, to predict the interference and then enforce action (reduce rate) if required. Chapter 4 explains the idea of using contextual integrity to enable the owner to decide what layer of integrity and therefore what layer of integrity validation is required based on the ambit and the decision tree and use that integrity verification method to store the data. Chapter 5 investigates data summarizing and retrievability using Bayesian inference and decision making process using context features as variables to estimate the error rate and availability probability and Chapter 6 concludes the dissertation.

2.0 Software Defined Secure Framework

In this dissertation, we design a suitable platform for implementing the services that requires gathering information from the context, performing contextual decision making, managing the parameters and enforcing actions based on them. The examples of these services we implement are interference management, integrity validation enforcing process, and data reliability mechanisms. We use an integration of SDN and SDP based architecture that accounts for the network requirements and gathers context information for a decision maker component that uses this information to chooses a suitable integrity verification method or manage interference or data reliability. In this chapter, SDN and SDP are comprehensively examined. We first review security challenges in SDN and explain how SDP can resolve those issues and how the combination of SDN and SDP provides a secure framework that meets the requirements of contextual decision making and action enforcing through the global network view and flexibility of SDN and security mechanisms of SDP.

In the following, we first review SDN and its security issues and solutions that resolve that single issue and then we review some papers that suggest SDP to resolve the SDN security issues. Later on in Section 2.2.1 we describe SDP and employ it to provide a comprehensive SDN-SDP architecture that resolves SDN security issues as well as meeting contextual decision making application requirements pertinent to this dissertation.

2.1 SDN security issues and solutions

In 2011, the "Open Network Foundation" (ONF) was formed to take the responsibility of leading SDN transformation by improving standardization (e.g., providing OpenFlow [110] interface for communication between data plane and control plane), improving availability, performance and scalability of SDN (e.g., by providing ONOS [16], a physically distributed and logically centralized SDN operating system) and providing an agile and scalable (requir-

ing few building blocks) service delivery platform for the network edge (e.g., CORD[1] [121]) [118]. ONF promotes SDN market for products, services, applications, users and consumers [87]. SDN is standardized by ITU-T [2] on November 2012, by adapting "Resolution 77 - Standardization work in ITU-T for SDN" and the road map for SDN standardization is maintained and is keeping updated by this organization [76].

An SDN decouples the control plane from the data plane, pushes network management, policy application, decision making, security, routing and other services to the programmable controller software. Switches and routers are now simple forwarding devices that receive instructions from the controller, which has the global view of the network status which is required to get the contextual information required for contextual decision making, a focus of this dissertation. When a switch receives a flow, it compares the flow header against the flow table inside the switch – if there is a match, it forwards the flow based on the matched rule; otherwise, it forwards the flow to the controller. The controller with a global view routes the flow and installs the proper rule in the corresponding switches [87]. One advantage of SDN, that we require in action enforcing, is the flexibility of the rules in the flow table. The rules include fields such as match, action and in some cases statistics. The match part is compared with the flow header. The action part in the typical SDN network includes forward, drop, quarantine, etc. For interference management the action is specifying data rate for each AP to avoid collision and for contextual integrity verification application to store/retrieve data we have to specify the correct verification process. Therefore, SDN is the environment that can addresses the distinct needs of contextual decision making and action enforcing applications. As shown in Figure 4 layer 2 controllers gathers information and are coordinated by main controller, applications make decisions based on the context information controller gathered. This decisions are towards improving performance (e.g., delay, storage communication, $\cdots$).

Some architectures include middle boxes as an SDN architecture element [126, 155, 88, 28, 60]. In these architectures, some suggest to use middle boxes as applications that the

[1]CORD is a new design of a Telco Central Office (at the edge of operator network) which unifies SDN, NFV (Network function virtualization) and Cloud technologies by re-architecturing a Central Office as a data center [121].

[2]International Telecommunication Union sector of Standardization: The ITU, formed in 1865, produces or revises standards which are necessary for network functionality.

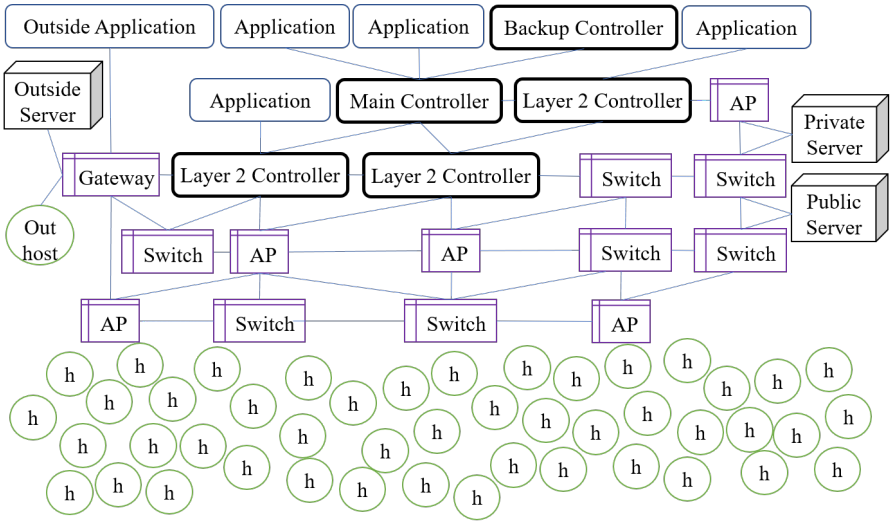

Figure 4: SDN general architecture

controller can trigger and use for making a decision and some suggest to have middle boxes in the lower level in the architecture and choose the action in the switches to forward the packet to these middle boxes if the service it provides, e.g., deep packet inspections, is required. SDN can perform perform load balancing between middle boxes which is a huge problem for middle boxes [126] today, and also is able to update policies and algorithms inside these middle boxes. We do not explicitly consider middle boxes in this dissertation although some of the contextual enforcement of actions may be performed by middle boxes.

Network security is one of the key necessities that helps to mitigate malicious attacks or even natural (benign) disasters [43]. Security issues in SDN have attracted attention since its advent. Although SDN architecture provides the global view of the network that helps traffic analysis and anomaly detection, SDN has many security vulnerabilities. In SDN, open programmability of network elements increases the trust issues between them [139] and against all the benefits that the central controller has, it is a single point of failure; therefore, DoS (Denial of Service) against the SDN controller can cause a lot of trouble; furthermore, when only flow headers are sent to the controller, to save bandwidth, it facilitates DoS attack on the switch nodes (since the flow should be stored on the switch until the controller replies).

Therefore necessary precautions should be taken into consideration. SDN security issues and corresponding solutions for them are depicted in Figure 5 and discussed in subsection 2.1.1 and 2.1.2

2.1.1 SDN security issues

Threat Model: An attacker in most cases is one/multiple malicious client host(s), which tries to perform a man in the middle attack, masquerades as another element, performs denial of service (DoS) on other elements, or interferes with legitimate behavior of the network. Figure 5 shows network elements and elements' connections in black boxes. These elements include application,main controller, layer 2 controller, the link between the controller and the switch, switch, gateway, access point, server and host. These elements and attacks against them are discussed in the following:

2.1.1.1 Element 1. Application We may have malicious, compromised, or flawed applications or conflicting policies. These problems may result in stopping the controller or overriding the rules of each other.

Issue 1.1. Failure in application/ DoS from application (availability breach)

A lack of trust establishment between the application and the controller causes many problems [93]. If an application crashes (e.g., because of software failure) or exits the program suddenly, it can kill the controller as a result. Most errors (e.g., fail stop (null pointer, division by zero, etc.), invariant violation (loop in the path, black hole, etc.)) occur when event handlers are being used (event handlers are threads that are triggered when the event happens, e.g., packet_in, link_up, etc.). A controller failure leads to stopping of the network functionality. When a crash happens, due to unexpected timing or consistency issues, a rollback is required to remove all of the affected actions. Performing a rollback requires additional primitives that are not available in the switch (e.g., previous rule's time out); to keep this information, one needs to log all switch activities which is a time-consuming process. A rollback may cause inconsistency in hosts as well, but there are mechanisms implemented in hosts to resolve it (e.g., queuing messages for reordered messages in TCP).

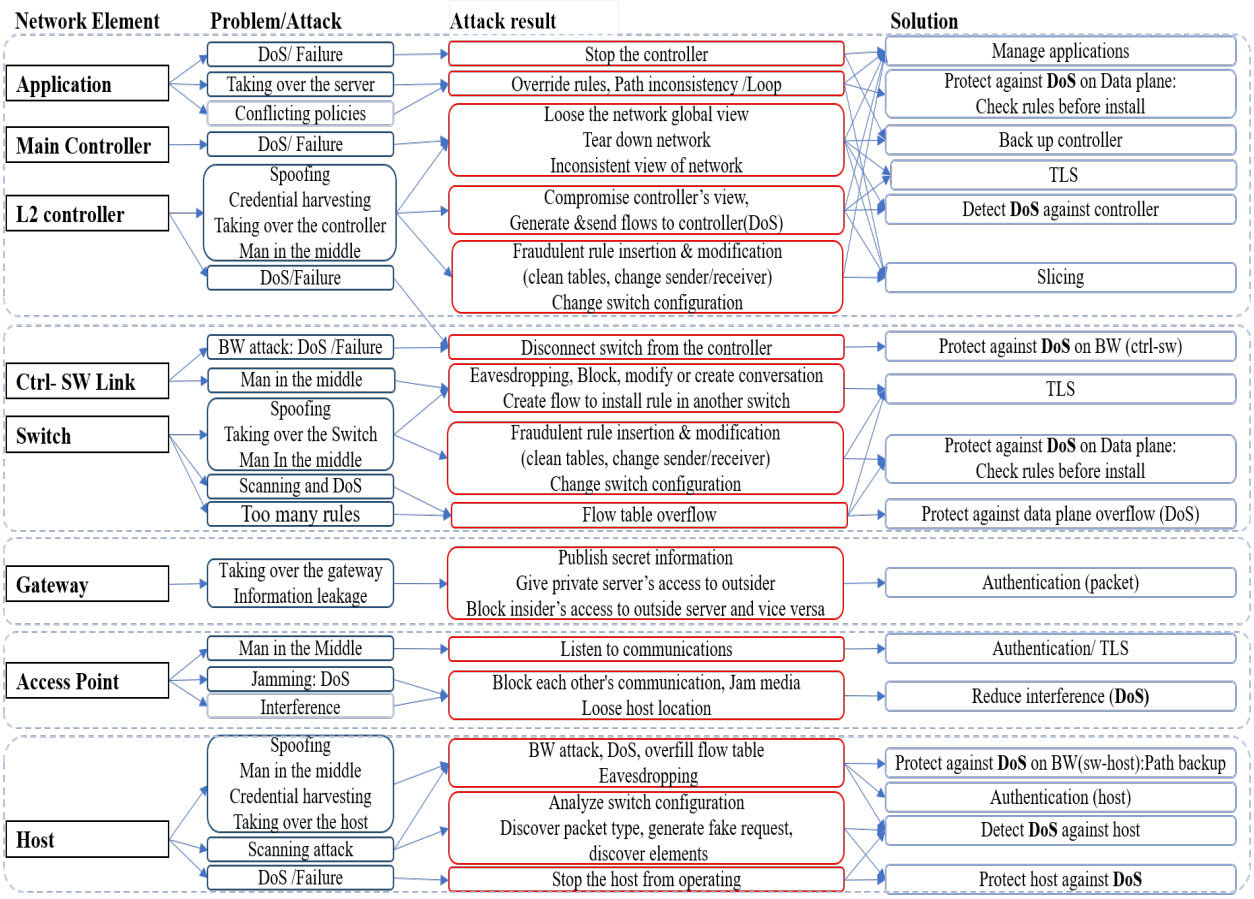

Figure 5: Attacks and problems (dark and light blue), effects (red) and solutions (blue)

After rolling back, buffered events should be replayed (buffer is cleared when the transaction is committed) which can cause the problem again. The controller should be able to decide which application is trustworthy, has the priority, and should be restarted.

Unknown traffic sent to the controller can cause DoS. Applications that send too many requests violate the resource utilization policy or the switch flow table specified limit. They can gradually allocate the memory or change the data.

Issue 1.2. Taking over the application webserver/conflicting policies (integrity breach) The infected or malicious application on the controller can change the flow (e.g., make the route longer) in a way that does not impact the forwarding function but decrease the performance. To mitigate this issue, system calls should be secured, limited, and done through interfaces.

Policy conflicts happen when there is a semantic gap between controller platform or access control actions; e.g., when there is flow aggregation, sharing one controller with many users, or using many controllers in one domain [3].

2.1.1.2 Element 2. Main controller

Efficient resource consumption attacks (a form of DoS) on the control plane are done by generating and sending crafted flows to the control plane through the data plane [146].

Issue 2.1. Failure/DoS in the main controller (availability breach) Distributed DoS (DDoS) attacks tend to drain the resources, for instance, deplete the bandwidth or exhaust the resources. DDoS attacks are initiated by the attacker or malicious code in the affected device (for instance, botnets). DDoS attacks are forged or fake traffic contains packets with spoofed source addresses, which causes switches to forward them to the controller and exhaust the resources. The main controller synchronizes and updates layer 2 controllers. It is a single point of failure; in the case of facing too many requests, DoS attacks, or any hardware or software issues, the controller will not be any longer available to process new packets. If we deploy a backup controller, it encounters a similar problem. These issues result in losing the global view of the network and stopping network functions and services; therefore, it affects the network availability and can tear down the whole network. Having layer 2 (distributed) controllers, mitigates this issue.

2.1.1.3 Element 3. Layer 2 controller When attackers masquerade themselves as the controller by spoofing, getting access to credentials, taking over the controller locating themselves between the layer-2 controller and the switch, and playing the role of the controller for the switch and vice-versa (man in the middle attack) or even when the controller is flawed or under DOS attack, the network management and control is lost.

Issue 3.1. Inconsistency in the main controller (identification and integrity breach)

Inconsistency can happen, when an attacker takes control or spoofs any of the layer-2 controllers, performs credential harvesting, or plays the role of a man in the middle. This attacker (that now can act as a layer 2 controller) can give a wrong view of the network to the main controller; therefore, the main controller receives inconsistent views from different controllers.

Issue 3.2. Installing malicious rules by element acting as L2 controller in front of switch (confidentiality, availability, and non-repudiation breach)

Installing and modifying rules in switches' flow tables can endanger many security services; the following discusses some examples. The attacker in the power (that act as a layer 2 controller, or the bug) may give the private server access to the outsider hosts by installing a rule that changes the flow header and sets a new destination for it. The attacker can perform eavesdropping actively by installing rules that redirect all or specific flows to the intended destination or passively by installing rules that send a copy of all or specific flows to the intended destination. The controller has a view of the network and can break the network apart by performing bandwidth (BW) attacks and installing rules that redirect traffic to a critical link. It can make the network collapse by clearing all the rules, setting all the rules to drop the packets, or installing nonsensical rules just to make everything messy. It can change the switch's configuration as well. Sometimes to increase scalability, developers can split tasks between the switch and the controller; in this case, the controller can modify the algorithm in the switch. The controller can also install too many flows on switches to fill flow tables. It can also generate and send messages on behalf of hosts, install rules that send messages to incorrect receivers, or install rules that change the message sender and destroys the non-repudiation (identification) service in the network. It can send too many requests

to another L2 controller and perform a denial of service attack. The device in the middle acts as a controller and can install or modify rules on the switch, change routes, or perform any of DoS, eavesdropping, or bandwidth attacks.

Issue 3.3. DoS on l2 controller (availability breach)

DoS attack on a layer 2 controller affects this controller's ability to route new flows coming from switches.

2.1.1.4 Element 4: The link between the controller and the switch The link between the controller and the switch is essential for south-bound communication in SDN.

Issue 4.1. Controller-switch link failure or bandwidth attack (availability breach)

If south-bound communication fails, the switch cannot route any new flows. The switch should be able to find another route to the controller or should be able to temporarily route flows independently, using an embedded algorithm.

Issue 4.2. Controller-switch link eavesdropping (confidentiality breach)

If the communication between a controller and a switch is not secure, eavesdropping on control packets can give an attacker a huge amount of information (such as host's IP addresses and information, flow table status, installed flows and required flows, etc.) that facilitates other attacks.

2.1.1.5 Element 5. Switch An attacker may logically locate itself between the controller and the switch and play the "man in the middle", spoof a switch, or take over the switch control. The switch's flow table overloads, if it is under DoS attack or simply receiving too many valid requests for rule installation.

Issue 5.1. Element acting as a switch in front of the ontroller (integrity, availability, and confidentiality breach)

If any element masquerades itself as a switch (e.g., by spoofing, taking over the switch, or man in the middle) or in the case of having a flawed switch, many security services are jeopardized. The malicious switch can give a compromised view from itself and its neighbors to the controller and make the view in the existing layer-2 controllers inconsistent and misguides the controller. It can generate flows and send them to the controller to perform

a DoS attack. In addition to the controller, it can create some problems for other network elements. It can intelligently generate and send flows to the controller to install specific rules in other switches or perform bandwidth attacks by redirecting traffic to a link by sending flows that pass that specific link (it needs to be intelligent and achieves the global view of the network) or it can provide access to the private server for outsiders.

Issue 5.2. Element acting as a switch in front of host (confidentiality, availability and identification breach)

Hosts are required to have some methods to authenticate the switch and the network; otherwise, a malicious switch (by spoofing, taking over the switch, or man in the middle) can eavesdrop on the receiving packets from the host, by forwarding (or duplicating) packets to the switch's malicious intended destination. Moreover, a malicious switch can drop packets and block conversations. It can also cause problem for hosts by generating and sending messages on behalf of hosts, sending messages to incorrect receivers, or sending messages to the correct receiver but change the message sender in the header which is an attack against non-repudiation service.

Issue 5.3. Too many rules to install in the flow table (availability breach)

The switch may receive too many requests. They can either be valid requests, from the controller to install required rules or efficient resource consumption attacks (DoS), on data plane by generating fake flows (e.g., by changing the receiver ID so that the new packet header does not match the flow table) to enforce inserting useless rules into the flow table [146]. Either way, the flow table overflows and the subsequent rules can not be installed. To perform a DoS attack, an attacker can discover the proactive/reactive configuration of a switch (the time it takes to process the packet, shows if the rule exists or is sent to the controller or whether the network is SDN or not). Having discovered the network type and flow table information, attackers can generate a high volume of fake requests leading to DoS on both the switch and the controller. Aggregating rules, detecting overloading flows, limiting rates, and considering priority for installing rules, can mitigate the issue.

2.1.1.6 Element 6. Gateway The gateway is a switch; therefore, it subsumes all the problems that can happen to the switch (issues2.1.1.5,2.1.1.5 and 2.1.1.5); moreover, since it

has a critical task and position in the network, it is vulnerable to some additional problems.

Issue 6.1 Taking over the gateway (availability, access control and confidentiality breach)

If the attacker takes over the gateway, they can provide private server access to outsiders and publish information throughout the Internet. They can also block insiders from getting access to the outside server or block outsiders from getting access to the public server.

2.1.1.7 Element 7. Access Point The access point acts as a switch; therefore, it may also absorb any of the problems that can happen to the switch (issues2.1.1.5,2.1.1.5 and 2.1.1.5). In addition to a generic switch's security issues, an AP may face some further issues, attributed to using wireless interface and mobility. Complications of updating host location, decreasing delay, and avoiding packet loss increase the security complexity.

Issue 7.1. Man in the middle in the access point (confidentiality breach)

Since wireless communications utilize air, water, etc. as media, it is easy for an attacker to capture the traffic; therefore, the communication between every two wireless devices must be encrypted (e.g., using TLS); otherwise, there would be no confidentiality in the communication.

Issue 7.2. Interference in the access point (availability breach)

Wireless access points may interfere with each other's communication. The existence of a malicious, hijacked or flawed AP increases the interference. They can block each other's communication, jam the media and block all surrounding communications. As a consequence, an honest AP may lose the location of hosts or the connection with the controller.

2.1.1.8 Element 8. Host and Servers The network exists to serve hosts and receive services from servers; however, a flawed or taken over (hijacked) host/server can cause problems. A non-trustworthy host/server can masquerade itself as another host/server or perform man in a middle attack between the host/server and the switch.

Issue 8.1 Wireless host (confidentiality and availability breach)

If the host communicates wirelessly, it may encounter all problems mentioned for APs (Issues 2.1.1.7 and 2.1.1.7).

Issue 8.2. Masquerading as host (identification, authentication and confidentiality breach)

The attacker can masquerade as a legitimate host by performing spoofing, a man in the middle attack, credential harvesting and, taking over the host. It can eavesdrop on the receiving packets or send messages on behalf of the victim host. The attacker can surface problems for other elements as well. As an example, it can intelligently generate and send flows to the controller through switches to install specific rules in other switches to redirect traffic. It can perform DoS attack by generating new and different flows and send them to the controller (DoS on the controller) or have the controller create new rules for the switches and fill the switch's flow tables (DoS on the switch). Depending on the intelligence of the attacker, if it gains the view of the network and knows both the switch configuration and how they install flows, it can perform bandwidth attacks

Issue 8.3 Scanning attack on the host (confidentiality breach)

An attacker should discover its victims before attacks (e.g., DoS, eavesdropping, etc.), using scanning attack. There are methods to detect attacks or protect hosts (e.g., IP mutation).

Issue 8.4 DoS attack on host (availability breach)

Attacks and malfunctioning can stop the host from operating. There should exist some methods to restart the host and retrieve the lost information.

2.1.2 SDN security solutions

In order to have a secure SDN controller, unauthorized access should be blocked, using a secure controller platform. Required network privileges and resources, should be provided to each application, meanwhile resources should be secured. Application isolation, authentication and authorization have important roles in the controller security. Transport Layer Security (TLS) is a necessary security protocol for SDN; however, when multiple controllers are communicating with the main controller (or any other single node), authorization and access control becomes more complicated and the risk of an unauthorized access increases [141]. Overall, although SDN has many security vulnerabilities, it has a good capacity to

mitigate security problems since it supports analysis and response, including changing policy or inserting security services [141]. The solutions to resolve the issues can be categorized into managing applications, slicing the network, using TLS and secure authentication, and detecting and mitigating DoS attacks on controller, controller-switch link, data plane, switch-host link and hosts. We described in very brief statements below, several proposed solutions, methods and approaches to show the variety of solutions that address SDN security issues.

2.1.2.1 Managing Applications: Checking for general correctness, performing symbolic execution and reducing input space using domain knowledge may reveal bugs [31]. IPC (inter process communication) or RPC (remote procedure call), data abstraction and microkernels should be used to isolate applications and having a secure access permission for authenticating application's system calls can protect the controller and setting different threshold for detecting resource utilization policy violations can find abnormal behavior. Messages and events should be logged, in case any serious problem happens, the controller should restart and roll back and decide which application is necessary to be restarted and which event should be rollback and run again in the same or different order([148, 35]. Traffic monitoring by recording samples of traffic and storing it in data stores for later replay makes it possible to add consistent records and coordinated replay ability for scalable software and configuration debugging, using OpenFlow split forwarding architecture [11, 162].

2.1.2.2 Slicing: FlowVisor mediate controller and switch can securely manage and isolate slices using VLAN priority. FlowVisor uses virtualization to share the same network hardware such as bandwidth, network view, traffic, CPU and flow table among logical networks [143, 144]. Netcore is a high-level language for tagging packages that uses VLAN tags to slice switches' links and ports and specify which packet may enter which slice [71].

2.1.2.3 TLS and Authentication: Secure authentication of switch and controller can avoid many security issues such as DoS and Man in the Middle. As an example, STRIDE analysis on sFlow and BigTap monitoring tools shows that in an SDN network they are vulnerable to spoofing (only sFlow), information disclosure (both) and tampering (only Big-

Tap) that can be resolved by using TLS. Authenticating packets, while entering and exiting the backbone network is important [70]. Ethane authenticate switches using certificates and Ethane and Resonance authenticate hosts using webform [115, 32]. Ethane leads to creation of OpenFlow. To identify malicious flows, anomaly detection can be used [156]. VAVE performs traffic analysis and dynamic updates on rules to protect hosts against source IP spoofing using source address authentication in the perimeter routers with filtering ability [164]. In wireless communications, [109] suggests having SDN controller, application controller and security controller use light weight symmetric cryptography for integrity, authentication and confidentiality.

2.1.2.4 DoS against controller: Authenticating controllers and having multiple replicated/distributed controllers can address DoS against an SDN controller [139]. Minimum required number of controllers that resist byzantine and guarantee maximum latency is an NP problem that can be resolved using the heuristic of assigning the controller with smallest capacity to the switch with highest demand [100, 99]. Replicating controllers on fault tolerant logically centralized data store can be done using state machine replication (SMR) and Paxos/VR and coordinates their actions through it. In order to tolerate f crashes $2f + 1$ replicas should exist and each switch reports to $f + 1$ of the controllers [22]. Mcad is a multi-controller decision making architecture that deals with mitigating flow rule attacks by assigning controllers dynamically and using votes to choose the correct flow rules [128]. Mcad-SA is a multiple controller and multiple back up controllers and different scheduling and switching algorithms are used to choose the controller [127]. In [127] a game theory model is used to describing when the attacker's probe on a controller increases, the defended switch moves to a backup controller. A method provided in [55] tolerates byzantine faults in both controller and data plane. The primary receiver sends "prepare message" to all replicas and waits for responses to form a message certificate with them. Then it calculates the response and sends it to client which waits for $f + 1$ similar responses [55]. Anomaly detection techniques using machine learning algorithms such as neural networks, Support Vector Machine (SVM), genetic algorithms, fuzzy logic, Bayesian networks, decision tree, etc. are used to detect intrusion [5]. As an example, Atlantic [44] uses a two phases method:

in Lightweight phase, it compares the entropy of flows features to quickly detect malicious flows by detecting deviations from normal flows; in Heavyweight phase, it gathers data every p seconds, selects features using Principal Component Analysis (PCA), summarizes the data and then classifies them using the union of SVM and K-means (in 3s). In [26] Self-Organizing Map (SOM) based model system is presented that has 3 detection loops: 1) flow collector from table entries, 2) statistics and feature extractor, 3) classifying of flows to normal and abnormal [26]. Traffic flows statistics can be gathered to acquire enough information for DoS detection. iStamp [108] gathers flow statistics and finds larger and more frequent flows by matrix indexing and sorting. NetFuse (detect overloading flows and improve performance and scalability) [161] uses both passive listening of OpenFlow controlling messages and active querying of flow tables and uses a list of reasonable aggregation dimensions to accurately detect and change the rate of aggressive flows to be rerouted to NetFuse proxy box to be delayed or dropped. DevoFlow (replacement for OpenFlow) [41] collects statistics using "sampling", "triggering and reports" and "approximate counters" (small memory and have high accuracy) and uses multi-dimensional flow statistic aggregation to detect and reroute elephant flows to the backup routes and perform load balancing.

2.1.2.5 DoS attack on data plane: FlowChecker is executed on top of NOX and performs priority-based matching and model checking by encoding flow table configurations into Binary Decision Diagrams (BDD). Checking policy conflicts, check flow aggregation of access control, analysis application queries, path inconsistencies, reachability, configurations and slicing isolation [3] is done. VeriFlow sits between a controller and data plane to aggregate rules in a tree and check forwarding actions for loops, inconsistencies and drops and sends an alarm and prioritizing rules to resolve it [122, 89]. Fresco authorizes applications and checks for policy conflicts [147]. FortNox validates signed roles by authorization and access control. It also changes the rules to alias, reduces rules and compares them pairwise before installation with state table rules to make sure they do not override security rules. If a conflict detected, the rules with higher priority are installed [122]. SE-floodlight has 5 components that mediate traffic between controller and data plane: 1-role base conflict analysis by priority (admin, security app, regular app). 2- conducting a binary tree for ac-

tions and integrating rule changes 3-constraint commands sent to the switch by state table manager. 4-using switch call back tracking to identify the application that issued the rule. 5-permission mediation to stop collapsing the network because of application errors. It analyses conflicts in rules using a rule chain algorithm [123]. Aggregation and distribution of flow table reduces its size. FFTA aggregates flow table rules in a binary tree and optimizes it by collapsing parents label to children and expanding prefixes to omit one child and merge one-bit differences by replacing that bit with a wildcard *. iFFTA updates the tree and redoes bit merging [106]. Palette [83] distributes flow tables between switches using either the notion of pivot bit value or draws tables as graphs and separates them into separate subgraphs. Then it distributes routes among nodes using graph node (switches) coloring (routes) algorithm heuristics.

2.1.2.6 Bandwidth attacks In WmSDN when the SDN controller is out of reach, switches can route the packets temporarily using OLSRD and wait for connection up [49]. As another solution, AVANT-GUARD resolves two challenges: 1- table saturation and DoS which are addressed by migrating timeout TCP sessions to limit the flows that are sent to the controller. 2-detect and respond to flow dynamicity by gathering statistics and for triggering the conditional flow rules [149]. FatTire [131] easily defines policies (e.g., multiple back up routes for each link failure) by providing a fault tolerant programming language. CORONET [90] suggests providing a backup link, detects failures using Link Layer Discovery Protocol and maps traffic to VLANS using packet classification at edge switches. It has 4 modules: 1) topology discovery 2) using Dijkstra to create multiple paths with disjoint link property minimizing the number of affected routes in link failure 3) assigning ports to VLAN IDs and 4) assigning paths to host's traffic randomly. Our work in [86] examines the potential of using historical data to manage interference in wireless SDN by predicting traffic bottlenecks and deciding rapidly to change the channel or limiting the rate of specific senders.

2.1.2.7 DoS attack on Hosts: SDN hosts are vulnerable to information disclosure, DoS and tampering [91]. OpenFlow Random Host Mutation (OF-RHM) assigns virtual IPs to

hosts frequently [80]. In paper [111] authors check home network hosts for traffic anomalies such as: rate of failed connections, too many requests to one machine, traffic distribution (packet type and dest port) and using NETAD in which more novels packets are considered safe [111]. Some solutions detect anomalies, block malicious flows and validate flows. OpenSAFE replicates traffic to middleboxes for traffic monitoring and provides a scripting language for network traffic monitoring [11]. CloudWatcher registers security services, allocate required services to flows that require them (match flow header and rule to forward and route to the security service) and install flows to enforce routing [145]. In OpenSec the flow table decides (based on the flow header and statistics) whether the action is forward to the security unit and the security unit decides the corresponding action (alert, quarantine, block) [96]. FRESCO provides a scripting language to incorporate in generating rules that provides security services and extend security functions on the controller [147].

2.2 Software Defined Perimeter (SDP) integrated with SDN

In a secure network, devices, data owners, third party entities and all network elements need authentication for access to the edge networks, services, and cloud servers. SDP verifies user identity, assesses the device status, authenticates users and services and then provides a secure connection between them [19, 114]. It performs application authentication, host authentication and protects the controller, switches and hosts from DoS attacks and enables secure communications. SDP creates an exclusive secure channel, using mutual TLS and a Virtual Private Network (VPN), between the communicating parties. As shown in Figure 6, in SDP, there are 3 main modules: SDP Initiating Host (IH), SDP Accepting Host (AH) and SDP controller (CTRL). The IH is typically installed on a client (e.g., on hosts or hosts' gateways). The AH is installed on an SDP gateway and manages the communication between IH and CTRL, while CTRL is authenticating the user and creating a secure channel between the user and the service. As one can see, this architecture resembles the SDN architecture. There has been a few recent works that suggest integrating SDN and SDP to take advantage of the SDN features while securing the whole network with SDP. The combination of SDN

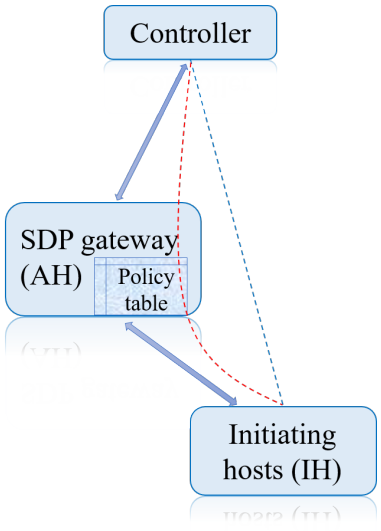

Figure 6: SDP architecture

and SDP can address the security requirements for implementing contextual decision making and action enforcing applications.

In [135] authors provide a secure and scalable framework using a combination of SDN and SDP, which provides flexibility and secure communication. They suggest two architectures, 1-client-to-server in which the server has an AH module and acts as both server and gateway which does not protect the server against any specific attacks but reduces the delay, 2-client-to-gateway in which the service is behind an AH gateway which is more secure but adds some delay. The authors also considers two scenarios in client to gateway architecture for controller placements:1-IH have direct access to CTRL: in this case in authentication first SDN controller should install the rule for route between IH (client) and CTRL, CTRL authenticates client to access the service and add forwarding rule to in AH gateway. The client is now authenticated and can send service request; therefore, first request is sent to SDN controller to install the rule for rout between client and AH gateway. When AH receives service request it will check its forwarding table and send the request to the server and send back the response to the client. 2-The CTRL is behind AH gateway for more protection

which also reduce flow installation delay (installing IH and AH route is enough and there is no need to install IH and CTRL route). The experiments showed that the combination of SDP and SDN can effectively block DoS flooding attacks.

In another proposed architecture of integrated SDN and SDP, authors suggest SDSec in which they use SDN controller to implement SDP CTRL. In this architecture SDSec Hosts stores parameters such as Trust, trusted Zone_ID, permission, resource consumption, and scope of other host it is authorized to communicate with. SDSec Switch is a User switch (among 3 type of Mininet switches- User, Kernel, OVS) that keeps a policies table that stores authenticated hosts IPs and access policy for each. SDSec controller is an OpenFlow POX controller with security checks and 9 security policies. Results show that it can reduce accepted requests and victim hosts resource consumption in the time of DoS attack [45].

Security attacks such as DOS attacks, backdoor exploits, VM Hopping and remote connection attacks in NVF, should be resolved [151]. To resolve these issues the gateway has a "drop all" policy initially; it only relays SPA authentication requests to the controller. The encrypted SPA includes Initiating Host ID (SDP ID), Accepting Host ID(Service ID), Gateway IP address, a timestamp and a 16-byte randomized data. They used Waverley Labs OpenSDP project as their SDP controller which is connected to a MySQL database to store approved hosts, gateways and services. The controller validates the SPA and the gateway updates its `iptables` to connect host to the requested service for a limited time.

