UNIVERSITY OF CALIFORNIA
SANTA CRUZ

SECURE COMMUNICATION INFRASTRUCTURES FOR CLOUD AND IOT

A dissertation submitted in partial satisfaction
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Xin Li

December 2018

The Dissertation of Xin Li
is approved:

Professor Chen Qian, Chair

Professor Katia Obraczka

Professor Ethan L. Miller

Lori Kletzer
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Secure Communication Infrastructures for Cloud and IoT

by

Xin Li

Cloud computing is a paradigm that enables the rapid provisioning of shared pools of hardware resources or high-level services. The cloud offers the flexibility to create, configure and cancel resources on demand. Third-party clouds have rich computing/storage resources and charge their tenants for resource usages. Despite its wide adoption, the cloud is not immune to security attacks. This dissertation attempts to enhance the security of cloud from two different aspects: 1) Fortify network security infrastructure in the cloud. 2) Fortify IoT Data in the cloud.

The first half of this dissertation presents an SDN-based modular NFV orchestration framework called APPLE, aiming for interference-free policy enforcement of security infrastructure in a resource-efficient manner. Several levels of mechanisms are leveraged in APPLE to incorporate traffic dynamics. Both simulation and prototype experiments using real network topologies and traffic traces show that APPLE is resource-efficient and can quickly react to traffic dynamics.

The second half of the dissertation describes two security protocol suits for verifiable data communication and management respectively. Both are specially optimized for IoT applications to fit into resource-constraint IoT devices. Compared to alternative solutions, both protocol suits reduce memory footprint on IoT devices, communication cost between IoT devices and the cloud as well as computing time on generation and verification of verifiable IoT data.

To my parents and wife.

Acknowledgments

I would like to express my best gratitude to my advisor, Professor Qian. He is the most important person for my Ph.D study and career path. It is Professor Qian that teaches me how to do solid research in computer networking starting from scratch. From Kentucky to California, he always gives me precious academic and life guidance in need. Without his encouragement and support, I would not have survived in the Ph.D pursuit journey. Professor Qian is the most knowledgeable person I have ever met in my life. He always encourages me to set foot in new research areas, to communicate with other scholars and more importantly to think out of box.

I would thank my qualifying-exam and dissertation committee, Professor Katia Obraczka, Professor Ethan L. Miller and Dr. Ying Zhang for their valuable guidance and constructive suggestions.

I would acknowledge Professor Song Han from University of Connecticut and Bo Han from AT&T Lab for their help shaping my research ideas into research projects. I also appreciate the mentoring of Dr. Wolfram Schulte and help of other colleagues during my internship at Facebook.

I am fortunate to have so many brilliant group members: Ye Yu, Huazhe Wang, Yu Zhao, Minmei Wang, Shouqian Shi, Haofan Cai, Ge Wang, Junjie Xie and Xiaofeng Shi. Many research challenges were tackled through discussions with them.

I am grateful for my parents. They not only give my life but also teach me how to be a good man. I would give special thanks to my beloved wife Ming. She understands and supports all of my critical life decisions. We together enjoy this chapter of our lives.

Chapter 1

Introduction

Cloud computing is a paradigm that enables the rapid provisioning of shared pools of hardware resources or high-level services. Cloud offers the flexibility to create, configure and cancel resources on demand. Third-party clouds have rich computing/storage resources and charge their tenants for resource usages. Cloud tenants could focus more on their core business and be free from the burden of expending and maintaining underlying computer infrastructure. Numerous companies start to deploy part of or even all services inside cloud [5, 36, 13].

Despite of its wide adoption, cloud is not immune to security attacks. According to a survey of 1400 IT decision-makers conducted by McAfee [25], 25% cloud tenants have experienced data theft from the public cloud and 20% cloud tenants have experienced an advanced attack against their public cloud infrastructure. This dissertation attempts to enhance the security of cloud from two different aspects:

1) **Fortify network security infrastructure in cloud.** Like enterprise network, various kinds of network security infrastructure such as firewall [89], intrusion detection system [109] are critical for the security of services hosted in cloud. This dissertation focuses on the orchestration of different security infrastructure

instances collectively enforce security policies to protect service operations. The first half of this dissertation presents an SDN-based modular NFV orchestration framework called APPLE, aiming for interference-free policy enforcement of security infrastructure in a resource-efficient manner. Several levels of mechanisms are leveraged in APPLE to incorporate traffic dynamics. Both simulation and prototype experiments using real network topologies and traffic traces show that APPLE is resource-efficient and can quickly react traffic dynamics.

2) **Fortify IoT Data in cloud.** Data storage is another driver for the adoption of cloud. Especially for emerging IoT, resource-restraint IoT devices send their sensing data to cloud for storage. Tenants retrieve data from cloud when needed. Ensure authenticity and integrity of data stored in the cloud is essential for the correctness and security of many applications. Although verifiable data outsourcing has been studied for over a decade, current solutions [110, 140, 180, 179, 120, 61] focus on general-purpose computing platforms such as PC and they are not fully suitable for the IoT due to stringent hardware constraints. The dissertation thus focuses on offer authenticity and integrity to IoT data in cloud. The second half of the dissertation describes two security protocol suits for verifiable data communication and management respectively. Both are specially optimized for IoT applications to fit into resource-constraint IoT devices. Compared to alternative solutions, both protocol suits reduce memory footprint on IoT devices, communication cost between IoT devices and the cloud as well as computing time on generation and verification of verifiable IoT sensing data.

1.1 Fortifying Network Security Infrastructure

Network security infrastructure such as firewalls and intrusion detection systems is indispensable for today's network. Security policies require traffic flow to traverse designated kinds of security infrastructure in sequence. The correct operation of network security infrastructure is critical to deliver security assurance. Network security infrastructure belongs to a broader category of appliances called network functions. Traditional network functions are shipped in dedicated hardware, but recent advances in general-purpose CPUs motivate network functions being implemented in software and then deployed through virtualization technologies, which is also referred to as Network Functions Virtualization (NFV). Though NFV brings flexible management of network functions in their placement, configuration and provisioning, off-the-shelf solutions from virtualization technology cannot directly address the orchestration of different kinds of network security infrastructure to collectively enforce security policies.

1.1.1 Network Function and Service Function Chain

Modern networks not only constitute of devices providing connectivity (*e.g.* switches and routers) but also a large number of devices that perform functions other than packet forwarding, which are called network functions (NFs). Therefore network security infrastructure such as the firewall and the intrusion detection system (IDS) belongs to NFs. Besides improving security, NFs (*e.g.* web proxy, video transcoder and load balancer) optimize performance. NFs are crucial in various modern networks, such as mobile networks [111, 171, 39, 182], enterprise networks [69], datacenter networks [37, 121] and broadband access networks [177, 6]. A recent study [162] reveals that the number of NFs is comparable to that of the switches.

Legacy NFs are usually shipped in dedicated hardware to trade manage flexibility for performance. The hardware NF introduces large capital expenses (CapEx) as well as expensive operational costs (OpEx). As a result, only limited number of hardware NFs are deployed in a network. Once NFs are deployed, it is unfeasible to redeploy them.

The correct operation of network security infrastructure is critical to deliver security assurance. Application or user requirements may specify security policies that require flows traverse through a given sequence of NFs, called *service function chain*. For example, an enterprise network administrator may specify a policy that outbound traffic should follow the service function chain: $firewall \rightarrow IDS \rightarrow compressor$.

To enforce service function chaining, *traffic steering* is utilized to make changes on traffic paths such that network flows can traverse through the designated NFs. However, traffic steering suffers from the following problems:

- Traffic steering may affect the performance of other network control applications such as traffic engineering. Ideally, policy enforcement should be completely orthogonal to other applications.

- Traffic steering introduces additional path length, which increases the probability of congestion and long delay, especially for WAN. The links near NFs are likely to become hot spots. The situation is aggravated if the NFs are not properly placed in the network.

- Much more forwarding rules are installed in the switches/routers forwarding table to support complex traffic steering [148]. Forwarding table space is a scarce resource.

On the other hand, the computing capability of general-purpose CPUs increase exponentially over time. As a result, NFV [106], or Network Functions Virtualization, has been proposed to rest on modern general-purpose hardware to realize functionality of various NFs. By leveraging virtualization technologies, NFV enables flexible management of virtualized network functions (VNFs) in terms of placement, configuration and provisioning. Nevertheless, off-the-shelf solutions from virtualization technology (*e.g.* Openstack [32]) cannot directly address the orchestration of different kinds of network security infrastructure to collectively enforce service function chains.

1.1.2 Desirable Properties of NFV Orchestration Framework

Besides **resource-efficiency** and **resistance to traffic dynamics** which are commonly identified by all hardware NF and NFV orchestration frameworks [177, 156, 139, 138, 74], this dissertation further take advantages of the flexibility of VNFs to achieve the following desirable properties that are impossible or irrelevant without NFV.

Order preserving. The partial order of service function chain should be preserved. Failure to preserve the order may nullify network security infrastructure and thus imposes great security risk on the network. For instance, IDS cannot inspect the payload of outbound traffic packets if the packets traverse the compressor first. Furthermore, network traffic should not traverse NFs not specified in the service function chain. At minimum, some resources such as CPU are wasted. Order preserving is not always retained for hardware NFs. As elaborated in the survey at Chapter 2, some placement strategies for hardware NFs prefer placing NF instances at certain vantage points and these NF instances directly tap into traffic links. In this case, without fine-grained control some traffic flows inevitably traverse irrelevant NF instance placed at vantage points.

Interference freedom. The framework should not change the existing flow routing paths determined by other network applications such as routing, access control, and traffic engineering. Since only limited number of hardware NFs are deployed in a network, traffic steering is adopted by most hardware NF orchestration frameworks [177, 148]. NFV brings the ability to instantiate VNF instances at hosts on demand, which opens up the desirable property of interference freedom. Moreover, interference freedom is the key enabler of modular NFV orchestration framework where different modules collectively control the network. Most current NFV orchestration frameworks (e.g. [97] [138]) jointly optimize path selection and instance placement. This tight coupling impedes novelty and continuous improvement brought by modular plugins.

Genericity. The NFV orchestration framework needs to support as many types of NFs as possible and requires no modification to the guest VNF to work. On one hand, imposing no limitations or assumptions on the virtualized network function can save human labors and hence boosting the adoption of the orchestration framework. More importantly, modern NFs are so diversified and the logic is so complicated, not all NFs are able to be ported to follow the programming paradigms/patterns of specified NFV orchestration frameworks.

None existing orchestration framework can satisfy the aforementioned requirements simultaneously. Chapter 3 demonstrates the design and implementation of APPLE, a NFV orchestration framework with such desirable properties. Before presenting APPLE, this dissertation first illustrates a survey on one of the most important aspect of NF orchestration, NF placement, in Chapter 2.

1.2 Fortifying the Internet of Things

Internet of Things (IoT) is being widely applied in a great number of everyday applications such as healthcare [11, 151], transportation [101], smart home [26, 14, 108], and surveillance systems [117, 75]. IoT devices usually generate a large amount of sensing data to reflect physical environments or conditions of objects and human beings. The majority of IoT hardware platforms are constrained in resources where only one small on-board flash memory (e.g., 1MB for TelosB [46] and 16MB for Z1 [47]) provides limited storage capacity. As most IoT devices carry constrained resources and limited storage capacity, sensing data need to be transmitted to and stored at resource-rich platforms, such as a cloud. On the other hand, analyzing historical sensing data is essential for decision-making in various IoT applications [26] [155]. For example, Nest Learning Thermostat [26], a system that controls the temperature of a smart home automatically and intellectually, learns a user's preference by analyzing historical data of the home. Hence IoT applications need to retrieve sensing data from the cloud for analysis and decision-making purposes. To this end, both state-of-art IoT proposals [108, 105] and industrial IoT practices [31] adopt the centralized data store residing in the cloud, aiming that the sensing data can be stored economically and retrieved effectively for analysis, as depicted in Figure 4.1.

Since the sensing data are stored in a third-party cloud, data authenticity and integrity, which guarantee that data are from these sensing devices and have not been modified, are important for trustworthy IoT applications [169]. However the data could be corrupted by outside attackers [3, 4, 8], malicious cloud employees [123], transmission failures, or storage loss [40]. Without data authenticity and integrity, IoT applications may make wrong decisions and cause economic and human-life losses.

A critical task of both IoT communication and data management systems are to allow the data consumers to verify the **correctness** of the IoT data. The "correctness" here includes two requirements: 1) **Authenticity and integrity**: the data should be collected from the sensing devices and not be tampered by any

Table 1.1: Performance of some cryptographic operations.

Metric	RSA	DSA	MD5	SHA1	SHA256
Time (ms)	203.6	192.8	0.041	0.051	0.098
Energy (mJouel)	37.6	36.2	0.007	0.009	0.017

third party; 2) **Completeness**: the data consumer should receive all and only the data satisfying the conditions in its query.

1.2.1 Digital Signature

Digital signature is a widely used method to protect data authenticity and integrity: The sender first computes a message digest D by hashing its original message m using a cryptographic hash function H, $D = H(m)$. H is also called message digest function. Note the length of D is significantly shorter than that of m. Then the sender uses its private key k to encrypt D and attaches the *signature* $E_k[D]$ to the original message. When the receiver gets m and $E_k[D]$, it decrypts $E_k[D]$ using the public key of the sender and verifies whether $D = H(m)$. However, applying the digital signature to every sensing record, called the *Sign-each* method, is not practical, because public-key encryption/decryption is considered slow and expensive, especially for sensing devices with limited resources.

The performance of some mostly used cryptographic operations is measured on M3, one mainstream IoT hardware platform available from one public testbed [12]. This hardware platform features one 32-bit ARM Cortex-M3 CPU@72MHz. The result shows the average time and energy to encrypt (RSA/DSA) or to compute hash (MD5/SHA1/SHA256) over a 10-byte string. The result is presented in Table 4.1. Even though there is great advances in hardware performance compared to prior platforms [144], directly applying RSA/DSA is still not suitable for resource-constraint IoT devices, especially for those powered by batteries.

1.2.2 Stringent Requirements

Although verifiable data outsourcing has been studied for over a decade, current solutions [110, 140, 180, 179, 120, 61] focus on general-purpose computing platforms such as PC and they are not fully suitable for the IoT due to stringent hardware constraints. Therefore, verifiable data outsourcing for IoT should possess the following properties.

- **Computation efficiency.** IoT devices are usually limited in computation power. Low-end CPUs are chosen for IoT because of their low prices and their energy-efficiency.
- **Memory efficiency.** IoT devices are constrained by memory capacity. Some IoT hardware platforms even do not equipped with off-chip memories.
- **Communication efficiency.** Communication of IoT devices is often more power-consuming than computation by orders of magnitude [93, 96]. Therefore, the sizes of updates from IoT devices to the cloud is an important metric.

The dissertation describes two security protocol suits for verifiable data communication and management respectively. Both are specially optimized for IoT applications to fit into resource-constraint IoT devices. Both protocol suits involves three key entities: *IoT devices*, *cloud*, and *data consumer*. The protocol suits for verifiable IoT data communication is called GSC. GSC enables data consumers to uniformly and efficiently fetch partial IoT data from the cloud. The protocol for verifiable IoT data management, VERID, further provides data consumers with the ability to retrieve data against SQL-like queries.

1.3 Dissertation Outline

In Chapter 2, a survey is conducted to show how NFs are placed in the network for different systems. This chapter stands on a system view. A system is developed to enforce policies in the network using VNFs and SDN. The design considerations and results are shown in Chapter 3. Chapter 4 presents detailed research approaches and challenges to resolve for design and implementation of a security protocol suit to provide authenticity and integrity. Chapter 5 further demonstrates a verifiable IoT data management protocol suit that provides data consumers with the ability to retrieve data on customized searching criteria. This feature is not supported in the security protocol described in Chapter 4. Finally, the summary and future work are presented in Chapter 6.