2.2.1 Integrated SDN-SDP Framework and Protocol

We integrate all these ideas and resolve SDN security issues with SDP by providing a secure protocol which can satisfy the needs of contextual decision making and action enforcing application requirements. In our architecture we integrate the SDN controller with the SDP controller to have a global view and comprehensive information of teh network. At the same time, such a controller decides and is aware of security specifications. We integrate the SDN switch with the SDP gateway (AH); therefore, integrating the flowtable with the policy table. The SDN-SDP Controller communicates with switches to gather further contextual information. The initial policy in the flowtable is "drop-all" packets except

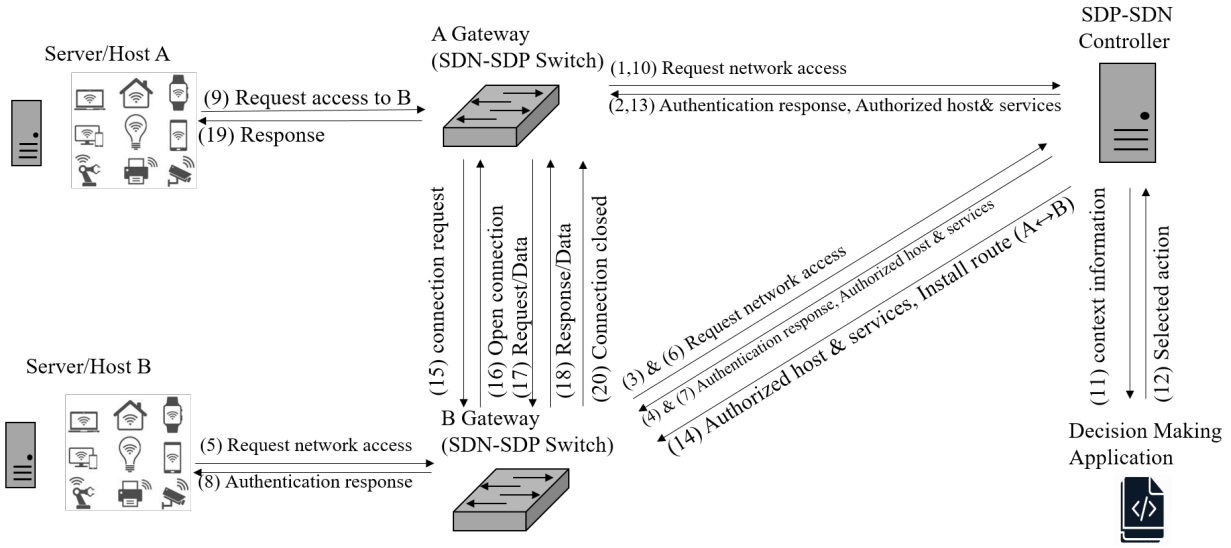

Figure 7: Protocol for Integrated SDN and SDP

SPA. The switch forwards the SPA to the controller to authenticate a host, make a decision about the packet forwarding, install routing rule in the tables, share updated authorized host and services with switches, provide requested services (such as deciding about the required integrity layer - see Chapter 4). This reduces the probability of illegitimate packets getting to the controller and keeps the controller safe and secure behind the gateway. This architecture authenticates all devices and secures their communications.

In SDP, Single Packet Authentication (SPA) is sent to request authentication and connection; it contains information such as host ID, requested service ID, gateway IP, timestamp, and randomized data. In this protocol all SDN-SDP switches drop all packets (from all devices, hosts, switches, services, etc.) except if they already have a rule for it or it is a legitimate SPA packet that is requesting access to the network.

Figure 7, shows the process of how a network element joins the network. If there are gateways connected to the controllers all SPA request should go to the controllers through the gateways, otherwise gateways should send their request to the controller directly as the "first" gateways that will handle future requests. When the network and devices are

initially setting up, the gateways transmit SPA packets and network access requests and are being authenticated and registered in an SDP-SDN controller (steps 1, through 4). The servers and services send SPA and network access requests through the gateways and are authenticated and registered in an SDP-SDN controller (steps 5, through 8)[3]. Each device generating data first communicates with the owner's gateway (SDP-SDN switch -A gateway- is called OG since this belongs to what we will call as a data owner in Chapters 4 and 5). 9) SPA (single packet authorization used for identifying clients in SDP), data, and metadata (update frequency, lifetime, tolerable latency and ID of the destination, etc.) reach the A Gateway. 10) A gateway sends SPA, data, and metadata to the SDP-SDN controller. 11) The SDN controller authenticates the owner and forwards the information to the application. The application uses the context information and the decision tree to choose the suitable action. 12) The application responds with the action. 13) The controller provides information about authorized services and connections to the A Gateway. 14)The controller also installs the proper rules (includes the action and route to B gateway) in the corresponding SDP-SDN switches. 15) A gateway sends SPA to the B gateway and requests connection to the destination. 16) B gateway opens the connection to the destination (cloud server, service provider or another host) and responds to A gateway. A gateway (and therefore Host/Server A) and destination can now exchange information for the session. Gateways exchange the data or request and response (17,18) and gateway A will sends the conformation to the Host A (19) and after every thing is done Server B closes the connection (20).

As shown in Figure 8, most SDN security issues and problems can be resolved using the suggested protocol in Figure 7. SDP authenticates applications and drops all packets before authentication therefore blocks DoS against controller, uses TLS for all communications, blocks unauthorized packets and hosts; therefore, protects against bandwidth attacks, data plane overflow and protects hosts from DoS attacks. Furthermore, providing a back-up and integrity verification for data set and controller, slicing, managing interference and providing path back-up can be done using contextual decision making and action enforcing applications. Chapter 3 and 4 discuss interference management and integrity verification as examples.

[3]leaving the network would follow the same process, devices send SPA's to accomplish this

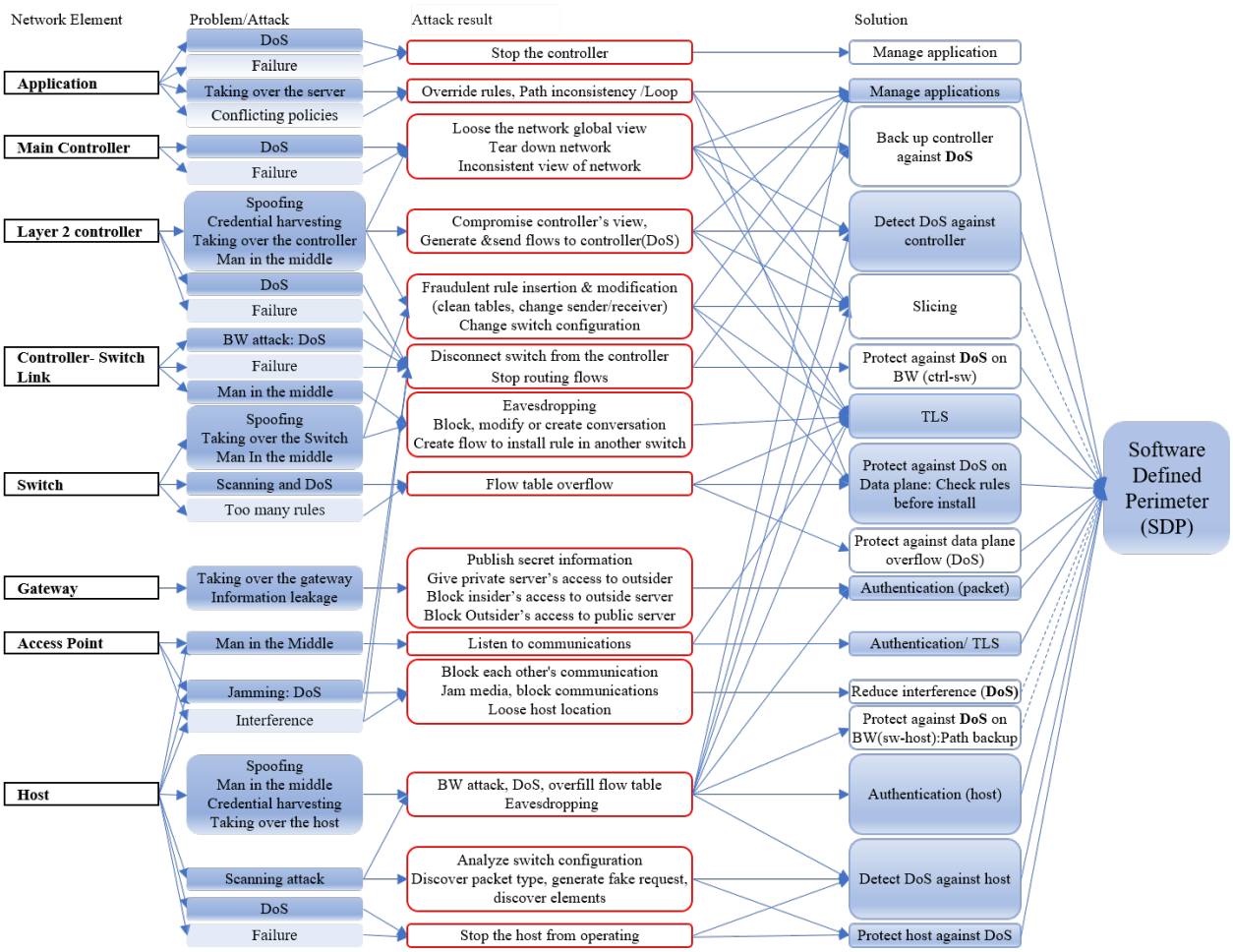

Figure 8: SDN issues resolved by SDP

2.2.2 Feasibility study

We implement a simple simulation of this protocol on Mininet 2.2.0 [112], Ubuntu 14.04.6 operating system, and Intel® Core™ i5-560M Processor 2.66 GHz, using POX SDN controller [6]. We use SDN firewall [47] to simulate the SDP controller on the SDN POX controller. We install agents on SDN-SDP switches that use POX messenger to relay the extra required information from user and switches to the controller [86] and develop a python decision making application (python 3.4.3) to use the context information acquired from the

user, SDN-SDP switch and SDN-SDP controller and select the matching action based on that context. As shown in Figure 9, the experiments show that the whole process of sending gathered information from OG to the controller and then to the application (here contextual integrity verification application ambit of data integrity (ADI) which is further discussed in Chapter 4) to select the proper action using a decision tree, and sending back the chosen ADI to OG, takes 37 ms on average. The code and details about the implementation are available online [84].

Figure 9: Feasibility study

3.0 Mining Historical Data towards Interference Management in Wireless SDNs

WiFi networks often seek to reduce interference through network planning, macroscopic self-organization (e.g. channel switching), or network management. In this chapter, we explore the use of historical data to automatically predict traffic bottlenecks and make rapid decisions in a wireless (WiFi-like) network on a smaller scale. This is now possible with software defined networks (SDN), whose controllers can have a global view of traffic flows in a network. Models such as classification trees can be used to quickly make decisions on how to manage network resources based on the quality needs, service level agreement, or other criteria provided by a network administrator. The objective of this chapter is to use data generated by simulation tools to see if such classification models can be developed and to evaluate their efficacy. For this purpose, extensive simulation data were collected and data mining techniques were then used to develop QoS prediction trees. Such trees can predict the maximum delay that results due to specific traffic situations with specific parameters. We evaluated these decision trees by placing them in an SDN controller. OpenFlow cannot directly provide the necessary information for managing wireless networks so we used POX messenger to set up an agent on each AP for adjusting the network. We explored the possibility of updating the tree using feedback from hosts. Our results show that such trees are effective in managing the network and decreasing maximum packet delay.

3.1 Introduction

Wireless infrastructures play an important role in a growing number of environments, some of which, such as e-health care environments [14], are rather critical in their demand for specific Quality of Service (QoS) for individual flows. In current WiFi networks, updating procedures require introducing new generations or versions of network protocols which in many cases forces the customer to exchange their device for a newer one; however if we

move the procedures to an SDN controller, not only can we change policies and procedures without changing the hardware, but also the update procedure could be done seamlessly without immediately affecting users. This can be accomplished by limiting the applied changes to any subnet of the network for test and getting feedback before global application of the changes. Most procedures and decisions in wireless networks are fixed, however with the use of SDNs we can use the controller's global view of the network information and by monitoring the cause and effects of each action in each context, we can train machine learning models to perform specific actions. These models can still be updated while running by receiving feedback and context from the network. In this chapter the aim was not to find the best set of parameters or algorithm for training the model or optimizing the network performance, but to show that using SDN controllers and historical data we can use machine learning models and use them for managing the network.

As an example, in hospitals, interference can impact the quality of a video which may be critical for a physician to make the correct diagnosis [1]. If the wireless network is controlled using a software defined network (SDN) controller, it is possible to achieve a complete view of the network in the controller which can monitor parameters. Through observation of relevant parameters, the controller can manage changes to the characteristics of a flow (in the extreme case by changing the communication channel or stopping some flows).

This begs the question as to whether a specific combination of factors such as number of APs, location of APs, power level, packet rate, packet size, number of flows, and so on (specific value ranges) can be used to develop a model that can assess the emergence of QoS problems. In other words, is it possible to develop a model based on historical data that can automatically and rapidly assess the possible QoS in various situations when multiple APs are simultaneously sending data. Then, we may be able to use this model to decide whether the QoS is unacceptable and react by lowering (for instance) the packet size and packet rate of flows from other APs (based on the requirements and priority of specific flows). Since there are numerous parameter combinations, we consider building classification trees that allow quick decisions to be made by the controller. The purpose of this chapter is to determine if this a viable approach. Our goal is not to develop new solutions to problems of flow

[1] Interference can increase delay and packet losses which mean a decrease in QoS.

control, scheduling, or congestion, but *to evaluate the potential of data mining-based models in wireless SDN controllers* to automatically and rapidly impact QoS for specific flows. In order to see if such classification guidelines can be developed, different situations that might happen in the network were simulated under different configurations. After collecting various types of information about specific parameters, we created a *cleaned database* of scenarios and observed QoS. Using this historical data (in this chapter from simulations), we applied data mining techniques using the Weka [72] data mining tool to create decision/prediction trees that can inform the controller what may happen to a flow under specific parameter ranges and network conditions.

We created a prediction tree that captures situations where there can be an observed decrease in QoS for flows and situations that may block communications. This "QoS tree" can be employed to predict delay based on QoS parameters (packet loss and delay) that a communication flow needs. Using the QoS tree, an SDN controller in the network can detect situations causing interference and may, for example, switch the channel of the flow (or neighboring flows) or lower the specified bandwidth allocated to APs that do not require high QoS. We then applied the tree in a POX [6] SDN controller. OpenFlow is sufficient for programming flow table rules but it cannot in general provide required information for wireless networks [137]. We used POX messenger and set up an agent on each AP. The controller used the POX messenger channel to receive information from APs, make a decision based on the QoS tree and to determine how it affected the QoS. As the historical data changes, we stored the leaves in an update-able data structure, making the tree dynamic. Agents were created on "Hosts" to communicate with the controller and send feedback to update the tree.

Our evaluation shows that such classification trees may be used to perform necessary management by the SDN controller. Using the dynamic data structure, we update the tree using online traffic. The results show that the tree is stable in the same network. We use the same approach when we have multiple flows and create decision trees for those configurations as well. Further work, when extended, may enable us to also determine which parameters are critical and need to be monitored more closely in the network to change (as needed) potential network slices/configuration, rather than managing only flows in one network.

The chapter is organized as follows. In Section 2, we provide a brief background of data mining, SDN, and some related work. In Section 3, we describe the experimental design and present the QoS Tree. Section 4 presents the results of applying the QoS tree in the SDN controller for simple situations with one flow. Section 5 provides similar experiments for more complicated situations with more flows and flow sizes. Section 6 provides a discussion of limitations and future work and Section 7 concludes the chapter.

3.2 Background and Preliminaries

In this section, we review some basic concepts of data mining and SDNs. We also discuss some related work in this area.

3.2.1 Data Mining

Data mining includes four main steps to create knowledge from collected data: selection, pre-processing, data mining, and interpretation/evaluation. Selection is the process of choosing tuples and attributes that are required for answering questions. Data pre-processing includes the following actions: *Cleaning:* detecting and correcting or deleting inaccurate or corrupts records; *Normalization:* reduction of data to any kind of canonical form; *Transformation:* conversion of a set of data values into the data format of a destination data system; *Feature extraction:* deriving some attribute values from an initial set of measured data; and *Selection:* selecting a subset of relevant features based on domain knowledge.

Data mining algorithms can be applied to the data to find patterns of interest. Classification and regression are considered important tasks in data mining. In this chapter, based on our continuous class variables (delay), we chose Random Tree, REP, and MP5 regression decision tree learners in Weka [72]. We also try to make the class variable "delay" categorical ($>$ 100ms and $<$ 100ms) and use the J-48 classification algorithm. We chose these algorithms because they are popular with data mining researchers (e.g. [4, 130, 152, 125, 30]). These methods (described next) help to extract information relationships and hidden patterns in

large data sets.

Random Tree: It is one tree from the set of possible trees, with k random features at each node [167]. The random tree generates many individual "learners". It constructs a decision tree by employing a random set of data. Each node is split using the best split compared to other variables. At each splitting step all attributes are selected randomly and the tree is grown as much as possible [29].

REP: It is a fast decision/regression tree builder that uses the regression tree logic to create multiple trees over different iterations. The algorithm uses a "gain" for splitting and pruning the tree by reduced error pruning and sorts numeric attributes. It uses the C4.5 method [2] of using fractional instances to deal with missing values. [167, 67].

M5P: M5P generates "M5 Model" trees and rules. M5 constructs a tree that relates the target value to other attributes using a divide-and-conquer method. First, it computes the standard deviation of the target value in a node. Then it will consider all possible splits and calculate their standard deviations and the reduction in error of the parent node with that split. The maximum reduction in error will specify which split should happen. The algorithm stops when the number of tuples in the node reaches a specific threshold. Then it uses standard regression techniques to provide a linear model for tree nodes. It uses a greedy search to minimize the number of effective parameters by removing the variable that contributes only a little to the model. Finally, it will prune the tree comparing the estimated error of each node with its parent node [129].

Following paths in random trees or REP trees will give us a result (by having specific parameter values). For example, a random tree or a REP tree may tell us: "if the packet size is smaller than N bits and the transmit power is smaller than P dBm, the delay will be t ms". In M5P trees, instead of getting a clear value as a flowchart result, we will have models left in the leaves.

J-48: The algorithm J-48 is a Weka implementation of the C4.5 classification algorithm with a categorical class variable. We use it with two categories of delay as described previ-

[2] C4.5 is an algorithm that splits data into smaller subsets by calculating "entropy" (the measure of data disorderliness) and "gain" (decrease in information entropy) for possible attribute splits, and makes a decision. For each split, it chooses the highest gain that is the lowest entropy to branch on. It stops when it reaches a completely pure subset that all instances have the same class attribute in a tree leaf. Then the tree will be pruned to eliminate outliers [29, 85]

ously [85].

We use 10-fold cross-validation to test the created decision trees. The data set was split into ten equal size subsets. Nine subsets are used to train the model, and the model is tested with the remaining tenth subset. The number of correct classifications over the number of all instances is used to estimate the accuracy of the tree [85].

3.2.2 Managing interference by using machine learning algorithms

Machine learning algorithms have been used in wireless interference management, some of which we briefly review next. In [46] authors process data from real-time reporting of sessions for network optimization. In order to predict packet drops, before the end of the session, machine learning was used on offline LTE data. In [105], the authors identify interference modulation order by using source automatic modulation classification. They use supervised learning techniques to achieve channel estimates in inter/intra cell interference, with/ without accurate information. This method can be used in the cancellation of interference in cellular networks.

In [153], the authors optimize radio resources with poor performance by using a statistical learning process which uses regression to extract relation between performances attributes. The objective is to heal inter-cell interference coordination. In [98] a Kalman-Filter approach is used to predict interference by deriving the correlation of co-channel interference. Based on interference prediction and path gain, the transmit power can be adjusted to achieve the required signal-to-interference ratio (SINR). None of these works have considered SDN networks and their control as their objective.

In [46] and [105], authors use historical data and libSVM to create models which predict interference but they did not apply their model to the network to see how it can improve performance metrics. In [153] and [98] they apply their model in the network but they did not use historical data. In these papers, authors consider signal-to-noise ratio and none of them examine QoS. In this chapter we simulate parameters that affect QoS and based on data from all APs across the network, we use models that decide how interference can affect the QoS. We also apply the models in an SDN controller to react to network conditions by

lowering the packet size and packet rates of flows from APs with lower priority to improve QoS.

3.2.3 Software Defined Wireless Network

SDNs push the control plane of the switches and routers to software. The data plane in SDNs is separated from the control plane. The high-level architecture is shown in Figure 1. The central controller in the SDN architecture provides the infrastructure for managing the network. Routing algorithms are placed in the controller. The SDN controller receives policies and instructions from the "application" via north-bound communications. [134]. Routing is performed for each flow by the controller and installed rules in the switch's flow tables. Switches forward the data according to these rules. When a flow enters a switch, the switch compares flow fields with the flow table. If it matches an existing entry, the corresponding action will be taken; otherwise, the switch uses the OpenFlow protocol to send the first packet of the flow to the controller. The controller then calculates the route for this flow and adds an entry with flow fields and suitable action to the flow table. SDN provides an intelligent and controllable architecture, less dependency on hardware or specific vendor, simple management, faster innovation, implementation, and testing [134, 1].

SDNs may be used in a variety of environments. As an example, consider a healthcare application where it is required to stream approximately 360 Mbps uncompressed video from two discrete endoscopic cameras [133]. Processing this data needs a high-performance real-time computing (HPC) environment, to minimize the risk to a patient. In [133], authors utilize an algorithm on OpenFlow SDN to use its capability of connecting multiple remote HPC servers and medical devices. Similarly, the use of SDN in wireless networks is possible. In particular, an SDN controller can set parameters in WiFi APs (which are the switches in Figure 11) such as the channel and the transmit power in addition to the flow tables [12].

In [137], the authors claim that there is not any uniformity of feature set solution available for wireless networks management, and OpenFlow does not address WiFi complexities such as interference management, mobility, and channel selection. They used Odin to propose Light Virtual Access Points (LVAP) which is per client AP with unique BSSID (mac

address of a wireless interface), it provides isolation in control logic. In the case of handoff, these LVAPs migrate between APs without triggering re-association in clients. Some applications were developed over Odin, such as mobility manager, load balancer, trouble shooting (Interference and jammer detection using channel snapshots using WiFiNet cards), channel selection, and energy-efficient WiFi networks (by selecting one AP as master with a couple of APs as slaves) and guest policy enforcement [137].

3.3 Experimental Design and Results

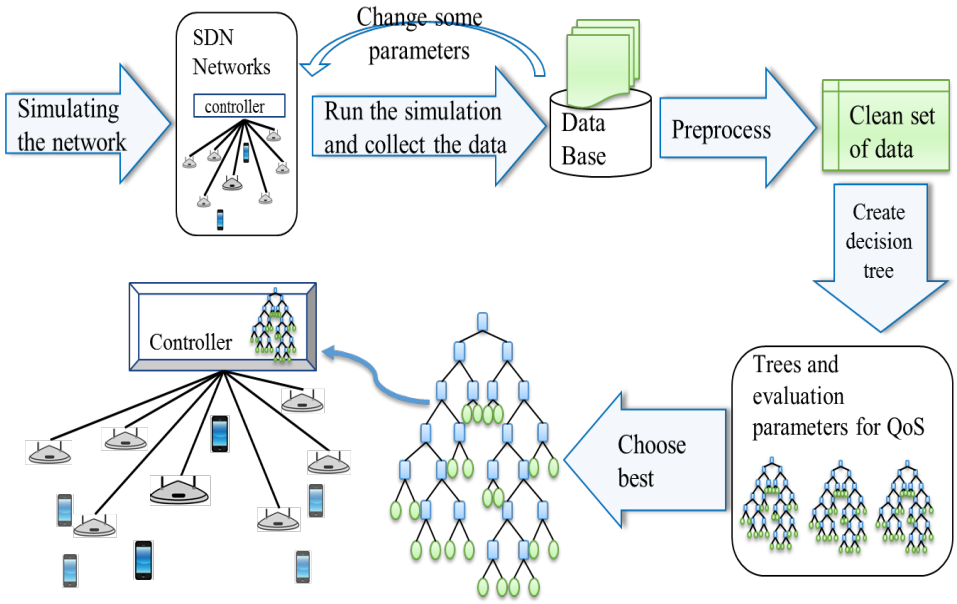

Figure 10: Steps in experiment

We simulate an SDN in which some hosts (2 to 6 in number) are communicating with each other (1 to 3 flows) through an AP and neighboring APs cause interference. We then examine different situations by changing the number of interfering APs, their power level, packet sizes, packet rates, their locations, different numbers of hosts, and different flow sizes. Then we use WEKA to apply data mining methods (specifically the random tree, REP, and M5P) to create a decision/prediction tree that considers the current state and predicts the

QoS of the tagged flow for that state. Each state is composed of the above parameters - number of APs, APs' locations, packet size, packet rate, power, etc. QoS is defined as the maximum delay that can happen in each state and it will be compared to the delay that the flow can tolerate based on the SLA. Based on the QoS flow needs, the prediction tree (described later) can be used to predict whether a situation can provide the necessary QoS or not. Figure 10 shows the steps used in our experiments.

3.3.1 Test Scenario

We first run the controller and create the WiFi network with the topology in Figure 11. Hosts (in this case h1 and h2) start to communicate while other WiFi APs continue to broadcast packets. This process is explained in more detail next.

Experimental environment: Experiments are done with Ubuntu 14.04, Python 2.7.6, Java 1.8.0_111. Tools that were used include but were not limited to OpenNet SDN simulator, POX controller, Weka, eclipse 3.8.1, and pydev.

Network setup: For each experiment, we first ran the POX [6] SDN controller, then OpenNet is started. OpenNet [34] is an open source simulator for wireless SDN formed by two simulators: Mininet [116] for simulating the SDN, using OpenFlow 1.3.1 and NS3 [132] for simulating wireless networks. We used OpenNet, without any change [3], to simulate 420 different configurations with a different number of APs (1 to 21) in different locations. In this simulation `ns3::YansWifiPhyHelper` is used to set-up WiFi PHY in the emulation, which uses `ns3::LogDistancePropagationLossModel` as the propagation loss model and `ns3::NistErrorRateModel` as the error rate model. The received power after adding the propagation loss is calculated as:

$$rx = 10\log(Pr_0(tx)) - n \times 10\log(d/d_0) \text{ [77]} \tag{3.1}$$

in which n is the path loss distance exponent, d_0 is reference distance (m), L_0 is path loss at reference distance (dB), d is distance (m), Pr_0 is the received power at d_0 (W), and tx is the

[3]The simulated network models the IEEE 802.11g standard.

current transmission power(dB) [77]. Different modulation/coding schemes have different error rate models [4].

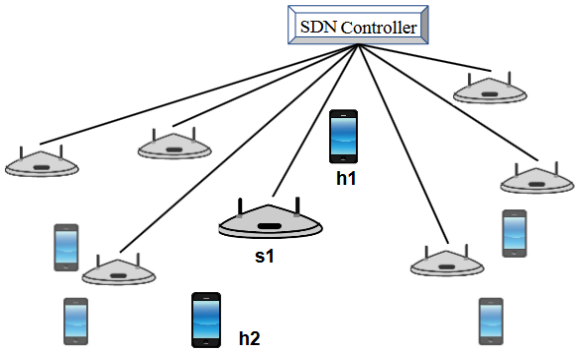

Figure 11: Experiment's topology

In Figure 11, the lighter/smaller APs are changed in numbers, locations, power, etc. with each configuration. We vary the number of APs from 1 to 21. The number of hosts varies between 2 and 6. APs and hosts are placed in a rectangular region between local coordinates of –120 m (lower / left) to 120 m (upper / right). The other parameter ranges are as the following: transmit power range between 0 and 40 dBm, packet size range from 0 to 100000 bytes, packet rate was between 10 and 1000 packets per second and packet size ranges from 64 bytes to 4000 bytes. All APs use channel 11 and simply broadcast packets to influence the QoS of the tagged flow(s). The APs were connected to a POX controller. Each configuration was defined in a Python script. Each Python script defines the AP's position, links, host's position, host's mobility, the channel characteristics, and other details about the simulation. Default parameters are used in most cases. As mentioned previously, our objective was not to develop any new algorithms for solving specific network problems, but to evaluate the feasibility of using data mining for network resource management in an automated manner.

Running experiments: Hosts **h1** and **h2** send packets with variable sizes to each other via the AP **s1**. The other APs and hosts may be there or not in different tests. In each

[4]Validation and description for OFDM modulation is presented in [120]. It calculates bit error rate (BER) for different modulations such as QAM, BPSK, QPSK at giver SNR after and before applying Forward Error Correction (FEC) [78].

experiment, the transmit power of each AP is specified. Hping3 [136] is used in each AP to create and send TCP/IP packets. This provides us the possibility of specifying different packet rates, packet sizes, packet counts, and other protocol details and varying them easily. The interference from these transmissions influences the delay, packet loss, and thus the QoS of the tagged flow. There are 1 to 3 tagged flows with different sizes that we monitor in the experiments, which is between hosts **h1** and **h2** that passes through AP **s1**. For each flow, we sent 50 packets to be able to see the changes.

Collecting the results: In each experiment, some outputs were gathered to form the database used as historical data. This database was analyzed to drive the prediction/decision trees (explained later). In each experiment, the following files were stored: (a) Python files of the simulated network consisting of APs and hosts' locations, (b) a script for setting the transmit power and executing Hping3, containing power, packet size, packet rate, and the number of packets, and TCP/UDP mode. Table 2 shows all of the extracted attributes.

In some cases, we decided to use the option "flood", which sends as many packets as it can with the maximum possible rate. The output response of every single packet delivery (between hosts) was stored along with the packet size, overall max delay, min delay, average delay, and packet loss. The results show that the delay range can be between 4.08 ms and 50672 ms. This experiment was repeated 1480 times with different configurations with different numbers of APs, packet size, packet rate, AP locations, transmit power, flows, and other variables. The total amount of gathered data was about 50 Gigabytes and the attributes in each file were surrounded by many unnecessary data fields. Thus the collected data needed to be pre-processed.

3.3.2 Pre-processing

For data pre-processing, we wrote Java programs to apply some string processing to extract features from the hosts' files and compute flow size, meanDelay, maxDelay, minDelay, and packet loss (see the tuple in Table 2). Next, we processed the APs' files to determine the power and Hping3 command parameters. Then we processed the network simulator Python files to get the APs and hosts locations. The result is a summarization and integration of each

experiment into one file. Later, another Java program was developed to integrate all files into one single Excel file. We further pre-processed the data by data cleaning (deleting some records in which the APs stop working), transformation, and normalization (to bring data into an acceptable range[5]), feature extraction (APs distances from hosts and s1 considering their location) and selection[6]. The result was a table with more than 200,000 tuples.

Table 2: Fields and Description

Fields	Description
Mean Delay	Average delay between h1 and h2
Max Delay	Maximum delay between h1 and h2
Min Delay	Minimum delay between h1 and h2
Packet loss	Number of the lost packet between h1 and h2
Num Of Ss	Number of APs that are sending packets
MeanDist Ss sH	Average distance between APs & Sender Host
MeanDist Ss rH	Average distance between APs & Receiver Host
Mean Dist Ss&S	The average distance between APs and AP1
TPacket Size	Total size of the packets that APs sent
TPacket Rate	Total rates at which APs sent packets
Mean Ss Power	The average of the power of sender APs
Num of flows	Number of Hosts that send packet
Packet size	The size of packets sent by the considering host
Total flow size	Summation of all flows in the network

[5]Some parameters like power level should be in an acceptable range. The maximum possible transmitter output power in most devices is 30 dBm and there is no legal device that can support a transmit power of more than 40 dbm [65, 37].

[6]Several data cleaning operations are not discussed here for lack of space. For example, floods were replaced with the maximum possible rate and size. Some standard techniques were applied to create better trees - the mean size and rate were multiplied by the number of Sender APs and the sum of the distance to **h1**, **h2** and AP **s1** were divided by the number of sender APs. Where some records are important and they should not get pruned, were duplicated. Also, since we have only one class variable, which is a delay when packet loss occurs we assume the packet's delay was more than a threshold, so we set the maximum delay in that record to that threshold.

QoS Tree
numOfSenders
meanDistOfSendersToH1
meanDistOfSendersToH2
meanDistOfSendersToS
meanPacketSize
meanPacketRate
meanPowerOfSenders

Table 3: Access Points' Attributes

3.4 Simple Decision Tree for one flow

We built a tree based on part of the data to see the effectiveness of our method. We implemented this tree in the controller and created sufficient agents for APs to communicate with the controller and change their bandwidth usage based on the prediction tree.

3.4.1 Creating the tree

We used Weka [72] to build decision trees to decide whether a configuration may affect the QoS. First, we built the tree for tuples that contained only a single flow of 64 bytes (between **h1** and **h2**), and we aggregated all packets of one flow to one tuple by considering the maximum delay as its class variable. If the tree predicted a block or very high delay in the flow, the controller should react to it by lowering the data rates of other APs. The attributes that were used are shown in Table 3.

The class attribute was "maxDelay" for the QoS tree. The attributes are used to split the tree branches in each step and the class attribute is used in the leaf nodes as the result for subsequently predicting the delay for packets in the tagged flow.

Since the class parameters were continuous, we needed regression classification algorithms

Table 4: Evaluation of QoS trees

QoS tree	M5P	REP	Random
Correlation coefficient	0.9303	0.968	0.9709

and in Weka, we used "M5P", "REP", and "Random Tree". The correctness of the trees was checked with 10 fold cross-validation (see Section 2). The summary of QoS prediction trees is shown in Table 4. For each tree in Weka, the correlation coefficient is calculated to estimate the effectiveness and correctness of the tree. There is the exact value of variables and the value that the model estimated for that variable; The correlation coefficient indicates how much these two variables are related [18].

Considering the Table4, comparing correlation coefficients, it turns out that the random tree has the highest correlation (0.9709). So we believe it can be chosen as the representative tree to predict QoS.

In the QoS tree, by following the trees' flowchart, the delay for the flow in a specific configuration with specific parameters can be predicted. The tree is shown in Figure 12. As an example, in the Random Tree for QoS, if the packet size is less than 12300 bytes and the mean distance to **h1** is less than 30 m, then the maximum delay that can result will be about 1180 ms. As shown in Figure 12, in each branch of the tree, just some of the parameters are needed. This allows the controller to quickly make decisions when necessary.

3.4.2 Measuring the effectiveness of the tree

In order to see the effectiveness of the tree, it is embedded in the SDN controller to predict network performance and make topology changes to see how this will impact the network performance in practice. It is necessary to determine the importance of various parameters in prediction to minimize the monitoring overhead. Weka created the decision tree for us. We wrote a program to convert that tree into "if and else" conditions to aggregate it into the POX controller. POX uses port 6633 to communicate with APs, but this communication is

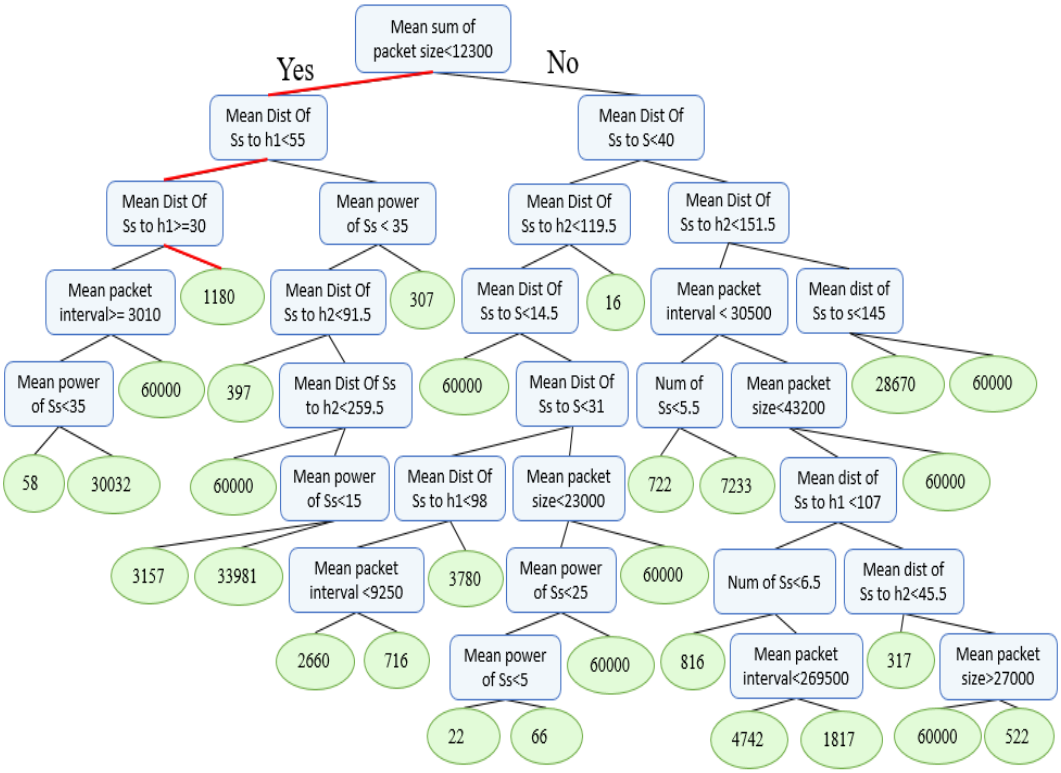

Figure 12: The selected decision tree

used for OpenFlow information. This is not enough for wireless networks and management and it cannot receive all required information. To provide a communication mechanism between the controller and APs, we used POX messenger, which uses port 7790 to receive information from APs. Then we created a Python agent on each AP which is responsible for communicating with the POX messenger. Agents build a message consisting of location information, packet rate, packet size, and power level and then send this message to the controller. Agents repeat this action each 100rtt (round trip time). The controller receives the information and uses the QoS tree to decide whether the required QoS is provided or not. If the maximum delay was more than acceptable for the tagged flow, it sends APs commands to reduce the rate by a factor of 10 the and size of packets by half. On the other hand, if there was extra bandwidth, APs may increase their packet size by 100 kilobytes and also increase the packet rate by a factor of 10. The agent on an AP will hear the commands and apply changes.

3.4.3 Results

Considering the amount of information that should be exchanged among the controller and APs, we repeated the experiment in 12 different situations, both before and after applying the QoS tree to see how it affects the QoS. The results are summarized in Figure 13. The percentage of changes in mean delay, max delay, and packet loss are negative which shows a reduction that concludes a higher QoS. There is a small increase in minimum delay in some cases due to the additional control packet and processing time in the controller. We calculated a 95% confidence interval for each parameter.

In Figure 13 the dots below the zero line shows a reduction in delay or loss. Considering the confidence interval, there was no big change in the case of minimum delay (a little increase due to extra communication between AP and controller), average delay, or packet loss. But the amount of maximum delay is decreased considerably, which is very good and beneficial especially in real-time applications or communications.

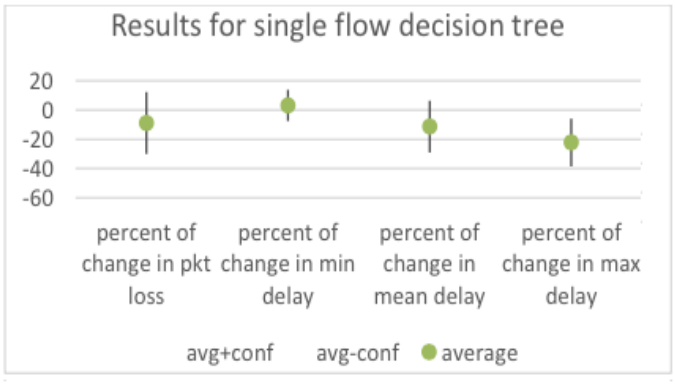

Figure 13: Results for single flow decision tree

3.5 Decision Trees for more flows

In this section, we used the entire database including the experiments with more than two hosts which are sending packets (multiple flows). Then we built different trees based on the whole database using REP, M5P, Random Tree, and J48 algorithms.

We first built a tree for the delay. In this tree, we can have more than one flow and the size of the flow can also change. In spite of the previous decision trees in this section, each packet of the flow formed a tuple, while in the previous one, each flow was summarized into one tuple. For each tuple, some of the other flow sizes are also added. The attributes that were used are as shown in Table 3.5.

As described in the following sections, we built 3 different trees using these attributes and different class variables. We first built regression trees for the delay and later we built classification trees.

3.5.1 Regression Tree for Delay

Here the class variable was the delay. The total number of instances in the data set was 206261. We do not show the tree due to space limitation (the size of the tree was large with 869 nodes). Table 6 shows the output of 10 fold cross validation on the tree. The important

QoS Tree
meanDistOfSendersToReceiver Host
meanDistOfSendersToSenderHost
totalPacketSize
totalPacketRate
meanPowerOfSenders
sumOfFlowSize

Table 5: Flow's attributes

variables in the REP tree were the distance of senders to the receiver, power level, and packet size. We also compare the decision tree methods with linear regression.

As we did in the previous section, we implemented this tree in the controller and created agents on APs. We also need to create some agents on hosts to send their packet sizes. We tested this tree in 12 different configurations and for 3 hosts which gives us 36 tuples (36 experiments) and we summarized the results. The results are shown in Figure 14. Like previous results, the maximum and mean delay and packet loss are all reduced while there is an increase in min delay in some cases due to the additional control packet and processing time in the controller. The dots below the zero line show a reduction in the delay or loss. The amount of average delay and maximum delay is decreased but based on the confidence interval, this tree was not as effective since the confidence range crossed zero.

Table 6: Evaluation of All Data Delay Decision Tree

QoS	Linear regression	REP Tree	M5P Tree	Random Tree
Correlation coefficient	0.39	0.804	0.795	0.80

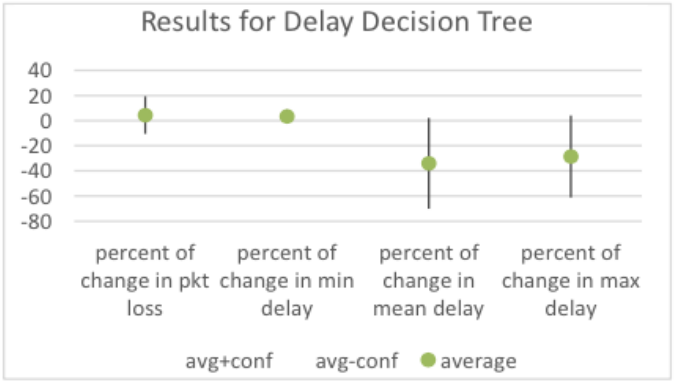

Figure 14: Results for delay decision tree

3.5.2 Delay Classification tree

In order to apply classification trees, we changed the class variable by setting a threshold of 100ms (the requirement for time-sensitive applications [166]) for the delay to label the class variable, considering the delay is more (false) or less (true) than this value. Then we built the tree using REP, Random Tree, and J48 algorithms in Weka.

The output of 10 fold cross validation on the tree is shown in Table 7. We also apply libSVM [46, 105] on our data to create the model and compare it with classification trees. The time it takes to build a tree for libSVM was two days, while it takes few minutes to build a tree with decision tree algorithms.

We chose J48 prediction tree since it has higher overall performance; REP has the lowest

Table 7: Evaluation of Classification Delay Tree

QoS	REP Tree	J48 Tree	Random Tree	libSVM
Correctly Classified Instances	82.95%	83%	83%	83%
Tree size	1709	1159	6541	---

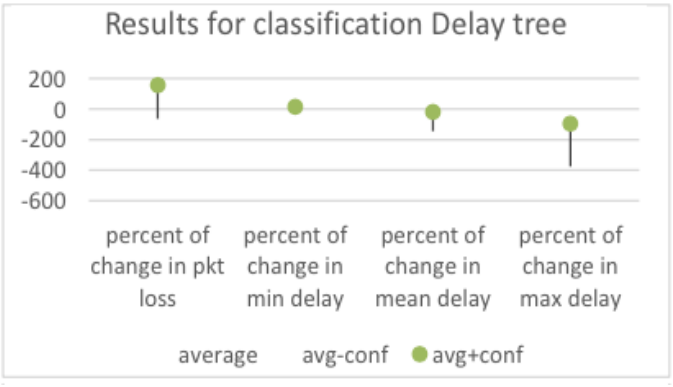

Figure 15: Results for delay decision tree

correctly classified items, libSVM was so slow (it takes two days to build the model) while it doesn't have higher correctly classified items. Random Tree has a tree 6 times bigger than J48 with the same correctly classified items. So in this case we pick J48 decision tree as our delay decision tree. The important variables in the J48 derived tree were packet size, the distance of senders to the receiver, and transmit power. We placed this tree in the controller as well and repeated the experiment as in previous sections. The results are shown in Figure 15. Although there is a slight increase in minimum delay due to sending control packets, we can see that the tree was much more effective and the average delay and maximum delay were both decreased. It especially has a big effect on decreasing the maximum delay. Although there is a slight increase in minimum delay due to sending control packets, you can see that the tree was much more effective and the average delay and maximum delay were both decreased. It especially has a big effect on decreasing the maximum delay.

3.5.3 Dynamic Tree

In order to update the tree by receiving feedback from hosts, each leaf is tagged with the value of the class attribute, the number of instances, and the number of incorrectly classified instances in that node. Then we store the conditions related to sequence number and leaves values in an update-able data structure (e.g., array).

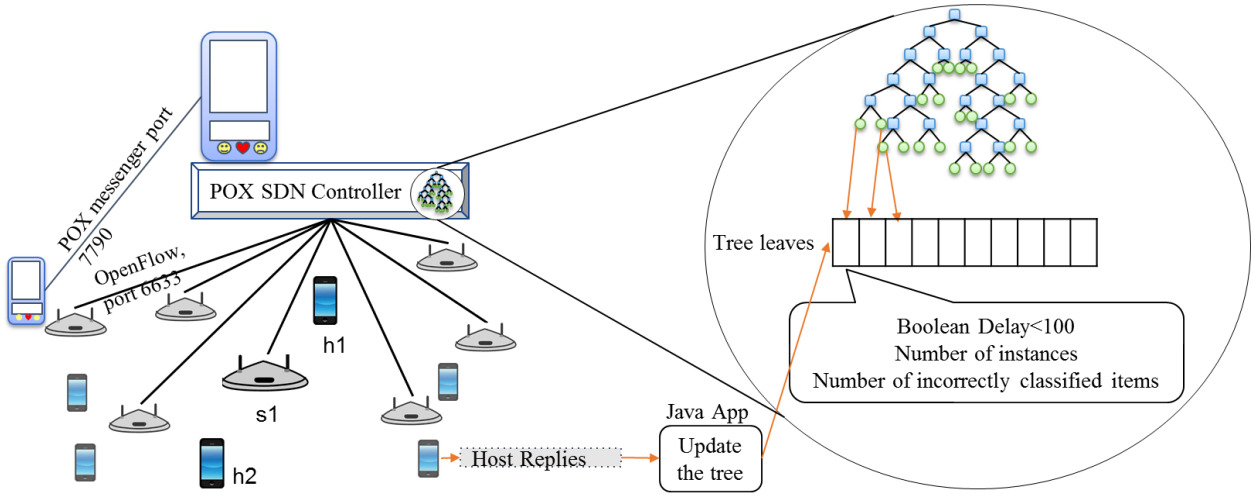

Figure 16: Dynamic tree

We put this tree in the controller and create agents on APs and Hosts to communicate with the controller and update the tree as new training instances get available as feedback from hosts. APs send their attributes such as location, packet rate, packet size, power level, etc. The controller reads values related to the condition from that data structure and replies with whether they should decrease their packet rate and packet size or not. Then the controller also receives feedback from the hosts to see how it affects the delay then it updates the tree using that feedback. The feedback is the number of instances and the delay values. In the dataset, the controller adds up the number of instances in one group and decides whether the class tag for that leaf should change. Based on the requirement the coefficient for old data (instances that are already in tree leaves) and new data (feedback from hosts) can be different.

We try this on the same network to see how it changes the tree, the results show that 19% of the times the statistics in tree leaves may change but the labels of the leaves do not change. This provides us with reasonable confidence in the stability of the trees and their ability to allow the SDN controller to manage the network.

3.6 Discussion and Limitations

Our objective in this work was to examine the potential of mining historical data towards wireless SDN management. SDNs aim to provide compatibility among different networks. In considering complex situations (e.g. in hospitals with different applications such as multimedia streaming, internet of things, medical devices, and personal area networks) we believe a good, yet simple, model based on historical data may be a great help in managing the network configuration, monitoring, troubleshooting, modifying and optimizing the network. SDNs can be applied in a large scale environment – management will be hard and it is necessary to make it automatic to reduce the amount of effort for network management.

Providing a reasonably accurate yet simple approach for SDN management is not trivial. Many different machine learning and artificial intelligence approaches have been applied in many applications, such as supervised learning (e.g. classification), unsupervised learning, and reinforcement learning (e.g. evolution and swarm algorithms and neural networks). In this work, we examined the applicability of simple models using historical data for automating wireless SDN management. We applied the models in the network to see whether there is an improvement in performance and also use feedback to update the decision trees. Previous works use historical data to create models but do not test the models in experiments nor do they get feedback from online data.

In this chapter, we focus on constructing a simple model with decision tree regression and classification techniques. Our reasoning is that trees allow an SDN controller to quickly check the important conditions for reconfiguration to see if the performance metrics (SLA) can be satisfied. This is in contrast to computationally expensive approaches that try to optimize the network performance. Our goal instead is to simply meet the performance metrics. We were also interested to make the tree more dynamic and use the feedback from hosts to optimize and update the tree and see the results over a longer duration of time. Decision trees may change in different networks with different configurations and different topologies. Also, multiple channels may need multiple trees.

A problem that is not covered in this chapter and planned for the future is to consider mobility and other complexities, more diverse data sets, and determining how much histor-

ical data is required for it to be effective in dynamic wireless networks. Clearly, this work is nascent and does not address several complex issues. In the simplest limitation, while considering mobile nodes that need heterogeneous QoS metrics to be satisfied, the decision trees may change substantially. This work also leads to hope for exploring automatic bandwidth management, the potential of using such models in dynamically slicing the network into partitions, and network reconfiguration. We can potentially use these kinds of models to automatically manage the network i.e., bandwidth allocation or slicing the network. Using real-world data sets which are not available at this time can also help us to confirm the results. Creating an optimal tree and evaluating the scalability and applying other dynamic tree approaches to update the model as new training samples are available are possible solutions to improve the models.

3.7 Conclusion

In this chapter, we used historical data to predict the quality of service and decide what flows to throttle (e.g. reducing packet size) towards managing an SDN wireless network where interference from competing transmissions may impact the quality observed by critical flows. In order to create a database consisting of different network configurations and traffic situations, we first simulated many SDNs with different topologies. In each topology, some hosts try to connect via an AP while other active APs cause interference, decrease the flow QoS and may block the media by overusing the bandwidth or sending with a high transmit power level. We collected the data and pre-processed them to achieve a clean set of data that consists of the number of actively interfering APs, APs' location, packet size, power, delay, etc. The data mining tool Weka is used to apply data mining methods to create one tree for the prediction of the QoS. Based on the QoS prediction tree, the SDN controller can decide whether a situation can meet the demanded QoS or not. The results show that such trees can be used by SDN controllers to rapidly managing the network to maintain QoS for critical flows. The controller can also use feedback from hosts to update the tree on a continual basis to improve the delay performance.

4.0 Software Defined Ambit of Data Integrity for the Internet of Things

On the Internet of Things (IoT), devices do not have the required computational power and storage capacity; and as a result, a variety of IoT devices may be required to outsource sensed or generated data to multiple heterogeneous cloud servers. We posit that it is the *Data Owner*'s responsibility to verify whether the stored data remain unchanged when the owner or some trusted third party further requires accessing this data. However, the "level" of this verification may be different under different *contexts* based on the application need. We propose four methods of integrity verification (which we call the *ambit of data integrity* – ADI) that considers the "toll" in terms of time, storage and communication by enlisting typically disparate integrity approaches under a single orbit. We adapt the notion of contextual integrity, previously used for assessing privacy grants, to extract important parameters required to decide on a suitable data integrity verification process. We propose a secure architecture using an integration of software defined perimeter (SDP) and software defined network (SDN) to perform authentication and gather each partition's context information for an SDN application to decide the proper integrity verification method that addresses the context requirements. To the best of our knowledge, this is the first time that the scope of integrity (or the data context) is used to determine the required layer of integrity verification in IoT.

4.1 Introduction

The Internet of Things (IoT), comprising of sensors and actuators, is increasingly changing human life by helping to solve new challenges. A multitude of IoT devices gather different kinds of data about the environment, the human body, social activities, etc., and enable people to make decisions or control actions [69].

In the near future, the environment around us is likely to see a profound transformation that can lead to a better quality of life, higher efficiencies through less waste, and reduced

costs. This transformation (described in [154] and paraphrased here) will involve user ownership of data that is generated (mostly) on their premises by a variety of wearable devices ("things" in the Internet of Things - IoT) and sophisticated, yet not very expensive connected devices. In an example of a healthcare scenario, the connected health devices could include blood pressure monitoring[1], electrocardiogram sensors[2], ultrasound probes[3] and even inexpensive DNA sequencing chips[4] that can identify inherited genes and chromosomes that may impact a patient's well being through drugs that are effective or dangerous to particular classes of patients. Although ensuring the self-integrity[5] of data being transmitted from the health monitoring devices is not within the scope of this work, recent advances show that there is no significant performance degradation between wearable devices and expensive medical tracking devices [157]; therefore, the information coming from an IoT device is equivalent in merit to the information generated by dedicated medical devices. Much of these data, while they may eventually be owned by a patient in an unforgeable blockchain, are today maintained in separate (perhaps multiple) cloud services that are owned by the vendors of the smart devices. The vendors may themselves be leasing storage, computation, and algorithms from various cloud services like Amazon's AWS or Microsoft's Azure or even lesser-known companies[6].

Increasingly, however, it is likely that the *owner* of the data may be freely able to move the data and/or allow other trusted parties to use them on the owner's behalf. These trusted parties should be able to retrieve the data from the cloud services that may not be necessarily reliable or trustworthy – in other words, the onus of assuring the integrity of data now lies with the owner. The owner may need to verify the integrity of data regularly *without having to download all of it or keeping a local copy of data from every device.* Besides, the trusted party (e.g., a healthcare or maintenance service or car service provider, etc. based

[1] `https://health.nokia.com/us/en/blood-pressure-monitor`

[2] `https://www.alivecor.com`

[3] see `https://www.lumify.philips.com/web/`

[4] `http://www.thermofisher.com`

[5] The data is correct and there are no flaws in generating or measuring data at any time so that knowledge that is gained from different parts of the data (measurements of different variables or different times of the same variable) make sense.

[6] It has been reported that increasingly companies are relying on multiple cloud providers (on average 8) for services [104].

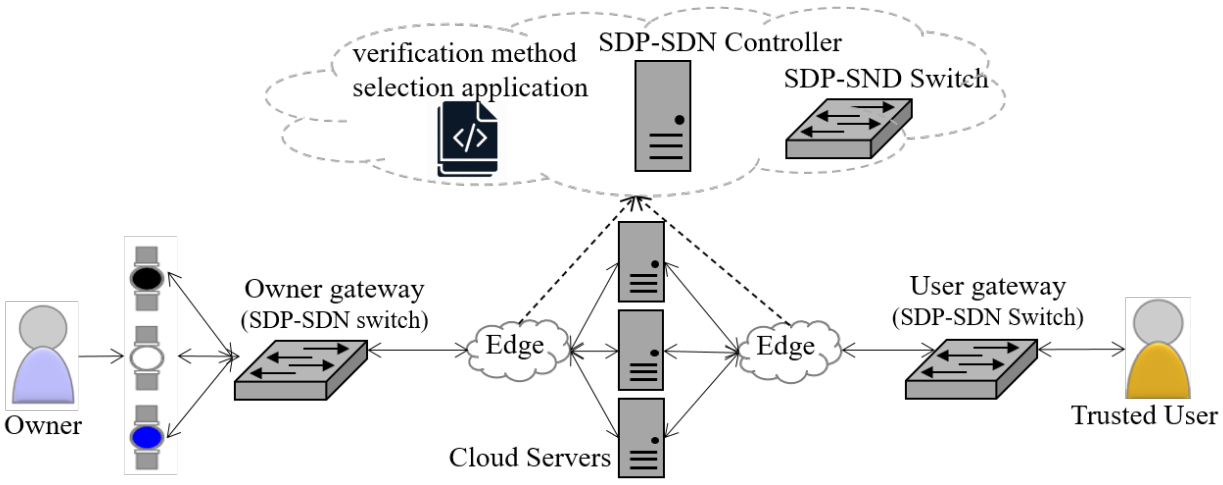

Figure 17: Motivating Scenario

on the nature of gathered data) may intermittently need to retrieve/use parts of the data. We capture this scenario in Fig. 17. The edge network, in our scenario, is a Software Defined Network (SDN) along with a Software Defined Perimeter (SDP) to provide security services for data storage, retrieval, and integrity verification. SDN and SDP allow network partitioning/security, gathering required context information for integrity verification (e.g., by exchanging information with installed agents on switches [86]), authenticating users/things, and provide secure access to cloud servers. The "things" are connected to the owner's gateway, which is an integrated SDP-SDN gateway and could even be a router/mobile phone with Open Virtual Switch (OVS) [124] installed on it. The gateway communicates with SDP-SDN controller which authenticates the owner and processes owners' request and provide access for transmitting the data to the appropriate cloud servers. Later, the gateway communicates with the cloud service and/or a trusted third party's gateway (with similar capabilities) to verify the integrity of the retrieved data.

The verification of the integrity of data may have different scopes or *ambit*[7]. Some data may not need strict cryptographic assurance of integrity but may need fast verification of

[7]The Mac OS Dictionary defines *ambit* as the "scope, extent, or bounds of something"

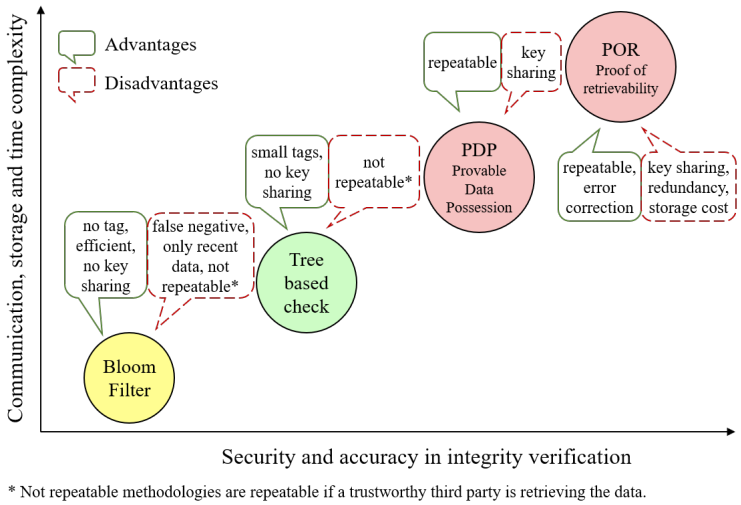

Figure 18: Integrity Ambit

the most recent data blocks or units, appropriate if the application and corresponding cloud service have higher obligations for reliability and security. An example of this is a healthcare provider (trusted party) who is looking at reports of a patient's (data owner) activity monitored by a wearable device. In this situation, the patient may be physically present for additional diagnosis – so fast verification is sufficient. Others may need strong cryptographic assurance to ensure that the decisions made are not based on falsified or modified information. Yet others may need protection against accidental or deliberate deletion (high availability). The "toll" on assuring data integrity and its verification needs to be commensurate with the scope/context of the situation. We develop a solution that considers what we call the *ambit of data integrity* or ADI to verify the integrity of data from multiple sources, stored in multiple clouds, under different conditions such as the time that the trusted party retrieves the data (e.g., to serve the owner) or the owner decides to check data integrity without having to download the data.

Security model of ADI relies on a trusted verifier (OGW) that belongs to the owner of the data. The SDN-SDP controller is responsible for authenticating parties (OGW, Servers, and UGW) and securing their communications. The OGW stores the data in (many) cloud

servers and generates and stores the key that can verify the integrity of data. A trusted third-party (UGW) that wants to retrieve the data, checks with the owner of the data through the gateway (OGW) to ensure that the data remain unchanged compared to what the data owner stored in the cloud. The UGW only shares small metadata that is sufficient for the OGW to verify the integrity of data for use by the UGW. Under these constraints, ADI is able to detect accidentally corrupted, forged, or fabricated data sent to the third-party retriever (UGW) instead of the original data. We assume that OGW is the trusted secure verifier and since UGW is the third party that requests the data, it only makes sense to trust this entity with sending the correct metadata to the verifier.

The contribution of this section can be situated in a manner similar to contextual integrity [117, 13], which, despite its name, has been used to indicate how the notion of *privacy* may change with context. We argue that data integrity checks should be context-dependent and scoped accordingly. We provide a hierarchical integrity verification framework that has 4 layers, including Bloom Filter (BF), encrypted Hash Tree, provable data possession (PDP), and proof of data retrievable (POR) as shown in Fig. 18 (These methods are explained in detail in Section 4.3). Each layer of integrity verification increases security or reliability but at the cost of higher communication, storage, and/or time overhead in comparison to the previous layer. We discuss a contextual integrity framework that determines the appropriate layer of integrity required based on the context.

The question we ask is when is it sufficient to use low cost but low latency methodologies (BFs or Hash Trees) and when to use highly secure methods (PDP and POR) that have significant overhead (in other words, what is the ambit?). Also, how can we systematically combine these approaches? Toward this, we adapt the concept of contextual integrity from Nissenbaum's theory [117, 13]. Here the determinant parameters are subject, sender, recipient, information type, and transmission principle (described later).

To the best of our knowledge, this is the first time that layers of data integrity verification have been considered for stored data from IoT in cloud servers. With ADI, the natural next step is to develop a secure architecture for it that addresses the requirements of its layers. We investigated various frameworks and after considering the requirements, we advocate

the combination of SDN-SDP in IoT networks with devices[8] acting as SDN switches. SDN supports network partitioning, gathering information about the context in each partition, variable packet flow headers, and flexible actions to be performed on flows. SDP supports authentication for the data owner and trusted parties to securely access the data they need from the cloud server.

The main **contributions of this chapter** are the 4-layer ADI approach, the use of contextual integrity to determine the required layer for data integrity verification for data from things stored in cloud servers, and an SDP-SDN architectural framework that addresses the requirements of ADI. To the best of our knowledge, it is the first time that the combination of SDN and SDP are proposed for IoT applications. These three contributions provide a novel and efficient solution to the problem of large-scale data integrity verification. We examine the parameters that can help decide what layer of integrity verification is required in each ambit and provide a secure framework for this process.

In the following, Section 4.2 summarizes the literature on data integrity and trust in IoT, contextual integrity, SDN, and SDP. Section 4.3 explores the building blocks of the selected methodologies including BFs, Hash Trees, PDP and POR. Section 4.4 explains the idea of using contextual integrity to decide what layer of integrity and therefore what layer of integrity validation is required based on the ambit. Section 4.5 provides the architectural framework for implementing this idea, Section 4.6 evaluates ADI and the architecture and finally, Section 4.7 concludes the chapter.

4.2 Literature Review

In this section, we first consider discussing some recent works in proving integrity verification in IoT networks. Then we briefly review the background knowledge required for ADI: the origins of contextual integrity and privacy and then SDN and SDP which are used to addresses the requirements of a network supporting ADI.

[8]These could be phones for a home network or Raspberry Pis or desktop all the way to an edge network with servers for smart grids or water systems.

4.2.1 IoT and data Integrity:

Providing data integrity, privacy, and trust in IoT networks has attracted much attention.

The authors in [68] used a cryptographic one-way hash to detect up to d defective items in a set of n items. They proposed a digital watermarking technique, to encode authentication information in the data structure D. They did this by modifying non-data fields, in a way that they should not be immediately identifiable by an adversary. In their model, the adversary modified the values of D but not the topology of D's pointers. The adversary had the knowledge of the algorithm but not the cryptographic master key. The authors built a program that identified up to *'d'* the number of changes and made it probabilistically difficult for an adversary to reproduce the database structure. The idea came from blood testing in which a test consists of selecting a sample including *'t'* items and performing a single experiment that determined if the sample contained bad blood. They produced a $t \times n$ matrix, that for any $d + 1$ columns there was one designated, with non-adaptive combined group testing scheme performed on each row. The column with a negative test result had 1 in a row and was removed. The remaining columns corresponded to bad elements. In this method, the only thing that needed to remain at the client-side was a key. In [68], the authors add some watermarks to the data in the cloud that has the complexity of $O(d^3 \log\ n\ \log\ d)$ in which *'d'* determines how many defective items can be identified and n is the number of bits.

As an example in the work in [15] discussed an amortized verifiable computation in which the client provided a function and an input to the server. The server replied with the answer and proof of the correctness of the result. The evaluation of polynomials was derived from very large data sets. Initially, the client stored the data on the server with some authentication information and kept a short secret key. The server computed a result with an authentication code. The client kept the clear text polynomial P and a vector of coefficients. The server had a vector of groups of the form $g_i^{ac} r_i$ in which r_i is the i^{th} coefficient of polynomial R and was calculated using a pseudorandom function. When queried, the server replied with $y = P(x)$ and $t = g^{aP(x)+R(x)}$ and the client accepts y if $t = g^{ay+R(x)}$. One application of [15] is in verifying outsourced computation to make predictions based on

polynomials fitted to many sample points in an experiment. Another application is in updating data and performing verifiable keyword searches and securing proofs of retrievability. For n variable for polynomial from degree d, assuming Decisional Diffie-Hellman the required time for the setup is $O((n+d)^d)$. After the setup, the required time is $O(nd)$ in the client and is $O((n+d)^d)$ in the server.

In another work, Catalano-Fiore [33] approach, a value m is encoded into 1-degree polynomial y, that $y(0) = m$ and $y(\alpha) = Fk(L)$, in which F randomizes the label. The server creates a new MAC with n authentication polynomials $(y1, y2, ...yn)$ that authenticate m as result of $f(m1, ...mn)$ and also $y(\alpha) = f(Fk(L1), Fk(L2), ..., Fk(Ln))$. Before this work, existing verification algorithms were not time efficient. The server sent m' to the client and he could test whether m' is the result of $f(m1, m2, ..., mn)$ by checking if $y(0) = m'$ and $y(\alpha) = f(Fk(L1), Fk(L2), ..., Fk(Ln))$. The authors wanted to avoid the time consuming $y(\alpha) = f(Fk(L1), Fk(L2), ..., Fk(Ln))$ part by safely reusing labels. They constructed a pseudo-random scheme that pseudo-computes a piece of the label. So they split the labels into 2-dimensions: the data set identifier and input identifier represented as (Δ, τ) that allows the same τ in labels. Also with pseudo-random function F using new amortized closed-form efficiency, if user pre-compute some information wf with same τs and different Δs it is possible to use wf to compute $W = f(Fk(\Delta, \tau), Fk(\Delta, \tau), ..., Fk(\Delta, \tau))$ in constant time. In this method, the client either should store the labels, which is a 2-dimensional matrix, and/or compute them, which has time and communication complexity of at least $O(n^2)$.

In [107] authors integrate new signature into current aggregated signature in which a single invalid signature invalidate the whole aggregated signature; however, in the secure aggregation scheme provided in [74, 73] the d-fault-tolerant aggregated signature verification algorithm can give the list of correctly signed messages providing that the number of incorrectly signed messages does not pass the bound (d). Secure aggregation may be more computationally intensive but the amount of communication for integrity verification will decrease. This scheme requires d-cover-free family (d-CFF). The characteristics of d-CFF is as the following: Consider set S with m elements, and set B with n subsets of S, it is not

possible to cover one subset with the union of any other d subsets[9]. Matrix M is used to show subsets which has rows for subsets and columns for elements. $M[i, j] = 1$ if the subset in row i includes the element on column j. Number of incorrect signatures should not pass d. Number of signatures in an aggregation (n) is bound by $\binom{n}{n-d} \leq 2^l$ where l is the length of aggregation. Claim is a pair of a public key and a message (pk ,m) which is used in the aggregation. The order and position of aggregating claims in each sequence of claims must be maintained. Considering that that C_1 and C_2 are two exclusively mergeable sequence of claims (have the same claim in each position and in one of them the claim should be empty) [74, 73].

The aggregation scheme is a an ordered list of Probabilistic Polynomial Time (PPT) algorithms $\Sigma = (KeyGen, Sign, Agg, Verify)$ that are defined as: $KeyGen$ creates a public private key pair or update key and securely erase the previous key. $Sign$ uses the key to create a signature σ for claim $c = (pk, t, m)$ with message m. $Verify$ takes claim and signature and outputs a validation (0 or 1). $AggSign$ creates an aggregated signature using C_1 and C_2 and their signature (τ_1 and τ_2) component-wise [74, 73]. The adversary may have pk and T and can query $Update$ (which updates the key and return Ok) and $AggSign$ for $T-1$ times, the adversary may also ask for current key which gives them sk but denied their further query. The scheme should be secure in the matter that no adversary should be able to pass the verification with the probability of at least ε in at most time t, by requesting at most q queries [74, 73]. Using d-CFF scheme, consider $C = (c_1, ..., c_n)$ and $C' = (c'_1, ..., c'_{n'})$, if there exists l different items among C and C' that has the same index but different values and $d = |n - n'| + l$; then there are d positions with errors in C'. If errors result in failure of verification of the subset that includes those d message, then due to d-CFF each correct messages can be verified in at least one row [74, 73].