Chapter 2

Network Function Placement

In this chapter, different NF frameworks and their placement issues are discussed. This chapter first reviews hardware NF placement problem in Section 2.1. Some recent advances in NFV are provided in Section 2.2, followed by irregular forms of NFs described in Section 2.3. Some NF placement-related discussions are presented in Section 2.4.

2.1 Hardware Network Function Placement

Until recently, most NFs were shipped in dedicated hardware which are deployed in certain vantage points in the network. Different types of hardware NFs have their own placement considerations and thus different placement strategies, as summarized in Table 2.1.

2.1.1 Independent Network Function

For some NFs, one instance of each NF can work independently to perform a task, without explicit interaction with other instances. For instance, the passive traffic monitor which taps into a communication link falls in this category. The placement strategies for this kind of NFs try either 1) to maximize the utility under fixed budget or 2) to minimize the total cost with a certain level of service guarantee. The research on placing passive traffic monitors has been extensively studied in literature [165, 76, 79, 150]. Particularly, Suk *et al.* formulated two integer linear programs aiming to maximize the fraction of traffic being monitored and to minimize total cost respectively [165]. The optimization variables include the number of monitors, their optimal positions and their sampling rates. This solution does not change the traffic routing paths. However varying traffic matrix renders the previously optimal solution sub-optimal. MeasuRouting [150] addresses this problem by strategically routing traffic to fixed monitoring points allowing moderate traffic path alteration.

2.1.2 Chained Network Functions

Some network traffic may be required to traverse through a given set of NFs in sequence, which is referred as service function chaining, or policy chaining. For example, for security purposes all http traffic should follow the service function chain: $firewall \rightarrow IDS \rightarrow proxy$. Existing solutions [115, 177] achieve service function chaining by leveraging traffic steering. PLayer [115] is a shim layer of network switches that explicitly forwards network traffic through its associated NFs in sequence to fulfill the requirement of service function chaining. PLayer does not use in-line processing fashion to avoid traffic being processed by unrelated NFs. Consequently, off-line processing is utilized, which means traffic is explicitly forwarded to NFs plugged into switches for processing. The latency due to the one-hop indirect path is negligible in datacenter networks. Inheriting the core idea of PLayer to achieve service function chaining by steering traffic to fixed NFs, the SDN-based solution StEERING [177] allows fine-grained per-subscriber

Table 2.1: Comparison between independent NFs & chained NFs.

NF type	Location	Traffic steering	Placement objective
Independent NFs	in-line	optional	max cov./min cost
Chained NFs	off-line	compulsory	min latency

and per-application service function chaining through centrally configured forwarding rules on SDN switches. With the aid of flexible traffic steering to achieve service function chaining, network operators are free to place NFs anywhere in the network. Nevertheless, the placement strategy impacts the user performance. StEERING provides one placement strategy aiming to minimize the traffic delay.

Both PLayer and StEERING do not consider the problem of NF mangling of service function chaining, which means some NFs such as NATs and web proxies, actively modify the traffic header or the payload. A packet may be not trivially recognized after mangling NFs.

2.2 NFV Orchestration and Placement

The new paradigm of NFV enables the great flexibility in terms of deployment, instantiation, configuration, and termination of virtual NFs on demand. Recently, numerous NFV frameworks have been proposed to efficiently manage the hardware resources. Each NFV framework comes with its own appropriate placement strategy. Table 2.2 compares different NFV frameworks and summarizes the impact of their the design choices on the placement [1].

[1]In the case of CSamp, it does not address service function chaining. Therefore, the properties of order preserving and mangling NFs are not listed for CSamp

Table 2.2: Comparison between different NFV frameworks

NFV form	NFV framework	Placement strategy	On path?	Mangling?	Location dependent?	Order preserve?
Thread-based	CoMb [156]	Monolithic consolidating	✓	✓	✗	✓
	MIDAS [54]	Cross-border placement	✓	✗	✓	✓
VM-based	E2 [138]	Path loosely-controlled	✗	✗	✗	✓
	Statos [97]	Path loosely-controlled	✗	✓	✗	✓
	VNP-OP [64]	Path tightly-controlled	✗	✓	✗	✓
	PACE [121]	Policy unordered	✗	✗	✗	✗
Irregular Forms	Slick [58]	Partial consolidating	✓	✗	✓	✓
	CSamp [159]	On-path distributed	✓	N/A	✓	N/A
	ETTM [91]	Monolithic consolidating	✓	✓	✗	✓

2.2.1 Thread-based Framework

Running software NFs as threads at generic hardware platforms enables resource multiplexing as well as components reuse.

Monolithic Consolidating

CoMb [156] is a representative of such NFV frameworks which consolidates multiple modular VNFs in a single server. To avoid duplicated functions performed at different modular VNFs, all incoming packets are first processed by functions shared by all CoMb VNFs, such as protocol parser and session reconstructor. CoMb then sends packets to their corresponding thread-based VNFs for further processing. The authors of CoMb assume that the resource footprint of each thread is proportional to its workload. CoMb [156] consolidates all the required NFs in a single thread for each traffic class. Such monolithic consolidating obviates the need for addressing mangling NFs, which drastically eases the management of the service function chaining.

Cross-border On-path Placement

Monolithically consolidated NFs may fall short in some cases. A service function chain may contain NFs that have preferred locations in the network. For example, web proxies are better to be placed closed to the end clients to reduce bandwidth consumption and network latency.

To allow service function chains to be accomplished by NFs distributed at different servers along the traffic routing path, MIDAS [54] is proposed to extend the scope of CoMb. MIDAS resolves the problems of CoMb servers discovery and on-path processing establishment across multiple NF service provider. MIDAS would be particularly helpful in the Internet, where location dependency impacts the performance significantly.

2.2.2 VM-based Framework

Isolation is an indispensable property for multi-tenant clouds, where the resources are shared among cloud tenants. Without isolation, busy VNFs may consume much more resources than other co-residing VNFs. In this case, QoS cannot be guaranteed. Most NFV framworks leverage virtual machines (VMs) as the container for isolation. The resource usage model is simplified in most VM-based frameworks such that one instance of VNF consumes fixed resources and possesses fixed processing capacity. Since VMs consumes substantial hardware resources, it is prohibitively expensive to consolidate NFs like CoMb for each traffic class to achieve service function chaining. Instead, VM-based frameworks steer traffic flows between VNFs to fulfill service function chaining. VM-based frameworks differ from traffic steering for hardware NFs in that the placement of VNFs could be decided at run-time, hence more freedom to optimize the performance and cost. Moreover, traffic steering could be augmented by other mechanisms to incorporate traffic dynamics, such as flow distribution (small time-scale load balancing), dynamic scaling in/out (resource efficiency), *etc.*.

Flowstream [104] is the first VM-based framework. The authors of Flowstream noticed the narrowed performance gap between commodity switches/servers and customized high-end switches/hardware NFs, and successfully predicted the rise of commodity general-purpose hardware to process traffic. Nevertheless, Flowstream is just a high-level idea without concrete design and implementation.

Path Loosely Controlled Placement

There are existing network-aware placement algorithms [127, 122] resolving the problem of mapping VMs onto physical machines in the cloud in such a way that the communication cost is optimized.

E2 [138], a VM-based NFV framework, relies on them to minimize intra-server traffic when mapping VNF instances to a servers. The NF placement of E2 involves four steps. Firstly, merge individual service function chains into a *single policy graph (pGraph)*. Next, E2 determines the number of instances for each NF in the pGraph given the estimation of the load on a NF and the per-instance capacity. The workload of a NF will be evenly distributed among its VNF instances. The third step is converting the pGraph to an *iGraph*, or the *"instance graph"*. In the iGraph, each node represents an instance of one NF and the edge weight captures the traffic demand between the associated two VNF instances. Finally, E2 computes an optimization problem with the objective to minimize inter-server traffic.

Statos [97] is another network-aware NFV platform being able to correctly forward traffic in face of mangling NFs. Statos exhaustively enumerates the possible downstream paths for each mangling NF in the policy graph to ensure correctness of service function chaining.

Path Tightly Controlled Placement

With emerging SDN technologies, some NFV frameworks could flexibly control the routing paths between NF pairs, enabling joint placement and routing optimization.

VNP-OP [64] attempts to minimize the cost, penalty for SLO violation and resource fragmentation by jointly optimizing placement of VNFs and traffic routing paths. This optimization is formulated as an Integer Linear Program (ILP) considering resource capacity constraints and service function chaining.

Charikar et al. first theoretically analyzed this joint optimization problem by recognizing it as a new extension to multi-commodity flow problem [78].

Unordered Placement

The aforementioned VM-based NFV frameworks all strictly preserve the sequential order specified by the service function chain. In some cases, the service function chain however can be partially ordered or even completely unordered. For instance, from the point view of security, there is little difference to put a traffic monitor before or after a DPI. PACE [121] addresses the VNF placement for unordered service function chains in cloud, with the objective to maximize the number of satisfied requests.

2.3 Irregular Forms of Network Functions

2.3.1 Element-based Framework

The aforementioned NFs, either hardware, thread-based or VM-based, which vertically integrate basic modules can independently complete a particular packet processing task like packet parsing. Slick [58] takes another approach which implements NFs as a chain of lightweight functions (e.g, checksums) across the network. These lightweight functions, referred as elements in Slick, are able to be reconfigured at runtime.

Element placement consists of two steps. The first step is to consolidate elements. The Slick controller decide whether to consolidate contiguous elements onto a single machine or distribute them across multiple machines based on each element's inflation factor, which is defined as $log(f_{out}/f_{in})$ where f_{out} and f_{in} are denoted as its output and input traffic volumes respectively. The second step is to place consolidated elements. The placement strategy is that consolidated elements with negative (positive) inflation factors are placed onto the node closed to sources on the longest common path of all traffic (destinations). The intuition is that placing elements with negative inflation factors near the sources can reduce the link bandwidth consumption.

2.3.2 Distributed NFs

CSamp [159] targets the scalability problem of fined-grained flow level monitoring by employing distributed coordinated NFs. In CSamp, the traffic monitoring modules are implemented as applications inside routers. To avoid duplicated traffic sampling on the routing path, CSamp uses a hash-based packet selection to achieve the implicit coordination. After distributed NFs are placed in the network, a network-wide goal, such as maximizing the coverage, is achieved by distributing the workload to monitor NFs across the network while respecting the resource constraints.

Apart from distributed monitoring, distributed redundant elimination [57] and distributed IDS/IPS [158] have been proposed on the same idea.

2.3.3 Host-based Framework

The NFs which have discussed are all residing inside the network. ETTM [91] are proposed to move NFs onto endpoints to exploit the increasing computing power with multi-core commodity servers. The NFs running on VMs residing in the endpoints, but they are logically controlled by a central controller. ETTM benefits from the trusted computing module (TPM) available in many current computers to provide NFs with the Attested Execution Environment (AEE). These distributed NFs residing in endpoints use Paxos [118] distributed consensus algorithm to provide consistent decisions, fault-tolerance and reliability. Since the traffic is only processed at endpoints, similar to CoMb [156], the required NFs in the service function chain are consolidated at the endpoints. The abundant resources in each endpoint are enough to support its own service function chains .

2.4 Network Function Placement Discussions

Can different forms of NFs be mixed together?

The hybrid environment has already been proposed in literature. VNF-P [129] focuses on a scenario where the service function chaining is provided by both fast dedicated hardware NFs for baseline workload as well as flexible on-demand VNFs for burst workload.

Can NFs be outsourced?

Sherry et al. proposed APLOMB [161], a service to oursource NFs in cloud without sacrificing performance. By leveraging DNS-based redirection, the incoming traffic of an APLOMB customer is routed directly to the cloud first for outsourced NF processing, then to the customer. The latency due to the indirection is negligible.

Where do service function chains come from?

PGA [147] answers this question in the realm of enterprise network. PGA is a fast automation tool to compose independent network policies into a global-scope one for each traffic class. PGA allows the detection and resolving of the conflict, if there is any.

2.5 Conclusion

In this chapter, a comprehensive survey is provided regarding the placement issues of both conventional hardware network functions and virtualized network functions. Different NF frameworks as well as how their design considerations impact the network function placement strategies are presented.

Chapter 3

Interference-free Policy Enforcement for NFV

As mentioned in Section 1.1.2, order preserving, interference freedom, and genericity, are desirable for NFV orchestration frameworks. However, none existing solutions can satisfy all of them. This chapter present the design and implementation of a novel SDN-based NFV orchestration framework, called APPLE (proActive aPProach for poLicy embEdding), which satisfies the aforementioned properties simultaneously. The core idea of APPLE is to distribute NFs on the unmodified traffic path to enforce service function chain policies, which does not belong to any placement strategy described in Chapter 2.

APPLE strictly preserve the order of policies. APPLE estimates the NF demand of network-wide flows and proactively installs VNF instances on traffic paths for potential flows to achieve interference freedom. VNF instances are contained in VMs or containers for the sake of genericity. Moreover, VMs or containers are able to separate fault domains which is especially useful for multi-tenant clouds where hardware resources are shared by multiple entities.

3.1 Prior Work

None existing orchestration framework can simultaneously satisfy all the requirements.

Policy enforcement of hardware NFs mostly relies on traffic steering. Two typical works are StEERING [177] and SIMPLE [148]. Both of them use the SDN technology to forward flows by assigned paths.

CoMb [156] and MIDAS [54] consolidate multiple VNF instances as threads in a physical machine. However, resource and fault isolation are not considered in these two orchestration frameworks which limits their applicable scenarios. NetBricks [139] solves the problem of isolation by taking the advantage of safe languages (e.g. Rust). However, such thread-based NFV framework also comes with the price of not being generic. The host network functions need to be modified to adapt their specific programming models.

VM/container-based NF orchestration frameworks are more widely adopted due to its generacity. PACE [121] proposes to use smart VM placement to deploy NFs. However PACE does not consider service function chains. Merge/split [149] accelerates the process of flow migration for stateful NFs by transferring the states associated with migration flows instead of the whole VM. OpenNF [98] follows the same idea. However, both frameworks require the modification to NFs. Stratos [97] and E2 [138] provide efficient and scalable NF provisioning by combining traffic engineering and NF placement. They utilize traffic steering to enforce policies and hence are not interference-free. This situation is true for SoftCell [112] as well, which focuses more on the packet processing in the mobile access network.

There are also irregular forms of NFs and their associated orchestration frameworks. Eden [62] is proposed to move the NFs onto participating endpoints to

Table 3.1: Comparison of NF orchestration frameworks

NF Form	**Framework**	**Policy Enforcement**	**Interference Freedom**	**Genericity**
Physical	StEERING [177]	✓	✗	✓
	SIMPLE [148]	✓	✗	✓
Thread-based	CoMb [156]	✓	✓	✗
	MIDAS [54]	✓	✓	✗
	NetBricks [139]	✓	✓	✗
VM/Container	PACE [121]	✗	✓	✓
	Merge/Split [149]	✗	✓	✗
	Stratos [97]	✓	✗	✓
	E2 [138]	✓	✗	✓
	SoftCell [112]	✓	✗	✓
Irregular forms	Eden [62]	✓	✓	✗
	OpenBox [74]	✓	✓	✗
	Slick [58]	✓	✓	✗

exploit the their increasing computing power. ETTM [91] is another host-based NFV orchestration framework. OpenBox [74] and Slick [58] separate the control and data planes of NF packet processing. These NF orchestration frameworks bear the same problem with thread-based ones: not generic and cannot reuse existing NFs directly without modification.

For these existing orchestration frameworks, the comparison is summarized in Table 3.1.

3.2 APPLE System Overview

Network Model. APPLE uses the SDN paradigm [126]. All physical nodes that host VNF instances are connected to one of the SDN-enabled switches. When a flow needs to be processed by a VNF, forwarding rules installed on the switch will guide the packets of the flow to the VNF instance and continue forwarding after receiving the packets again from the VNF instance. A central controller obtains network information and installs forwarding rules onto switches via standard APIs such as OpenFlow [126].