Authors in Secure Logging the scheme used this fault tolerant scheme in detecting modifications in log files using sequential aggregation. Logging scheme must remain secure even if the malicious party gets the secret key; therefore, in each time epoch secret key is calculated based on the previous secret key and the previous secret key will be deleted securely [25].

[9]examples of d-CFF's constructions are based on concatenated codes, polynomials, algebraic-geometric Goppa and randomized constructions

This scheme has a security parameter of κ that specifies the running time of PPT. In this scheme concatenation ($C||C'$) would result in $(c_1, ..., c_n, c'_1, ..., c'_n)$ in which each claim for message m in epoch i is $c = (pk, i, m)$. The goal is to aggregate different messages signature, in any order, into one equal size signature that can authenticate all those messages, and still can be aggregated further [74, 73]. Aggregation signature required a lot of key sharing and key updates and hence is not suitable for internet of things.

In more recent works, In [160], authors provide a method to balance user privacy, data integrity in cloud servers, and computational cost by adding user-arbitrary weights while calculating the mean. They also provided biometric elliptic curve cryptography (using users' identity, password, and imprints) for authentication and used enhanced truth discovery technique (which includes assigning random weights) to still keep users' privacy [160]. In [61] and [2], authors modeled trust and reputation in IoT networks. In [103], a data integrity monitored method is provided to detect and isolate failures in sensor system (such as fault in time, format, and value). In [75], authors design a provable data possession scheme that can perform multiple updates at a time by using a Merkle Hash Tree that enables updating multiple leaves at the same time and updating the values of all their parents up to the root. From the above old and recent *integrity* related solutions and considering previously provided PDP and POR solutions [9, 8, 42, 7, 57, 59, 142, 168, 24, 23] –and to the best of our knowledge–, the context or scope of integrity has never been used to determine the required extent of integrity verification in IoT. In the following some PDP and POR methods are reviewed.

Proof of Data Possession (PDP) This section discusses methods from the literature that provide data verification class called provable data possession. [8] provides a provable data possession (PDP) at untrusted stores, using a fully additive homomorphic signatures; however, since it is not secure, they added a one time indices to make it secure. In Homomorphic message authentication, the user generated a set of tags that authenticated some values using a secure key. This method is able to dynamically adding blocks without re-tagging the entire file, support unlimited verifications, and with some variations it also supports public verifiability; However even in publicly verifiable modified version of this work it still requires to share the key with other entities which we avoid in HCI. Other works in PDP

such as [10, 63, 42, 9, 138] all tried to provide more efficient PDPs. In [66] a Diffie-Hellman and merkle Hash Tree is used which has the storage overhead as large as the file it self to reduce the server computation; however, this method increased communication. [63, 48] are RSA based PDP approaches that has the communication complexity and client storage complexity of $O(1)$; however, they have heavy computation on the server and performing RSA over a file is so slow. In [138], authors provided a method in which the signature of the parity blocks is equal to parity of the signature of the data blocks and use this method to provide a PDP approach; however, the communication complexity is $O(n)$. In [140, 163], authors used Diffie-Hellman based approaches; however, in both approaches the client has to store n bits per data block; therefore, these methods would not be efficient if data blocks are small. All these methods involve public key cryptography and they all require key sharing.

Block-chain based integrity checks remove the requirement of Third Party Auditors (TPA) [102]. The authors in [165] proposed a blockchain based Merkle tree for data integrity verification, those in [158] proposed a decentralized collective trust protocol that allows users to trace the history of data. The work in [101] proposed a decentralized data provenance auditor to verify data security through information in the block. However, these methods have large computational and communication costs due to large data sizes. In [159] authors provided a block chain based PDP that tried to reduce communication and computation load using homomorphic verifiable tag (HVT) which requires key sharing.

In Dynamic Provable Data Possession (DPDP) [57, 59, 56, 58], clients store data and a Skiplist in an untrustworthy server. The Skiplist is a data structure that keeps the meta-data of n blocks as leaves and rank upper layers as the number of accessible leaves. This method is based on rank-based authenticated Skiplists. The nodes in the search path are affected in the case of the insertion, modification, or deletion of blocks. The client keeps the label of the top-leftmost element in the Skiplist which is the root of the list. The validation of data integrity consists of the hash of nodes in the verification path (leaf to root) with the size of $O(\log(n))$. For updating the data, the client verifies the new proof and computes the new label of the root node after the update. The updating process affects nodes along the verification path with the length of $O(\log(n))$. This PDP method is used as third level of integrity verification in this chapter, which is investigated further in Section 4.3.

Proof of Retrievability (POR) In [82, 142, 53, 24] proof of retrievablity (POR) is provided which is applicable on encrypted files only and supports limited number of challenges. High Availability and Integrity Layer (HAIL) [23] provided a proof of retrievablity (POR) for a trusted verifier that checks the integrity of data and correct the errors where the file is distributed across multiple servers with redundancy across servers and a Byzantine adversary can corrupt multiple servers at a time. In HAIL the client stores the keys. It assures granularity of a full file by detecting server faults in a challenge-response reactive cryptographic system and recover corrupted files using cross server redundancy. The file is publicly verifiable even if it is encrypted. They build IP-ECC (Integrity protected error correcting code) which combines MAc and parity and aggregate responses by combining MACs across multiple blocks. In HAIL the client is the one that verifies the data and it does not support public verifiability. In HAIL the server code overhead is 9% of the file size for 1G data. HAIL is used as forth level of integrity verification in Section 4.3.

4.2.2 Contextual Integrity:

In Nissenbaum's theory, the term contextual integrity examines whether privacy is violated as context changes - i.e., the flow of information is *contained* as long as five parameters determine that the *integrity of the context* remain unchanged: subject, sender, recipient, information type and transmission principle, that conform to norms and ethical concepts [117, 13]. Three main domains affect privacy policies: government intrusion in individual privacy, sensitive personal information, and personal space. People perceive information as private based on social appropriateness, distribution of information, and changing norms. The desire to keep information private or to release it depends on parameters such as information flow (e.g., in social media, people share information to seek attention or snoop on others), awareness (e.g., the accuracy/type of location information is unclear), modesty (e.g., not sharing information to avoid bothering others) and maybe secrets [13].

In [81], a trigger action mechanism is used to provide contextual integrity, for permission requests by IoT platforms, based on users' opinions in different contexts. The permission system has two steps: first, it collects the information about the context (including infor-

mation about execution path, data flow, inter-procedure control, run time data value, user interface activity), and then it searches the dataset of mapped contexts and grants permissions approved by the user. If no match is discovered, it prompts the user and inserts a new record into the mapping dataset. The authors detect IoT attack execution paths while reducing the frequency of prompting the user [81].

In [40], authors present an information assisting agent that provides *implicit* contextual integrity. This agent uses an information model that considers relationships and information norms. It helps users to be aware of the information they are sharing in each context and avoids sharing sensitive information (previously unknown to the receiver). Context is defined by users, the topic of the information, and the knowledge of each user. Permission to share is granted based on appropriateness; appropriateness(B, t) is the probability that user B is willing to share information about topic t. The appropriateness is updated after each information sharing event, based on the knowledge each entity knows or learns [40].

As discussed above, the term contextual integrity has been used to maintain the integrity of the "context" to ensure privacy violations are minimized. We suggest *contextual data integrity* that uses the right mechanisms to verify the integrity of data based on the scope or ambit. One ambit may need quick verification, but if there is a potential for mischief, this may be mitigated by changing the ambit.

4.3 ADI Layers

In this section, we describe the hierarchical data integrity framework that contains four layers of integrity verification, as shown in Fig. 18, to ensure if the data, sent by an owner to be stored in one or more cloud servers, remain unchanged. The data verification process has 2 phases – Phase 1: Initialization (creating metadata or tags and storing the data along with tags in the server) and Phase 2: Challenge-Response (request and receive proof from the server, and verify the proof). In this framework, higher ambit layers have integrity verification methods that are more secure and/or reliable at the cost of increased overhead. The factors that we consider include latency, communication overhead, storage overhead,

tagging requirement, key sharing requirement, and repeatability (the server should not be able to use previous proofs in response to a new challenge or use them to generate new proofs; therefore, the server cannot claim to have the correct data without actually having the intact data [8]).

4.3.1 Threat Model

We have the following elements in our scenario:

- Adversary: This entity sends corrupted data to the third party (as an example, in the healthcare scenario, this hurts the patient because the doctor has incorrect information). This adversary can be of two forms: (1) The cloud server storing the data, which may be under attack, itself wants to forge the data or accidentally provides corrupted data blocks that harm the data owner. (2) An adversary that is a man-in-the-middle that sends wrong data to the third party instead of the cloud server and thus harms the data owner.
- Fully Trusted OGW: This entity is the data owner's gateway which is secure and trustworthy. It stores the generalized Bloom Filter and the key used for HMACs in building the tree. It performs data updates by storing data in the cloud, hashing them into the Bloom Filter prior to storage, and also hashing them securely in the tree.
- Trustworthy UGW: The UGW is the entity that needs to retrieve and use the data to service the data owner. If the UGW is not trustworthy it does not have to get any data and ask for verification from the OGW. It can harm the data owner directly without any of these hassles; therefore, the only scenario that makes sense is to assume that the UGW is trustworthy.

4.3.2 The first layer: Nested Bloom Filter

This approach uses a novel Nested Bloom Filters suitable for a low ADI. A Bloom Filter is a data structure that verifies whether an element is a member of a set or not using an empty array and set of k (unkeyed) hash functions (e.g., MD-5 - that are also not secure). It begins with an array of all 0s and then inserts a data block by hashing it using all k functions

with the hash outputs as an index. The BF sets the values in the array cells with the output index to 1. Later on, to check the membership of a data block, it hashes the block using all k hash functions and compares the output indexes with the array values in those indexes. If there is a match, i.e., they are all 1, the membership test is passed. Considering that the best number of hash functions k is $k = m \cdot \ln(2)/n$ (see [27]), the minimum required space in the BF for storing n items is m, which is calculated as $m \geq n \log_2 e \cdot \log_2(1/\epsilon)$ [27], where ϵ corresponds to the maximum fraction of the universe of false positives that is tolerable. As an example, if we use $n = 12$ and allow at most 1% false positive, m should be at least 115 bits. This method is fast, but it allows false positive and data cannot be deleted or modified in the case of change, because they remember old data; therefore, over time, by adding more data a BF can be saturated (all values turn to 1) and therefore the false positive probability can dramatically increase.

A solution to saturation is to use a Generalized Bloom Filter (GBF). In GBF [97], the initial filter is randomly filled by 0's and 1's. Then, k_1 hash functions set the bit and k_2 other hash functions reset the bits. This method limits false positives, but it introduces false negatives. Concatenation of GBFs (CBF) [113] which consists of multiple GBFs improves robustness and capacity, since there is less insertion and therefore fewer false negatives in each sub filter [97, 113]. GBFs (and therefore CBFs) tend to forget old data. Thus CBFs also address the problem of advertising saturated filters (all-one attacks). In this chapter, we use CBFs and we assign a dedicated part (subfilter) for each device that generates data for the owner; therefore, for each device the overall BF has the most recent data. Further, a device that generates data more frequently does not overwrite data from other devices that may make less frequent updates. The False Negative probability (FN) and False Positive probability (FP) calculation is discussed in Section 4.3.2.3, using formulas from [97]).

4.3.2.1 Formal Preliminaries for Nested Bloom Filter The verification system's protocols in Hierarchical bloom filter has the following functions for storing data and for verifying the integrity of retrieved data:

- $UpdateCBF(n, \{F_i\}_{i=0}^{n}, k, \{H_i\}_{i=0}^{k}, CBF, k_1, k_2, \{H'_i\}_{i=0}^{k_0}, \{H''_i\}_{i=0}^{k_1}) \rightarrow CBF$: This process is performed in the OGW and it updates the Concatenated Bloom Filter. Here, F

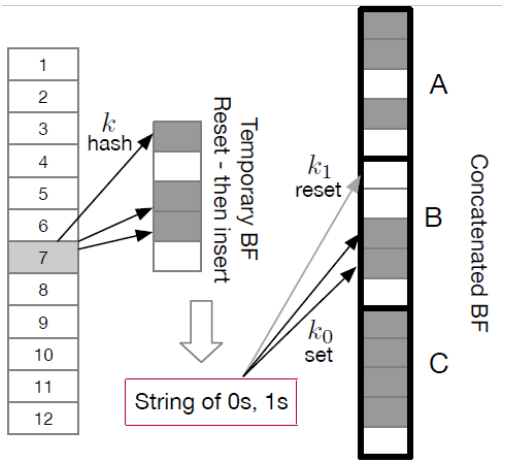

Figure 19: Layer 1. Bloom filter verification

is the file that has n blocks (F_is) of data. Each block has information about time and ID of the device that created the data. k is the number of hash functions in $(\{H_i\}_{i=0}^{k})$ which are used set bits in a simple Temporary Bloom Filter (TBF). CBF is the main Bloom Filter, k_0 is the number of hash functions $(\{H_i'\}_{i=0}^{k_0})$ that set bits in the CBF, and k_1 is the number of hash functions $(\{H_i''\}_{i=0}^{k_1})$ that reset the bits in the CBF. This `Update Concatenated Bloom Filter` function is implemented using Algorithm 1.

- $RequestQuery(time, ID_{device}) \rightarrow time, ID_{device}, ID_{cloud}$: This function finds the corresponding cloud based on the requested data ID_{device} (the ID of the device that generated that data). UGW forwards the request for the data from specific device in a specific timeline to the corresponding cloud services.
- $Respond(time, ID_{device}) \rightarrow \{F_i\}_{i=0}^{n}, n, subtree$: The cloud responds to the UGW with the matched blocks and number of those blocks.
- $PreVerification(n, \{F_i\}_{i=0}^{n}, k, \{H_i\}_{i=0}^{k}) \rightarrow \{TBF_i\}_{i=0}^{n}$: This process is done in the third-party user gateway (UGW). $\{F_i\}_{i=0}^{n}$ includes the set of blocks that are retrieved, n is the number of blocks, k is the number of hash functions and, $\{H_i\}_{i=0}^{k}$ are the hash functions for simple Bloom Filter. The output values which are the simple Bloom Filters $(\{BF_i\}_{i=0}^{n})$ are sent to the OGW. `Pre-Verification` function is implemented using

Algorithm 2.

- $Verification(n, \{TBF_i\}_{i=0}^{n}, CBF, k_1, k_2, \{H'_i\}_{i=0}^{k_0}, \{H''_i\}_{i=0}^{k_1}) \rightarrow \{0, 1\}$: The owner gateway (OGW) receives simple Bloom Filters ($\{TBF_i\}_{i=0}^{n}$) from the UGW and verifies the data. The output is reject (0) or verify (1). `Verification` function is implemented using Algorithm 3.

The algorithm for updateCBF, Pre-verification, and Verification methods are described in the following.

In Fig. 19 and Algorithm 1, we illustrate a nested BF approach. To initialize, each data block is hashed into an empty fixed-size temporary bloom filter (TBF). This TBF is hashed into a CBF where each section corresponds to a category of data (e.g., data from one wearable device is stored in its own section of the CBF). The CBF is stored in the owner's gateway.

Algorithm 1 `Update Concatenated Bloom Filter`

1: **input:** $(n, \{F_i\}_{i=0}^{n}, k, \{H_i\}_{i=0}^{k}, \text{CBF}, k_1, k_2, \{H'_i\}_{i=0}^{k_0}, \{H''_i\}_{i=0}^{k_1})$
2: **output:** CBF
3: **for each** $i \in \{1 \dots n\}$ **do**
4: $TBF : \{\forall j \in \{1 \dots k\} : \upsilon(H_j(F_i)) \leftarrow 1 \}$ ▷ Hash i^{th} block of data in TBF
5: $CBF : \{\forall j \in \{1 \dots k_0\} : \upsilon(H'_j(TBF)) \leftarrow 1 \}$ ▷ Hash TBF in CBF (set bits to 0 for k_0 number of hash functions)
6: $CBF : \{\forall j \in \{1 \dots k_1\} : \upsilon(H''_j(TBF)) \leftarrow 0 \}$ ▷ Hash TBF in CBF (reset bits for k_1 number of hash functions)
7: $TBF : \{\forall j \in \{1 \dots size_{TBF}\} : \upsilon(j) \leftarrow 0 \}$ ▷ Reset the TBF for future blocks
8: **end for each**

*$\upsilon(i)$ denotes the value of bit i in the corresponding Bloom Filter

During the challenge-response phase, as shown in Figure 20 the owner asks for the TBF of the intended block (Algorithm 2) and verifies the data (Algorithm 3) by checking the received TBF's against the locally stored CBF. If they match, a confirmation is sent otherwise the data is rejected and an alert is issued.

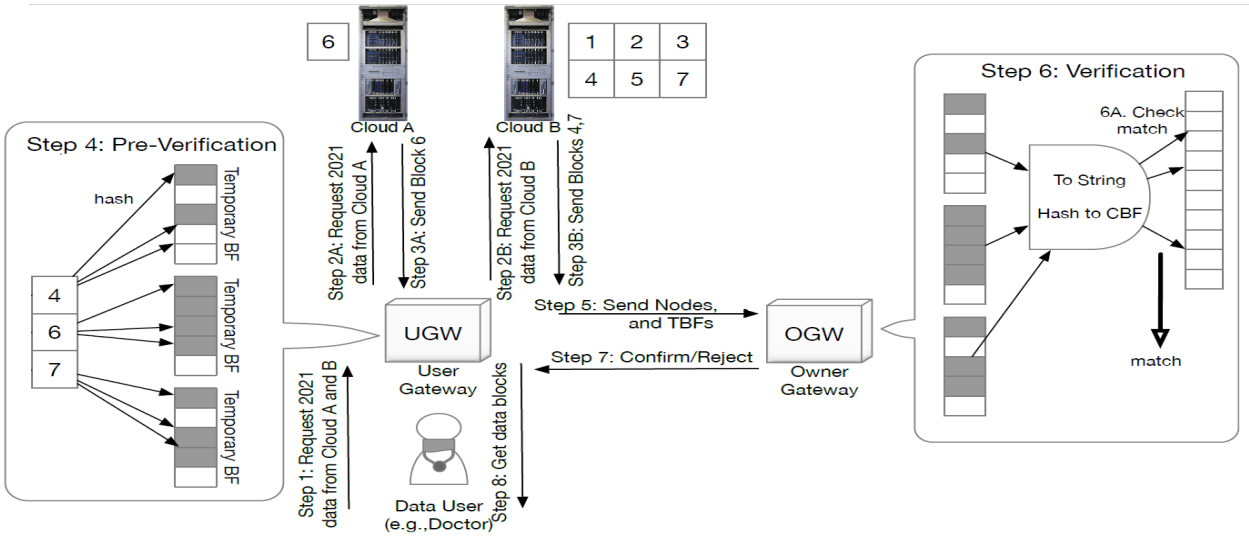

Figure 20: Verification with Nested Bloom Filter

Algorithm 2 `Pre-Verification in UGW`

1: **input:** $n, \{F_i\}_{i=0}^{n}, k, \{H_i\}_{i=0}^{k}, subtree$

2: **output:** $\{TBF_i\}_{i=0}^{n}, \{\text{SHA-3}(F_i)\}_{i=0}^{n}, subtree$

3: **for each** $i \in \{1 \dots n\}$ **do** ▷ For each new block

4: $TBF_i : \{\forall j \in \{1 \dots k\} : \upsilon(H_j(F_i)) \leftarrow 1\ \}$ ▷ Hash i^{th} block of data in TBF

5: **end for each**

*$\upsilon(i)$ denotes the value of bit i in the corresponding Bloom Filter

4.3.2.2 False Positive Probability for Simple Temporary Bloom Filter The size of the Bloom Filter is m and the number of data blocks inserted into the Bloom Filter is n; however, only one data block ($n = 1$) is inserted into the Temporary Bloom Filter. For such a simple Bloom Filter, the false positive probability (that a block that does not exist is believed to be correct) is calculated as $FP = (1 - e^{(\frac{-k \times n}{m})})^k$ [97]. Based on this formula,

Algorithm 3 `Verification in OGW`

input: $n, \{TBF_i\}_{i=0}^{n}, CBF, k_1, k_2, \{H'_i\}_{i=0}^{k_0}, \{H''_i\}_{i=0}^{k_1}, \{\text{SHA-3}(F_i)\}_{i=0}^{n}, \kappa, \ subtree$

output: $\{0, 1\}$

for each $i \in \{1 \dots n\}$ **do**

    $result_i \leftarrow min(\{\forall j \in \{1 \dots k_0\} : min\{\upsilon(H_j(TBF_i))\}\}, \{\forall j \in \{1 \dots k_1\} : min\{1 - \upsilon(H_j(BF_i))\}\})$ ▷ Check if received TBF exists in CBF

end for each

$resultBF \leftarrow \{\forall i \in \{1 \dots n\} : min\{result_i\}\}$ ▷ 1 if all TBFs exist in CBF

our analysis shows that for $m = 20$ the false positive probability would be less than 0.01 when k is between 2 and 50. As an example, the FP for $m \in (0, 20)$ and $k = 3$ is shown in Figure 21. For $k = 3$ and $m = 20$ we have $FP = 0.0027$.

4.3.2.3 False Positive and Negative Probability for a General Bloom Filter

The false-negative probability for a GBF is calculated in [97] and summarized in this section. Let us assume that k_0 is the number of hash functions that turn the bit to 1 and k_1 is the number of hash functions that change the bit to 0. The false-negative rate for a GBF is calculated as shown in the set of equations below (Equation Set 1) with comments:

Equation set 1: False Negative probability in Generalized Bloom Filter [97].

$$q_0 = (1 - e^{(-k_0/m)}) \quad \triangleright\ q_0 \text{ determines the probability of a bit reset}$$

$$q_1 = (1 - e^{(-k_1/m)}) \times e^{(-k_0/m)} \quad \triangleright\ q_1 \text{ determines the probability of a bit set}$$

$$b_0 = m \times q_0 \quad \triangleright\ b_0 \text{ is the probability of resetting } m \text{ bits}$$

$$b_1 = m \times q_1 \quad \triangleright\ b_1 \text{ is the probability of setting } m \text{ bits}$$

$$p_{00}(n-i) = [(1 - q_0 - q_1)^i + \frac{q_0}{q_0 + q_1} \times (1 - (1 - q_0 - q_1)^i)]^*$$

$$p_{11}(n-i) = [(1 - q_0 - q_1)^i + \frac{q_1}{q_0 + q_1} \times (1 - (1 - q_0 - q_1)^i)]$$

$$F_n = \frac{1}{n} \sum_{i=0}^{n-1} [(1 - p_{00}(n-i)^{b_0}) \times (p_{11}(n-i)^{b_1})] \quad \triangleright\ F_n \text{ is false negative probability}$$

* $p_{00}(n-i)$ is the probability of a bit that is reset in the $n-i$-th insertion remains 0 after the ith insertion; $p_{11}(n-i)$ is the same for a bit that is set.

Similarly, the calculation of the false positive for GBF is calculated in Equation set 2 below with comments:

Equation set 2: False Positive probability in Generalized Bloom Filter [97].

$$p = p_0(1-q_0-q_1)^n + \frac{q_0}{q_0+q_1} \times (1-(1-q_0-q_1)^n)$$

▷ p is probability of a bit remaining zero after n insertions

$$F_p = (p^{b_0}) \times ((1-p)^{b_1})$$

▷ F_p is the false positive probability

Based on Equation sets 1 and 2, and assuming that each owner generates one data block (each data block is 16 KB as suggested in [56]) each day and also assuming that in most cases users may require the data from the past year, in order to keep false positive as low as 1%, we calculate the minimum required size of the Bloom Filter for $n = 365$. With $m = 10700$ the false positive probability is less than 0.01 for $k \geq 3$[10] (where $k = k_1 + k_2$); Therefore, $m \geq 10700$ bits satisfies $FP \leq 0.01$. Figure 21 shows the FN and FP rates for $k \in \{3, 5, 7, 10\}$ as example. As shown in Table 8, in the case the owner is storing more data blocks each day, if we maintain $m = 10700$ bits, in order to keep FP less than 0.01, k should increase. As an example, if the owner (e.g., a patient) is creating 10 data blocks each day of the year (3650 data blocks), in order to satisfy $FP \leq 0.01$, k should be 7. $k = 7$ also satisfies $FP \leq 0.01$ for 100 data blocks a day (36500 data blocks each year). The reason that we do not select a higher value of k for is the FN. The false-negative probability increases with increasing k with a small number of blocks (check $k = 7$ FP and $k = 7$ FN in both Figures 21). It is more efficient to optimize k based on the application requirements as this reduces the time needed for integrity verification.

[10]The upper bound is an 8 digit number.

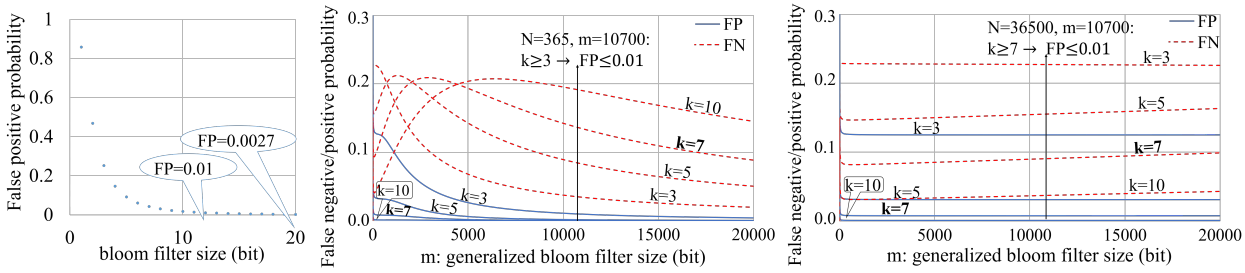

Figure 21: FP & FN probability; left: $n = 1$ & $k = 3$, middle: $n = 365$, right: $n = 36500$

Table 8: FP and FN probability for CBF size of 10700 bit

	$k = 3$		$k = 5$		$k = 7$		$k = 10$	
Blocks per day	FP	FN	FP	FN	FP	FN	FP	FN
365 (one block)	0.01	0.03	0.001	0.08	0.00026	0.13	0.000029	0.19
3560 (10 blocks)	0.10	0.17	0.028	0.21	0.0076	0.16	0.00097	0.09
36500 (100 blocks)	0.12	0.22	0.031	0.15	0.0078	0.09	0.00098	0.03

4.3.2.4 Analysis We investigate security (collision), storage, time and communication for Nested Bloom Filter verification method.

Security: In Nested Bloom Filter, data is first hashed into a simple temporary Bloom Filter (TBF). Then the TBF will be transformed into a string, and the string is hashed into a concatenated Bloom Filter. The probability of detecting a forged element as a valid one is the same as the probability of false-positive, which in Nested Bloom Filter would be equal to $FP_t = P_1 + (1 - P_1) \times (P_2)$ where P_1 is the probability of having a collision in the TBF and P_2 is the probability of having a collision in the CBF. As discussed in Section 4.3.2.2, the false positive for TBF is calculated as $FP = (1 - e^{(\frac{-k \times n}{m})})^k$. For $n = 1$, $k = 3$ and $m = 20$ bits, we have the FP or P_1 of 0.0027 (Figure 21). As discussed in Section 4.3.2.3 in Equation sets 1 and 2, in CBF, for $n = 365$ (one year data, assuming that each day 1 data block is created.), $k = 7$ and $m = 10700$ bits the FP or P_2 would be 0.0002 therefore, the overall false positive rate would be $FP_t = P_1 + (1 - P_1) \times (P_2) = 0.0027 + (1 - 0.0027) \times 0.0002 = 0.0029$. This value will change with time – for example, after two years the data is doubled and this value would be $FP_t = P_1 + (1 - P_1) \times (P_2) = 0.0027 + (1 - 0.0027) \times 0.0014 = 0.004$ when the number of blocks that are hashed into the Bloom Filters are doubled. Therefore, it is suggested to use the Hash Tree (layer 2 in ADI) for older data[11].

Storage: The cloud services store the data; the CBF can be stored either in the cloud server or the client gateway (OGW). The CBF has a fixed size (m) which has a storage complexity of $O(1)$. As shown previously, with $n = 36500$, to get a FP of 0.01 or less (with $k = 7$), m should be 10700 $bits$. In summary, the storage has the complexity of $O(1)$ (CBF) in addition to the data blocks.

Time: We have storing time, verification time and updating time. Storing the data includes the process of inserting the data into the CBF. For storing n data blocks, the time complexity of inserting data nodes into a CBF using k hash functions is $O(kn)$. The integrity check process includes comparisons between BFs and CBF; which is done in $O(1)$. The CBF does not have to be updated since it tends to forget old data.

[11] We tuned out variables so carefully (e.g., $k = 7$ and m=10700) that with n increasing, the FP would not exceed 0.01. As an example, for $n = 36500$, the FP is $0.0027 + (1 - 0.0027) * (0.0078) = 0.0104$ and from this point, increasing n does not affect FP. Choosing efficient k and m, based on the application requirement, can make a huge difference.

Communication: In the process of retrieving the data and verifying the integrity of data, there are two places that have communication overhead. The communication between the cloud server and UGW (fetching the data) which is 0 in Nested Bloom Filter (the cloud only send the data to the third party), and communication between the UGW and OGW (for verifying the integrity of the retrieved data blocks), for which, the UGW sends a temporary BF ($O(1)$) per block.

4.3.2.5 Experiments An implementation in hardware can help us to make a better sense of the performance of the Nested Bloom Filter method in the real world. We implemented a server (to emulate the cloud) on a desktop computer using a Mac operating system (with 2.66 GHz Quad-Core Intel Xeon processor and 8 GB 1066 MHz memory), and Java version 8. We also implemented the OGW and UGW on two Raspberry Pis version 3 Model B, with a Raspbian operating system, running Java 8. Our justification is that a Raspberry Pi can emulate an IoT gateway (inexpensive but perhaps running a full OS).

The elements (server, OGW, and UGW) communicate with each other using Java sockets[12]. We tested this implementation on different networks including a wireless network (WiFi with a data rate of 28 Mbps using NET GEAR N750 wireless dual band gigabit router), wired Ethernet network (using Cisco gigabit smart switch SG 200-08), and finally, in order to eliminate the effect of network connection speed and processing power of Raspberry Pis, we also performed experiments on a single system (PC) for comparison.

We used crowd-sourced fit bit data sets from [62]. The devices in our experiment include HRM (heart rate monitor), activity tracker, and calorimeter. We assumed that each device stored its data in a corresponding server (on the desktop). We assumed 36500 data blocks (each block 2^{14} bytes as suggested in [56]) as information from a single year (each day generates 100 blocks of data) to select the tuned Bloom Filter size (m value) for tolerable FP and FN rates. As discussed in Section 4.3.2.3, to satisfy an FP rate of at most 0.01, in CBF the value of $m = 10700$ bits is enough (considering that k should be chosen based on the number of data), as shown in Figures 21. In the temporary BF, as shown in Figure 21,

[12]The implementation is available on Github and can be reached at https://github.com/Maryam-mary-karimi/HDI

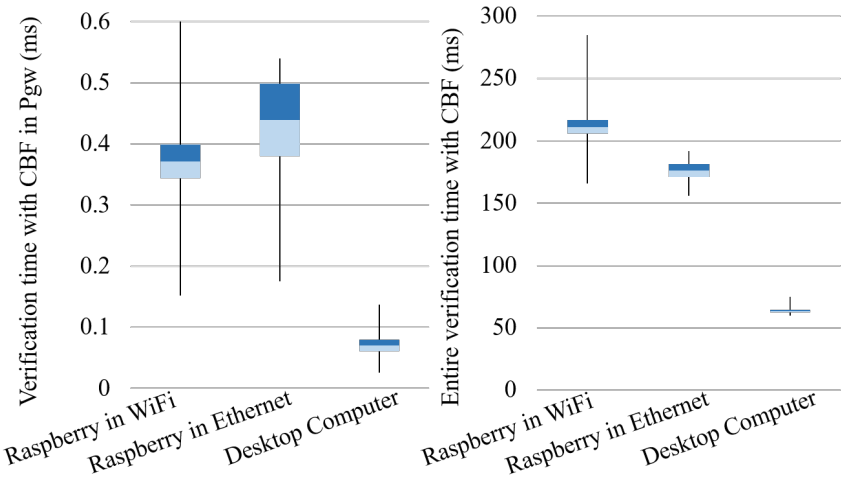

Figure 22: Verification time with bloom filter

the size as small as $m = 20$ bits gives us the overall FP of less than 0.01.

In order to evaluate the performance, we measured the execution time of verification processes. We repeated each experiment 100 times and calculated the average with a 95% confidence interval (as shown in Figure 22). In this experiment, we measured the verification time, starting from the time that the UGW receives the data and metadata from Servers. The UGW then starts negotiating with the OGW and the OGW verifies/rejects the data and replies to the UGW. We also measured the required processing time for the OGW to verify the data with CBF. In all cases, the integrity verification time using only the CBF took less than 0.5 ms. Considering the real-world situation (using WiFi as the communication network protocol and Raspberry Pis as gateways), verifying the integrity of data blocks using CBF took only 0.37 ms on average in the OGW, and in this situation, the entire verification process took only 207 ms (considering the PC experiment which eliminates the network delays this time would be 63 ms).

As our simulations show Nested Bloom Filter verification is very fast. This method however is not "repeatable" if the recipient is a cloud server that stores the data. It works well with a trusted third-party retrieving the data and asking the owner's gateway to verify the data for it, without having to send the data to the owner's gateway, without sharing any

keys or tagging of the data, requiring a small fixed size extra storage on the owner's gateway (or in the server). The nested BF reduces the communication cost of integrity validation since the third party can send the TBF of the data block instead of the whole block. This layer has the lowest storage cost and delay; however, the drawbacks of this method are the false positive (FP) and false-negative (FN) probability in the TBF and CBF (which is negligible for more recent data) and lack of explicit security guarantees. This method is suitable for situations where the owner has an extremely low resource for computation, the bandwidth is limited, sessions are short, the storage server is most trusted, the stored data is not highly sensitive, the most recent data is relevant, historical data is not usually needed and a trusted third party (or the owner) is retrieving the data itself (not simply checking for its existence). Experiments show that this method can verify data in less than 1 ms, the communication cost is 20 *bits* (size of TBF) per data block, the storage cost in the owner's gateway is 1.3 KB (independent of the number of the blocks) and no storage overhead in the cloud server.