The overview of APPLE is shown in Figure 3.1.

APPLE host. Each physical node hosts multiple VNF instances, which is also called an APPLE host. A virtual switch (vSwitch), such as Open vSwitch [29], is installed in the node to switch packets to different VNF instances, which is also connected to the outside network.

VNF Instance. VNF instances run as VMs on physical nodes. VMs guarantee the CPU and memory resource isolation.

Optimization Engine. It is a traffic-aware VNF placement algorithm. It runs periodically to make adjustments according to the large time-scale network dynamics. It takes the traffic rate, forwarding path, and service function chain of each flow, together with the available hardware resources, as input. It computes the proper placement of VNF instances and the particular VNF instances each flow is supposed to traverse. The objective functions of the Optimization Engine are defined by other modular control programs. Optimization Engineer is also able to resolve conflicts between different control programs. The Optimization Engine interacts with the Resource Orchestrator to install VNF instances accordingly and obtains the information about available resources at APPLE hosts from the Resource Orchestrator. It sends the information about how new flows are assigned to different VNF instances to Rule Generator for the computation of data plane forwarding rules.

Resource Orchestrator. It allocates sufficient resources and launches VNF instances according to the result of Optimization Engine. In addition, it monitors the available resource on APPLE hosts and reports this information to Optimization Engine.

Rule Generator. It gathers the outputs from Optimization Engine and generates the data plane forwarding rules. These rules are installed at physical and virtual switches through the SDN API.

Dynamic Handler. It may receive an overloading notification from SDN monitoring applications. It will then re-balance the workload to resolve overloading by requesting the Rule Generator to install new forwarding rules and possibly by instantiating additional light-weight VMs (they are ClickOS instances in our prototype implementation) through Resource Orchestrator.

Local Agent. It can either be a dedicated host or a VM inside one APPLE host. Local Agent is used to detect new flows for the sake of *flow affinity* when some traffic is migrated to other NFs. For switches supporting new standard of SDN API (e.g. P4 [33]), they can match on the SYN flag in the TCP packet to directly detect new flows. In this case, the functionality of local agent can be replaced by these switches.

3.3 Design Challenges

There are several challenges involved in the design of APPLE: (1) There are different and possibly conflicting strategies to place VMs on the substrate infrastructure. It is challenging to satisfy many resource allocation goals while enforcing policies. (2) Traffic is highly dynamic. APPLE should avoid both under-provision during peak load and over-provision during base load to strive for a balance of performance and resource-efficiency. One specific challenge here for stateful VNFs is that *flow affinity* needs to be preserved during load balancing, which means all packets of each flow are required to traverse the same VNF instances. (3) Proper forwarding rules are required to install in SDN switches and vSwitches in the APPLE hosts to accomplish service function chaining. The challenge is how to efficiently use scarce forwarding table entries to fulfill this task.

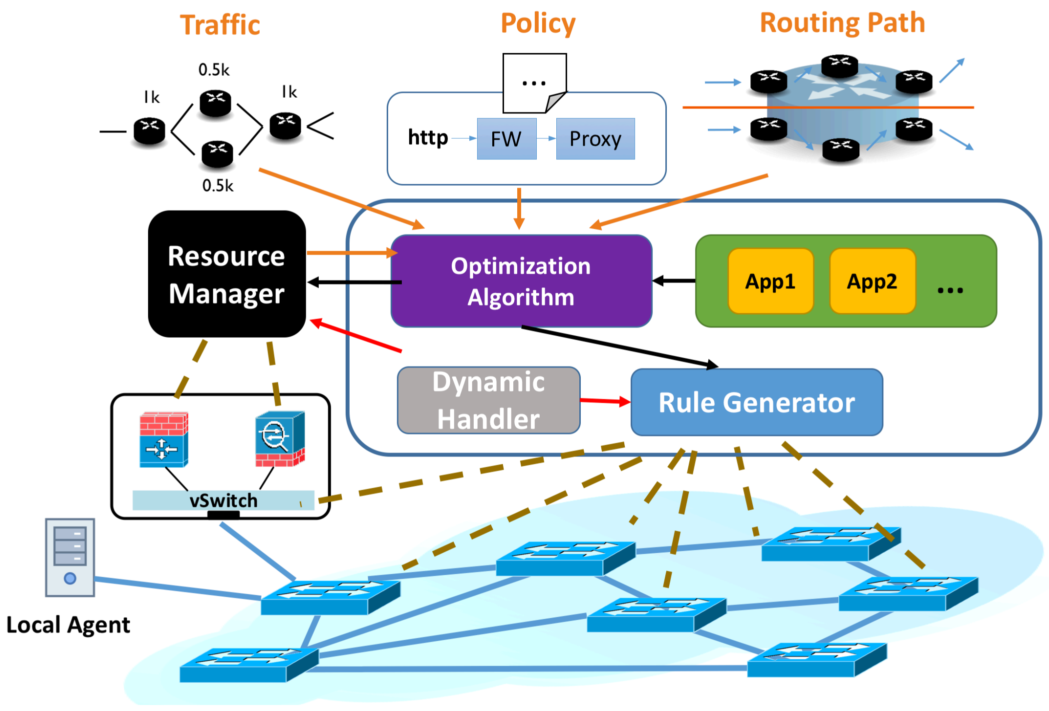


Figure 3.1: Overview of APPLE

3.4 Optimization Engine

The functionality of Optimization Engine is to find a NF placement plan while still respecting the various resource constraints. Most of existing NF resource management strategies tightly couple path selection and NF placement which impede continuous improvements and an open ecosystem. To this end, the most important design choice of Optimization Engine is to support modularity while policy enforcement is still preserved. Conflict resolution is essential for modularity. There are existing works trying to address the conflicts among modules for path forwarding [146] and resource allocation [59, 60] respectively. Nevertheless, the integration of the two branches remains unexplored.

The high-level idea of conflict resolution for a combination of path selection and resource allocation is depicted in Figure 3.2. FortNOX [146] is leveraged for path selection conflict resolution. However, FortNOX operates on low-level Open-

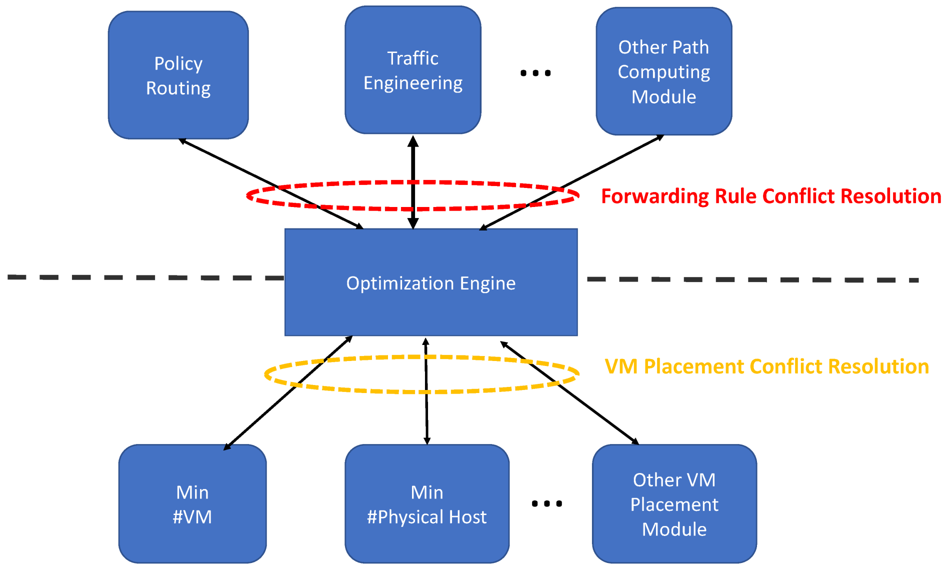

Policy Routing
Traffic Engineering
...
Other Path Computing Module
Forwarding Rule Conflict Resolution
Optimization Engine
VM Placement Conflict Resolution
Min #VM
Min #Physical Host
...
Other VM Placement Module

Figure 3.2: Optimization Engine decouples path selection and resource allocation.

Flow rules and therefore the routing path for each flow is not directly provided. A recently developed tool AP-Classifier [175] can help to efficiently compute the forwarding path for each network flow based on low-level OpenFlow rules. For each network flow, the virtualized NFs are installed on the resulting forwarding path to meet the requirement of policy enforcement. On the other hand, Optimization Engine provides flexibility by allowing multiple modular modules to jointly control how resources are allocated on the forwarding path. Resource allocation is mathematically model as an optimization problem which jointly optimizes different and even conflicting goals of various resource allocation modules. Optimization Engine ensures that other resource constraints (e.g. bandwidth, host computation/storage resources, etc.) are not violated in the meantime. Optimization Engine applies global optimization that computes a VNF placement plan for all current flows or online placement for any new flows.

3.4.1 Traffic Aggregation for Scalability

Optimization Engine of APPLE determines VNF placement driven by service function chainins for all traffic flows. Kandula et al. [116] found that 100K flows arrive every second on a 1500-server cluster. APPLE aggregates traffic into *equivalence classes*. The flows having the same path and policy chain are aggregated into a class. The input size of Optimization Engine and thus the time to solve the optimization problem can be reduced significantly.

In this chapter, each class is denoted as $h \in H$, where H is the set of all classes. The recently developed atomic predicate based analysis tool [170] is used to classify flows into equivalence classes. Detailed explanation can be found in [170].

3.4.2 Spatial Distribution

With a centralized network-wide view, APPLE spatially distributes the workload such that the responsibility of each VNF instance can be more balanced. Load distribution is essential to process jumbo classes whose traffic rates are beyond the capacity of one single VNF instance.

3.4.3 Mathematical Model Objectives and Variables

Objectives. Optimization Engine moderates multiple resource allocation modules. The possible objective of one module is to minimize the resource utilization for policy enforcement in order to make more room for production VMs co-located in Apple hosts. Another modules may aim at minimizing the number of running physical hosts for energy efficiency reasons. As a result, there is no fixed objective function for the mathematical model. The resource allocation module is required to have an evaluation function $f(S)$ to interact with Optimization Engine, where S represents the *resource allocation state*. Resource allocation state includes the resource reservation information for all Apple hosts. The interface $f(S)$ returns a value in range $[0, 1]$ representing a rating. A smaller value indicates higher preference. In order to summarize all resource allocation modules, the objective function is to minimize the weighted sum of evaluation functions. Note that the detailed implementation of $f(S)$ is not restricted to a liner function.

VNF capacity. The capacity of an instance of VNF n is denoted as Cap_n, which captures the number of packets being processed per second. APPLE can measure the capacity of each VNF instance in advance, which is one-shot effort.

Available Resource. Each VNF instance occupies hardware resources in its APPLE host. Optimization Engine needs to ensure there are enough hardware resources to launch new instances. The available hardware resources of all APPLE hosts connected to switch v is represented as A_v. Note that A_v is a vector, where each element denotes a type of hardware resource. Optimization Engine acquires such information from Resource Orchestrator. Likewise, R_n is to denote a resource requirement vector in which each element is the requirement of a type of resource of VNF n.

Service function chains. Service function chains are determined by some external mechanisms. They specify the sequence of NFs that each class of flows need to traverse in order. $C_h =< c_h^j >$ represents the service function chain for class $h \in H$, where c_h^j means the the jth NF on the policy chain C_h.

Flow Paths. Forwarding paths are provided by pipelining existing SDN rule-based confilicts resolution tool [146] and forwarding path analysis tool [170]. We use $P_h =< p_h^i >$ to donate the path, where p_h^i is the ith switch class h encounters.

Traffic Rate. It can be estimated by other applications [95]. T_h captures the traffic rate of class $h \in H$.

Table 3.2: Notations in the optimization problem.

Notations	**Explanation**
p_h^i	i-th switch on the path of class h
c_h^j	j-th VNF on the policy chain of class h
$d_{h,j}^i$	portion of class h processed in c_h^j connected to p_h^i
$\sigma_{h,j}^i$	cumulative portion of class h processed in c_h^j until p_h^i
T_h	traffic volume of class h
Cap_n	process capacity of VNF n
q_n^v	quantity of VNF n connected to switch v
l_v^v	existing traffic load on VNF n connected to switch v
R_n	the resource requirement vector of VNF n
A_v	available resource of APPLE host connect to switch v
$f_k(\cdot)$	evaluation function for kth resource allocation module
S	resource allocation state
w_k	weight for kth resource allocation module
$\|P(h)\|$	path length of class h
$\|C(h)\|$	number of VNFs at the policy chain of class h
$i(P,h,v)$	index of switch v on the path of class h
$i(C,h,n)$	index of VNF n at the policy chain of class h

The network topology is represented by a graph $G = (V, E)$, where V and E are the set of switches and links in the network respectively. Let N denote the universal set of all VNFs. The optimization variable $d_{h,j}^i$ determines the portion of traffic of class h to be processed in instances connected to switch p_h^i for VNF c_h^j. Another optimization variable $q_v^n \in \{0, 1, 2, ...\}$ quantifies the number of VNF instances instantiated for VNF n connected to switch v. $\sigma_{h,j}^i$ is a derived variable, which means the cumulative portion of traffic that has been processed, from beginning to p_h^i on the network path for VNF c_h^j for class h. $|P_h|$, $|C_h|$ are the length of P_h and C_h, respectively. $i(P, h, v)$ gets the index of switch v on the sequence $P_h =< p_h^i >$. Likewise, $i(C, h, n)$ is the index of VNF n on the sequence of $C_h =< c_h^j >$. The notations used in this optimization problem is listed in Table 3.2. The optimization formulations are stated as follows.

$$Minimize \qquad \sum_k w_k * f_k(S) \tag{3.1}$$

s.t.

$$\sigma^{i}_{h,j} = \sigma^{i-1}_{h,j} + d^{i}_{h,j} \qquad \forall i, j \tag{3.2}$$

$$\sigma^{i}_{h,j-1} - \sigma^{i}_{h,j} \geq 0 \qquad \forall i, j \tag{3.3}$$

$$\sigma^{|P(h)|}_{h,|C(h)|} = 1 \tag{3.4}$$

$$\sum_{h:v\in P_h} T_h d^{i(P,h,v)}_{h,i(C,h,n)} + l^{v}_{n} \leq Cap_n \times q^{v}_{n} \qquad \forall v, n \tag{3.5}$$

$$\sum_{n\in N} R_n \times q^{v}_{n} \leq A_v \qquad \forall v \tag{3.6}$$

$$0 \leq d^{i}_{h,j} \leq 1 \qquad \forall i, j \tag{3.7}$$

$$q^{v}_{n} \in \{0, 1, 2, ...\} \qquad \forall n, v \tag{3.8}$$

Eq. (3.3) makes sure that the order of service function chain is preserved. Eq. (3.4) assures that 100% traffic of class h are processed by the service function chain. Eq. (3.3) and Eq. (3.4) collectively define the requirements of service function chaining. VNF capacity limits and resource constraints are captured by Eq. (3.5) and Eq. (3.6). Even though $i(P, h, v)$ and $i(C, h, n)$ in Eq. (3.5) seem to make the optimization problem nonlinear, actually once the input is given, the values of $i(P, h, v)$ and $i(C, h, n)$ are known immediately, without solving the whole optimization problem.

This optimization problem is an Integer Linear Program (ILP) when the objective function is also a linear function. This optimization problem can be reduced from *Set Cover Problem*, which is known to be NP-hard. Furthermore, Dinur et al. [90] have proved that Set Cover Problem cannot be approximated within $\Big(1 - o(1)\Big) \cdot \ln n$, unless $P = NP$, where n is the number of subsets. Therefore, a heuristic based on simulated annealing is proposed in this chapter to solve this problem.

3.4.4 Simulated Annealing (SA)-Based Heuristic

Simulated Annealing (SA) is utilized to compute the VNF placement plan for Optimization Engine. Simulated Annealing is an optimization problem solving framework, where the user needs to customize some variables and functions. The following content briefly introduces some important concepts and the corresponding customization.

State. State describes the values of all variables in the optimization problem. For the case of VNF placement, the state is about the number of each type of VNF installed in each Apple host as well as where the policy chain is enforced for each equivalent class.

Energy Function. This function summarizes how "good" the state is. For Optimization Engine, it only focuses on resource allocation. It shares the same idea of objective functions in optimization problems. Therefore, Optimization Engine reuses the objective function Eq.(3.1) as the energy function of Simulated Annealing.

Neighbor Generator Function. This function generates a new feasible state from current state. To be more specific regarding APPLE, this function randomly deletes one existing installed VNF or randomly installs a VNF instance of a random type to a randomly chosen Apple host. In the meantime, the newly generated state is checked whether it still meets the requirement of policy enforcement and various resource constraints. If the two requirements are not satisfied, the neighbor generator function keeps on proposing new states until the two requirements hold.

The customization of State and Energy Function is trivial. However, Neighbor Generator Function requires further discussion. By design every newly accepted state is forced to meet the constraints. Once one flow deviates from these two requirements, it is hard to cater the needs of policy enforcement again, because it is unlikely to randomly allocate proper VNF types on its path when the network topology is complex. Another key question is how packet processing positions (i.e. $d^i_{h,j}$ and $\sigma^i_{h,j}$) change with respect to the new NFV placement plan. When a new VNF instance is added to the network, $d^i_{h,j}$ and $\sigma^i_{h,j}$ do not change since all constraints still hold. However, in the situation where one VNF instance is deleted, the traffic load imposing on that VNF instance should be shifted to other VNF instances. During the migration, the policy enforcement and bandwidth constraints (Eq.(3.2) - Eq.(3.5)) should be taken into consideration. Since all constraints are continuous and linear, it is computational efficient to find a feasible solution.

3.5 Enforcing Optimization Results

APPLE needs to enforce the optimization decisions made by Optimization Engine. The installation of VNF instances at APPLE hosts are handled by Resource Orchestrator as ordinary VMs. However, it is non-trivial to enforce the forwarding rules from the result of spatial distribution $d^i_{c,j}$ directly. Thus, a new concept, *sub-class*, is introduced to help APPLE generate forwarding rules.

3.5.1 Sub-class

Note that the policy enforcement is on per-flow basis, even though Optimization Engine solves the optimization problem in the scale of classes. All packets of a flow are required to traverse the same stateful VNF instances, which is referred as *flow consistency*. Therefore, a new concept, *sub-class*, is defined which is the aggregation of flows within a class that traverse the same VNF instances. d^s_c represents the fraction of sub-class s in class c. Clearly, $\sum_s d_c^s = 1$. The responsibility for each sub-class is accepted as long as the space distribution from Optimization Engine is satisfied. One way is to identify sub-class is to split the rule space. For example, if $< 10.1.1.0/24 >$ is a class, a sub-class contributing 50% of the class could be $< 10.1.1.128/25 >$. However, one problem occurs when APPLE changes the spatial distribution of a class due to network dynamics: changing the assignment of VNF instances for existing flows may violate flow affinity. The problem is resolved by a novel data plane scheme called flow tagging, presented in the following subsections.

3.5.2 Tagging Scheme

A tag is an identifer written to a packet header. The header modification could be easily customized in SDN-enabled switches. The unused bits in the packet header can be used as the tag field, such as the 6-bit DS field and 12-bit VLAN ID (if VLANs are not used). APPLE currently cannot support VNFs that dynamically modify the packet header (e.g. NAT).

The main idea of the tagging scheme is to avoid duplicated classifications on physical switches, which consume substantial TCAM resource. Current physical SDN-enabled switches implements forwarding table in TCAM. In the APPLE tagging scheme, the tag consists of two fields. One field specifies the *host ID* which is the next host to process this packet. If one packet has traversed all the required VNF instances, this tagging field is Fin. The other field encodes *sub-class ID* within a class. Note that sub-class ID can be multiplexed by different classes. The Sub-class tagging field remains unchanged in the network. Figure 3.3 shows the flowchart on how to process a packet arriving at a physical SDN switch. Upon reception of a new packet, the switch checks the host ID field. If host ID

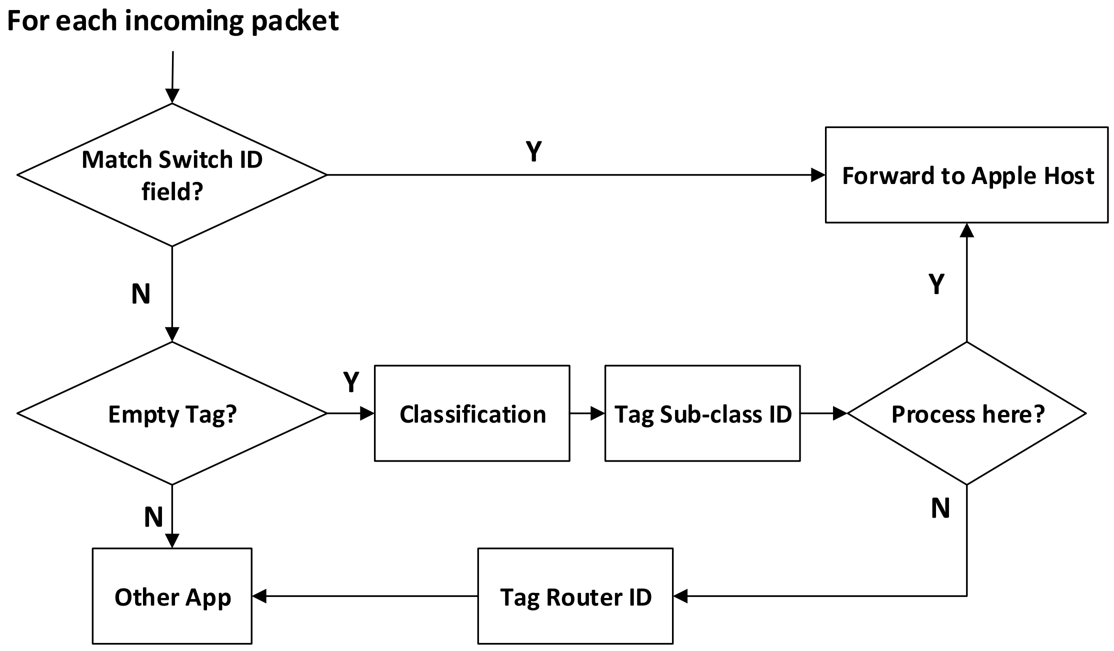


Figure 3.3: Data plane framework at an SDN switch.

indicates one APPLE host connected to the switch, it would forward the packet to the APPLE host for NF processing; otherwise, the switch would forward the packet to the correct next hop. If this field is empty, it means that this packet just entered the network and the switch should tag a sub-class ID first. After that, if the packet is to be processed in any APPLE host connected to the switch, the switch should forward it to the APPLE host.

Such semantics can be easily encoded in TCAM, when flow table pipelining is supported. Table 3.3 illustrates the TCAM layout at the physical switch. Rules of other applications are stored in the next table. Here, the sub-class matching rules are a bunch of wildcard rules to achieve the target distribution computed by Optimization Engine. For switches not supporting pipeline processing, the semantics can still be retained by the cross-product of the two tables, but the TCAM consumption would increase. Note that the classification rules are just installed at the corresponding ingress switch for each sub-class to reduce TCAM

Table 3.3: Layout of TCAM at physical switches

Type	**Switch ID field**	**Match**	**Action**
Self match	Self switch ID	*	Fwd to APPLE host
Classification	Empty	Sub-classes	Tag sub-class ID, Fwd to APPLE host
	Empty	Sub-classes	Tag sub-class ID, Tag switch ID, Go to next table
Pass by	*	*	Go to next table

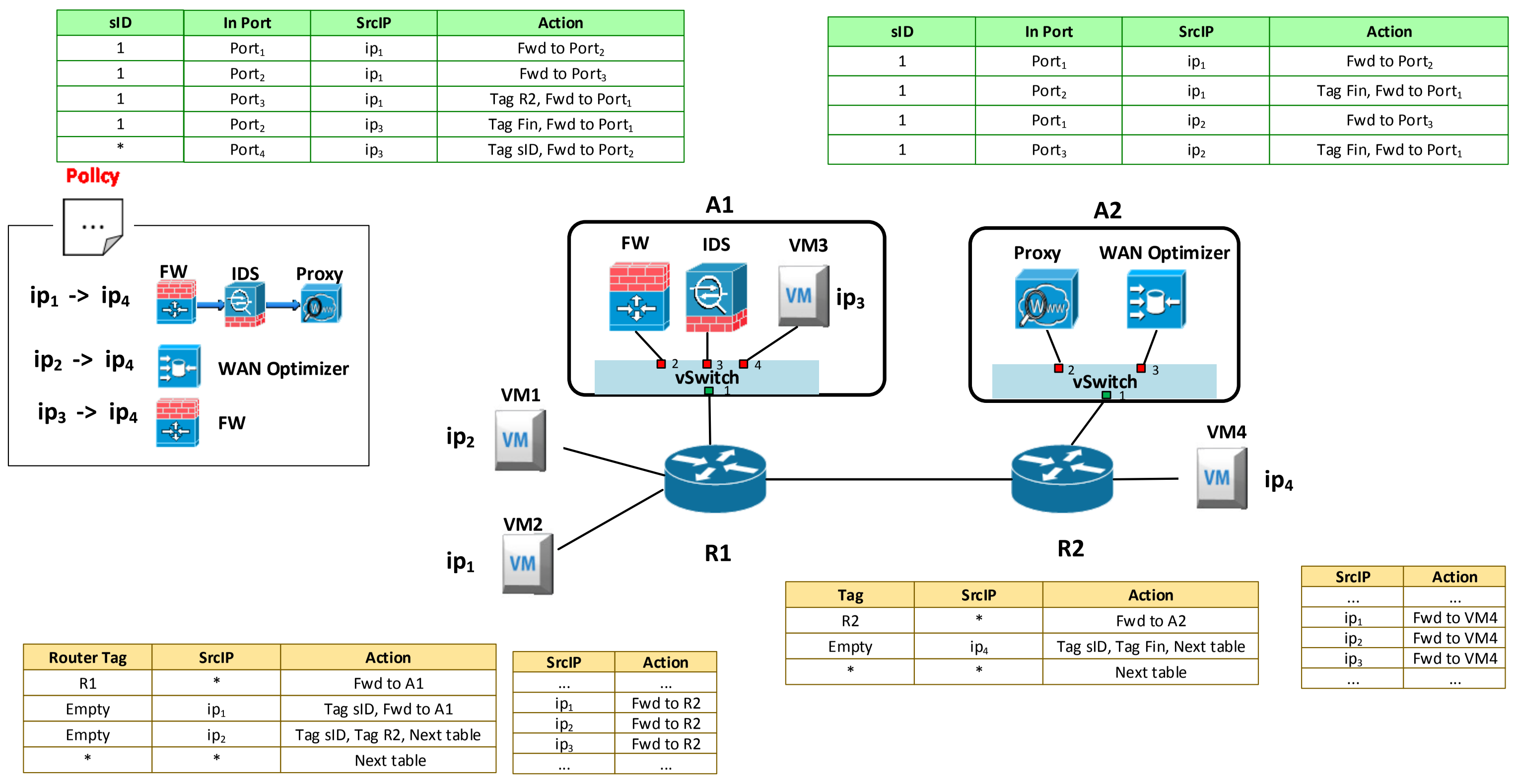
consumption.

Forwarding rules are also needed in vSwitch embedded in APPLE hosts to direct packets to desired VNF instances. The matching rule is based on three tuples, <IncomePort, class, sub-class>, where class is specified by matching rules. A packet may traverse multiple VNF instances in one APPLE host, and IncomePort is enough to identify which VNF instances the packet has traversed. (This dissertation assumes that a packet does not traverse a same instance twice.) The combination of class identifier and sub-class IDs distinguishes different sub-classes. APPLE does not directly match sub-classes, because sub-classes matching rules change occasionally to adapt to the new spatial distribution. Hence it is an effective way to avoid VNF inconsistency caused by rules updating, which violates flow affinity.

When the packet leaves an APPLE host, it also needs to be tagged to indicate the next APPLE host to process it. Since production VMs can also be installed in APPLE hosts, the vSwitch adopts a similar tagging scheme and the processing pipeline to store rules of other applications. One difference is that IncomePort is enough to distinguish whether a packet has been tagged or not: the packets from the ports connect to production VMs are not tagged yet. Figure 3.4 gives three common scenarios for tagging scheme. There are 3 classes, and they have the same path, $S1 \to S2$. The classes can be distinguished by the srcIP field. Each class only has only one sub-class (denoted as sID in the figure). The traffic $ip_1 \rightarrow ip_4$ represents the scenario where the packets traverse multiple APPLE hosts. The traffic $ip_2 \rightarrow ip_4$ represents the scenario where the packets are processed in APPLE hosts not connected to the ingress switch. The traffic $ip_3 \rightarrow ip_4$ represents the scenario where the packets originate within an APPLE host.

3.5.3 Flow Affinity Retaining

When the distribution of a class changes, the matching rule for each sub-class may adjust accordingly, which in turn changes the sub-classes that some flows belong to. Statos [97] retains flow affinity by installing a new exact microflow rule in the virtual switch everytime a new flow is encountered. This method does not work for APPLE because TCAM in physical switches do not possess enough space to host all microflows [130]. To this end, a new method to retain flow affinity is proposed in this chapter. The main idea of this method is to use *SYN* packets as the indication of new flows and only new flows are allowed to be classified into new sub-classes. *FIN* flag on the other hand is leveraged to detect the termination of flows. However, current OpenFlow switches cannot perform matching of TCP flags. Hence the controller installs higher-priority rules at OpenFlow switches to forward all incoming packets of affected sub-classes to a local agent to check their TCP flags. In this local agent, packets with a SYN flag would be tagged with the new sub-class IDs and then are sent back to the OpenFlow switch. On the other hand, the local agent also records which flows have switched to the new sub-classes, so that their following packets can also be tagged with new sub-class IDs. For other packets, the local agent only gives them the old sub-class IDs in order to retain flow affinity. After 60 seconds, which it is sufficient for all old flows to terminate, the controller deletes the forwarding rules that direct all packets to the



sID	In Port	SrcIP	Action
1	$Port_1$	ip_1	Fwd to $Port_2$
1	$Port_2$	ip_1	Fwd to $Port_3$
1	$Port_3$	ip_1	Tag R2, Fwd to $Port_1$
1	$Port_2$	ip_3	Tag Fin, Fwd to $Port_1$
*	$Port_4$	ip_3	Tag sID, Fwd to $Port_2$

sID	In Port	SrcIP	Action
1	$Port_1$	ip_1	Fwd to $Port_2$
1	$Port_2$	ip_1	Tag Fin, Fwd to $Port_1$
1	$Port_1$	ip_2	Fwd to $Port_3$
1	$Port_3$	ip_2	Tag Fin, Fwd to $Port_1$

Router Tag	SrcIP	Action
R1	*	Fwd to A1
Empty	ip_1	Tag sID, Fwd to A1
Empty	ip_2	Tag sID, Tag R2, Next table
*	*	Next table

SrcIP	Action
...	...
ip_1	Fwd to R2
ip_2	Fwd to R2
ip_3	Fwd to R2
...	...

Tag	SrcIP	Action
R2	*	Fwd to A2
Empty	ip_4	Tag sID, Tag Fin, Next table
*	*	Next table

SrcIP	Action
...	...
ip_1	Fwd to VM4
ip_2	Fwd to VM4
ip_3	Fwd to VM4
...	...

Figure 3.4: Illustration of three common scenarios for tagging scheme.

local agent and install the new sub-class matching rules in the OpenFlow switch. From then on, all incoming packets match on the new rules without violating flow affinity.