4.3.3 The second layer: Hash Tree

Supporting ADI is a novel Hash Tree verification. We use secure hashes – SHA-3 [17] (unkeyed) and HMAC [92] (keyed) hash functions to build a secure tree which we employ in data integrity verification.

4.3.3.1 Formal Preliminary for Hash Tree The Hash Tree has the following functions for storing data and for verifying the integrity of retrieved data:

- $KeyGen(1^{\lambda}) \rightarrow \kappa$: The OGW Generates a random private key κ of size of λ from a keyspace K. κ is used for building the tree hash values using HMAC [13].
- $UpdateTree(n, \{F_i\}_{i=0}^{n}, subtree, \kappa, d) \rightarrow subtree$: This process is done in OGW. F and n are defined previously. *subtree* is received from the cloud and will be updated using SHA-3 values of the tree nodes and HMAC values of the cluster of nodes (nodes with similar attributes (device ID, creation time, etc.)). The key κ is used in the HMAC and

[13] We assume that the key size is commensurate with the needed security.

d is the maximum number of children each node in the tree can have. The `Update Tree` function is implemented using Algorithm 4.

- $RequestQuery(time, ID_{device}) \rightarrow time, ID_{device}, ID_{cloud}$: This function finds the corresponding cloud based on the requested data ID_{device} (the ID of the device that generated that data). UGW forwards the request for the data from specific device in a specific timeline to the corresponding cloud services.
- $Respond(time, ID_{device}) \rightarrow \{F_i\}_{i=0}^{n}, n, subtree$: The cloud responds to the UGW with the matched blocks and number of those blocks along with the *subtree* corresponding to those blocks.
- $PreVerification(n, \{F_i\}_{i=0}^{n}, subtree) \rightarrow \{\text{SHA-3}(F_i)\}_{i=0}^{n}, subtree$: This process is done in the third-party user gateway (UGW). $\{F_i\}_{i=0}^{n}$ includes the set of blocks that are retrieved and n is the number of blocks. The output values includes the SHA-3 values ($\{\text{SHA-3}(F_i)\}_{i=0}^{n}$) for each block and the *subtree* are sent to the OGW. This function is implemented using Algorithm 5.
- $Verification(n, \{TBF_i\}_{i=0}^{n}, \{\text{SHA-3}(F_i)\}_{i=0}^{n}, subtree, \kappa) \rightarrow \{0, 1\}$: The OGW receives the SHA-3 values and the *subtree* from the UGW and verifies the data. The OGW uses SHA-3 values, the HMAC function, and the key (κ) to rebuild a tree and compares it to the received *subtree*. The output is reject (0) or verify (1). `Verification` function is implemented using Algorithm 6.

The algorithm for UpdateTree, Pre-verification, and Verification methods are described in the following. As shown in Algorithm 4 and Figure 23, in order to initialize, the owner builds a tree by first securely hashing the data blocks (say SHA-3) and then grouping SHA-3 values based on features such as the device, date, and then applying HMAC on SHA-3 values in one group. As an example, in Fig. 23, the nodes $1, 2, 3, 5$ were created by device B in year 2020. In stage 2 of the initialization, the leaves of the tree (SHA-3 values) are omitted and the HMAC tree is stored in the corresponding cloud server along with the data. The HMAC key remains in the owner's gateway.

Algorithm 4 `Update Tree`

1: **input:** $(n, \{F_i\}_{i=0}^{n}, subtree, \kappa, d)$
2: **output:** subtree
3: **for each** $f \in \{F_i\}_{i=0}^{n}$ **do** ▷ For each new block
4: insert the node n under the proper node ▷ Proper node is leaves' parent (second layer) with proper device ID and date
5: split the branch if the number of nodes exceeds d ▷ shown in Figure 24
6: s ← SHA-3(f) ▷ Calculate SHA-3 value for the new block
7: $n \leftarrow HMAC((s|\forall x \in \{siblings\} : s = s \oplus \text{SHA3}(x)), \kappa)$ ▷ new HMAC
8: **for each** np in the path from n to subtree root **do**
9: $np \leftarrow HMAC((s|\forall x \in \{npsiblings\} : s = s \oplus x), \kappa)$ ▷ Update HMAC value for nodes in the path to the root
10: **end for each**
11: **end for each**
12: omit leaves from the subtree

In the challenge-response phase (as shown in Figure 25), the owner asks for SHA-3 values of data blocks and the HMAC tree. The owner builds a tree using the SHA-3 values and compares it with the received tree to verify its integrity. Integrity verification can be done reasonably fast; however, it is not repeatable. Previous SHA-3 values could be used as proof that the data exists unless the data are retrieved by a third party (e.g., the healthcare provider) that then generates the SHA-3 values from them. The owner's gateway simply verifies the data integrity for the third party, without having to retrieve the data itself and without sharing any key. This method is suitable when the cloud server is trusted and is not overloaded, the stored data needs a level of integrity and a trusted third party is retrieving and needs data from several time periods, not necessarily the latest. Our analysis and experiments show that this method can verify data in 26 ms, communication cost is 2.94 KB (tree and SHA-3 values) for each data block and the storage overhead cost in the server is 35 KB for storing 1G file including 2^{16} blocks.

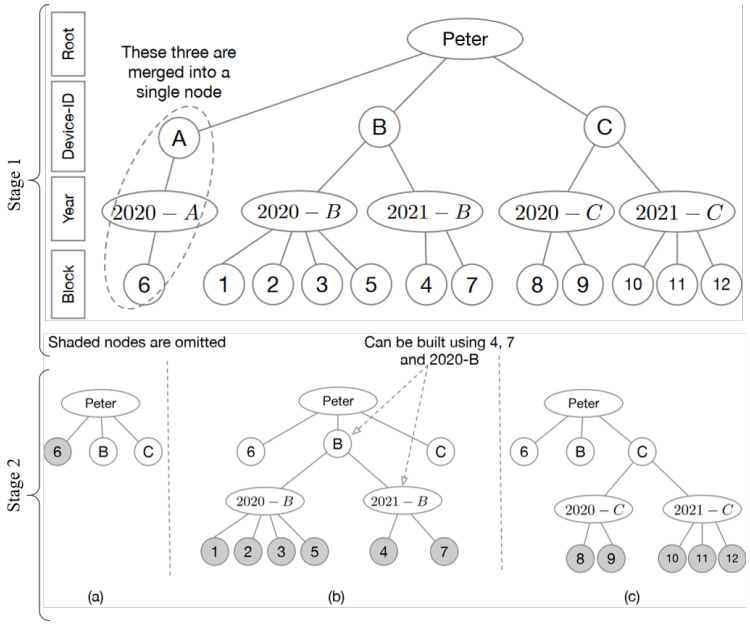

Figure 23: Layer 2. Hashing tree verification

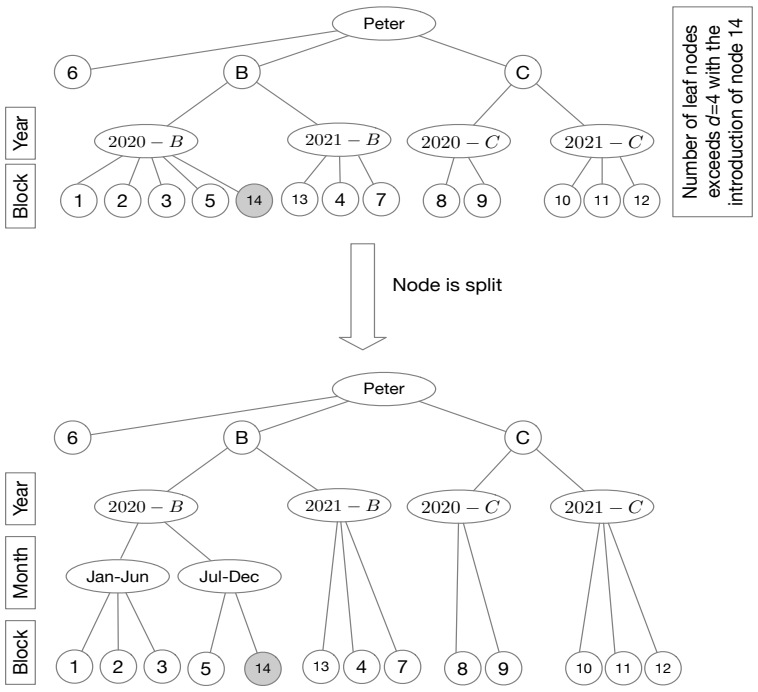

Figure 24: Splitting nodes when the number of children exceeds d (here $d = 4$)

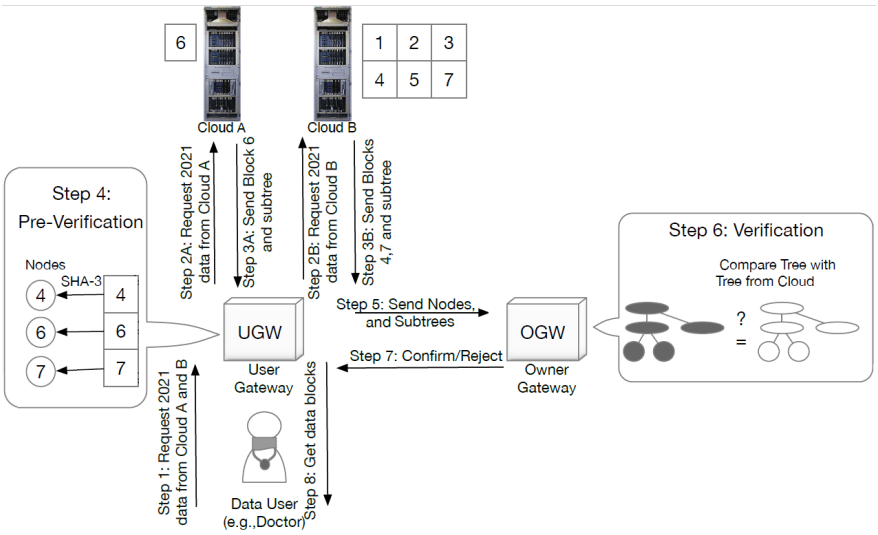

Figure 25: Verification with Hash Tree

Algorithm 5 `Pre-Verification in UGW`

input: $n, \{F_i\}_{i=0}^{n}, subtree$

output: $\{\text{SHA-3}(F_i)\}_{i=0}^{n}, subtree$

for each $i \in \{1 \dots n\}$ **do** ▷ For each new block

    $SHA\text{-}3_i \leftarrow SHA\text{-}3(F_i))$ ▷ Calculate SHA-3 values

end for each

Algorithm 6 `Verification in OGW`

input: $n, \{\text{SHA-3}(F_i)\}_{i=0}^{n}, \kappa, \ subtree$

output: $\{0, 1\}$

Build a new-subtree using SHA-3 values:

    Merge SHA-3_i values with the same device-ID and time interval to create HMAC parents

    Merge HMAC parents with the same device-ID to create a parent

$resultTree \leftarrow compare(subtree, new\text{-}subtree)$ ▷ Compare the built subtree and received subtree, output is 1 if they are equal

In order to reduce the amount of data required to be fetched from the server for verification, we suggest choosing d (the number of children for each parent in the tree) between 2 and $\log(n)$; The lower bound is 2, in view of the fact, that it is more efficient to merge the child and parent into a single node if a node has only one child. The upper bound is $\log(n)$, since it limits the number of extra sibling data blocks that have to be fetched when data is being retrieved and verified; therefore, the communication complexity between cloud server and UGW would be more efficient and is limited to $O(\log(n))$ (considering that subtree size complexity is also $O(\log(n))$). As shown in Figure 24, if any node has more than d children, the tree splits into two branches (in this example, we selected d to be 4). Here we have shown this split happening between the first six months and the next six months of the year for illustration only - the split can be accomplished using other factors.

Another refinement that can be done to reduce the communication complexity during the verification process is to avoid sending the middle nodes of the subtree that can be rebuilt using the received SHA-3 values; therefore, in such a case, it is sufficient to send SHA-3 values and some high-level common parents. As an example in Figure 23(b) nodes "$2021 - B$" and "B" can be rebuilt using current information (SHA-3 for blocks 4 and 7 and HMAC values for 2020-B) and sending them as a part of the subtree will not add to the security of the integrity verification; furthermore, in the verification phase step $6 - B$, it is not required for those nodes (2021-B and B) to be compared to the rebuilt tree.

Yet another possibility that reduces the communication overhead is to send the SHA-3

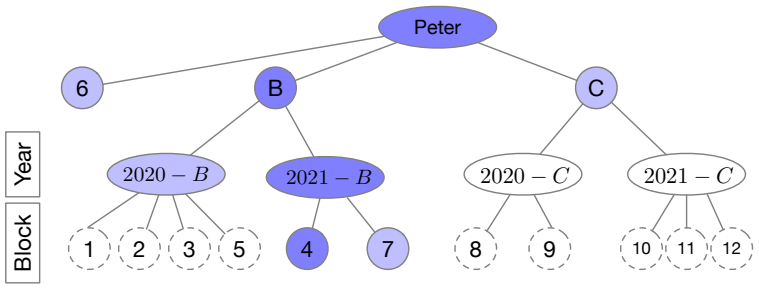

Figure 26: Subtree for retrieving block number 4

values and the lowest common parent(s). As an example, in Figure 26 to retrieve and verify the integrity of node 4, only the SHA-3 value of nodes 4 and 7 and the $HMAC$ value of the node $2021 - B$ are needed [14].

4.3.3.2 Analysis We analyses security (linearity and collision), storage, time and communication for Hash Tree.

Security: One of the advantages of Hash Tree is that the hash functions **do not have to be linear or homomorphic** as in most (if not all) provable data possession (PDP) and proof of retrievability (POR) provided solutions. Here we chose SHA-3 and HMAC but other secure hash functions can be used as well (e.g., SHA-512). In the tree, the collision happens based on the collision probability in SHA-3, which is negligible ([119]).

Storage: The cloud services store the data and the tree. The number of children each node of the tree can have is between 2 and d ($2 \leq d \leq \log(n)$). The number of nodes in the tree can be calculated using a geometric progression sum as $S = \frac{\left(1-d^h\right)}{1-d}$, in which h is the height of the tree (considering the value of d, h is between $\log_d(n)$ and $\log_2(n)$). Assuming we have n data blocks, the number of nodes in the worst case, when each node has only 2 children, is $(n-1)$ (note that the leaves are omitted). Thus, the cloud storage of the tree is

[14] As another example, in Figure 26, to retrieve and verify the integrity of nodes 5 and 7, UGW sends SHA-3 values for 1,2,3,5,4,7 and HMAC for B to OGW. In a more complicated example to retrieve and verify the integrity of nodes 7 and 8, UGW sends SHA-3 for 4,7,8,9, and to create node Peter we also need the HMAC of node A. There is no need to send node B since it can be created using SHA-3 values. However, OGW still needs 2020-B (there is no need to transmit node 2021-B since it can be created using SHA-3 values), the same situation is true for node C that needs node 2021-C for verifying data integrity.

$O(n)$. In summary cloud storage has the complexity of $O(n)$ (tree) in addition to the data blocks.

Time: Storing the data includes the process of creating the tree. To create the tree, all blocks need hashing (i.e., a calculation of SHA-3 or HMACs values) which is fast to compute [119, 92]. For a tree with $2n-1$ nodes,[15] this process has the time complexity of $O(n)$.

The integrity check process includes sub-tree rebuilding. There is $\log(n)-2$ number of HMAC calculations to rebuild the sub-tree; therefore, the overall verification process time complexity is $O(\log(n))$

To update the tree (change, insert, delete), the SHA-3 or HMAC for the siblings is retrieved and the nodes along the path to the root should be updated $(\log(n)-2)$. The CBF does not have to be updated since it tends to forget old data. The overall time complexity for updating process is $O(\log(n))$.

Communication: In the process of retrieving the data and verifying the integrity of data, there are two places that have communication overhead. The communication between the cloud server and UGW (fetching the data) and communication between the UGW and OGW (for verifying the integrity of the retrieved data blocks).

In fetching the data, the cloud server sends the data ($O(n)$) and the sub-tree to the UGW $O(\log(n))$. The sub-tree is the certain overhead in Tree, which has the complexity of $O(\log(n))$ (n is the size of the retrieved data). Based on the nature of the user's query, the data may need to include not only the requested data but also sibling blocks of the requested data. In the scenarios, we considered, this situation happened rarely in experiments, where the user requested data from a random time period and random device and mostly included all of the children of one parent node. In rare cases that sibling data blocks are required, in a worst-case scenario, the number of overhead sibling data blocks would be $d-1$ data blocks (each block is 16 KB as suggested in [56] and considering $2 \leq d \leq \log(n)$.); therefore, the complexity of communication overhead between the cloud and the UGW including sibling data blocks ($O(\log(n))$) and subtree ($O(\log(n))$) still would be $O(\log(n))$.

[15]Building the tree involves n number of $SHA-3$ computations and at most $n-1$ number of $HMAC$ computations. Although the $SHA-3$ values are omitted from the tree, they affect the time complexity of computation.

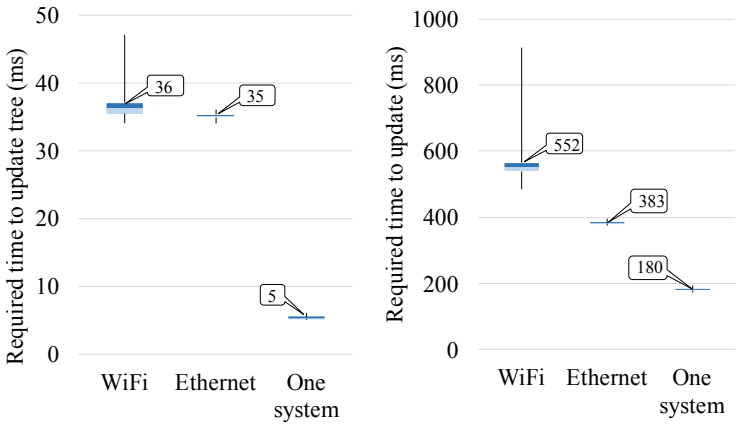

Figure 27: Updating time with Hash Tree

In verifying the data, the UGW sends the sub-tree ($O(\log(n))$) and SHA-3 of blocks ($O(1)$ per block – we have at most d blocks due to the limited number of children in the tree ($2 \leq d \leq \log(n)$); therefore, the complexity is $O(\log(n))$ to the OGW. Hence, the entire communication overhead complexity between OGW and UGW is $O(\log(n))$.

In order to evaluate the performance, we measured the execution time of updating and verification processes using the same framework we used for evaluating Nested Bloom Filter. We repeated each experiment 100 times and calculated the average with a 95% confidence interval.

The experiment outcome (time) for the updating process is shown in Figure 27. The required time for updating the tree (left) and the entire updating process (right) was measured. Updating the tree involves requesting the related part of the tree from the server, creating new leaves (SHA-3 values) for new blocks of data, and rebuilding the tree. This process took 36 ms using the wireless network, 35 ms in a wired network, and 5 ms in a single system, on average. The entire updating process including receiving data, updating the tree, and sending the new data and the tree to the server took 552 ms using WiFi connection, 383 ms in wired connection, and 180 ms in a single system, on average.

Figure 28 shows the required time to verify the data in owner's gateway using tree in the left chart and the entire verification time in the right chart. The verification process

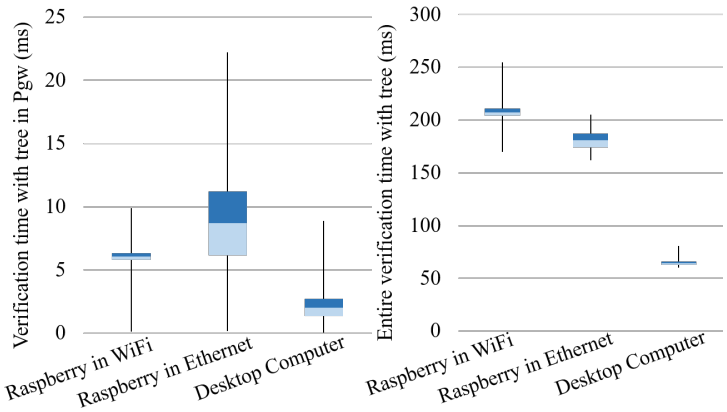

Figure 28: Verification time with Hash Tree

is shown in Figure 25. We measured the verification time, starting from the time that the UGW receives the data and metadata from Servers. The UGW then starts negotiating with the OGW and the OGW verifies/rejects the data and replies to the UGW. We also measured the required processing time for the OGW to verify the data both with CBF or the tree. The averages in the experiments yield that the number of nodes in the tree is 377.7 and the average size of the sub-tree that should be sent to the UGW contains 50.19 nodes. As shown Figure 28, considering the real-world situation (using WiFi as the communication network protocol and Raspberry Pis as gateways), it takes 6.1 ms for the OGW to verify the data using the tree.

4.3.4 The third layer: Provable Data Possession

In Provable Data Possession (PDP), the problem is that the owner needs to know that the correct data still *exists*. We choose the provided method in [57, 59, 56, 58], which is called dynamic provable data possession (DPDP) in which clients store data in a rank-based authentication Skiplist (a data structure that keeps the meta-data of n blocks as leaves and ranks upper layers as the number of accessible leaves) in an untrustworthy server. The nodes in the search path are affected in the case of the insertion, modification, or deletion

of blocks. It tags the data using fully additive homomorphic signature (In homomorphic message authentication, the owner generates a set of tags that authenticates some values using a secure key and using one-time indices to make it secure. Later on, the owner challenges the server for proof of data possession along with the tag. The tag and response are used to verify that data are still stored in the cloud server. This method can dynamically add blocks without re-tagging the entire file and can detect 1% of data change with a probability of 99%. It is repeatable, meaning that, the proof should be different each time to make it impossible for the server to use the same proof for further challenges while it no longer owns the (intact) data and supports an unlimited number of verifications; however, it is computationally expensive. Formal Preliminaries and evaluation and experiments for DPDP is available in [57].

This method is appropriate when a cloud server cannot be trusted to have reliably kept the data and may try to manipulate the data or deliberately or accidentally delete the data and use the storage space for other purposes and a paying owner wants to randomly verify that the data exists. It also encourages the storage service to care for customer data. The owner must be able to perform key establishment and complicated cryptographic operations. Experiments show that this method can verify data in 40 ms, the communication cost is 412.5 KB (the size of the proof), the required storage in the client side is 18.13 KB and the storage overhead cost in the server is 84 KB for storing a 1G file including 2^{16} blocks [57, 56, 59].

4.3.5 The fourth layer:Proof of Data Retrievability

For Proof Of Retrievability (POR) [82] we choose HAIL [23], which protects data integrity against an active dynamic, byzantine adversary that tries to corrupt the data, in up to b servers out of n servers, in each time epoch. As shown in Fig. 29, in HAIL, the data file is divided into l segments (3 in this example) and parity, called server code, is added to each segment, which adds 9% redundancy. Then a dispersal code (tag) which is a combination of a universal hash function (as parity) and a pseudorandom function is added to the data; it is calculated as $F_{ij}^{d} \leftarrow RS - UHF_{k_j}(F_{i1}....F_{il}) + g_{k'_j}(\tau_{ij}) for\ i \in [1.....m], j = [l+1, n]$ in which RS-UHF is defined as $(m_1 k_1^{l-1} + ... + m_1 + ... + m_1 k_n^{l-1} + ... + m_1)$ and τ_{ij} the input to

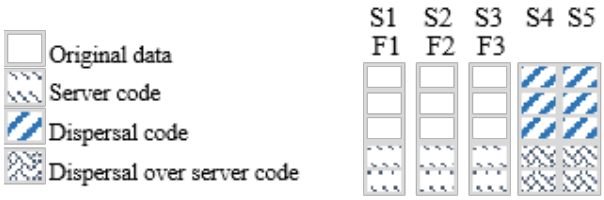

Figure 29: Layer 4. POR verification using HAIL[23]

the random number generator, it is the file name and off set of the file in the position (i, j). The original data along with the corresponding server codes are stored in l primary servers ($S1, S2$ and $S3$) and the dispersal codes are stored on the secondary servers ($S4$ and $S5$). HAIL verifies the data and if verification fails it uses the error correction codes in dispersal and server code to recreate the data [23]. However, the storage overhead is much larger here. Formal Preliminaries and evaluation and experiments for HAIL is available in [23].

This method is useful when the data are critically important, an active dynamic adversary is trying hard to corrupt the data, the server price and high redundancy do not matter and additional servers are available. The owner's gateway must be able to perform key establishment and complicated cryptographic operations. The latency is high (the initialization phase, for storing 1G file on b primary servers and $n - b$ secondary servers, is likely to be several orders of magnitude larger than simply communicating BF arrays). The owner's gateway must be able to challenge all servers and if required, gather parts of the corrupted data from servers, retrieve the intact data and redistribute it all in a one-time epoch. This may also work only with static files [23]. Experiments show that this method has a 9% server storage overhead cost which will be 92 MB for storing a 1G file [23]. HAIL generate n sets of keys, each key is 64 byte and assuming that only keys are stored on client side, the required storage on client would be 4096 KB.

4.3.6 Comparison

We compare Nested Bloom Filter, Hash Tree, DPDP and HAIL in this section in the manner of storage, time and communication complexity and efficiency. For comparison we are storing a 1 GB file which is divided to 2^{16} blocks of data. As shown in Figure 30, HAIL stores parity and tag which adds 9% redundancy which would be 1.090 GB. DPDP stores its Skiplist in the cloud with the complexity of $O(n^{\epsilon}\log(n))$ (ϵ is expected amortized and is between 0 and 1 [56, 58].) while Hash Tree stores the tree with the complexity of $O(n)$. Storing 2^{16} blocks in DPDP consumes 1.0084 GB while in Hash Tree it takes at most 1.0035 GB. Bloom filter stores a fixed size CBF (10700 bit=1.3 KB) either in cloud server or client side. Storage in the client (OGW) in both Tree and DPDP methods has the complexity of $O(1)$. DPDP stores the root node and a key and Tree stores the key. Although they both have the same complexity of $O(1)$, experiments show that for 2^{16} blocks DPDP needs 18.13 KB, while, Tree requires 4.13 KB (as discussed in Section 4.3.2.3, the required size of the CBF is 10700 *bits*), to make sure that the integrity of one whole year worth of data is verifiable with a false positive probability of less than 0.01, without requiring to use the Tree or other methods for verification. Tree requires less storage than DPDP in client storage as depicted in Figure 30. HAIL stores n (n is number of data blocks) sets of keys in the client, which has the complexity of $O(n)$, and for storing our file, the client need 4 MB space for storing keys.

Considering that DPDP and HAIL are implemented in C++ and Nested Bloom Filter and Hash Tree are implemented in Java and they used different platforms, comparing the experimental execution time may not be valid; however, the analytical comparison can give us a good estimation. The integrity verification time in BF is $O(k)$ where k is the number of hash functions and is a fixed number; therefore, it has the complexity of $O(1)$, both Hash Tree and DPDP and Hash Tree has the complexity of $O(log(n))$; Of course, we cannot compare the security implications completely in the case of using Bloom Filters. Tree and DPDP both have the time complexity of $O(log(n))$ for updating the data. The design of an efficient update for HAIL is left for future work and there is no complexity analysis provided for HAIL in [23]; however, authors showed that HAIL is able to encode 2 to 4 MB per

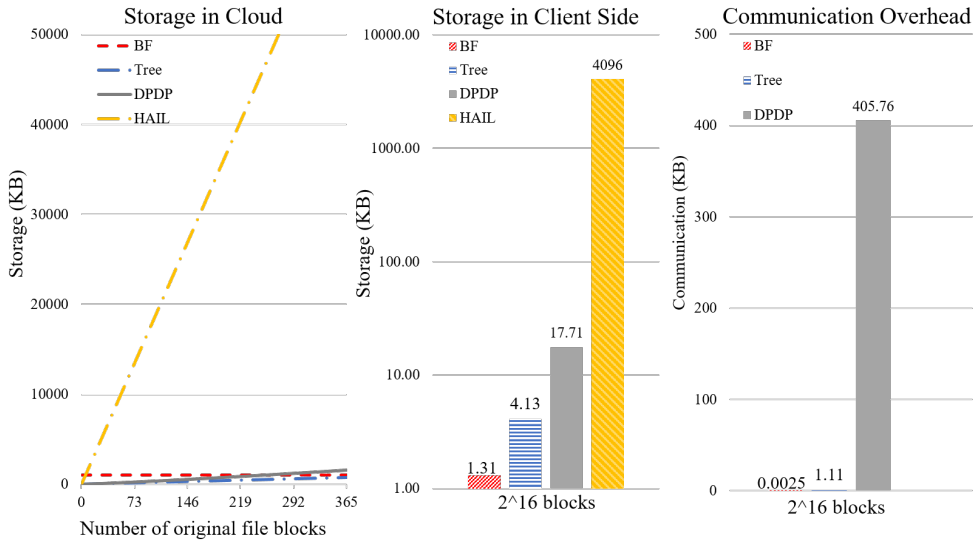

Figure 30: Cloud Storage(left), Client Storage(middle), and Communication(right) overhead

second [23].

We compared communication overhead as well. Communication complexity in DPDP includes sending the "proof" (the path from the root to leaf) and the key which has the complexity of $O(\log(n))$ which would be 415.5 KB overhead in experiments [56]. In our experiments for Hash Tree, the user requests data blocks from a random device and random time period. In assessing the communication complexity between the cloud and UGW, based on such experiments, the query includes all leaf-level children of one parent (recall that leaves are SHA values that are omitted and only pointers remain to determine which data blocks belong to this lowest-level parent.); therefore, the communication overhead between the cloud and UGW includes only the subtree (with the size complexity of $O(\log n)$ which was 480 bytes on average in the experiments– n is the number of nodes in the tree and each node size is 32 bytes); however, imagine the case where the user queried data that does not include all children of the lowest parent. In this case, the cloud server should also send the requested sibling blocks to the UGW. In the worst-case scenario, the number of sibling data blocks is $(d-1)$ and each block is 16 KB. In this case, considering the communication between cloud and UGW, with $d \leq 28$, Tree is still better than DPDP. Consider also that DPDP

is assumed to be working with a single source of data stored in a single cloud; hence the assumption is to have one skip list for all data; however, if we apply DPDP to the multiple clouds scenario we may need multiple skip lists which may also increase the communication overhead by a factor of c which is the number of cloud servers.

The communication complexity between UGW and OGW, for Nested Bloom Filters includes temporary BFs ($O(n(1))$, for Hash Tree includes sub-tree ($O(\log(n))$) and SHA-3 values ($O(1)$) which is $O(\log(n))$ in overall and for DPDP is the proof which is $O(\log(n))$; however, as shown in Figure 30, experiments showed that for 2^{16} blocks, DPDP has 415.5 KB overhead while Tree has only 1.11 KB, overhead. The number of siblings does not change this communication overhead considerably, since each extra sibling will add only 32 bytes (size of SHA-3 output value) to the communication overhead.

Analytical and experimental results showed that Hash Tree performs as well or somewhat better than DPDP in time, storage, and communication overhead, Nested Bloom Filter is more efficient than both of them and HAIL is the most expansive one. In addition BF and Tree do not need to share the key any key with any party; however, HAIL and PDP requires to share the public key with the cloud while provide repeatability.

4.4 ADI and Contextual Integrity

In this section, we introduce parameters inspired by Nissenbaum's theory [117] that can be used to define the ambit and context to assess which ADI layer is suitable for a certain situation. As mentioned in [117, 13] the determinant parameters (there for privacy) are (i) subject, (ii) sender, (iii) recipient, (iv) information type, and (v) transmission principle. These are relevant to the *integrity of the context* to ensure that privacy is maintained. In our work, we use comparable factors to define similar parameters, but for data integrity. These factors will likely have to be refined further based on specific applications. But we emphasize that it is important to have clear context to specify the right ADI.

Imagine a scenario in which data are generated by things that belong to the owner, a diabetic patient, and are delivered to multiple cloud servers. These could include an auto-

matic insulin pump, a smartwatch to monitor heart rate and walking steps, a smart weighing machine, a sleep cycle monitoring device, and her cell phone to capture her symptoms and test results. In the integrity verification phase, which is of interest here, the cloud server is the sender and the doctor (trusted party) is the initial recipient. The trusted party may send (see Section 4.3) a verification tag to the owner of the data, the patient. Alternatively, the patient may herself be the recipient of data.

(1) **Subject:** The person or the thing that this information is about. The subject can be specified by merit, start time (birthday or production date), and functionality (career, relation, or function). When the subject is a diabetic patient, information like glucose level is critical but the number of walking steps may not be important.

(2) **Sender:** The sender can be either the data owner or the device that generated the data. The device can be an insulin pump, smartwatch, weighting machine, sleep cycle monitoring, etc.

(3) **Receiver:** The receiver is the cloud server. The security and trustworthiness of the server and the storage price, are important factors.

(4) **Information Type:** The information type depends on the device: from the insulin pump, the information type is the glucose level, from the smartwatch it is heart rate, blood pressure, number of steps, etc. The information type is defined by merit (see below) of the information (e.g., blood glucose for a diabetic person has a high ADI while the number of walked steps do not), a timestamp, useful lifetime/history, and update rate.

(5) **Transmission Principle:** The characteristics of the channel that is carrying the information to the cloud include availability, confidentiality, rate of data loss, etc.

Each of the above may have specific attributes:

(i) **Awareness and consent of owner (c):** The data should not be stored if the owner of the data does not provide consent. Therefore (as we explain in Section 4.5), the first time that a new information type (or from a new device, entity, etc.) is sent, the gateway needs user permission and information about the factors that would be used for selecting a suitable ADI layer. This is similar to a situation when a new flow comes to an SDN switch and the controller is polled.

(ii) **Merit (m):** Let D be the data, whose ADI is under consideration. Associated with

these data may be their merit m, which suggests how important the data are for the scenario. The data may be used for quick decisions on a variety of mundane or specialized problems with low merit, or be used for auditing and forensics that may have lead to a spectacular event in which case the merit is high. The former may have a low ADI and the latter a very high ADI to make sure of the retrievability/integrity/correctness of the data. We anticipate a continuum or at least a quantized set of m values in between a minimum and maximum - for instance, there may be critical healthcare data that has an ADI somewhere in between, where more recent data is important compared to older data. Merit defines how important is the integrity of data and how much damage in the terms of life and price it has if the information is eliminated or manipulated. The higher the cost of damage, the higher the merit.