3.6 Rule-based Load Balancing

Even though VNFs are contained in software and hence flexible to be installed on demand, to redeploy VNFs after each optimization described in Section 3.4 also imposes a heavy burden on Resource Orchestrator to install/delete virtual network function instances. On the other hand, changing the forwarding rules is much more agile than virtual machine management. As a result, this section present a proposal that when the traffic matrix is updated, the VNF placement remains unchanged and the forwarding rules are modified to accommodate the new traffic matrix. The location of VNFs may change if no feasible solution is available given the current placement.

The new optimization problem is very similar to the one described by Eq. (3.1) - (3.8). Given fixed VNFs, the objective of the previous optimization problem does not make any sense. Instead, the objective now is to minimize the maximal load of any network function instance in order to balance the workload. Let μ be the maximal load of any network function instance. The load of virtual network function n at router v is represented as L_v^n. The new objective and the associated constraints are given in Eq. (3.9) - (3.11). The other constraints are brought from Eq. (3.2) - (3.7).

$$Minimize \qquad \mu \tag{3.9}$$

s.t.

$$\mu \geq L_v^n \qquad \forall v, n \tag{3.10}$$

$$L_v^n = \frac{\sum_{h:v \in P_h} T_h d_{h,i(C,h,n)}^{i(P,h,v)}}{Cap_n \times q_n^v} \qquad \forall v, n \tag{3.11}$$

Since all decision variables in this optimization problem are fractional numerical values, this optimization problem can be solved using linear programming.

Thus, the overall procedure to take when traffic matrix is updated is first to solve the linear optimization problem defined in this section. If there is no feasible solution, a new NF placement plan should be given by solving the mixed integer linear programming problem described in Section 3.4; otherwise the forwarding rules are modified according to the optimization result.

The purpose of NF load balancing is to adapt to traffic dynamics without VNF placement changes. It is possible that NF load balancing can absorb traffic dynamics with minimal number of VNFs deployed in the network even though redundancy improves the chance of successfully absorbing traffic dynamics.

3.7 Incorporating Traffic Dynamics

The large time-scale traffic dynamic domonstrates clear daily or weekly patterns [68]. Since the traffic pattern changes slowly, for this kind of traffic dynamics, it can be easily handled by periodically running Optimization Engine and placing VNF instances accordingly.

The difficult part is to efficiently handle small time-scale traffic dynamics is both fast and vigorous. Even if load balancing can help in this case, the time to collect traffic matrix information takes up to tens of minutes [95]. In this chapter, another mechanism is proposed to adapt to the traffic dynamics quickly, called *fast failover*. The core idea is to temporally re-balance the distribution of sub-classes to relieve the overloaded VNF instance. Since overloading is transient, the distribution will roll back to the normal state when the VNF instance is no longer overloaded.

Fast failover can react quickly to small time-scale traffic dynamics, because it only temporarily changes the TCAM matching rules and if possible installs light-weight VNF instances (*i.e.* ClickOS[125]). When a VNF instance is overloaded, it sends an overloading notification to Dynamic Handler which in turn spreads half workload of all sub-classes that traverse this VNF instance to other sub-classes. If such re-balance is expected to result in overloading of another light-weight VNF instance, the Dynamic Handler installs new instances to absorb traffic dynamics. Also, when the VNF instance is no longer overloaded, the temporarily installed instances are canceled to save hardware resources. During fast failover, if there is any stateful VNF involved, flow affinity should be ensured. Figure 3.5 illustrates the steps to achieve fast failover for a particular example, where the firewall is a light-weight VNF instance. Initially, there are two IDS and one firewall instances on the path. When the *master IDS* instance is overloaded, APPLE builds a new sub-class by installing a new firewall to accommodate traffic from previous sub-class.

3.8 Implementation Details

3.8.1 ClickOS VM Initiation

A prototype system of APPLE is developed based on open-source software tools including OpenStack [32], ClickOS [125], Open vSwitch [29], and OpenDay-Light SDN controller [30]. A step-by-step procedures to initiate a new ClickOS

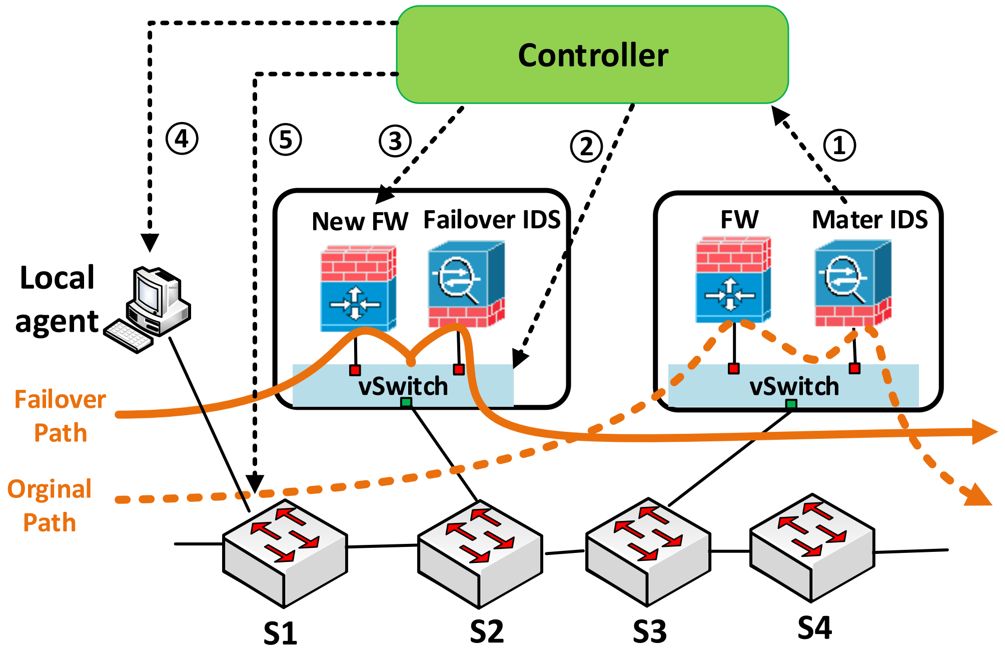


Figure 3.5: Steps of fast failover for a sub-class whose policy chain is $FW \rightarrow IDS$: (1) Overloaded VNF instance sends an overloading notification. (2) New light-weight instances are initiated. (3) Controller installs forwarding rules for the new sub-class. (4) Inform the local agent of the old and new sub-class IDs. (5) Update rules to forward half traffic to the new sub-class.

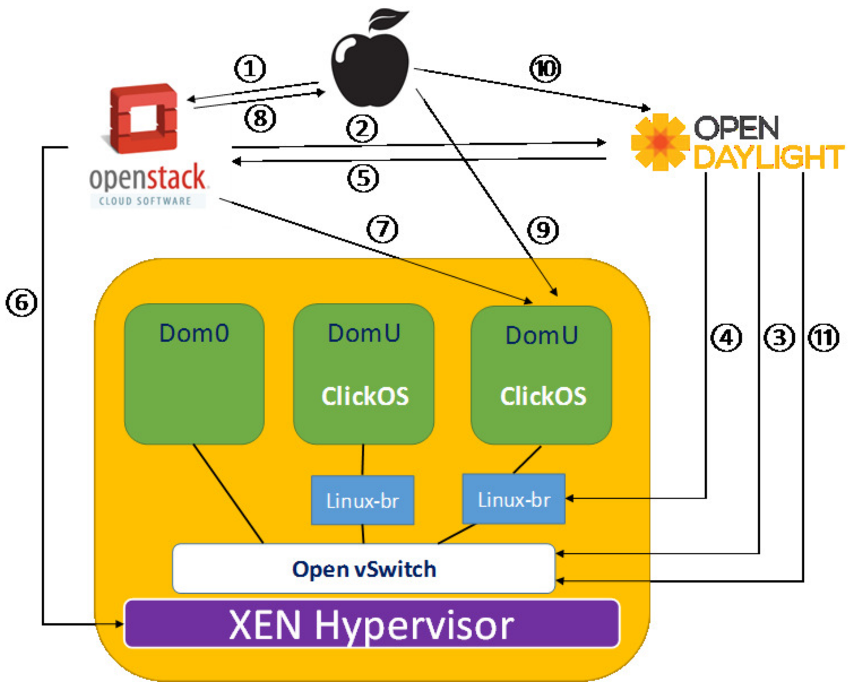


Figure 3.6: Implementation of initiating a new ClickOS VM for a VNF instance.

[125] VNF instance is illustrated in Figure 3.6. The central controller is a standalone application that calls services provided by Openstack [32] and Opendaylight [30] via their REST APIs. It is worth noticing why Openstack delegates the networking part to Opendaylight. An Openstack controller contains a component called Neutron, which is responsible for the management of the embedded virtual network. However, Neutron exposes no APIs for the users to install customized forwarding rules to Open vSwitches [29] used in APPLE hosts. However, if the network opertor manually sets Opendaylight as the controller for Open vSwitches, Openstack will seize control of Open vSwitches intermediately. Explicitly configuring that Opendaylight handles the networking for Openstack is used in the prototype. Upon VM initiation request, Openstack notifies Opendaylight to prepare the networking via REST API (**Step 2**). In **Step 3**, Opendaylight calls OVSDB [143] South-bound RPC to create a new port on the Open vSwitch. Since Xen VMs do not support Open vSwitch directly, a Linux Bridge [21] is added between one Xen [63] VM and the Open vSwitch (**Step 4**). Augmented with the networking information from Opendaylight (**Step 5**), Openstack leverages libvirt driver [20] to create a new VM (**Step 6**). After that, the newly created VM fetches the ClickOS image from Openstack and installs it (**Step 7**). Once APPLE is notified the completion of the VM installation (**Step 8**), it configures the ClickOS VM into the desired VNF through a customized tool describe in [125] (**Step 9**). Finally, APPLE proactively installs the forwarding rules in the Open vSwitch by calling Opendaylight's REST API (**Step 10&11**).

For normal VMs other than ClickOS, the procedures to initiate them are almost the same. The only different is that in **Step 9**, generic configuration tools are utilized (e.g. the tools from Openstack).

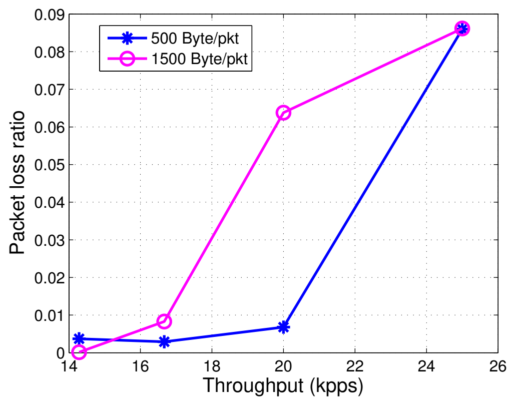


Figure 3.7: Performance of VNF

3.8.2 Overloading Detection

It is found that for most of the VNFs, the performance is closely related to the packet receiving rate as illustrated by Figure 3.7 which shows how the loss rate of a ClickOS-based traffic monitor changes.

3.8.3 Local Agent

The performance of the local agent is fairly important. A kernel module is built based on Netfilter [28] to check the 'SYN' flag and tag sub-class ID at the IP DS field. This kernel module inserts the hook function at the point where the packets just enter the kernel, and thus obviating the resource-consuming but unnecessary Linux network stack. Once the packets are tagged, they are directly sent to the NIC output queue, bypassing Linux network stack again. Another optimization to further accelerate the processing rate is that instead of consuming CPU cycles, the calculation of the new checksum is off-loaded to the NIC. If all the traffic flows consist of 1500-Byte packet, the local agent can sustain 4.2Gbps.

3.9 Prototype Evaluation

3.9.1 Experiment Setup

Openstack (Liberty release), Opendaylight (Lithium release), Xen Hypervisor (version 4.4.2) and Open vSwitch (version 2.0.2) are installed on a VirtualBox VM with quad cores@3.4G and 8GB memory. Two network namespaces [22] in Dom0 and all Xen VMs are connected to a same Open vSwitch. Here, network namespaces, light-weight containers, are created to emulate production hosts or VMs. One network namespace sends packets to the other one via a ClickOS VM that is configured as a passive monitor.

3.9.2 ClickOS VM Setup Time

The prototype experiment indicates that the booting time is much longer if Openstack is involved. The setup time is estimated by measuring the duration when the throughput drops to zero: new forwarding rules are installed on the Open vSwitch (which consumes only negligible time, as little as 70ms) right before ClickOS VM creation, meanwhile the namespaces are sending UDP packets (Figure 3.8). The experiment is conducted 10 times. The approximate booting time ranges from 3.9 seconds to 4.6 seconds, with an average of 4.2 seconds. The main reason for the longer booting time is that Openstack and Opendaylight consume substantial time to orchestrate and prepare the networking before actually initiating a new VM.

3.9.3 Waiting For Five Seconds

One solution to obviate the overhead introduced by failover is to change the forwarding rules after the complete creation of the ClickOS VM. In this subsection,

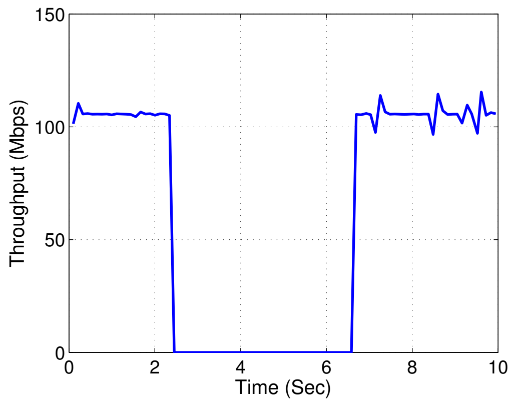


Figure 3.8: ClickOS booting time

the controller modifies the forwarding rules on the Open vSwitch 5 seconds after VM initiation request. According to previous VM setup time measurement, 5 seconds is enough to completely boot a new ClickOS VM.

This subsection evaluates the overhead of failover for TCP and UDP flows, by using Iperf [17] and Netcat [27] to send UDP packets and to transfer a 20MB file via TCP, respectively. Both experiments are conducted 10 times. As expected, there is no overhead associated with failover. For all 10 times of the UDP experiment with 1500-Byte UDP packets sent at 10Kpps, the loss rate for the UDP flow is always 0%. Figure 3.9 shows the CDF plot of the time to transmit a 20MB file with and without failover, which indicates that failover does not bring extra overhead. The performances of the three situations in Figure 3.9 are approximately the same and their differences are due to the statistical fluctuation.

3.9.4 Reconfiguring Existing VMs

Even though the solution in Section 3.9.3 introduces no performance degradation, the 5-second waiting time constrains the flexibility of the system to adapt to network dynamics. The booting timer can be saved by configuring existing ClickOS VNFs. The micro-measurements shows that the time to install forward-

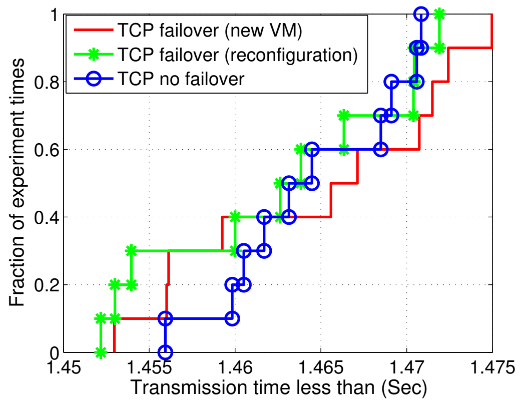


Figure 3.9: File TX time distribution

ing rules is 70ms and reconfiguration only takes 30ms. A similar experiment to Section3.9.3 is conducted. The only difference is that reconfiguring an existing ClickOS VM replaces initiating a new one. Still, the UDP packet flow rate is 0% for each time of the UDP experiment. There is also no noticeable difference in TCP performance (Figure 3.9).