(iii) **Time (t):** Time plays several roles in ADI; the rate at which data are updated (t_u), the tolerable latency of data verification (t_l), and the remaining useful lifetime of data (t_h).

(iv) **Storage price and affordability (p):** This factor considers the price for the storage space considering the data owners budget, and if the storage is free

(v) **Trust (i):** Trust is calculated based on the integrity of the cloud server and integrity of the transmission channel.

(vi) **Redundancy (r):** Redundancy in data may be from aggregated values that still have important information (e.g., avg., min, max, etc.) even if the fine-grained data is corrupted.

In Fig. 31, we built a decision tree using these factors. For a diabetic patient, information about glucose and insulin dosage is critical. A health care provider who needs to attend to an emergency may need these data with the highest ADI (layer-4), considering the server price is affordable and there is no redundancy in the data. The owner should verify this information regularly. Heart rate and blood pressure measurements between doctor visits may be useful for diagnosis but are not necessarily the only sources of data if the doctor can talk with the patient. The integrity of such data should be checked when retrieved by the doctor, but corrupted values may waste time. Hence using Hash Trees (layer 2) is a good option (low merit, low latency). Historical values of the number of steps in a day and sleep cycles may not be required and are not critical if they are corrupted. If a sleep monitor

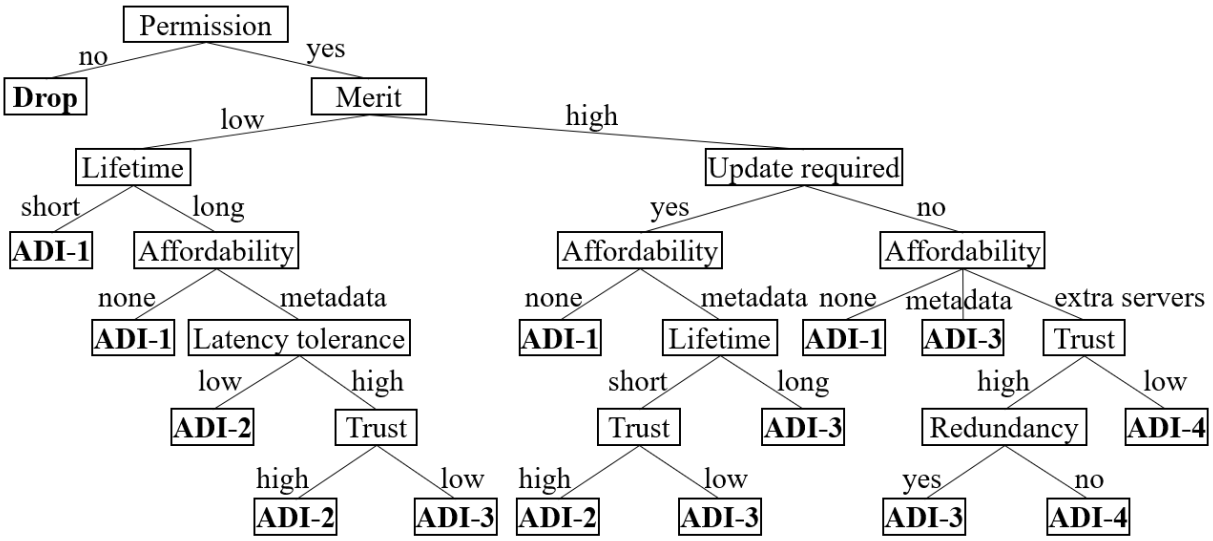

Figure 31: The decision tree selects required ADI layer based on context.

server does not allow any extra storage for tags, ADI layer 1 is a sufficient option where the CBF stays in a patient's gateway.

4.5 SDN-SDP Architecture for ADI

We now discuss a framework to gather context variables, select the required ADI layer and store data and metadata in the cloud server(s) securely. We integrate SDN and SDP [135, 45, 54, 151] to meet the framework requirements. We chose Software Defined Networks (SDN) to allow us to partition and gather information to select the ADI layer appropriate for incoming data from a device. An SDN decouples the control plane from the data plane, pushes network management, policy application, decision making, security, routing, and other services to the programmable controller software. Switches and routers are now simple forwarding devices that receive instructions from the controller, which has the global view of the network status. When a switch receives a flow, it compares the flow header against the flow table inside the

switch – if there is a match, it forwards the flow based on the matched rule; otherwise, it forwards the flow to the controller. The controller with a global view routes the flow and installs the proper rule in the corresponding switches. This advantage of SDN, which we require in the integrity verification process, is the flexibility of the rules in the flow table. The rules include two parts: match and action. The match part is compared with the flow header. For storing/retrieving data we need the ADI layer information in the header. The action part in the typical SDN network includes forward, drop, quarantine, etc. Here, to store/retrieve data we have to specify the correct verification process.

Devices, data owners, third-party entities need authentication for access to the cloud servers; therefore, we chose the SDP [114, 19] as the architecture to authenticate users, devices, and services. SDP creates an exclusive secure channel (e.g., Virtual Private Network (VPN)) between the communicating parties. In SDP, there are 3 main modules: SDP Initiating Host (IH), SDP Accepting Host (AH), and SDP controller (CTRL). The IH is typically installed on a client. The AH is installed on an SDP gateway and manages the communication between AH and CTRL, while CTRL is authenticating the user and creating a secure channel between the user and the service. The combination of SDN and SDP in IoT [135] can address the security requirements for implementing ADI.

In our architecture, we integrate both SDN and SDP controllers and implement all of their required functionality in a single SDP-SDN controller. The *SDP-SDN controller* has global view and comprehensive information about network. It decides –and therefore is aware of– many management and security specifications of the network. It communicates with switches, devices and users to gather required information for ADI application to choose the correct ADI. It authenticates the users, devices, services and creates a secure exclusive channel (e.g., VPN) between them and SDP-SDN switch (as in SDP controller) and also is responsible for partitioning the network, routing the packets, and providing required services to the network elements using SDN applications (as in SDN controller). A partition can include *IoT devices* (devices that belong to one user, devices in a residence, devices for monitoring an infrastructure). An *SDN application* selects the suitable integrity verification process based on the security policies and contextual information. The *SDP-SDN switch* forwards the packet based on its flow/policy table. If the host is not authenticated (matching

rule for a packet does not exist in the table) the switch drops the packet, unless it is a valid SPA (Single Packet Authentication)–SPA requests authentication and connection, contains information such as host ID, requested service ID, gateway IP, timestamp, randomized data.– the switch forwards the SPA to the controller to authenticate host, make a decision about the packet forwarding, install routing rule in the tables, share updated authorized host and services with switches and provide requested services such as deciding about the ADI layer. The *Owner Gateway (OG) and Server Gateway (SG)* are both SDP-SDN switches (Fig. 32). It can simply be a residential SDN switch or it can even be a mobile phone with OVS and AH installed on it [124]. An agent installed on the OG performs partially the functions of the controller. It gathers information and monitors the changes in the integrity ambit. The ADI layer (from Section 4.3) is defined by the cross-tabular set of parameters (from Section 4.4). When a user wants to store a new type of data in the *cloud server*, the OG uses this context in the packet header and sends it to the controller. The controller uses the flow header information and sends "user-defined policy" and "ADI layer information" to the integrity verification application to select the suitable integrity verification process. The controller installs a new rule in the user gateway that defines the routing path to the cloud and the ADI is recorded.

In Fig. 32, when the network and devices are initially setting up, the gateways and servers and services are sending SPA and network access request and are being authenticated and registered in an SDP-SDN controller (steps 0-1, through 0-8 shown by dashed arrows and gray text). Each device generating data first communicates with the owner's gateway (SDP-SDN switch called OG here). 1) SPA (single packet authorization – used for identifying clients in SDP), data, and metadata (update frequency, lifetime, tolerable latency and ID of the destination cloud server) reach the OG. 2) based on requested services in SPA, a questionnaire may be sent to the owner to gather information about the owner (in the cased of ADI: permissions, data merit, and storage affordability). 3) The owner responds. 4) OG Sends SPA, data, and metadata to the SDP-SDN controller. 5) The SDN controller authenticates the owner and forwards the information to an ADI application. The ADI application uses the decision tree (Fig. 31) to choose the suitable ADI layer. 6) The application responds with the required ADI level. 7) The controller installs the proper rules (includes the required

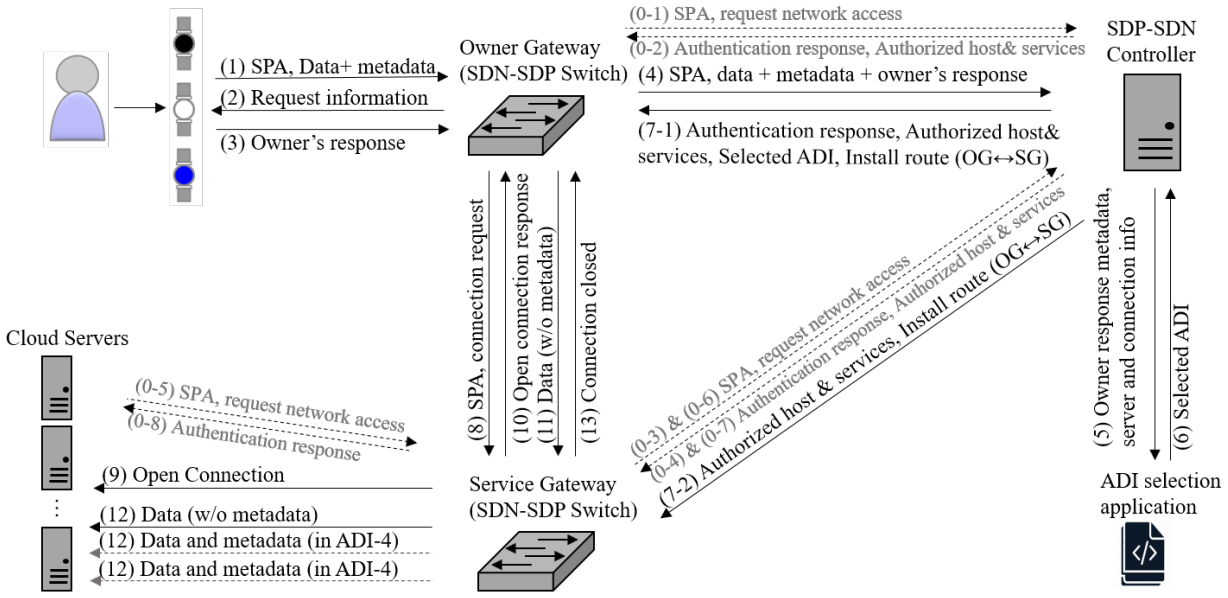

Figure 32: The designed IoT data integrity verification protocol using SDP-SDN

level of ADI and route to SG) in the corresponding SDP-SDN switches (e.g., OG, SG, etc.). The controller also provides information about authorized services and connections to the OG. the agent installed on OG uses this information to create metadata (one of these based on the verification method: CBF, HashTree, PDP tag, POR tag or parity) and adds it to the data. 8) OG sends SPA to the SG and requests connection to the cloud server(s). 9,10) SG opens the connection to the cloud server and responds to OG. 11,12) OG and cloud server(s) can now exchange information (data and meta data) for the session (13). The data retrieval process is pretty much the same; however, when a third party wants to retrieve the data (user gateway - UG instead of OG) the SDP-SDN controller should communicate with the OG (and owner) to request permission to share the information with the UG. Once the third party retrieves the data, the OG has to verify the integrity using the correct ADI.

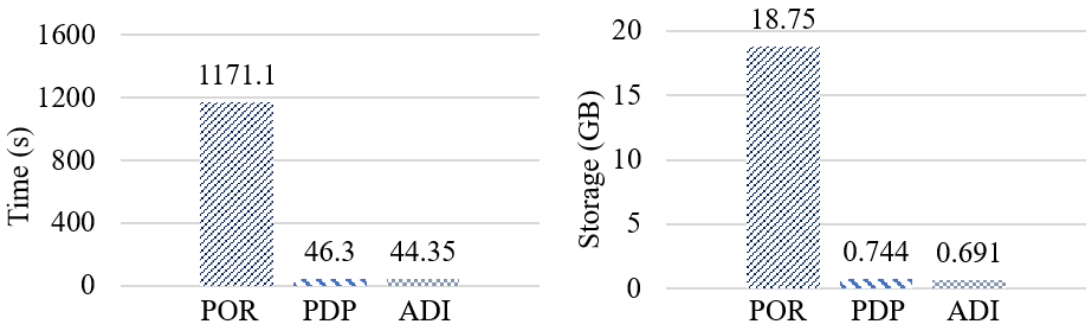

Figure 33: Time and storage using in ADI versus pure POR and PDP

4.6 Evaluation

In this section, we evaluate the feasibility and efficiency of ADI. We run the whole process explained in the above framework and the protocol in Section 4.5 to determine how much time this process requires and if this time is reasonable. In the efficiency study, we store a dataset using ADI integrity verification and we compare the storage overhead with the methods that purely use PDP or POR. We implement a simple simulation of this protocol on Mininet 2.2.0 [112], Ubuntu 14.04.6 operating system, and Intel® Core™ i5-560M Processor 2.66 GHz, using POX SDN controller [6]. We use SDN firewall [47] to simulate the SDP controller on the SDN POX controller. We install agents on SDN-SDP switches that use POX messenger to relay the extra required information from user and switches to the controller [86] and develop a python ADI selection application (python 3.4.3) to use the context information acquired from the user, SDN-SDP switch and SDN-SDP controller and select the matching verification level based on that context. The experiments show that the whole process of sending gathered information from OG to the controller and then to the ADI application to select the proper ADI layer, and sending back the chosen ADI to OG, takes 37 ms on average. The code and details about the implementation are available online [84].

To evaluate the efficiency, we implement all four verification methods in python. We

use the data set of "Hospitalized patients with heart failure" [169, 64] with the size of 1.38 MB, 2007 patients, and information such as admission ward, emergency status, occupation, whether the patient died, discharged or still in care, body temperature, blood pressure, pulse, respiration, history of diseases such as heart failure, diabetes, leukemia, tumor, liver disease, AIDS, and many other features (total of 164 features). We gather domain knowledge from different resources[16] to annotate the data and extract the required information to be used in the decision tree (Figure 31) and select the proper verification method for each record. For instance, the occupation feature and the average wage of that job are used to estimate the affordability of storage for that specific patient, and emergency admission is used to determine the low latency tolerant data. Data for the history of the disease are required for a long time and have high merit and should be stored with proof of retrievability if affordable, while the weight, height, blood pressure, and heart rate can be stored in a less expensive way depending on how normal or abnormal they were (blood pressure higher than 180/110 is critically dangerous and should be stored with more protection and be accessible in the case of need). Results show that 1.2% of data is selected for being evaluated with POR, 65% of data is selected to be evaluated by PDP, 4.6% of data is evaluated by the Hash Tree and 20% of the data is evaluated by BF (the remaining data was null values). More information about data processing along with the implementation is available on GitHub [84]. Figure 33 compares the ADI storage overhead with the methods that purely use PDP or POR. PDP uses 4.4% more time and 7.4% more storage and POR uses 26 times more time and storage than ADI. The results show that using ADI can reduce the required time and storage while still protecting the more critical data that need higher levels of secure integrity verification. This it does by keeping the ambit or scope of the integrity needs of data.

[16]Medical domain knowledge is achieved from the following websites:
https://www.hopkinsmedicine.org,
https://oxfordmedicine.com,
https://www.heart.org/en/health-topics/heart-failure,
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3891119

4.7 Conclusion

In this chapter, our purpose is to provide an architectural framework for IoT that will eventually automate the data integrity verification process using the right scope or ambit of data integrity – ADI. This framework allows us to choose the best integrity verification method based on contextual parameters of the data and its usage, as well as the characteristics of the intervening entities. We consider 4 layers of integrity verification in which higher ADIs cost more in terms of higher storage, time, and communication, but better protect the information from malicious or accidental corruption. To the best of our knowledge, this is the first time that the contextual integrity concept has been applied for determining the required layer of integrity verification. Also to the best of our knowledge, this is the first approach at all to determine the required layer of integrity verification. ADI efficiently reduces the required time and storage space while still storing the critical data using high integrity assurance methodologies.

To support this data integrity verification scheme we suggest a secure and flexible SDN-SDP framework that supports network partitioning, gathering context information from each partition, connect to the application to select the suitable verification process that selects the ADI layer, and stores data in the cloud supporting the corresponding integrity verification process. As part of the future work, we expect to evaluate the approach by implementing the framework and performing experiments with data owners being able to define parameters for ADI carefully and investigate a range of possible values.

5.0 Software Defined Ambit of Data Reliability for the Internet of Things

As the Internet of Things (IoT) grows, new challenges and opportunities arise. Various IoT devices may need to outsource sensed or generated data to multiple heterogeneous cloud servers and con-firm data availability at all times at varying granularities. Previously, reliability has been achieved by client-side processing, encryption, and storage, which is computationally infeasible for IoT devices. We propose the "ambit of data reliability (ADR)" which (i) summarizes the data (ii) adds low computation erasure code (iii) adapting the notion derived from contextual privacy to extract important parameters that determine the tradeoff between reliability and performance. The summary allows data at a coarser granularity when acceptable. The erasure codes protect against data damage. The context determines the "levels" of coarseness and reliability that correspond to the "ambit" of the data. A suitable architecture that can extract this context information and employ data-driven decisions for ADR, comprises of software defined networks (SDN) and software defined perimeter (SDP). With Bayesian Regression, we demonstrate how to determine the preferred error rate and availability probability for each context, and as a result how much storage, bandwidth, and computation are needed and how they can be reduced to maintain high performance based on application requirements. Integration of SDN and SDP can determine network parameters' values, perform decision making and enforce summarization, verification, and proof of retrieval. ADR can improve storage and bandwidth usage while sacrificing granularity in comparison with existing data reliability methods.

5.1 Introduction

In this chapter, we consider a final case study of approaches to ensure that data is available at a variable granularity at all times, trading off the reliability with performance. Previously we described HAIL[23] that added an erasure code to the data and provided Byzantine data availability. HAIL increases the storage (both on client-side and server-side)

and bandwidth usage exponentially and it also requires a lot of processing at the gateway in an IoT network. In this chapter, we want to assure data availability as well; however, at the same time we would like to be able to trade-off the storage, bandwidth, and time for update and data retrieval - which is possible using a global view of the generated data. Toward this, we summarize the data into a RACE-Sketch (described below) where possible, we use the hash tree integrity checks as before (see Chapter 4), and then we add an erasure code to data where required. We demonstrate the simple use of Bayesian Regression to determine the actions based on the preferred error rate and availability probability, how much resources (storage, bandwidth, computation) have to be used and how they can be reduced based on the application requirement (storage time, update time, retrieval time) while keeping the performance as high as possible. We show how the integration of SDN and SDP helps us to determine the network parameters value and how to perform decision making and enforce summarizing verification, and proof of retrieval.

Once again, we assume the use of an "owner's gateway" to manage data from "things" in the network and a "service gateway" that retrieves this data for the use by the owner or a third-party service. In IoT networks managing the limited resources is crucial and considering this limitation many applications may not be feasible with sensors or things. In this dissertation the idea is to enable performance tradeoffs using the integration of a framework combining SDN and SDP for IoT or wireless applications. As seen previously, SDN provides a global view of the network and SDP provides security information (and mechanisms for protection and authentication). With the architecture we suggest, using integration of SDN and SDP, we move the decision process to the controller and application and use historical data to make procedures more efficient and thus suitable for IoT networks.

In this section we use this framework to enhance data reliability. In other words, we want to assure the ability to retrieve data and keep it available. The basic idea comes from the purpose of Proof of Retrievability (POR) [82, 142, 53, 24] which is to ensure the integrity of data in a network with high failure probability storage nodes. Most of the previously proposed methods require huge storage and computation overhead. Furthermore, most POR mechanisms can only be applied to encrypted files and supports a limited set of challenges (challenges are data queries that look for data corruptions and if they fail and the failure

number exceeds a threshold, it triggers a redistribution of shares, which revokes all files from server, retrieves the file and redistributes). An example of such methods is the "high availability and integrity layer" (HAIL) [23] that demonstrated a proof of retrievability (POR) for a trusted verifier to check data integrity, correct errors when files are distributed across multiple servers with redundancy between servers and byzantine adversaries can corrupt up to b servers out of n servers, simultaneously. HAIL verifies the data and if verification fails it uses the error correction codes in dispersal and server code to recreate the data. Clients in HAIL only store the key. By using a challenge-response reactive cryptographic system, HAIL ensures the granularity of a full file and recovers corrupted files by using cross-server redundancy. Although the file is encrypted, it is publicly verifiable. With IP-ECC (Integrity Protected Error Correcting Code), multiple message authentication codes (MACs) are merged and aggregated responses are derived. HAIL verifies data only by the client, and public verification is not supported. Experiments show that this method has a 9% server storage overhead cost which will be 92 MB for storing a 1G file. HAIL generates n sets of keys, each key is 64 bytes, and assuming that only keys are stored on client side, the required storage on the client has the complexity of $O(n)$ and would be 4096 KB. The design of an efficient update for HAIL is left for future work and there is no complexity analysis or communication analysis in [23].

However previous methods such as HAIL are not suitable for IoT since they require key generation by multiple devices, key sharing, and client-side computation to store the data and a huge storage overhead. In this work we suggest to use data summarization where possible to reduce the cloud server storage, then apply hash tree integrity verification to detect data missing and then use network coding to provide an erasure code with a light computation in the client side gateway. In this process, both summarization and network coding require multiple parameters that specifies the tolerable error rate, available storage and frequency of updates for summarizing and tolerable unavailability, available storage and bandwidth for network coding. Providing this data -if not impossible- is a huge burden on a data owner and data owner's gateway (the architecture is similar to that in Chapter 4). We can use SDN-SDP framework to use decision making applications using trained models to provide these parameters to client gateway and facilitate data storage as well as make

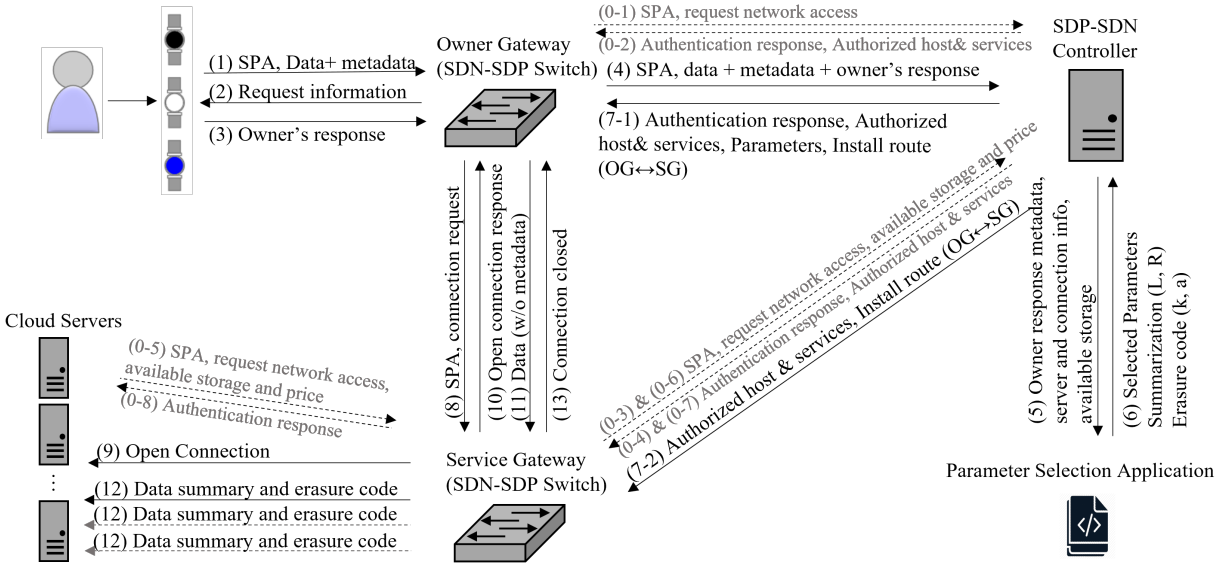

Figure 34: Ambit of Data Reliability Framework

sure that the data is available; while changing the granularity and increasing the availability probability for more sensitive and critical data and reducing storage, time and bandwidth requirements for less critical data. In this way, we are changing the reliability "level" to improve performance, based on the ambit or scope of the data, similar to Chapter 4. We call this the *Ambit of Data Reliability* or ADR.

The overall scenario is depicted in Figure 34. For both summarization and erasure code first data owner provide some information to the gateway which forwards them to the SDN-SDP integrated controller, SDN-SDP controller has global view of the network so it has access to the information about cloud servers, their available storage, their prices, available bandwidth, the security of the network devices, failure frequency and connections. The controller forwards its information along with the information received from the gateway to ADR application to estimate the tolerable summarization error rate and tolerable unavailability error rate and calculates the required parameters for them. The application sends this information to the controller which forwards them to the owner gateways along with

the connection and storage information. The gateway uses this information to summarize the data and apply the network coding erasure code and send them to the proper storage node for storage. Whenever a storage node fails it is the controller's responsibility to notify the service gateway and provide it with the required erasure parameters to rebuild a storage node.

5.2 Background

In this section, we provide some background on RACE-Sketch which we used for data summarising and network coding which we used for providing the ability to retrieve data in an IoT network.

5.2.1 RACE-Sketch

Repeated Array of Count Estimators known as RACE Sketch [39] is a recent approach used for providing differential privacy in previous works, can provide data summaries that are linear, merge-able and suited to IoT devices with small resources unlike other KDEs (kernel density estimates[1]). Considering a dataset $D = x_1, x_2, ...x_N \subset R^d$ and kernel k in which $k(x, y) \in [0, 1]$ is the collision probability of x and y; the KDE for query q is defined as $KDE(q) = \frac{1}{N}\sum_{x \in D} k(x, q)$ which requires $O(Nd)$ memory. However, RACE Sketch compresses N high dimensional vectors into small counter arrays indexed using locally sensitive hash functions. To store an element, this algorithm hashes the element with hash functions and for each hash value, the counter in the corresponding index will be incremented. When requesting the data, the average counter value is returned. Figure 35 shows this approach with an example of storing and querying the data. Imagine matrix A is our sketch – in order to store an element $x \in D$, we hash x with L hash functions $h_1(x), h_2(x), ...h_L(x)$ and increment the counter in $A(i, h_i(x))$. Here, $h_i(x)$ is a local hash function in which the similar points x and y would have the same hash with high probability of $k(x, y)$ and

[1]Methods for estimating the probability density function of a random variable.

would be in the range of $(1, R)$. If we concatenate the hash function for p times then the collision probability for this new hash function will be $k^p(x, y)$ and the hash outputs would be in the range of $(1, R^p)$. In this case the RACE estimator for query q would be $\mathbb{E}\{A[h(q)]\} = \sum_{x \in D} k^p(x, q)$ and the variance would be $var(A[h(q)]) \leq \left(\sum_{x \in D} k^{p/2}(x, q)\right)^2$. Two sketches A_1 and A_2 can be merged by simply adding the counters in two matrix.

Coleman et al. analysed RACE sketch for the locally sensitive hash (LSH) family. Such LSH functions are a family of hash functions that tends to have similar hash for similar data items and therefore similar items x and y closer than a distance d (can be defined as Euclidean or Manhattan or other distances) tends to collide with a probability of $k(x, y)$. In a simple version in angular Kernel, where the range of data is between $(-1, 1)$ the estimate for query q is calculated as $K(q) = \frac{1}{|D|} \sum_{x \in D} k^p(x, q)$ while the collision probability is $k(x, y) = 1 - \frac{1}{\pi}\theta(x, y)$ where $\theta(x, y)$ is the angle between x and y. The memory bound is $L = O\left(\left(\frac{\tilde{K}(q)}{K(q)}\right)^2 \frac{1}{\epsilon^2} log \frac{1}{\delta}\right)$ with probability of $(1 - \delta)$ where $\tilde{K}(q) = \frac{1}{|D|} \sum_{x \in D} k^{p/2}(x, q)$ [39].

In such cases we can limit the range of hash output by using rehashing. Then the estimator would be $\hat{K}(q) = \left(\frac{A[h(q)}{|D|} - \frac{1}{R}\right) \frac{R}{R-1}$ and $var(\hat{K}(q)) \leq \left(\frac{R}{R-1}\right)^2 \left(\sqrt{\frac{R-1}{R}}\tilde{K}(q) + \frac{1}{\sqrt{R}}\right)^2$. When we bound the memory to range $[0, R]$ then with the probability of $1 - \delta$ and error of $1 \pm \epsilon$, we have the equation 5.1 in which L is the number of LSH functions (and therefore number of rows) [39]. The implementation is available online at `https://github.com/brc7/RACEKernels` in c++.

$$L = O\left(\left(\frac{R}{R-1}\right)^2 \frac{1}{K(q)^2} \frac{1}{\epsilon^2} \log \frac{1}{\delta}\right) [39] \tag{5.1}$$

In [38], authors uses the RACE sketch to provide differential privacy to address the user's privacy challenge when building a model for a data set with many users' records. Differential Privacy (DP) provides privacy-utility trade off. The definition is that a function A can provide ϵ differential privacy if for data base D and D' which differs only in one element, it can be proven that $Pr[A(D) \in S] \leq exp(\epsilon) Pr[A(D') \in S]$ in which S is the sets of A's domain. A's sensitivity is defined as $\triangle = sup \parallel A(D) - A(D') \parallel$ over all neighboring D and D's. For instance, adding Laplace noise ($z \sim Lap(\triangle/\epsilon)$) to function $A(D)$ provides ϵ differential privacy; also Fourier, Bernstein, trigonometric polynomial and various other mechanisms have been used to provide differential privacy [38].

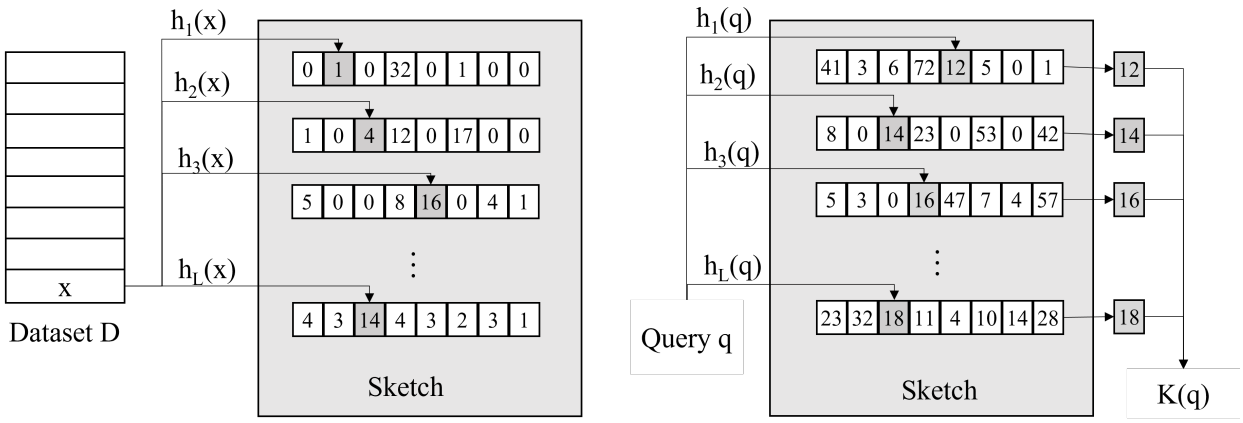

Figure 35: How RACE Sketch works

Authors in [38] provided an efficient one pass private sketch that works with most machine learning methods using RACE sketch (i.e., approximate pairwise sums) and Locally Sensitive Hash (LSH) kernel (i.e., hash functions in which collision probability $k(x, y)$ decreases as distance $d(x, y)$ increases). The process of Private RACE Sketch is as the following: It starts with $S_D \leftarrow 0^{R \times W}$, where R and W are the number of rows and columns correspondingly. R functions from LSH family $\{l_1(x), ...l_R(x)\}$ are defined and used to hash x (coming from the streaming data) to get R hash values for each row of S_D and it increments r in $S_D[r, l_r(x)]$. Finally $S_D = [S_D + Z]$ where $Z \sim LAP(R\epsilon^{-1})$. This process computes $O(NR)$ hash values thus had the run time of $O(dN)$, since R is assumed to be fixed. In RACE, the query q response is the process of getting the mean of $S_D[r, l_r(x)]$ over R rows to approximates the quantity $f_D(q)$ [38].

With a data set $D = \{z_1...z_N\}$ and loss function $L(\theta, z)$, RACE can approximate Empirical Risk (as in Empirical Risk Minimization (ERM) [2]). Authors proved that the output of $A(D)$ is ϵ-differential private. This differential privacy mechanism can be used in applications such as Kernel Density Estimation, Mode Finding, Naive Bayes Classification,

[2]Estimate statistical theoretical performance bound of learning algorithms using known training data sets.

Anomaly Detection and Sampling, and Linear Regression. The implementation is available online [38].

5.2.2 Data Retrievability

Many of the the proof of retrievability methodologies use Maximum Distance Separable (MDS) codes to generate erasure code to tradeoff the redundancy and reliability. One of the most well known branches of MDS codes is Reed-Solomon codes [51]. In order to understand Reed-Solomon codes let us have an overview of the algebraic structure called Fields. Finite Fields ($\mathbb{F}$) also know as Galois Fields are closed, Associative, commutative, there should exist an identity elements and each element should have an inverse in the set. The size of a Field is determined by p^m where p is a prime number, if $m = 1$ it is a prime field, otherwise if $m > 1$ it is an extension field $GF(p^m)$ that each element has the form of $a_{m-1}X^{m-1} + ... + a_1X^1 + a_0$ where $a_i \in (0, p-1)$. $\mathbb{F}_p[X]$ is a Finite Field when operations are defined modulo irreducible polynomial $g(x)$. The elements of F_{p^m} are the roots of $X^{p^m} - X \in \mathbb{F}_p[X]$ [20].

Now let us define Reed-Solomon codes. For integer k between 1 and n and Field $\mathbb{F}$ with the size bigger than n and the set $S = \{\alpha_1, ..., \alpha_n\} \subseteq \mathbb{F}$ Reed-Solomon is define as $RS(n, k) = \{p(\alpha_1), p(\alpha_2), ..., p(\alpha_n)\} \in \mathbb{F}^n$ in which the code for message $m = (m_0, m_1, ...m_{k-1})$ would be $p(X) = m_0 + m_1X + ... + m_{k-1}X^{k-1} \in \mathbb{F}[X]$ [21, 23].

In another definition, for integer k between 1 and n and Field $\mathbb{F}$ with the size $n + 1$ and the set $S = \{1, \alpha^1, ..., \alpha^{n-1}\} \subseteq \mathbb{F}$ Reed-Solomon is defined as $RS(n, k) = \{(c_0, c_1, ..., c_{n-1} \in \mathbb{F}^n \mid C(X) = c_0 + c_1X + ... + c_{n-1}X^{n-1}\ satisfies\ c(\alpha) = c(\alpha^2) = ... = c(\alpha^{n-k} = 0)\ \}$ This means that RS code with evaluation points $1, \alpha, ..., \alpha^{n-1}$ is equal to zero at the points $\alpha, \alpha^2, ..., \alpha^{n-k}$ [21, 23].

BCH codes and Reed-Muller codes have a similar definition as RS-codes; however, the coefficients in RS-codes are from $\mathbb{F}_{2^m}$, in BCH are from $\mathbb{F}_2$ and in Reed-Muller are from $\mathbb{F}_q$ [21].

Concatenated codes are defined by the outer code which converts the message to codewords and inner code that convert symbols to codewords($C = C_{out} \diamond C_{in}$). $C_{out}[n, k, n-k+1]$ can be implemented using Reed-Solomon codes (RS) with block length of $n = 2^m$ and

$C_{in}[m/r, m, d]_2$ can be a binary linear code [21].

The high availability and integrity layer (HAIL) [23] used Reed-Solomon coding to provide proof of retrievability. In order to do that, it divides the data file into l segments ($l = 3$ in Fig. 29) and parity, called server code, is added to each segment. Then a dispersal code (tag) which is a combination of a Reed-Solomon universal hash function (as parity) and a pseudo-random function is added to the data; it is calculated as $F_{ij}^{d} \leftarrow RS - UHF_{k_j}(F_{i1}....F_{il}) + g_{k'_j}(\tau_{ij})$ for $i \in [1.....m], j = [l+1, n]$ in which RS-UHF is defined as $(m_1 k_1^{l-1} + ... + m_1 + ... + m_1 k_n^{l-1} + ... + m_1)$ and τ_{ij} the input to the random number generator, it is the file name and off set of the file in the position (i, j). The original data along with the corresponding server codes are stored in l primary servers and the dispersal codes are stored on the secondary servers ($S4$ and $S5$) [23].

HAIL is useful when the data are critically important, the network is unstable and storage nodes have high probability of failure or an active dynamic adversary is trying hard to corrupt the data, the server price and high redundancy do not matter and additional servers are available. The owner's gateway must be able to perform key establishment and complicated cryptographic operations. In Hail, a client should generate $n-l$ pairs of dispersal keys (erasure), n server code keys (parity) and challenge keys. Considering each key is 64 bytes, overall, a client has to store 4096 KB of keys for a 1G file which is not suitable for IoT devices with limited storage. The process of adding server code and dispersal code using Reed Solomon Universal Hash function is way too complicated for most IoT devices. In addition, the latency is high. If the number of detected corruptions pass a threshold, the owner's gateway must be able to challenge all servers and if required, gather parts of the corrupted data from servers, retrieve the intact data and redistribute it all in a one-time epoch, which is a huge burden on bandwidth and processor and most IoT low rate protocols cant support such heavy communication. This may also work only with static files [23]. HAIL requires a lot of processing which makes it unsuitable for the IoT applications; therefore we suggest network coding to build a more suitable erasure code for IoT.

5.2.3 Network Coding

Network coding is originally used to maximize network traffic throughput. Considering the network(V, E, c) in which V is set of nodes, E is set of edges (directed links) and $c(e)$ is the capacity of the link $e \in E$. The question is having set of sessions $(s_1, T_1), ..., (s_N, T_N)$ in this network can we achieve the network rate $(r_1, ...r_N)$ where r_i is the achievable rate for session between source s_i and and destination T_i, where $s_i \in V$ and $T_i \in V$. In unicast session the maximum possible rate is $r(s,t) = MinCut(s,t)$. Min cut is the sum of weight of the edges that if removed the network will be disconnected. In 1972 Edmonds proved that $r(s,V) = min_{\nu \in V} MinCut(s,\nu)$ is achievable using maximum number of edges disjoint spanning trees rooted at s. In multicast this rate is upper bound and mostly achievable. Steiner tree or multicast tree, is the tree that starts from s and reaches every node in T. In traditional methods the number of edge disjoint Steiner tree determines the maximum rate, which is an NP-hard problem and even if can be calculated still cannot reach the upper bound. However using network coding at source (addition, subtraction and XOR over finite field) and decoding at destination the upper bound is achievable and the efficient coding can be computed in polynomial time. Assuming $\forall e \in E : c(e) = 1$, single senders, set of receiver $T \subset V$, and there are $h = Mincut(s,T)$ symbols $x_1, ...x_h$ that are going to travel from s to T. We have edges $e'_1, ..., e'_h$ that entering s and carry symbols $y(e'_1), ...y(e'_h)$, let's consider them as $x_1, ...x_h$. $y(e)$ denotes the symbol carried by e that is the combination of all $y(e')$ on e that entered node ν; therefore, $y(e) = \sum_{i=1}^{h} g_i(e)x_i$ where $g(e) = [g_1(e), ..., g_h(e)]$ is global encoding vector. Calculating $y(e_i)$ on all edges e_i can be performed using the following equation [36]:

$$\begin{bmatrix} y(e_1) \\ \vdots \\ y(e_h) \end{bmatrix} = \begin{bmatrix} g_1(e_1) & \dots & g_h(e_1) \\ \vdots & \ddots & \vdots \\ g_1(e_h) & \dots & g_h(e_h) \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix} = G_t \begin{bmatrix} x_1 \\ \vdots \\ x_h \end{bmatrix} \tag{5.2}$$

G_t is invertible and its coefficients are chosen randomly, independently and uniformly. The receiver should apply the inverse of G_t to decode the symbol[36]. For tagging the u_i form the global vector will be prefixed to the x_i. The tag $g(e)$ add h symbols as overhead to the information. If we consider that there are h packets in a block related to the k^{th} generation

and tagged with k, blocks inside a generation can be synchronized using buffering[36].

At each node after receiving h packets in a generation and extract the tags to form G_t and apply inverse G_t to the symbols, the message can be decoded[36]. Network coding can also minimize the amount of energy required for each packet as well as minimized the delay (number of hops to the destination). Network coding can be accomplished with 3 simple ideas: random coding (distributed randomly and uniformly), packet tagging to enable decoding, and buffering for dealing with asynchronous packet arrivals[36].

In erasure codes using Reed-Solomon coding, the file is divided into k parts of size M/k and these parts are encoded to n pieces and are distributed over n servers. In case of server failure, data from at least k servers should be decoded and re-encoded to replace the failed servers. However if network coding is being used, with a little more storage overhead by the factor of $\beta \leq 2$ the new node can be generated by getting randomly re-encoded data of size $\beta M/k^2$ from random k servers[50].

5.3 Summarizing Data with RACE Sketch

We chose RACE sketch for summarizing because it is efficient and also experiments in [38] shows that the summarized data can still be used for training machine learning models and decision making. More specifically RACE can create classifiers using both maximum likelihood and maximum posteriori rules.

In RACE sketch The choice of L and R make the difference in memory computation trade off. While minimizing R optimizes the memory ($R = 3$ is the optimum range), update and query has the time complexity of $O(L)$; therefore two out of three parameters of error, memory and update, cost should be chosen for optimization. For example, both ($R = 3$, $L = 10 \times 10^3$) and ($R = 4^3$, $L = 2 \times 10^3$) have an error rate of 0.5%; however the first one uses 260 times less memory and the second one has 5 times faster updates. In order to obtain the KDE with less than 1% error rate, for a 5GB dataset, RACE sketch only requires 4MB. Experiments show that RACE compression ratio is 10 times better than random sampling (RS), hash based sampling (HBS) and sparse kernel approximation (SKA) [39].

In this work, we used RACE Sketch to summarize the data. We have the dataset D that we summarize it in a matrix A, with L rows (number of LSH), each one is an array of size R in which the similar data would face collision with the probability of δ (The collision probability is calculated using $k(x, y)$). When we add a data element, where collision happens, we add one to the counter. When we query an element q, the element q is hashed using those L hash functions, as shown in Figure 35, and the median or mean of the counters will be returned as an estimation. The memory usage is calculated using equation 5.1. Equation 5.1, can be used to estimate L, R, ϵ and collision probability δ; however we have to calculate $K(q)$, therefore in order to optimize the value of L and R based on the required ϵ, we need to have an estimation of $K(q) = \frac{1}{|D|}\sum_{x\in D} k^p(x, q)$ (which eventually is mean of the counters). For simplicity we can imagine that the data has uniform distribution, in which case we would know that each counter in hash arrays can be estimated as $|D|/R$ and considering that the data is normalized it would be $1/R$, so we can use this value as mean of the counters. However, Authors in [150] suggest using $K(q) = \frac{0.1}{\sqrt{|D|}}$ for other distributions.

From equation 5.1 we can derive *absolute error* which is $|K(q) - \tilde{K}(q)| < 8 * (\frac{R}{R-1}) * \sqrt{\log \frac{1}{\delta}} * \frac{1}{\sqrt{L}}$. The equation 5.1 uses absolute error. To get relative error (ϵ) we replace $\epsilon = absolute\ error/kde(q)$ and as we discussed $K(q) = \frac{0.1}{\sqrt{|D|}}$ therefor the relative error is $\epsilon < 8 * (\frac{R}{R-1}) * \sqrt{\log \frac{1}{\delta}} * \frac{1}{\sqrt{L}} * \frac{\sqrt{|D|}}{0.1}$.

It is also possible to estimate typical values of $K(q)$ for an application by computing the ground-truth KDE for a few hundred queries and then taking the average [39]. Furthermore, if data has a timeline and maybe considered as a time series, in order to store and summarize the data using RACE sketch, we suggest to use an array of time instead of counters.

5.4 Data Reliability with Network coding

In [52] a decentralized erasure code using linear codes with probabilistic structure that leads to sparse matrix is introduced for reliable distribution storage where multiple nodes generate the data and store it in multiple storage nodes. There are k data generating nodes (each node generates 1 packet) storing data in $n > k$ storage nodes (each storage node has

the capacity of 1 packet data) with the condition that the data retriever can query any arbitrary k storage nodes and rebuild the original k data packets. For $d(k)$ times each node i randomly, uniformly and independently is assigned to an storage node, (some storage nodes may be chosen more than once for each node so $N(i)$ the number of storage nodes for node i may be less than $d(k)$) each storage node creates a random linear combination of data nodes connected to it and stores it along with the random coefficients from Field F_q used in that linear combination (see Figure 36) which leads to storage overhead of $N(j)(\log_2(q)+\log_2(k))$ bits for storage j. It can be formulated using $s = mG$, in which s is stored data, m is data vector and $G_{k\times n}$ is an sparse matrix representing bipartite graph of data nodes and storage nodes. A data retriever query random k storage nodes and use maximum likelihood (ML) decoding solved by inverting the $G'_{k\times k}$ sub-matrix of G. In order to optimize the storage overhead $d(k)$ should be minimized while ensure that the sub-matrix is full rank (rows are linearly independent and thus matrix is invertible). It is proven that $d(k) = c\ln k$ is optimal for any $c > 5\frac{n}{k}$ (each node sent data to $5\frac{n}{k}lnk$). The storage overhead complexity is $O(\log(k)$ and the overall overhead is $O(\log(k)(\log(q) + \log(k))$ which is negligible using large packets. Most common erasure codes are Reed-Solomon, the benefit of network coding erasure code is that it is decentralized and no matrix is explicitly stored [52].

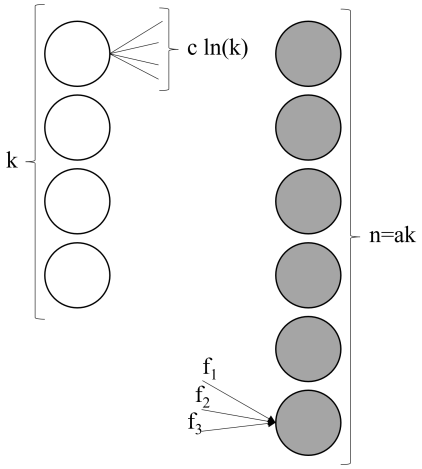

Figure 36: Network coding used to form an erasure code using randomly uniformly and independently storage selections and linear combinations with random coefficients [52]

Using network coding as an erasure/regeneration code to recover from node failure, au-

thors in [51] tried to find the optimum tradeoff between storage and repair bandwidth. The repair process is shown in Figure 37. There are two extremal points on the tradeoff curve which are minimum storage regenerating (MSR) also known as maximum distance separable (MDS) [3] and minimum bandwidth regenerating (MBR). Previous erasure codes required M bit bandwidth communication to repair a node; however, MBR reduces repair bandwidth significantly by storing slightly more than M/k bits per storage node. Analysis show that in a graph $G(n, k, d, \alpha, \gamma)$ where M is the whole data size, n is the number of storage nodes, each node store α bits, k is the number of nodes require to recover data, $d \leq n-1$ is the number of survived nodes available and can be used to repair and build a new node β bits, which requires $\gamma = d\beta$ bandwidth, $G(n, k, d, \alpha, \gamma)$ is feasible for any $\alpha \geq \alpha^*(n, k, d, \gamma)$.

$$\alpha^*(n,k,d,\gamma) = \begin{cases} \frac{M}{k} & \gamma \in [f(0), +\infty) \\ \frac{M-g(i)\gamma}{k-i} & \gamma \in [f(0), f(i-1)) \end{cases} \quad [51] \tag{5.3}$$

where $f(i) = \frac{2Md}{(2k-i-1)i+2k(d-k+1)}$ and $g(i) = \frac{(2d-2k+i+1)i}{2d}$ and $d \leq n-1$; as a result the minimum bandwidth for repair is $\gamma min = f(k-1) = \frac{2Md}{2kd-k^2+k}$.

A couple of methods are compared using this analysis, including:

- Hybrid: Use one full replica for rebuilding failed nodes and multiple erasure code fragments. This method reduce the repair bandwidth process to M/k however if the replica is lost, new fragments cannot be created until it restored.
- MSR: For minimum storage $\alpha = \frac{M}{k}$ and $\gamma = \frac{Md}{k(d-k+1)}$ where communication for repair would be M for $d = k$ and it would be $\frac{M}{k}.\frac{n-1}{n-k}$ for $d = n-1$.
- MBR: For minimum repair bandwidth $\alpha = \frac{2Md}{2kd-k^2+k}$ and $\gamma = \frac{2Md}{2kd-k^2+k}$ so $\alpha = \gamma$ and for $d = n-1$, $\alpha = \gamma = \frac{M}{k}.\frac{2n-2}{2n-k-1}$; therefore communication is exactly the same as amount of stored data.

Evaluation analysis assumed that a fraction of f nodes fails in each time period and the node is available with a probability of a and assume that we have R replicas.

- Simple replication: the amount of data has to be stored is $R.M$ and $f.R.M$ bits has to be replaced in each time unit the unavailability probability is $(1-a)^R$.

[3] Reed-Solomon codes are also MDS

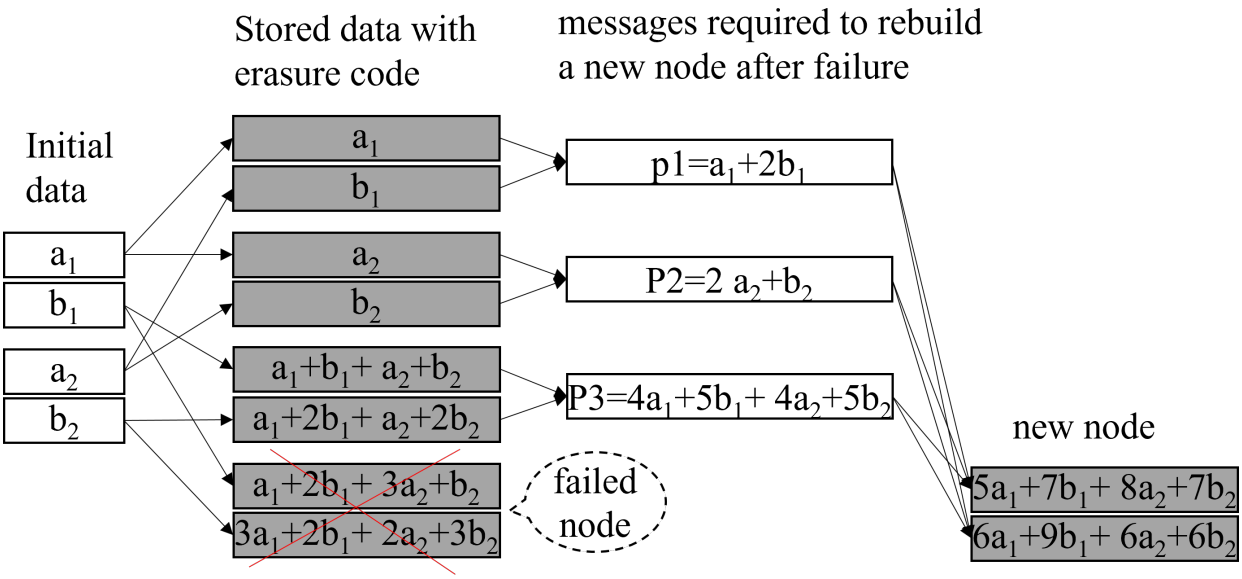

Figure 37: Network coding repair process

- Ideal erasure code: $n = k.R$ and $f.R.M$ bits has to be replaced, which gives the unavailability of $U_{ideal}(n,k) = \sum_{i=0}^{k-1} \binom{n}{i} a^i (1-a)^{n-i}$.
- Hybrid: store $R.M$ bits and replace $f.R.M$ bits and has the unavailability probability of $(1-a).U_{ideal}(n,k)$. It has an asymmetric design and can cause bottleneck in disk I/O. They make the system over complicated, for a negligible bandwidth efficiency that is minimal in a more stable environment.
- MSR: $R = n/k$ and store $R.M$ (same is ideal erasure code) and replace $f.R.M$, therefore extra storage would be $\frac{(n-1)\beta}{M/k} = \frac{n-1}{n-k}$ and unavailability of $U_{ideal}(n,k)$.
- MBR: ideal fragment size is $b = \frac{(n-1)\beta}{M/k} = \frac{2(n-1)}{2n-k-1}$ which stores $M.n.b$ bytes and replaces $f.M.n.b$ bytes to provide the unavailability of $U_{ideal}(n,k)$.

Experiments show that the MSR storage is the same as ideal erasure code storage. MSR and MBR have the same unavailability probability and both provides better availability that Hybrid (0.000059 vs 0.00018).

5.5 Bayesian Regression

We use Bayesian Regression [94, 95] with specific data sets to improve the performance of decision-making and automate the decision process at the SDN controller for reliability. We use Bayesian Regression in two different steps; first in data summarizing to determine the proper values to limit the error rate while minimizing the resources based on priorities (storage/ update time) and second in providing erasure code with network coding to maximize data availability probability while minimizing the used resources based on the priorities (storage/bandwidth). Bayesian inference describes the data and extracts the relative credibility of features (effect of one feature on another) considering their prior probabilities. Bayesian inference can help in assessing how critical each context variable is and how it would affect the probability of choosing an action to be enforced.

We use a data set of "Hospitalized patients with heart failure" [169, 64] with the size of 1.38 MB, 2007 patients, and information such as admission ward, emergency status, occupation, whether the patient died, discharged or still in care, body temperature, blood pressure, pulse, respiration, history of diseases such as heart failure, diabetes, leukemia, tumor, liver disease, AIDS, and many other features (total of 164 features).

Please note that the use of Bayesian Regression will be just one example of using ML in the SDN/SDP framework for performance improvements and tradeoffs. The objective here is to show a proof of concept. Other techniques and their systematic evaluation are beyond the scope of this dissertation. We use Bayesian Regression to decide what values of parameters will be effective and give us the sufficient and required summarizing and erasure codes for reliability under failure.

As we previously discussed, in order to provide efficient reliability, we first want to summarize the data, then we hash the data using hash tree for data verification and finally we add erasure code using network coding. We are using Bayesian Regression in two steps: (i) in summarizing to figure out the tolerable error rates for data and based on that we can choose the proper storage variables (L and R), and (ii) in erasure codes to figure out the tolerable unavailability probability and based on that to determine how much storage and bandwidth are required. We used the data set of "Hospitalized patients with heart failure" [169, 64]

with the size of 1.38 MB, 2007 patients, and information such as admission ward, emergency status, occupation, whether the patient died, discharged or still in care, body temperature, blood pressure, pulse, respiration, history of diseases such as heart failure, diabetes, leukemia, tumor, liver disease, AIDS, and many other features (total of 164 features)

In this Bayesian task we want to find the tolerable summarization error rate and tolerable unavailability rate. For this task we change the dataset format and build a synthetic dataset based on the features and feature values of "Hospitalized patients with heart failure" dataset. For each Feature belong to each person (for every cell of the data) we considered describers such as "permission", "merit", "afford", "lifetime", "update", "latency", "trust", "redundancy", "tolerable error rate" to help us decide how to handle that cell of data. I (the human), manually annotated the data values considering the range of the feature value in combination with other feature values. For example, for a patient heart rate value, based on the fact that the value is bound in a normal range (low merit) or it is in danger zone (high merit), the occupation of the patient is used for estimating affordability, the the admission ward is used for estimating trust, the emergency is used for latency, and so on. We ended up with having a dataset with 325734 records and a few records are shown in Table 9.

5.5.1 Bayesian Regression for Data Summarizing

For data summarizing our objective is to estimate the tolerable error rate and then use that to determine L (number of hash functions) and R (length of each array) based on the requirement of whether we need to optimize storage (minimizing R minimize the storage) or we need to optimize retrieval and update time (reducing the number of hash functions reduce the update time). We first find the distribution of the "tolerable error rate" which was logarithmic Laplace (a.k.a logarithmic double exponential). We declared parameters and descriptive mathematical model. We modeled the data with Bayesian linear regression and used $linear_{model}.BayesianRidge()$ from Sklearn package in Python to to estimate regression parameters using a linear model in Bayesian Regression. We used 70% of the data for train and 30% for test The Coefficients and the intercept are as follows:

$coef\ =\ [-1.21536796e-01, -7.06715953e-02, -1.01147732e+00, -7.75073535e-$

Feature	Value	permission	merit	afford	life time	update	latency	trust	redundancy	error rate
Killip grade	I	yes	high	some	long	no	low	yes	no	0.12
NYHA cardiac function class	IV	yes	high	some	long	no	high	yes	no	0.1
gender	Male	yes	low	some	short	no	low	yes	no	0.94
visit.times	1	yes	low	some	short	no	low	yes	no	0.95
respiration	21	yes	high	some	short	no	low	yes	no	0.16
occupation	Urban Resident	yes	low	some	short	yes	high	yes	no	0.80
admission way	Non Emergency	yes	low	some	short	yes	high	yes	no	0.97

Table 9: Error rate estimation data for Bayesian Regression model

$01, 6.41217771e-02, -2.10275960e-02, 5.97534438e-12, 3.87825318e-03]$, and

$intercept = 1.5085833599387743$ and finally the mean square error (MSE) is 0.0256. After creating the model we can feed the new upcoming data to the model to get an estimation of error rate before summarizing and storing it.

The estimated error rate determines how fine grained the data should be (ϵ the error rate); therefore, if it is zero we wont summarize at all. The next task is to find L and R, and based on the overall available storage ($L \times R$ maximum possibility) and considering data requirements -whether it require minimum storage or minimum computation (when the data needs a lot of query or update)-we optimize the values of L and R. We use the formula 5.1 where $k(q)$ can be estimated as $\frac{0.1}{\sqrt{|D|}}[150]$ and $|D|$ is the size of the data. So we have $\epsilon < 8 * (\frac{R}{R-1}) * \sqrt{\log(\frac{1}{\delta})} * \frac{1}{\sqrt{L}} * \frac{\sqrt{|D|}}{0.1}$. In this equation ϵ, is determined with trained Bayesian regression model, $|D|$ is defined by the size of dataset and δ is assumed to be 0.05 (when availability probability is 95% the $\log(\frac{1}{\delta})$ is equal to 1.3)[38].

$$L < \left(\frac{8 * log1/\delta * |D|}{\epsilon^2 * 10^2}\right)\left(\frac{R}{R-1}\right)^2 \text{ [39]} \tag{5.4}$$

Considering equation 5.4 Let's assume constant $c = \left(\frac{8*log1/\delta*|D|}{\epsilon^2*10^2}\right)$. Therefore to optimize storage (S) we should minimize $S = L \times R * 4 + (R * 4)$ (we need 4R to store hash seeds).

$$S(R) = \left(c * \left(\frac{R}{R-1}\right)^2 + 1\right) * 4R \tag{5.5}$$

I replaced $L = c * \left(\frac{R}{R-1}\right)^2$ in storage equation 5.5 and solved $S'(R) = 0$. The roots were $R = \frac{\sqrt{13}+5}{2}$ and $R = \frac{-\sqrt{13}+5}{2}$. The trade-off between error rate and storage when we focus on optimizing storage is shown in Figure 38.

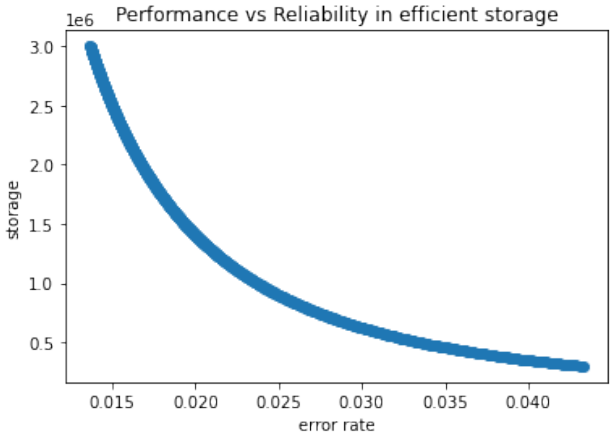

Figure 38: Performance vs error rate with efficient storage

To optimize update we should minimize L. Minimizing L increases the storage, so in this case we consider the maximum possible storage value that is available and user is willing to pay for, which is achieved from the SDN controller (one scenario can be that we provide the min required storage-as mentioned above- to SDN controller and the controller provides us with how much more capacity is available while ensuring that giving us at least the minimum requirement) and then calculate R based on that.The trade-off between error rate and storage when we focus on optimizing update is shown in Figure 39.

As an example, for storing 1GB File with 0.01 error rate the required storage would be $4 * 10^6$ bits.

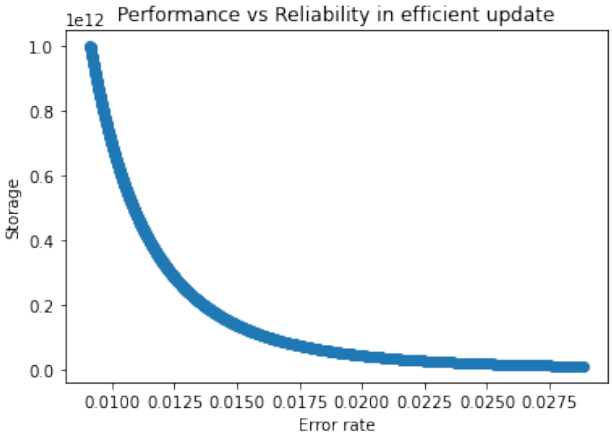

Figure 39: Performance vs error rate in efficient update computation

5.5.2 Bayesian Regression for for Network Coding Erasure code

For data erasure code, our objective is to estimate the tolerable unavailability and then use that to determine k (number of data blocks) and n(number of storage nodes) based on the requirement of whether we need to optimize storage (minimizing R minimizes the storage) or we need to optimize retrieval and update time (reducing the number of hash functions reduce the update time). The unavailability is calculated using Equation 5.6 where a is the availability probability of each node and should be achieved from SDP (or in our frame work integrated SDN-SDP) controller. Figure 40 is an example of different choice of n and k and their effects on Unavailability [4].

$$U_{ideal}(n,k) = \sum_{i=0}^{k-1} \binom{n}{i} a^i (1-a)^{n-i} \tag{5.6}$$

MSR: for minimum storage $\alpha = \frac{M}{k}$ and $\gamma = \frac{Md}{k(d-k+1)}$ where communication for repair would be M for $d = k$ and it would be $\frac{M}{k}.\frac{n-1}{n-1}$ for $d = n-1$. In MSR $R = n/k$ and store $R.M$ (same is ideal erasure code) and replace $f.R.M$, therefore extra storage would be

[4]To have a better sense of proper range of parameters, here are some pairs of (n,k) that gives us the unavailability of 0.01 (14, 10), (25, 19), (31, 24), (37, 29), (50, 40), (57, 46), (64, 52), (71, 58), (78, 64), (85, 70), (92, 76), (99, 82)

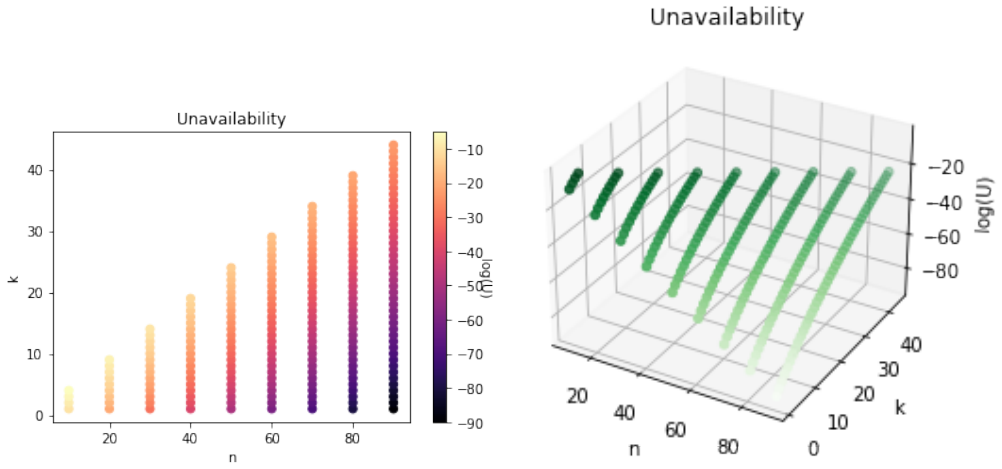

Figure 40: Different choices of n and k and their effects on U in 2D and 3D plots

$\frac{(n-1)\beta}{M/k} = \frac{n-1}{n-k}$ with unavailability of $U_{ideal}(n,k)$.

MBR: for minimum repair bandwidth $\alpha = \frac{2Md}{2kd-k^2+k}$ and $\gamma = \frac{2Md}{2kd-k^2+k}$ so $\alpha = \gamma$ and for $d = n-1$, $\alpha = \gamma = \frac{M}{k}.\frac{2n-2}{2n-k-1}$; therefore communication is exactly the same as amount of stored data. In MBR ideal fragment size is $b = \frac{(n-1)\beta}{M/k} = \frac{2(n-1)}{2n-k-1}$ which stores $M.n.b$ bytes and replaces $f.M.n.b$ bytes to provide the unavailability of $U_{ideal}(n,k)$.

For this case we took the same approach as data summarization and we ran Bayesian regression (Bayesian Ridge). We obtained the following model coefficients:

$coef = [-1.87567410e-02, -6.42720467e-01, -1.04731402e+00, -4.95920567e-01, 5.86055624e-02, -4.23610894e-01, 3.45508294e-12, 3.52801638e-03]$ and

$intercept = 1.645630640548738$ and Mean Square error is 0.053493169576964844. We use this model to estimate the tolerable unavailability and based on that we can estimate k and n and then based on the requirement (minimum storage or minimum bandwidth) choose either of MSR or MBR to calculate the other required storage parameters. Considering the provided formulas for α and γ and assuming that we get the tolerable unavailability of 0.01, with the choice of $n = 14$ and $k = 10$, Figure 41 depicts the effect of different choices of d and *mode* (either we require MBR or MSR) and the required storage in each node (α), in addition to the required bandwidth for each node transmission (γ).

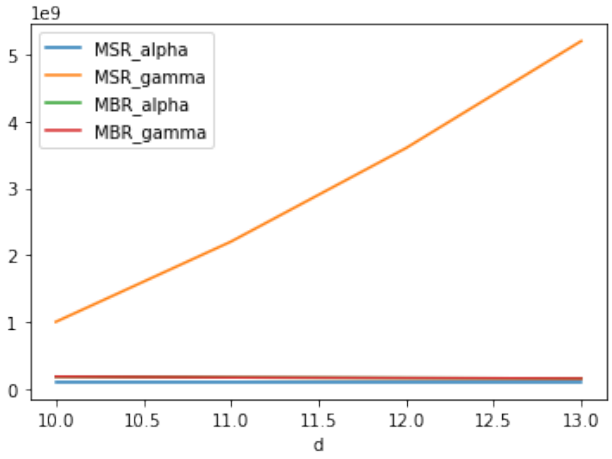

Figure 41: Different choices of d and its effects on *storageandbandwidth* in different modes

As an example, let us suppose we want to store a 1GB file: In the summarizing phase with an error rate of 0.01, the storage requirement would be 4×10^6 bytes of storage. Adding an erasure code with $n = 14$, $k = 10$ and unavailability probability of 0.01 in MSR mode, each storage node should store 4×10^5 bytes and the overall storage cost would be 56×10^6 bytes. Compared to HAIL, which was 1.09 GB 42, the cost with ADR is almost 20 times smaller. Moreover ADR does not have complicated client side computation which makes it suitable for IoT networks. This all is gained at the cost of losing data granularity.

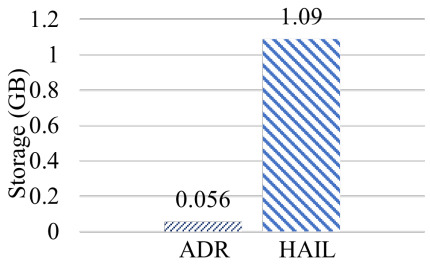

Figure 42: Comparison of HAIL and ADR storage

5.6 Discussion

We have not performed a detailed systematic analysis of the trade-offs between granularity of data and storage costs, but in the proof-of-concept analysis, we show that this approach outperforms HAIL, but at the cost of data granularity. If applications can tolerate this lack of granularity, which we believe is the case when data from "things" may not change significantly over time, this approach would yield efficiencies that are not possible with HAIL, while maintaining application needs. Moreover decisions about which reliability method is required are made in real-time for streaming data at the time that we are storing the data; however once the decision is made for stored data, the erasure code would be there and when retrieving the data the system does not make any decision, it just uses the erasure code to retrieve the data under failure. If the decision making model is updated then the update would be effective on new data to come and older data will not change. If for any reason there is a need to update the decision, it is possible to change the erasure code for the data by rebuilding the file and doing the entire process of creating the erasure code and distribution from the beginning. In the case of changing the decision for summarizing, once we summarize the data the only possibility is to summarize it further (increase the tolerable error rate and hence decrease storage and/or update time), and there is no mechanism to decrease the error rate.