3.9.5 Overloading Detection

One namaspace use pktgen[136] to send 1500-Byte UDP packets to another one via a ClickOS instance that is configured as a passive monitor. The passive monitor is viewed as being overloaded if the receiving packets rate is greater than 8.5 Kpps. The distribution will roll back to the normal state if the packets rate drops to 4 Kpps or lower. Figure 3.10 illustrates how fast our system detects overloading. Initially, the source sending rate is 1 Kpps. To mimic network dynamics, the source sending rate soars to 10 Kpps. The overloading is immediately detected. Therefore, another ClickOS instance is quickly configured as a passive monitor and the traffic is evenly split to the two passive monitors. 50% is indeed a very conservative number. However, the workload is also hopefully evenly distributed to two VNF instances to prevent further overloading as much as we can.

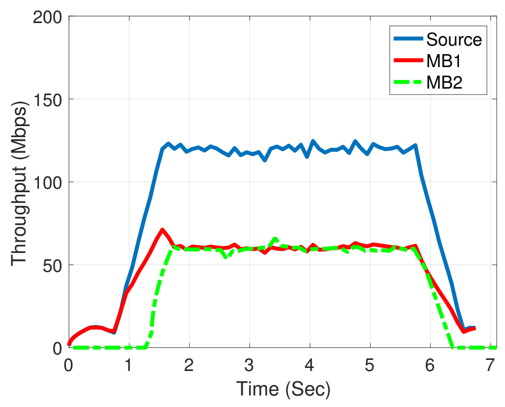


Figure 3.10: Illustration of fast failover

After 5 seconds, the source sending rate becomes 1 Kpps again, which causes the network to roll back to the normal state. During the whole process, the packet loss is 0%.

3.9.6 Performance of Local Agent

In this experiment, two physical machines each with 1Gbps NIC are interconnected with a switch. One physical machine utilizes pktgen to send UDP packets to the other machine that runs the local agent. Given a bandwidth of only 1Gbps, The packet sending rate is controlled by changing the packet size between 60 Byte and 1500 Byte. Three different experiments are conducted. The first experiment is a control group in which the local agent does nothing and the packets are forwarded to Linux network stack. In the second experiment, the kernel module is installed in the local agent, but this kernel module calculates the checksum by its own. The third experiment is similar to the second one. The only difference is that the calculation of checksum is off-loaded to NIC. Figure 3.11 shows that with the kernel module, the throughput of the local agent that processes packets is even greater than that of the local agent doing nothing. It can also be inferred from Figure 3.11 that the offloading provides considerable improvement. The kernel

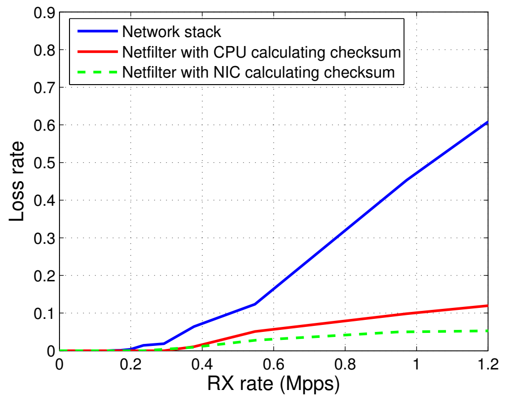


Figure 3.11: Local agent performance

module can receive packets at a rate of 0.3 Mpps without any packet loss, which is enough for our system. One experiment further monitors the latency overhead introduced by flow affinity by computing the timestamp when one packet is sent to the local agent and the timestamp the packet is sent back from the local agent. packet stamps are from tcpdump. Nevertheless, the latency is so negligible that the accuracy of tcpdump is not able to capture the negligible difference which indicates that the latency overhead can be ignored due to the indirection path to the local agent.

It is found that the processing overhead of flow tagging in software switches is low (latency increase $< 1\%$ and memory increase $< 1\%$) and independent of the load.

3.10 Simulation Evaluation Results

Extensive simulation experiments are performed using real traffic trace data and real network topologies.

3.10.1 Simulation Setup

Topology and data set. Three representative topologies for campus network, enterprise network, and data center network are leveraged. Internet2 research network (12 nodes and 15 links) represents campus network. Time-varying traffic matrices for internet2 are provided in [41], which consist of snapshots of 12×12 traffic matrices. The totem data set [166] is adopted to represent enterprise network. It contains an intradomain network, GEANT (23 nodes and 74 links) and associated time-varying traffic matrices. A 2-tier campus data center network, UNIV1 (23 nodes and 43 links) [68] is used for the data center network experiment. To illustrate that Optimization Engine is scalable even for large topologies, a Rocketfuel router-level ISP topology, AS-3679 [164] is used too. The traffic matrices for UNIV1 and AS-3679 are synthesized using FNSS tools [153].

Optimization Engine is run against one traffic matrix input which is the mean value of 672 snapshots. After that, VNFs are placed in the network according to the result from Optimization Engine. At last, all the traffic matrices are replayed in time order and how APPLE reacts to traffic changes is measured. For each topology, experiments are conducted multiple times with different traffic matrices. Internet2 traces consist of $672 * 24$ snapshots. Therefore, there are 24 sets of experiments for Internet2. Likewise, the experiments on GEANT and UNIV1 will run 15 rounds and 20 rounds respectively.

Service function chains. Due to the lack of publicly available information on NF related policies, service function chains are synthesized based on one real-network study [157] and case studies [38]. The service function chains are the sequences of 4 different NFs: firewall, proxy, NAT and IDS.

VNF specifications. The information on capacity and resource requirements for each NF is from the survey in [64], which is listed in Table 3.4. It is assumed

Table 3.4: VNF data sheets

NF type	**# Cores**	**Capacity**	**ClickOS**
Firewall	4	900Mbps	✓
Proxy	4	900Mbps	✗
NAT	2	900Mbps	✓
IDS	8	600Mbps	✗

that the firewall and NAT are implemented in ClickOS, while the proxy and IDS are contained in normal VMs. It is assumed that there are 64 cores at each APPLE host.

Metrics. The following metrics are measured: TCAM and hardware consumption, the algorithm computation time, and packet loss ratio during traffic dynamics.

3.10.2 Computation Time

Short computing time is crucial to timely VNF provisioning. In order to compare with optimal solutions computed by CPLEX [15], the objective function is chosen to be a simple linear function : minmizing the number of installed VNF instances. Another comparison method is based on LP relaxation, a universal technique to approximate the integer programming by continous linear programming. The optimization problem discussed in Section 3.4 is solved by CPLEX [15] on a quadcore@3.40G desktop with 16GB memory. Table 3.5 compares the average time to solve the problem for different topologies. The running time of SA is configurable and it determines the quality of the solution. We set the running time such that the resulting objective value is at most 35% larger than the optimal value. As inferred from Table 3.5, the computation speed of SA lies between those of CPLEX and LP relaxation based method. However, CPLEX and LP relaxation based methods are not able to support unlinear objective functions.

Table 3.5: Average computation time of different topologies.

Topology	**Nodes**	**Links**	**LP relax**	**SA**
Internet2	12	15	0.029 s	0.056 s
GEANT	23	74	0.1 s	0.31 s
UNIV1	23	43	0.235 s	0.52 s
AS-3679	79	147	3.013 s	16.11 s

For small and medium topologies (Internet2, GEANT and UNIV1), Optimization Engine is fast: the computation time is less than 1 second. Even for the large topology (e.g. AS-3967), the computation time is acceptable.

3.10.3 Rule-based Load Balancing

By utilizing rule-based load balancing, the overhead of resource orchestrator is reduced significantly. There are 24 sets of data for Internet2, and 15 sets, 20 sets of data for GEANT and UNIV1 respectively. In this set of simulation experiments, for each topology, the VNF placement is first computed by using the average traffic rate of the first dataset as traffic matrix input. Two VNF placement algorithms are used: one getting the optimal result using CPLEX directly; the other resulting in approximate result utilizing LP relaxation and SA. After that, rule-based load balancing is performed by feeding the average traffic rate of other datasets with fixed VNF placement. The number of rounds having feasible solution without VNF redeployment for each topology is illustrated in Table 3.6, which clearly shows that the VNF placement based on the heuristic is more robust to traffic dynamics compared to the optimal result. After all, from the result of the heuristic, there are more instantiated VNF instances which provide redundancy and flexibility.

The maximum workload of any VNF (e.g. μ in Eq. (3.9)) of feasible solutions is demonstrated in Figure 3.12. It can observed that the maximum workload

Table 3.6: Number of rounds having feasible solution without VNF redeployment.

Topology	**Rounds**	**Optimal**	**LP relax**	**SA**
Internet2	23	19	21	21
GEANT	14	1	12	7
UNIV1	19	6	19	19

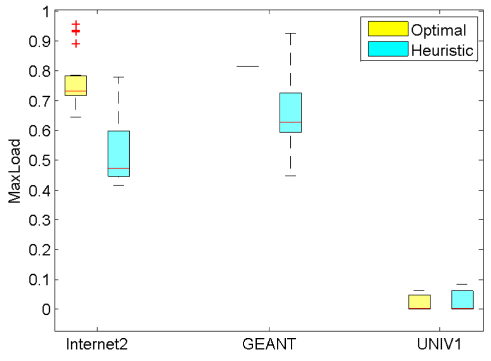


Figure 3.12: Maximal Load

obtained from optimal VNF placement is generally larger. This observation again illustrates the flexibility and redundancy provided by the heuristic. Note that even it seems that the optimal VNF placement yields smaller maximum workload in UNIV1, it has much smaller feasible solutions.

3.10.4 TCAM Usage

By leverage the tagging scheme, the TCAM consumption is reduced. Figure 3.13 gives the boxplot of the TCAM usage reduction ratio compared to that without tagging scheme, for three topologies under different traffic matrices . There is a least 4X reduction for all three topologies. The reduction ratio for UNIV1 topology is more impressive than the other two, because traffic exploits multi-paths in data center networks. Therefore, we are more motivated to tag these traffic at their ingress switch, rather than match them on all multi-paths,

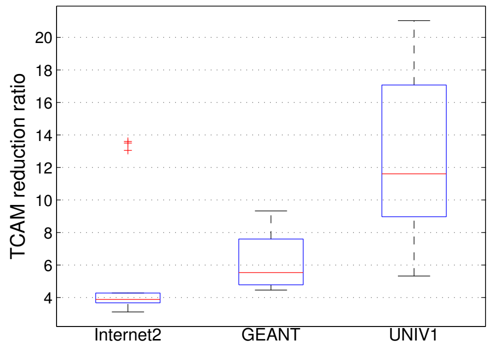


Figure 3.13: TCAM usage reduction by tagging

resulting in less TCAM consumption.

3.10.5 Hardware Resource Usage

Since APPLE is the first VM-based orchestration framework that introduces no interference to the network, this subsection compares the hardware resource usage for APPLE with an alternative strawman solution called *ingress*, which consolidates all the VNFs of one service function chain in the ingress switch and enforce policy there for each class. For simplicity it is assumed that CPU core is the only required resource for launching a new VNF instance. Figure 3.14 plots the hardware usage for the two solutions. There is 4X reduction for internet2 and 2.5X reduction for GEANT. This benefit comes from the resource multiplexing between different classes. The gap in UNIV1 is not that significant, because UNIV1 only has two core switches. Therefore, the limited hardware capacity at the core switches force APPLE to place VNFs at the ingress switches.

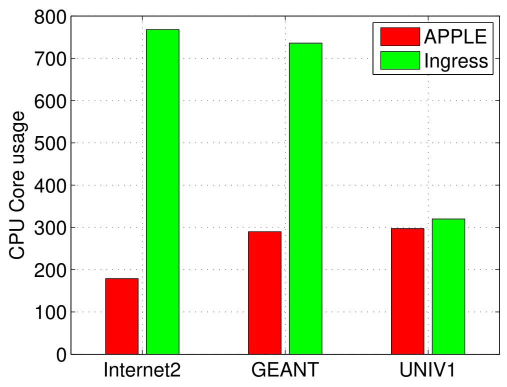


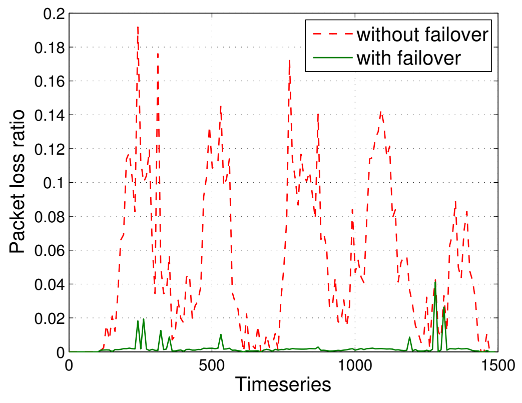
Figure 3.14: Average CPU Core usage.

3.10.6 React to Traffic Dynamics

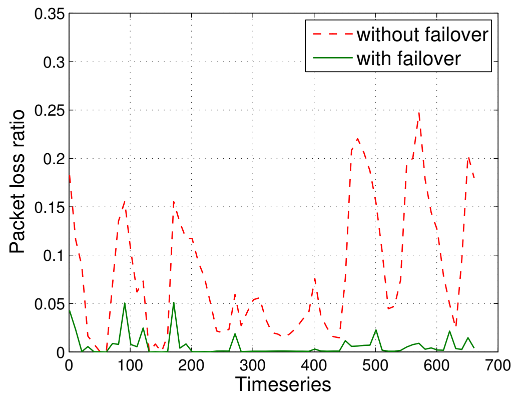
With fast failover, APPLE can quickly react to traffic changes with low packet loss rate. Figure 3.15 depicts the packet loss rate over time for three different topologies with/without fast failover. Thanks to fast failover, the packet loss rate remains much lower for all three topologies even in the face of fiercely changed traffic. This plot illustrates the ability of fast failover to absorb traffic burst efficiently. In the mean time, only a few new ClickOS instances are installed to support fast failover. The average additional cores to support fast failover is less than 17 for all topologies.

3.11 Conclusion

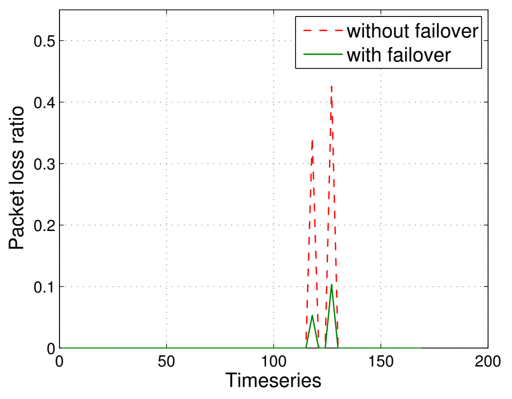
APPLE is the first implementation of an NFV orchestration framework based on VMs that satisfies three requirements, namely order preserving, interference freedom, and genericity. APPLE applies an optimization engine to determine VNF placement and a flow tagging scheme to reduce TCAM consumption and retain flow affinity. Results from both prototype and simulation experiments using real network topologies and traffic matrices show that APPLE achieves



(a) internet2



(b) GEANT



(c) UNIV1

Figure 3.15: Packet loss rate over time for APPLE with and without fast failover.

good performance.

Chapter 4

An IoT Data Communication Framework for Authenticity and Integrity

The design of an IoT data communication framework involving the three key entities: *IoT devices*, *cloud*, and *data consumer*. The following key requirements or challenges of the IoT data communication framework distinguishes it from traditional data collection and management methods.