6.0 Conclusion

In this dissertation, we consider the use of a secure and flexible architecture using the integration of SDN and SDP to improve decision making and action enforcing in WiFi and IoT networks considering the resource limitation of IoT. We apply this architecture in three different scenarios to show possible applications with proof of concept. In the first scenario we used historical data to predict the quality of service by building a decision tree and deciding what flows to throttle (e.g., reducing packet size) towards managing an SDN WiFi network where interference from competing transmissions may impact the quality observed by critical flows. We use the global view in the SDN controller and install agents on Access Points to gather required information for predicting maximum delay using the decision tree. In the second scenario we choose the best integrity verification method (among four provided methods including Nested Bloom Filter (new), Hash Tree (new), PDP (previous), POR (previous)) based on contextual parameters of the data and its usage, using an static decision tree. Finally in the third scenario we consider trade-offs between data reliability and performance using data summarizing and network coding and we used Bayesian Regression for bounding the error in data summarizing and deciding the probability of data availability.

6.1 Limitations

One significant issue in this study was the lack of a suitable public dataset to use for model training and a concrete method of extracting the required features from the dataset. While SDP and SDN networks have a range of applications such as managing Data Centers and IoT networks, as far as we know, there is not any large scale implementation of SDN that exploits machine learning for performance trade-offs. Also, SDP indeed reduces the burden on the controller and avoid unauthenticated packets to get to the controller. Yet it still has a single point of failure that if for any reason crashes, the whole network will go down. All data about the network will get lost as well which brought up the idea of

using distributed controllers or simply having a backup controller. We compared our work with existing approaches to demonstrate the trade-offs, but did not systematically try to determine the optimum parameters or algorithms. Finally, we do not consider a rapidly changing dynamic environment (especially in scenario 1).

6.2 Future Works

In this dissertation we provided a framework using the combination of SDN and SDP concepts, that secures the network, provide global view of the network and can manage nodes and makes decision for them. We used three different scenarios as examples and demonstrate proof of concept for each one of them. Performing a comprehensive human subject study to create a real data set and complete software implementation, is in the plan to be done in future. Also in this dissertation we annotated the data using human input in order to change the data format and extract the values of decision merit ("merit", "trust", "affordability",....) ; however, we plan to use Natural Language Processing (e.g., word2vec and cosine similarity and similar methodologies) to automate feature extraction.

We also did not consider in detail the relationship of this work to emerging paradigms of IoT like the Helium People-Powered Networks [79]. In Helium, a crypto-token is used to incentivize coverage toward creation of a decentralized wireless infrastructure for IoT. Tokens are used to verify coverage provided (the consensus mechanism is called "Proof-of-Coverage"). However the objective here is to provide Internet connectivity using low power. The LoRAWAN base stations simply relay the IoT data to the Internet. As far as we know, the network does not support additional services such as resource management or data integrity or data reliability, nor are they required to use SDN/SDP. One can imagine *services* built on top of Helium - perhaps a "Proof of Service" token with different valuations for different edge services. This is also a project for the future.

6.3 Publications

- Karimi, M., Krishnamurthy, P., Joshi, J., & Tipper, D. (2017, November). Mining historical data towards interference management in wireless sdns. In Proceedings of the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks (pp. 81-88).
- Karimi, M., & Krishnamurthy, P. Hierarchical Data Integrity for IoT Devices in Connected Health Applications. Open Journal of Internet of Things (OJIOT), Volume 7, Issue 1, 2021, ISSN 2364-7108
- Karimi, M., & Krishnamurthy, P. Software Defined Ambit of Data Integrity for the Internet of Things, 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), 2021, pp. 737-745, doi: 10.1109/CCGrid51090.2021.00089.

Bibliography

[1] Adnan Akhunzada, Ejaz Ahmed, Abdullah Gani, Muhammad Khurram Khan, Muhammad Imran, and Sghaier Guizani. Securing software defined networks: taxonomy, requirements, and open issues. *IEEE Communications Magazine*, 53(4):36–44, 2015.

[2] Nuray Baltaci Akhuseyinoglu, Maryam Karimi, Mai Abdelhakim, and Prashant Krishnamurthy. On automated trust computation in iot with multiple attributes and subjective logic. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 267–278. IEEE, 2020.

[3] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM Workshop on Assurable and Usable Security Configuration*, SafeConfig '10, page 37–44, New York, NY, USA, 2010. Association for Computing Machinery.

[4] Gaurav Ambekar, Tushar Chikane, Shiben Sheth, Abhilasha Sable, and Kranti Ghag. Anticipation of winning probability in poker using data mining. In *Computer, Communication and Control (IC4), 2015 International Conference on*, pages 1–6. IEEE, 2015.

[5] Javed Ashraf and Seemab Latif. Handling intrusion and ddos attacks in software defined networks using machine learning techniques. In *Software Engineering Conference (NSEC), 2014 National*, pages 55–60. IEEE, 2014.

[6] Murphy McCauley at UC Berkeley. The pox network software platform. `https://github.com/noxrepo/pox`. Accessed:11/21/2019.

[7] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 14(1):1–34, 2011.

[8] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 598–609, 2007.

[9] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–10, 2008.

[10] Michael Backes, Dario Fiore, and Raphael M Reischuk. Verifiable delegation of computation on outsourced data. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 863–874. ACM, 2013. `https://dl.acm.org/citation.cfm?id=2516681`.

[11] Jeffrey R Ballard, Ian Rae, and Aditya Akella. Extensible and scalable network monitoring using opensafe. In *INM/WREN*, 2010.

[12] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 109–114. ACM, 2012.

[13] Louise Barkhuus. The mismeasurement of privacy: using contextual integrity to reconsider privacy in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 367–376, 2012.

[14] Petros Belsis, Dimitris Vassis, Stefanos Gritzalis, and Christos Skourlas. W-ehr: a wireless distributed framework for secure dissemination of electronic healthcare records. In *Systems, Signals and Image Processing, 2009. IWSSIP 2009. 16th International Conference on*, pages 1–4. IEEE, 2009.

[15] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Annual Cryptology Conference*, pages 111–131. Springer, 2011. `https://link.springer.com/content/pdf/10.1007/978-3-642-22792-9_7.pdf`.

[16] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[17] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 313–314. Springer, 2013. `https://link.springer.com/content/pdf/10.1007/978-3-642-38348-9_19.pdf`.

[18] Albert Bifet. Teaching material, comp423/523a 2012 stream data mining/regression, 2012. https://speakerdeck.com/abifet/regression.

[19] Brent Bilger, Alan Boehme, Bob Folres, Zvi Guterman, Mark Hoover, Michaela Iorga, Junaid Islam, Marc Kolenko, Juanita Koilpilla, Gabor Lengyel, et al. Sdp specification 1.0, 2014.

[20] Eric Blais and Venkat Guruswami. Introduction to coding theory, basics of finite fields, 2010. http://www.cs.cmu.edu/~venkatg/teaching/codingtheory/notes/algebra-brief-notes.pdf.

[21] Eric Blais and Venkat Guruswami. Introduction to coding theory, notes 6: Reed-solomon, bch, reed-muller, and concatenated codes. *CMU: Spring 2010*, 2010. http://www.cs.cmu.edu/~venkatg/teaching/codingtheory/notes/notes6.pdf.

[22] Fabio Botelho, Fernando Manuel Valente Ramos, Diego Kreutz, and Alysson Bessani. On the feasibility of a consistent and fault-tolerant data store for sdns. In *Software Defined Networks (EWSDN), 2013 Second European Workshop on*, pages 38–43. IEEE, 2013.

[23] Kevin D Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198, 2009.

[24] Kevin D Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 43–54, 2009.

[25] Xavier Boyen, Hovav Shacham, Emily Shen, and Brent Waters. Forward-secure signatures with untrusted update. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 191–200, 2006.

[26] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408–415. IEEE, 2010.

[27] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. *Internet mathematics*, 1(4):485–509, 2004. https://projecteuclid.org/download/pdf_1/euclid.im/1109191032.

[28] Kai Bu, Yutian Yang, Zixuan Guo, Yuanyuan Yang, Xing Li, and Shigeng Zhang. Securing middlebox policy enforcement in sdn. *Computer Networks*, 193:108099, 2021.

[29] Lakshmi Devasena C. Comparative analysis of random forest, rep tree and j48 classifiers for credit risk prediction. *International Journal of Computer Applications (0975 -8887), International Conference on Communication, Computing and Information Technology (ICCCMIT-2014)*, 2014.

[30] Lakshmi Devasena C. Article: Comparative analysis of random forest, rep tree and j48 classifiers for credit risk prediction. *IJCA Proceedings on International Conference on Communication, Computing and Information Technology*, ICCCMIT 2014(3):30–36, March 2015. Full text available.

[31] Marco Canini, Daniele Venzano, Peter Peresini, Dejan Kostic, and Jennifer Rexford. A nice way to test openflow applications. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, number EPFL-CONF-170618 in 9, 2012.

[32] Martin Casado, Michael J Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 1–12. ACM, 2007.

[33] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 336–352. Springer, 2013. `https://link.springer.com/content/pdf/10.1007/978-3-642-38348-9_21.pdf`.

[34] Min-Cheng Chan, Chien Chen, Jun-Xian Huang, Ted Kuo, Li-Hsing Yen, and Chien-Chao Tseng. Opennet: A simulator for software-defined wireless local area network. In *Wireless Communications and Networking Conference (WCNC), 2014 IEEE*, pages 3332–3336. IEEE, 2014.

[35] Balakrishnan Chandrasekaran, Brendan Tschaen, and Theophilus Benson. Isolating and tolerating sdn application failures with legosdn. In *Proceedings of the Symposium on SDN Research*, page 7. ACM, 2016.

[36] Philip A Chou and Yunnan Wu. Network coding for the internet and wireless networks. *IEEE Signal Processing Magazine*, 24(5):77–85, 2007.

[37] Ronald H Coase. The federal communications commission. *The Journal of Law and Economics*, 56(4):879–915, 2013.

[38] Benjamin Coleman and Anshumali Shrivastava. A one-pass private sketch for most machine learning tasks. *arXiv preprint arXiv:2006.09352*, 2020. implementation available at: `https://github.com/brc7/PrivateRACE`.

[39] Benjamin Coleman and Anshumali Shrivastava. Sub-linear race sketches for approximate kernel density estimation on streaming data. In *Proceedings of The Web Conference 2020*, pages 1739–1749, 2020.

[40] Natalia Criado and Jose M Such. Implicit contextual integrity in online social networks. *Information Sciences*, 325:48–69, 2015.

[41] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 254–265. ACM, 2011.

[42] Reza Curtmola, Osama Khan, Randal Burns, and Giuseppe Ateniese. Mr-pdp: Multiple-replica provable data possession. In *2008 the 28th international conference on distributed computing systems*, pages 411–420. IEEE, 2008.

[43] Anderson Santos da Silva, Paul Smith, Andreas Mauthe, and Alberto Schaeffer-Filho. Resilience support in software-defined networking: A survey. *Computer Networks*, 92:189–207, 2015.

[44] Anderson Santos da Silva, Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville, and Alberto Schaeffer-Filho. Atlantic: A framework for anomaly traffic detection, classification, and mitigation in sdn. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 27–35. IEEE, 2016.

[45] Ala' Darabseh, Mahmoud Al-Ayyoub, Yaser Jararweh, Elhadj Benkhelifa, Mladen Vouk, Andy Rindos, et al. Sdsecurity: A software defined security experimental framework. In *2015 IEEE international conference on communication workshop (ICCW)*, pages 1871–1876. IEEE, 2015.

[46] Bálint Daróczy, Péter Vaderna, and András Benczúr. Machine learning based session drop prediction in lte networks and its son aspects. In *Vehicular Technology Conference (VTC Spring), 2015 IEEE 81st*, pages 1–5. IEEE, 2015.

[47] Jash Desai. Sdn-openflow-pox-firewall, 2018. https://github.com/jashdesai95/SDN-OpenFlow-Pox-Firewall.

[48] Yves Deswarte, Jean-Jacques Quisquater, and Ayda Saïdane. Remote integrity checking. In *Working Conference on Integrity and Internal Control in Information Systems*, pages 1–11. Springer, 2003.

[49] Andrea Detti, Claudio Pisa, Stefano Salsano, and Nicola Blefari-Melazzi. Wireless mesh software defined networks (wmsdn). In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, pages 89–95. IEEE, 2013.

[50] Alexandros G Dimakis, P Brighten Godfrey, Martin J Wainwright, and Kannan Ramchandran. The benefits of network coding for peer-to-peer storage systems. In *Third Workshop on Network Coding, Theory, and Applications*, 2007.

[51] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE transactions on information theory*, 56(9):4539–4551, 2010.

[52] Alexandros G Dimakis, Vinod Prabhakaran, and Kannan Ramchandran. Decentralized erasure codes for distributed networked storage. *IEEE Transactions on Information Theory*, 52(6):2809–2816, 2006.

[53] Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography Conference*, pages 109–127. Springer, 2009.

[54] Zeal Egaesiri Ekrebe. Security of software defined network with software defined perimiter. *ERA (Education and Research Archive),University of Alberta*, 2020.

[55] Karim ElDefrawy and Tyler Kaczmarek. Byzantine fault tolerant software-defined networking (sdn) controllers. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual*, volume 2, pages 208–213. IEEE, 2016.

[56] C Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. *ACM Transactions on Information and System Security (TISSEC)*, 17(4):15, 2015. https://user.eng.umd.edu/~cpap/published/cce-alp-cpap-rt-09.pdf.

[57] Chris Erway, Alptekin Küpçü, Charalampos Papamanthou, and Roberto Tamassia. Dynamic provable data possession. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 213–222, New York, NY, USA, 2009. Association for Computing Machinery.

[58] Ertem Esiner, Adilet Kachkeev, Samuel Braunfeld, Alptekin Küpçü, and Öznur Özkasap. Flexdpdp: Flexlist-based optimized dynamic provable data possession. *ACM Transactions on Storage (TOS)*, 12(4):1–44, 2016. `https://crypto.ku.edu.tr/wp-content/uploads/2019/05/flexdpdp.pdf`.

[59] Ertem Esiner, Alptekin Küpçü, and Öznur Özkasap. Analysis and optimization on flexdpdp: A practical solution for dynamic provable data possession. In *International Conference on Intelligent Cloud Computing*, pages 65–83. Springer, 2014.

[60] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Extending {SDN} to handle dynamic middlebox actions via {FlowTags}. In *Open Networking Summit 2014 (ONS 2014)*, 2014.

[61] Giancarlo Fortino, Lidia Fotia, Fabrizio Messina, Domenico Rosaci, and Giuseppe ML Sarné. Trust and reputation in the internet of things: state-of-the-art and research challenges. *IEEE Access*, 8:60117–60125, 2020.

[62] Robert Furberg, Julia Brinton, Michael Keating, and Alexa Ortiz. Crowd-sourced fitbit datasets 03.12.2016-05.12.2016, May 2016. `https://doi.org/10.5281/zenodo.53894`.

[63] Décio Luiz Gazzoni Filho and Paulo Sérgio Licciardi Messeder Barreto. Demonstrating data possession and uncheatable data transfer. *IACR Cryptology ePrint Archive*, 2006:150, 2006.

[64] A Goldberger, L Amaral, L Glass, J Hausdorff, PC Ivanov, R Mark, JE Mietus, GB Moody, CK Peng, HE Stanley, PhysioBank, PhysioToolkit, and PhysioNet. Components of a new research resource for complex physiologic signals. *PhysioNet*, 2000. Circulation. 101 (23), pp. e215–e220.

[65] Shyamnath Gollakota, Haitham Hassanieh, Benjamin Ransford, Dina Katabi, and Kevin Fu. They can hear your heartbeats: non-invasive security for implantable medical devices. *ACM SIGCOMM Computer Communication Review*, 41(4):2–13, 2011.

[66] Philippe Golle, Stanislaw Jarecki, and Ilya Mironov. Cryptographic primitives enforcing communication and storage complexity. In *International Conference on Financial Cryptography*, pages 120–135. Springer, 2002.

[67] Mihaela Göndör and Vasile Paul Bresfelean. Reptree and m5p for measuring fiscal policy influences on the romanian capital market during 2003–2010. *International Journal of Mathematics and Computers in Stimulation*, 6(4):378–386, 2012.

[68] Michael T Goodrich, Mikhail J Atallah, and Roberto Tamassia. Indexing information for data forensics. In *International Conference on Applied Cryptography and Network Security*, pages 206–221. Springer, 2005. `https://link.springer.com/content/pdf/10.1007/11496137_15.pdf`.

[69] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[70] Tang Guodong, Qin Xi, and Chang Chaowen. A sdn security control forwarding mechanism based on cipher identification. In *Communication Software and Networks (ICCSN), 2017 IEEE 9th International Conference on*, pages 1419–1425. IEEE, 2017.

[71] Stephen Gutz, Alec Story, Cole Schlesinger, and Nate Foster. Splendid isolation: A slice abstraction for software-defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 79–84. ACM, 2012.

[72] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[73] Gunnar Hartung, Björn Kaidel, Alexander Koch, Jessica Koch, and Dominik Hartmann. Practical and robust secure logging from fault-tolerant sequential aggregate signatures. In *International Conference on Provable Security*, pages 87–106. Springer, 2017.

[74] Gunnar Hartung, Björn Kaidel, Alexander Koch, Jessica Koch, and Andy Rupp. Fault-tolerant aggregate signatures. In *Public-Key Cryptography–PKC 2016*, pages 331–356. Springer, 2016.

[75] Jialing He, Zijian Zhang, Meng Li, Liehuang Zhu, and Jingjing Hu. Provable data integrity of cloud storage service with enhanced security in the internet of things. *IEEE Access*, 7:6226–6239, 2018.

[76] Chaesub Lee (head). Software-defined networking (sdn). `https://www.itu.int/en/ITU-T/sdn/Pages/default.aspx`. Accessed: 2019-08-23.

[77] Tom Henderson. ns3::logdistancepropagationlossmodel, 2015.

[78] Tom Henderson. ns3::nisterrorratemodel, 2015.

[79] Helium Systems Inc. Helium - introducing people powered networks. `https://www.helium.com/`. Accessed:4/26/2022.

[80] Jafar Haadi Jafarian, Ehab Al-Shaer, and Qi Duan. Openflow random host mutation: transparent moving target defense using software defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 127–132. ACM, 2012.

[81] Yunhan Jack Jia, Qi Alfred Chen, Shiqi Wang, Amir Rahmati, Earlence Fernandes, Zhuoqing Morley Mao, Atul Prakash, and Shanghai JiaoTong Unviersity. Contexlot: Towards providing contextual integrity to appified iot platforms. In *NDSS*, 2017.

[82] Ari Juels and Burton S Kaliski Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597, 2007.

[83] Yossi Kanizo, David Hay, and Isaac Keslassy. Palette: Distributing tables in software-defined networks. In *INFOCOM, 2013 Proceedings IEEE*, pages 545–549. IEEE, 2013.

[84] Maryam Karimi. Adi, 2020. `https://github.com/Maryam-mary-karimi/ADI`.

[85] Maryam Karimi and M Ahmadzadeh. Mining robocup log files to predict own and opponent action. *International Journal of Advanced Research in Computer Science (IJARCS)*, 5(6):27–32, 2014.

[86] Maryam Karimi, Prashant Krishnamurthy, James Joshi, and David Tipper. Mining historical data towards interference management in wireless sdns. In *Proceedings of*

the 13th ACM Symposium on QoS and Security for Wireless and Mobile Networks, pages 81–88. ACM, 2017.

[87] Maryam Karimi, Mohammad Sadegh Najafi, Reza Akbari, and Manijeh Keshtgari. Presenting a new method, using topology virtualization for stabilizing flow tables in sdwn. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 219–226. IEEE, 2017.

[88] Pradeeban Kathiravelu and Luís Veiga. Sdn middlebox architecture for resilient transfers. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 560–563, 2017.

[89] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 49–54. ACM, 2012.

[90] Hyojoon Kim, Mike Schlansker, Jose Renato Santos, Jean Tourrilhes, Yoshio Turner, and Nick Feamster. Coronet: Fault tolerance for software defined networks. In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–2. IEEE, 2012.

[91] Rowan Kloti, Vasileios Kotronis, and Paul Smith. Openflow: A security analysis. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–6. IEEE, 2013.

[92] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication, 1997.

[93] Diego Kreutz, Fernando Ramos, and Paulo Verissimo. Towards secure and dependable software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 55–60. ACM, 2013.

[94] John Kruschke. Doing bayesian data analysis: A tutorial with r, jags, and stan. *Academic Press*, 2014.

[95] John K Kruschke. Bayesian estimation supersedes the t test. *Journal of Experimental Psychology: General*, 142(2):573, 2013.

[96] Adrian Lara and Byrav Ramamurthy. Opensec: Policy-based security using software-defined networking. *IEEE transactions on network and service management*, 13(1):30–42, 2016.

[97] Rafael P Laufer, Pedro B Velloso, and Otto Carlos MB Duarte. A generalized bloom filter to secure distributed network applications. *Computer Networks*, 55(8):1804–1819, 2011. `https://www.gta.ufrj.br/ftp/gta/TechReports/LVD11.pdf`.

[98] Kin K Leung. Power control by interference prediction for broadband wireless packet networks. *IEEE Transactions on Wireless Communications*, 1(2):256–265, 2002.

[99] He Li, Peng Li, Song Guo, and Amiya Nayak. Byzantine-resilient secure software-defined networks with multiple controllers in cloud. *IEEE Transactions on Cloud Computing*, 2(4):436–447, 2014.

[100] He Li, Peng Li, Song Guo, and Shui Yu. Byzantine-resilient secure software-defined networks with multiple controllers. In *Communications (ICC), 2014 IEEE International Conference on*, pages 695–700. IEEE, 2014.

[101] Xueping Liang, Sachin Shetty, Deepak Tosh, Charles Kamhoua, Kevin Kwiat, and Laurent Njilla. Provchain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability. In *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 468–477. IEEE, 2017.

[102] Bin Liu, Xiao Liang Yu, Shiping Chen, Xiwei Xu, and Liming Zhu. Blockchain based data integrity service framework for iot data. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 468–475. IEEE, 2017.

[103] Gong-Xu Liu, Ling-Feng Shi, and Dong-Jin Xin. Data integrity monitoring method of digital sensors for internet-of-things applications. *IEEE Internet of Things Journal*, 7(5):4575–4584, 2020.

[104] Angus Loten. CIOs contend with ever-expanding range of cloud services. *The Wall Street Journal*, December 1st 2017.

[105] Tze-Ping Low and Jangwook Moon. Interference modulation order detection with supervised learning for lte interference cancellation. In *Vehicular Technology Conference (VTC Fall), 2015 IEEE 82nd*, pages 1–5. IEEE, 2015.

[106] Shouxi Luo, Hongfang Yu, et al. Fast incremental flow table aggregation in sdn. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[107] Di Ma and Gene Tsudik. A new approach to secure logging. *ACM Transactions on Storage (TOS)*, 5(1):1–21, 2009.

[108] Mehdi Malboubi, Liyuan Wang, Chen-Nee Chuah, and Puneet Sharma. Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp). In *INFOCOM, 2014 Proceedings IEEE*, pages 934–942. IEEE, 2014.

[109] Cintia B Margi, Renan CA Alves, and Johanna Sepulveda. Sensing as a service: secure wireless sensor network infrastructure sharing for the internet of things. *Open Journal of Internet Of Things (OJIOT)*, 3(1):91–102, 2017.

[110] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[111] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *International workshop on recent advances in intrusion detection*, pages 161–180. Springer, 2011.

[112] Mininet-Core-Team. Mininet, 2020. `http://mininet.org`.

[113] Marcelo Duffles Donato Moreira, Rafael Pinaud Laufer, Pedro Braconnot Velloso, and Otto Carlos MB Duarte. Capacity and robustness tradeoffs in bloom filters for distributed applications. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2219–2230, 2012. `https://ieeexplore.ieee.org/abstract/document/6171165`.

[114] Abdallah Moubayed, Ahmed Refaey, and Abdallah Shami. Software-defined perimeter (sdp): State of the art secure solution for modern networks. *IEEE Network*, 33(5):226–233, 2019.

[115] Ankur Kumar Nayak, Alex Reimers, Nick Feamster, and Russ Clark. Resonance: dynamic access control for enterprise networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 11–18. ACM, 2009.

[116] G Neri, RCS Morling, GD Cain, E Faldella, M Longhi-Gelati, T Salmon-Cinotti, and P Natali. Mininet: A local area network for real-time instrumentation applications. *Computer Networks (1976)*, 8(2):107–131, 1984.

[117] Helen Nissenbaum. *Privacy in context: Technology, policy, and the integrity of social life*. Stanford University Press, 2009.

[118] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[119] Christof Paar and Jan Pelzl. Sha-3 and the hash function keccak. *Understanding Cryptography A Textbook for Students and Practitioners, www. crypto-textbook. com*, 2010. `http://professor.unisinos.br/linds/teoinfo/Keccak.pdf`.

[120] Guangyu Pei and Thomas R Henderson. Validation of ofdm error rate model in ns-3. *Boeing Research Technology*, pages 1–15, 2010.

[121] Larry Peterson, Ali Al-Shabibi, Tom Anshutz, Scott Baker, Andy Bavier, Saurav Das, Jonathan Hart, Guru Palukar, and William Snow. Central office re-architected as a data center. *IEEE Communications Magazine*, 54(10):96–101, 2016.

[122] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and Guofei Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 121–126. ACM, 2012.

[123] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software defined network control layer. In *NDSS*, 2015.

[124] Konstantinos Poularakis, Qiaofeng Qin, Erich Nahum, Miguel Rio, and Leandros Tassiulas. Bringing sdn to the mobile edge. In *2017 IEEE SmartWorld (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–6. IEEE, 2017.

[125] Narasimha Prasad, Kishor Kumar Reddy, and Ramya Tulasi Nirjogi. A novel approach for seismic signal magnitude detection using haar wavelet. In *Intelligent Systems, Modelling and Simulation (ISMS), 2014 5th International Conference on*, pages 324–329. IEEE, 2014.

[126] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 27–38, 2013.

[127] Chao Qi, Jiangxing Wu, Hongchang Chen, Hongtao Yu, Hongchao Hu, and Guozhen Cheng. Game-theoretic analysis for security of various software-defined networking (sdn) architectures. In *Vehicular Technology Conference (VTC Spring), 2017 IEEE 85th*, pages 1–5. IEEE, 2017.

[128] Chao Qi, Jiangxing Wu, Hongchao Hu, Guozhen Cheng, Wenyan Liu, Jianjian Ai, and Chao Yang. An intensive security architecture with multi-controller for sdn. In *Computer Communications Workshops (INFOCOM WKSHPS), 2016 IEEE Conference on*, pages 401–402. IEEE, 2016.

[129] John R Quinlan et al. Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, volume 92, pages 343–348. Singapore, 1992.

[130] Manickam Ramasamy, Shanthi Selvaraj, and M Mayilvaganan. An empirical analysis of decision tree algorithms: Modeling hepatitis data. In *Engineering and Technology (ICETECH), 2015 IEEE International Conference on*, pages 1–4. IEEE, 2015.

[131] Mark Reitblatt, Marco Canini, Arjun Guha, and Nate Foster. Fattire: Declarative fault tolerance for software-defined networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 109–114. ACM, 2013.

[132] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and Tools for Network Simulation*, pages 15–34. Springer, 2010.

[133] Zahra Ronaghi, Edward B Duffy, and David M Kwartowitz. Toward real-time remote processing of laparoscopic video. *Journal of Medical Imaging*, 2(4):045002–045002, 2015.

[134] Shiva Rowshanrad, Sahar Namvarasl, Vajihe Abdi, Maryam Hajizadeh, and Manijeh Keshtgary. A survey on sdn, the future of networking. *Journal of Advanced Computer Science & Technology*, 3(2):232, 2014.

[135] Ahmed Sallam, Ahmed Refaey, and Abdallah Shami. On the security of sdn: A completed secure and scalable framework using the software-defined perimeter. *IEEE Access*, 7:146577–146587, 2019.

[136] Salvatore Sanfilippo. hping3(8) - linux man page, 2016.

[137] Julius Schulz-Zander, P Lalith Suresh, Nadi Sarrar, Anja Feldmann, Thomas Hühn, and Ruben Merz. Programmatic orchestration of wifi networks. In *USENIX Annual Technical Conference*, pages 347–358, 2014.

[138] Thomas SJ Schwarz and Ethan L Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 12–12. IEEE, 2006.

[139] Sandra Scott-Hayward, Sriram Natarajan, and Sakir Sezer. A survey of security in software defined networks. *IEEE Communications Surveys & Tutorials*, 18(1):623–654, 2016.

[140] F Sebe, A Martinez-Balleste, Y Deswarte, J Domingo-Ferrer, and JJ Quisquater. Time-bounded remote file integrity checking. *Technical Report 04429*, 2004.

[141] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[142] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107. Springer, 2008.

[143] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, 1:132, 2009.

[144] Rob Sherwood, Glen Gibb, Kok-Kiong Yap, Guido Appenzeller, Martin Casado, Nick McKeown, and Guru M Parulkar. Can the production network be the testbed? In *OSDI*, volume 10, pages 1–6, 2010.

[145] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6. IEEE, 2012.

[146] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 165–166. ACM, 2013.

[147] Seungwon Shin, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, Guofei Gu, and Mabry Tyson. Fresco: Modular composable security services for software-defined networks. In *NDSS*, 2013.

[148] Seungwon Shin, Yongjoo Song, Taekyung Lee, Sangho Lee, Jaewoong Chung, Phillip Porras, Vinod Yegneswaran, Jiseong Noh, and Brent Byunghoon Kang. Rosemary: A robust, secure, and high-performance network operating system. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 78–89. ACM, 2014.

[149] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013.

[150] Paris Siminelakis, Kexin Rong, Peter Bailis, Moses Charikar, and Philip Levis. Rehashing kernel evaluation in high dimensions. In *International Conference on Machine Learning*, pages 5789–5798. PMLR, 2019.

[151] Jaspreet Singh, Ahmed Refaey, and Abdallah Shami. Multilevel security framework for nfv based on software defined perimeter. *IEEE Network*, 34(5):114–119, 2020.

[152] Mayank Taneja, Kavyanshi Garg, Archana Purwar, and Samarth Sharma. Prediction of click frauds in mobile advertising. In *Contemporary Computing (IC3), 2015 Eighth International Conference on*, pages 162–166. IEEE, 2015.

[153] Moazzam Islam Tiwana, Berna Sayrac, and Zwi Altman. Statistical learning in automated troubleshooting: Application to lte interference mitigation. *IEEE Transactions on Vehicular Technology*, 59(7):3651–3656, 2010.

[154] Eric Topol. The smart-medicine solution to the health-care crisis. *The Wall Street Journal*, July 7 2017.

[155] Renlong Tu, Xin Wang, Jin Zhao, Yue Yang, Lei Shi, and Tilman Wolf. Design of a load-balancing middlebox based on sdn for data centers. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 480–485, 2015.

[156] Ricard Vilalta, Raluca Ciungu, Arturo Mayoral, Ramon Casellas, Ricardo Martinez, David Pubill, Jordi Serra, Raul Munoz, and Christos Verikoukis. Improving security in internet of things with software defined networking. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016.

[157] Matthew P Wallen, Sjaan R Gomersall, Shelley E Keating, Ulrik Wisløff, and Jeff S Coombes. Accuracy of heart rate watches: Implications for weight management. *PloS one*, 11(5):e0154420, 2016. `https://www.ncbi.nlm.nih.gov/pubmed/27232714`.

[158] Chao Wang, Shizhan Chen, Zhiyong Feng, Yanan Jiang, and Xiao Xue. Block chain-based data audit and access control mechanism in service collaboration. In *2019 IEEE International Conference on Web Services (ICWS)*, pages 214–218. IEEE, 2019.

[159] Haiyan Wang and Jiawei Zhang. Blockchain based data integrity verification for large-scale iot data. *IEEE Access*, 7:164996–165006, 2019.

[160] Tian Wang, Md Zakirul Alam Bhuiyan, Guojun Wang, Lianyong Qi, Jie Wu, and Thaier Hayajneh. Preserving balance between privacy and data integrity in edge-assisted internet of things. *IEEE Internet of Things Journal*, 7(4):2679–2689, 2019.

[161] Ye Wang, Yueping Zhang, Vishal Singh, Cristian Lumezanu, and Guofei Jiang. Netfuse: Short-circuiting traffic surges in the cloud. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3514–3518. IEEE, 2013.

[162] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. Ofrewind: enabling record and replay troubleshooting for networks. In *USENIX Annual Technical Conference*, pages 327–340. USENIX Association, 2011.

[163] Go Yamamoto, Satoshi Oda, and Kazumaro Aoki. Fast integrity for large data. In *Proc. ECRYPT Workshop Software Performance Enhancement for Encryption and Decryption*, pages 21–32, 2007.

[164] Guang Yao, Jun Bi, and Peiyao Xiao. Source address validation solution with openflow/nox architecture. In *Network Protocols (ICNP), 2011 19th IEEE International Conference on*, pages 7–12. IEEE, 2011.

[165] Dongdong Yue, Ruixuan Li, Yan Zhang, Wenlong Tian, and Chengyi Peng. Blockchain based data integrity verification in p2p cloud storage. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 561–568. IEEE, 2018.

[166] University of Michigan Z. Morley Mao. Network service model, 2010.

[167] Yongheng Zhao and Yanxia Zhang. Comparison of decision tree methods for finding active objects. *Advances in Space Research*, 41(12):1955–1959, 2008.

[168] Qingji Zheng and Shouhuai Xu. Fair and dynamic proofs of retrievability. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 237–248, 2011.

[169] Zhang Zhongheng, Cao Linghong, Zhao Yan, Xu Ziyin, Chen Rangui, Lv Lukai, and Xu Ping. Hospitalized patients with heart failure: integrating electronic healthcare records and external outcome data (version 1.2). *PhysioNet*, 2020.