1) **Time series data and event data.** IoT sensing data can be classified into two types: time series data and event data [178]. Time series data are generated by each device for every fixed time period, such as 1 second. They are used to conduct continuous monitoring tasks such as temperature reports. Event data are generated whenever a certain type of events occurs, such as a vehicle appearing in a smart camera. They are used to monitor discrete events. Note time series data can be viewed as a special case of event-based data driven by clocks. *Hence this chapter focuses on finding a solution for event-based data. The proposed methods*

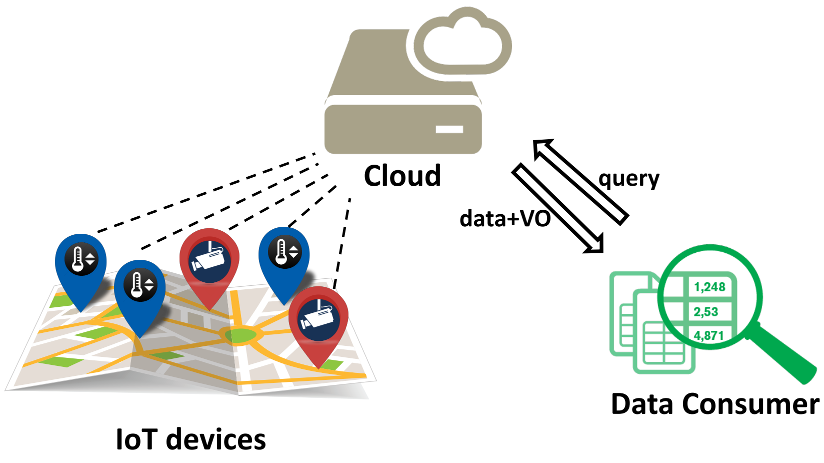


Figure 4.1: Overview of the IoT data communication framework

are applicable to time series data as well.

2) **Data random sampling.** A common but critical problem shared by state-of-art IoT designs is that the resources for transmitting and storing data (e.g. network bandwidth, storage quota) are limited in presence of massive IoT data. By 2022, the IoT data is expected to constitute 45% traffic in the Internet [92]. Cloud providers charge users for storage, retrieval and transferring of data [2]. It is desired to have *predictable cost* for both users (in finance) and the cloud (in resource) [87]. Hence only a fixed resource budget can be allocated to the sensing data over a time period, called an *epoch*. For example, the cloud can only keep 100 data records from any device collected during every minute. Random sampling is widely used in IoT [65]. To guarantee representative samples, every event should have an equal probability to be sampled, which is called the *uniformity* property. Uniformity is essential for unbiased statistics estimation such as histogram [80], median [124] and average [134].

3) **Authenticity and integrity.** Since the sensing data are stored in a third-party cloud, data authenticity and integrity, which guarantee that data are from these sensing devices and have not been modified, are important for trustworthy

IoT applications [169]. However the data could be corrupted by outside attackers [3, 4, 8], malicious cloud employees [123], transmission failures, or storage loss [40]. Without data authenticity and integrity, IoT applications may make wrong decisions and cause economic and human-life losses. Authenticity and integrity should be *verifiable* by data applications.

4) **Flexible application requirements.** Different applications may have different requirements on sending data granularity. For example, applications like self-driving cars need fine-grained road information, while other applications like road-traffic estimation only need a few sampled data. Even if the cloud can store up to 100 records, some applications only retrieve part of them, e.g., 10 records, due to bandwidth limit, memory limit, or application requirements. In addition, these 10 records should have verifiable authenticity, integrity, and uniformity. We call this feature *partial sample-rate data retrieval*.

There are mature solutions for any individual requirement that has been studied by researchers for over decades. However, no existing solution collectively resolves all requirements of emerging IoT applications. The difficulty stems from the combination of seemly conflicting requirements. The main contribution of this chapter is to present two holistic solutions, namely DTC and GSC, which strive for a balance of security and performance in IoT settings. *Digital signature* is a widely used method to protect data authenticity and integrity: The sender first computes a message digest D by hashing its original message m using a cryptographic hash function H, $D = H(m)$. H is also called message digest function. Note the length of D is significantly shorter than that of m. Then the sender uses its private key k to encrypt D and attaches the *signature* $E_k[D]$ to the original message. When the receiver gets m and $E_k[D]$, it decrypts $E_k[D]$ using the public key of the sender and verifies whether $D = H(m)$. However, applying the

Table 4.1: Performance of some cryptographic operations.

Metric	RSA	DSA	MD5	SHA1	SHA256
Time (ms)	203.6	192.8	0.041	0.051	0.098
Energy (mJouel)	37.6	36.2	0.007	0.009	0.017

digital signature to every sensing record, called the *Sign-each* method, is not practical, because public-key encryption/decryption is considered slow and expensive, especially for sensing devices with limited resources.

A more efficient method, *concatenate*, is to compute the message digest D for a large number of records and sign once on D. This approach requires each sensing device to cache all records and has the all-or-nothing feature: if some applications only require part of the records, the signature cannot be verified. A well-known method to sign a data stream is hash chaining [100]. However, it does not fit the IoT framework either, because sampling and partial sample-rate data retrieval will break the chain and hence make the signature unverifiable. The authors test the performance of some mostly used cryptographic operations on M3, one mainstream IoT hardware platform available from one public testbed [12]. This hardware platform features one 32-bit ARM Cortex-M3 CPU@72MHz. The result shows the average time and energy to encrypt (RSA/DSA) or to compute hash (MD5/SHA1/SHA256) over a 10-byte string [144]. The result is presented in TABLE 4.1. Even though there is great advances in hardware performance compared to prior platforms [30], directly applying RSA/DSA is still not suitable for resource- constraint IoT devices, especially for those powered by batteries.

A qualitative comparison on GSC, DTC, and other possible methods are is presented in Table 4.2.

Table 4.2: Overall comparison of different signature schemes.

Signature Scheme	Computation efficiency	Partial data retrieval	Constant space cost	Sampling uniformity
Sign-each	✗	✓	✓	✗
Concatenate	✓	✗	✗	✗
Hash chaining	✓	✗	✓	✗
DTC	✓	✓	✗	✓
GSC	✓	✓	✓	✓

4.1 Problem Statement

4.1.1 Network Model

The life cycle of IoT sensing data is demonstrated in Figure 4.1. The communication model consists of three different kinds of entities:

IoT devices are resource-constraint devices that generate sensing data. Programs running on the IoT devices should abide stringent resource limits in computation, memory, and power resources. IoT devices transmit sensing data to the cloud at a fixed time interval called *epoch*. IoT devices do not require perfect synchronization, but one synchronization protocol is needed to loosely synchronize clocks on different IoT devices with bounded drift.

Cloud is a third-party storage provider who has rich resources. It stores the sensing data from IoT devices and exposes an interface for Data Consumers (DCs) to make queries.

Data Consumers (DCs) are a vast variety of software systems, devices and human clients that retrieve the IoT sensing data for analysis purposes. DCs may also be IoT devices. The data consumers are able to make ad-hoc queries on history data.

IoT sensing data can be classified into two types: time series data and event data [178]. Time series data, generated periodically, are used to describe continuously changing environment parameters such as temperature. Event data are generated whenever a certain type of events occurs, such as a door with one smart lock being opened. Note time series data can be viewed as a special case of event-based data driven by clocks. Hence this chapter uses event-based data in the model for generality. At the end of each epoch, a device sends all events data to the cloud.

4.1.2 Threat Model

IoT sensing devices and data consumers are trustworthy and any entities in between including the cloud are subject to attack or may perform functionalities in a dishonest way. The correctness of range selection are two-folded:

1. Each data item in the query result should be from the intended database and not tampered by any third party. This property is called as **authenticity** and **integrity**.
2. Each data item in the result must satisfy the query. This second requirement is denoted as **completeness**.

This work does not address the issue of privacy and confidentiality. They are orthogonal to the database query correctness and there are exiting works [145] on solving these two aspects of data outsourcing.

The cloud can claim that it stores no data to a data consumer. However this cheating is relatively easy to audit and detect. As long as the cloud claims it stores sensing data, the returned data to a DC must be correct and complete.

To make the system robust against device compromise, each IoT device hosts and uses its own private key. The work assumes the existence of a well-functioning PKI which manages the distribution of the public keys. There is also an external

Table 4.3: Important Notations.

Notation	**Definition**
D_i	Message digest of sample block i or event i
D_{ij}	Message digest summarizing ith till jth events
$H(\cdot)$	Message digest function
pk	Public key
pk^{-1}	Private key
$\{\cdot\}_{pk^{-1}}$	Encrypt using private key pk^{-1}
$\{\cdot\}_{pk}$	Decrypt using public key pk
m	Number of sampled events
n	Number of monitored events
K	Number of sensing devices
S_a	Numerical interval between 2^{-a-1} and 2^{-a}
$h(\cdot)$	Hashing function whose range is between 0 and 1
B	Budget limit

mechanism for the data consumer to get the relation schema and the names of all collaborative IoT devices in the task groups of interest.

4.2 System Design

At first it is assumed that there is no budget constraint as it is a common scenario in current IoT settings. This work relaxes this assumption and incorporate budget constraints in Section 4.3.

4.2.1 Existing Signature Schemes

Digital signature is widely used to ensure data authenticity and integrity. However, none of existing signature schemes are appropriate for the IoT setting.

First, the straightforward Sign-each method causes expensive computational cost on both the signer and the verifier owe to excessive public-key encryption/decryption operations. Since data selection is completely executed in the cloud, if the cloud selects event samples or the partial data with bias, data consumers are unaware of it. Furthermore, the Sign-each method may not be able to detect data loss if an event record is entirely disappeared.

The *concatenate* signature scheme can amortize the signing and verification cost to multiple messages, but it is not suitable for sensing devices which may be lack of buffer space to accommodate all messages. In addition, it does not support partial sample-rate data retrieval.

Hash chaining [100] reduces the buffer space complexity from $O(m)$ to $O(1)$ for both the signer and verifier, where m is the number of messages buffered in the sensing device to be jointly signed. In the hash chaining signature scheme, only the first message is signed and each message carries the one-time signature for the succeeding message. However, hash chaining fails when some events are dropped due to sampling or partial sample-rate data retrieval.

Two signature schemes are proposed in this chapter to address the aforementioned problems when applying digital signature in the IoT scenario: 1) the Dynamic Tree Chaining (DTC) that is developed based on Merkle tree [128] and a later more detailed design by Wong and Lam [172]. DTC serves as the baseline in this chapter. 2) a novel signature scheme specifically designed for the IoT data communication framework, called Geometric Star Chaining (GSC).

4.2.2 Dynamic Tree Chaining (DTC)

Let start from the Tree chaining designed by Wong and Lam [172], one variation of Merkle tree [128]. The digest of each event report is one leaf node in binary *authentication tree* presented in Figure 4.2. The value of the internal node is computed as the hashing of the concatenation of its two children. Take the authentication tree in Figure 4.2 as an example. D_{12} is the parent of D_1 and D_2 and $D_{12} = H(D_1||D_2)$, where $H(\cdot)$ is the message digest function, such as

SHA-1 [50] or MD5 [24], used for tree chaining. Likewise, $D_{14} = H(D_{12}||D_{34})$ and $D_{18} = H(D_{14}||D_{58})$. As a result, the root summarizes all the leaf nodes. The root node is regarded as the block digest. The block digest is appended with *epochID* and then signed by the private key to create the block signature. EpochID is used to identify which epoch the data are generated; otherwise, the cloud returns events from other epochs without being detected.

The verification process is on a per-event basis. In order to verify the integrity/authenticity of an event e, the verifier requires the block signature, the position of event e in the authentication tree and the sibling nodes in the path to the root, which are all appended to event e. As a result, the overhead to transmit this metadata is $O(\log n)$, where n denotes the number of events.

Basically, the verification algorithm is to replay the process to build the authentication tree and to verify the nodes in the path to the root. Imagine the receiver begins to verify event e_3 which is represented as the dashed circle in Fig. 4.2. First, the receiver computes $D'_3 = H(e_3)$ and then its ancestors in order: $D'_{34} = H(D'_3||D_4)$, $D'_{14} = H(D_{12}||D'_{34})$, $D'_{18} = H(D'_{14}||H_{48})$. Event e_3 is verified if the decrypted block signature equal D'_{18}, that is to say $\{\{D_{18}\}_{pk^{-1}}\}_{pk} = D'_{18}$, where $\{\cdot\}_{pk^{-1}}$ denotes singing using private key whereas $\{\cdot\}_{pk}$ is the function to decrypt signature with public key. In this case, all the nodes in the path, as well as their siblings, are verified and they could be cached to accelerate the verification process. Suppose the 4th event e_4 arrives after e_3 has been verified. Event e_4 is verified directly if $H(e_4) = D_4$.

Note that the expensive encryption operation is amortized to all events in one authentication tree and thus tree chaining is computationally efficient. More importantly, since every single event is verifiable in tree chaining, it is fully compatible with partial sample-rate data retrieval without resource waste. The most severe issue that impedes the adoption of the original tree chaining in IoT environment is that all events should be buffered in the IoT device before the building of authentication tree, since each event ought to be appended with auxiliary authentication information from the authentication tree.

Introducing cloud can greatly reduce the memory footprint at IoT devices. The IoT device only maintains the message digest of each event and stores all events to the cloud directly without caching. At the end of each epoch, with all leaf nodes available, the IoT device builds the authentication tree, which is then sent to the cloud. The cloud in turn attaches essential authentication information to each event received in the current epoch. The memory footprint can be further optimized if the authentication tree grows in an online fashion: The IoT device transmits to the cloud internal nodes no longer needed for calculating the rest of authentication tree. An internal node is generated when its two children are available. In the meantime, these two children are transmitted to the cloud. Figure 4.2 illustrates the online authentication tree building process. Verifying D_3 requires sibling nodes in the path to the root (D_4, D_{34}, D_{58}), signature of the root ($\{D_{18}\}_{pk}$) and the position of e in the tree (3). $D_1 - D_8$ represent the message digests of the events in timely order. D_{12} is calculated immediately when D_2 comes into play. In the meantime, D_1 and D_2 cached in the sensing device are transmitted to the cloud. Likewise, when D_4 is available, D_{34} is computed, which in turn immediately contributes to the calculation of D_{14}. As a result, at that time D_3, D_4, D_{12} and D_{34} are dismissed from the sensing device. This optimization reduces the space complexity in the sensing device to host nodes of authentication tree from $O(n)$ to $O(\log n)$, where n denotes the number of events monitored in one epoch.

For DTC, data selection is completely executed in the cloud when the data

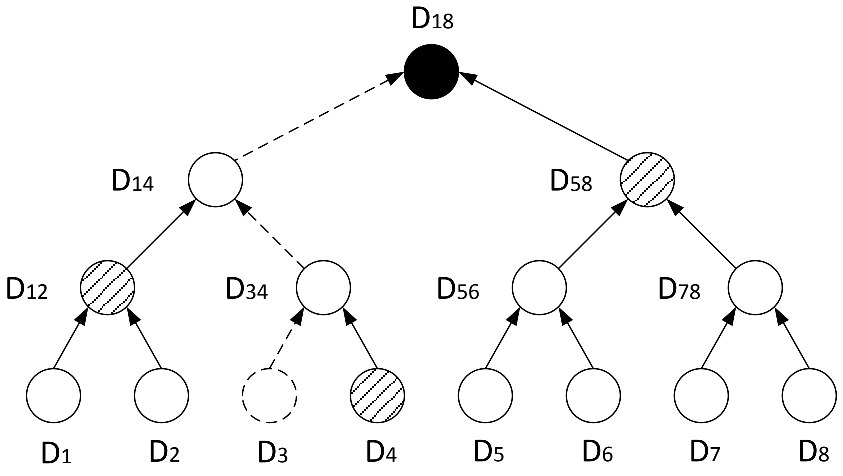


Figure 4.2: Illustration of tree chaining.

consumer retrieves partial sample-rate data. Without additional mechanisms, if the cloud selects event samples or the partial data with bias, data consumers are unaware of it. The plausible solution is to allow the data consumer to specify the sequences of the interesting events. It is the data consumer's own responsibility to guarantee uniformity. Apparently, this straightforward method is not scalable. This solution could be optimized by expressing the sequence of requested events in a succinct way. A sequence number is appended to each event to indicate its position in the authentication tree. The data consumer sends a number m which specifies the number of events requested as well as a seed s which uniquely determines one random permutation. The cloud returns the events specified by the top m elements in the random permutation.

The data consumer checks whether these events sequence numbers confine the random permutation. In this proposal, the uniformity is guaranteed by the random permutation. The pseudocode of random permutation algorithm in Algorithm 1, which is an implementation of Fisher-Yates shuffle [154]. The random

Algorithm 1: Random Permutation

```
Input: An array of of events $E$; A seed $s$ for the pseudorandom number
       generator; A positive number $n$
Output: A list of $n$ elements from $E$
1 $l \leftarrow$ E.size();
2 rng.init($s$); // Initialize random number generator
3 for $i \leftarrow 0$ to $n-1$ do
4     $j \leftarrow$ random integer with $i \leqslant j \leqslant l-1$;
5     swap ($E[i]$, $E[j]$);
6 end
7 return E[0:n-1]; // return first $n$ elements
```

permutation algorithm ensures that the m randomly sampled events confirm to uniformity.

Furthermore, using different seeds enables *re-sampling*, which means that the receiver can request for different sets of uniformly drawn events from the cloud.

This feature is useful for some machine learning applications. One example is ensemble learning [183] in which each weak learner requires one instance of training dataset generated by one round of re-sampling from the whole dataset.

Nevertheless, the number of generated events is unpredictable and may be unbounded. Once the buffer in the sensing device is full, the root node in the authentication tree is signed and the remaining nodes are flushed to the cloud to spare space for upcoming events. In this case, one IoT device may apply digital signature more than once in one single epoch. The verifier also requires additional space to cache the verified nodes. The verifier stops caching new verified nodes when the buffer is full. As a result, the buffer space constrains the performance of DTC, which is a particularly severe problem in IoT environment where most devices possess little buffer space. DTC can be also extended to k-degree Merkle tree.

4.2.3 Geometric Star Chaining (GSC)

This subsection presents a more efficient and secure data communication framework in this chapter, called Geometric Star Chaining (GSC). The basic idea of GSC is inspired by one observation that any arbitrary fraction value can be represented or closely approximated by a few number of binary digits. For instance, $5/8 = (0.101)_2$. Thus, partial data with sample rate p, where $p = \sum 2^{-b_i}$ and b_i is the position of the i-th 1 in the binary expression of p, is equivalent to the union of multiple data blocks each corresponds to one set bit in the binary representation. The data block is called *sample blocks*. For instance, to retrieve a sampled data with sampling rate 5/8, the cloud can send the data consumer two blocks containing (approximately) 1/2 and 1/8 of the samples respectively.

The events included in the sample blocks are in *geometric distribution*. Each sample block should draw events uniformly from the IoT data stream. In order to ease the presentation of how sample blocks form, this dissertation defines a set of successive numerical intervals $\{S_i\}$ where $S_i \triangleq \{x \in \mathbb{R} : 2^{-i-1} < x \leq 2^{-i}, i \in \mathbb{N}\}$, which are visually represented as rectangles in Figure 4.3. On receiving a new event e, the sensing device computes which numeric interval in $\{S_i\}$ that $h(e)$ falls in and event e is inserted into the corresponding sample block, where $h(\cdot)$ is a non-cryptographic uniform random hashing function and $\forall x : 0 \leq h(x) \leq 1$.

Note that events in the same data block are either completely retrieved or not retrieved at all. Each of such data block could be viewed as an atomic "giant event". GSC computes one message digest for every block and concatenates these digests to a single digest for digital signature, as is depicted in Figure 4.4. Verifying the second sample block requires the events in it, D_1, D_3 and D_4. The digest of one sample block is computed in an online fashion. One variable D_i is allocated to each sample block to capture the newest value of message digest. Suppose a

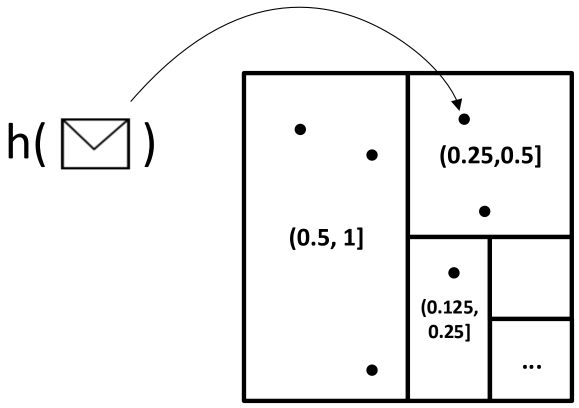


Figure 4.3: Visual representation of numerical intervals.

new event e observed at the device which belongs to the ith sample block. The message digest updates as $D_i = h\left(h(e)||D_i\right)$. This online updating proceeds until the end of the epoch. At this time, concatenate approach is applied to all the message digests $\{D_i\}$. The result summarizes all events generated in one epoch. Note the value i, which indicates the sampling rate of each block, should also be stored and hashed with the block. In this way, the data consumer that receives the block can verify the sampling rate.

In fact, any random function can be used to implement the geometric distribution for GSC, such as continuous coin-tossing, but using a uniform random hash is convenient. One practical issue about hashing is that the raw output of hashing functions is one finite-length bit sequence. Computing which numerical interval in $\{S_i\}$ that $h(e)$ falls in is equivalent to counting leading zeros (CLZ) in that bit sequence, which is intrinsically supported in many hardware platforms including X86 and ARM. Therefore, $|\{S_i\}|$ and hence $|\{D_i\}|$ are bounded by the size of the bit sequence. For the case of xxHash64 [44], this function produces 64-bit hash values and thus $|\{S_i\}_{0\leq i\leq 64}| = 65$ and $|\{D_i\}_{0\leq i\leq 64}| = 65$. It is evident that space cost for this signature scheme at the sensing device is constant.

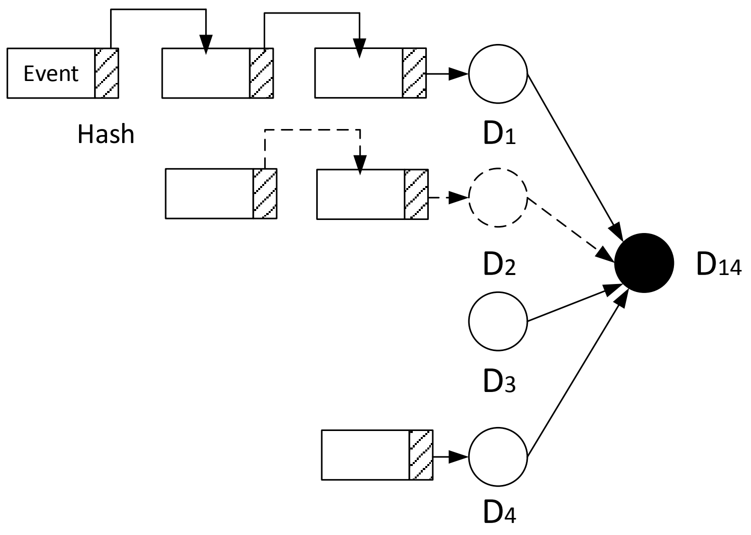


Figure 4.4: Illustration of GSC.

4.2.4 Data Retrieval and Verification of GSC

A sampled fraction of sensing data is usually sufficient for most IoT applications [34]. In the network model presented in Section 4.1.1, a data consumer requests for a certain fraction of events observed at a particular sensing device from the cloud. GSC provides verifiable authenticity, integrity, and uniformity for partial data retrieval with an arbitrary sampling rate.

Based on the application requirement, a data consumer first determines the maximum number of events of each sensing device for an epoch it wants to receive, called a portion number. It then sends all portion numbers to the cloud. For each portion number, the cloud converts it to a sampling rate p and constructs the binary expression of p, such that $p = \sum 2^{-b_i}$ where b_i is the position of the i-th 1 in the binary expression of p. Then the cloud sends the corresponding sample blocks to the data consumer.

For example, if the data consumer requests for data with a sample rate of

$5/8 = (0.101)_2$, it should fetch two sample blocks correspond to sample rate of 2^{-1} and 2^{-3} respectively. The message digests associated with all sample blocks from one epoch are stored in a single file, such that it is convenient for the data consumer to access the necessary information to verify the data. For the received sample blocks, the data consumer first computes their digests as the final digest used for the signature. It then compares the final digest and the decrypted signature. This step verifies the following properties. 1) The received blocks were not modified or partially dropped and 2) The data were indeed uniformly sampled based on the given sampling rates and the uniform random hash function.

Compared to DTC, GSC requires smaller buffer size on each sensing device. It also provides verifiable uniformity. In addition, retrieving GSC-signed IoT data from the cloud can be achieved by sequential reads which are much faster than random reads [42, 35]. GSC does not support random permutation as in DTC because GSC by design ensures verifiable uniformity which is the motivation for random permutation; otherwise, random permutation would offset most of the performance gain of GSC.

4.3 Incorporating Budget Limit

With ever-growing volume of IoT data, storing all raw IoT data in the cloud poses a heavy monetary burden on the users. In the previous section, the system design without restricted budget limits is discussed. This section relaxes this assumption and incorporates the budget limit. The solution presented in this section to address the issue of budget limit is compatible with DTC and GSC.

This section describes a distributed sampling protocol incorporating the budget limit, in which any event, no matter which IoT device generates it, has the same probability to be sampled and finally stored in the cloud. The total number of sampled events does not exceed the budget limit.

4.3.1 Sampling Protocol Design

This sampling protocol introduces a new entity, called *coordinator*, in the network model described in Section 4.1.1. One coordinator is a software working as a sampler which sits between the sensing devices and the cloud. A coordinator can be installed on an IoT hub or a server at the edge of the Internet. It maintains communications with all sensing devices on behalf of the cloud and temporarily buffers IoT data samples.

The sampling protocol presentation focuses on one single epoch since at the beginning of each epoch, the *sampling protocol* (SP) is reset to the initial state. At the end of each epoch, the coordinator signals all sensing devices to advance to the next epoch. The straightforward solution is to buffer all the events in the coordinator and uniformly sample them based on the budget limit. However, the number of these events could possibly be very large, and therefore the storage capacity of the coordinator may be not enough to accommodate them all. Thus, a sampling protocol with space bound for both the sensing device and the coordinator is desired. The challenge of such sampling protocol design derives from the combination of the distributed setting and the unpredictability of data streams. If only one stream of data is considered, the problem is regressed to classic *reservoir sampling* [167], which has been studied extensively in the literature. Also, as long as the number of elements in each stream of data is known in advance, the central coordinator can decide how many samples are allocated to different sensing devices, each of which runs an instance of reservoir sampling.

The basic idea of this sampling protocol is to dynamically maintain events with the smallest hash values on the coordinator, also known as bottom-k sampling [85].

Suppose B is the sampling budget per epoch. For the simplest implementation, all IoT devices upload generated events to the coordinator directly. The coordinator only maintains the B events with smallest hashing value and discards others in an online fashion. As long as the hashing is uniform, the events maintained in the coordinator are drawn uniformly from all events already observed from the epoch.

In order to reduce network bandwidth consumption, the coordinator could broadcast to all sensing devices current global B-th smallest hash value, denoted as τ, so that the IoT devices could discard the events locally whose hash value is greater than τ. Let σ denotes the total number of events sent to the coordinator and K is the number of sensing devices. One straw-man sampling protocol is that the coordinator broadcasts the new value of τ every time it changes. Since τ changes $O(B \log \sigma)$ times, the communication cost between the coordinator and the sensing devices is $O(KB \log \sigma)$. Cormode et al. proposed a distributed sampling algorithm [87], which is proved to be optimal in terms of communication cost, which is $O(K \log_{K/B} \sigma + B \log \sigma)$ with high probability. Basically, the coordinator accepts any event from all IoT devices until the budget limit is exceeded. The coordinator discards half of the received events (*i.e.* the coordinator halves the sample rate) to accommodate new events. The coordinator repeats this process until the end of the epoch and then uploads the stored events to the cloud. The detailed distributed sampling protocol is presented as follows.

The sampling protocol executes in multiple rounds. The coordinator as well as the sensing devices maintain a variable which represents which round the sampling protocol is in, and the coordinator ensures that all devices are kept up to date with this information. Initially, the sampling protocol begins at round 0. Suppose the sampling protocol is at round j. Round j indicates a sample rate of 2^{-j}. This protocol involves two algorithms at the sensing device and the coordinator respectively. The pseudocode for the sensing device and the coordinator is presented in Algorithm 2 and Algorithm 3 respectively.

Sensing device: On receiving a new event e, the sensing device first computes which numeric interval in $\{S_i\}$ that $h(e)$ falls in, and updates the *local counter* associated with this set, where $h(\cdot)$ is a uniform random hashing function and $\forall x : 0 \leq h(x) \leq 1$. Computing the numeric interval can still be visually interpreted by Figure 4.3 where the result presented the $i-th$ largest rectangle. Let l_i^k be the local counter for S_i at device k. Each sensing device and the coordinator maintain their own local counters. The local counters at devices are used for auditing the coordinator. It is worth mentioning that all sensing devices and the coordinator use the same hashing function. Suppose $h(e) \in S_i$. If $i \geq j$, which implies $h(e) \leq 2^{-j}$ (sample rate), the device instantly forwards event e to the coordinator; otherwise, the event is discarded locally. At the end of each epoch, the sensing device signs both sampled events and all counters it maintains. Note that none events are buffered at the device in any case.

Coordinator: The coordinator maintains queues $\{Q_i^k\}$, each of which corresponds to one numerical interval in $\{S_i\}$ of each sensing device. Upon receiving an event e, the coordinator first computes i, such that $h(e) \in S_i$, followed by comparing the value of i and j. In the case of $i < j$, event e is discarded; otherwise, it is buffered at queue Q_i^k (suppose the event is from kth sensing device) followed by updating both the counter associated with numerical interval S_i and the global counter g, which records the total number of events buffered at the coordinator. At this moment, as long as the value of the global counter g exceeds the budget limit B, all event queues associated with S_i are discarded, the global counter is updated accordingly and the sampling protocol advances to the next round (i.e. $j \leftarrow j+1$). The coordinator then signals all sensing devices to promote to the

Algorithm 2: SP at sensing device k in round j

```
foreach event $e$ do
    $i \leftarrow \min\{x \in \mathbb{N} : h(e) \geq 2^{-x-1}\}$;
    $l_i^k \leftarrow l_i^k + 1$;
    if $i \geq j$ then
        Forward $e$ to the coordinator;
    else
        Discard $e$;
    end
end
```

Algorithm 3: SP at the coordinator in round j

```
foreach event $e$ do
    $i \leftarrow \min\{x \in \mathbb{N} : h(e) \geq 2^{-x-1}\}$;
    $k \leftarrow e.source$;
    if $i \geq j$ then
        $Q_i^k$.add($e$);
        $l'_i \leftarrow l'_i + 1$;
        $g \leftarrow g + 1$;
        while $g > B$ do
            Discard queues $\{\forall \hat{k}, Q_j^{\hat{k}}\}$;
            $g \leftarrow g - l'_j$;
            $j \leftarrow j + 1$;
            Broadcast $j$ to all sensing devices;
        end
    else
        Discard $e$;
    end
end
```

newest round j. It is evident that coordinator buffers at most $B + 1$ events all the time. Hash chaining cannot coexist with the sampling protocol, because the coordinator is allowed to discard events that are essential for the verifier to validate the received data. DTC and GSC, on the other hand, do not bear the same problem. Algorithm 3 is the pseudo-code for the coordinator part of this sampling protocol.

4.3.2 Copping with Network Latency

In the last subsection, it is assumed that all communications between the coordinator and the sensing device are instantaneous. However, it is not the case in real networks and the network latency attributed to propagation and processing is inevitable. Consequently, different sensing devices and the coordinator miss synchronization when the coordinator signals all sensing devices to advance to the next epoch. Lagged round promotion does not impact the eventual correctness of the sampling protocol, even though a lot of network bandwidth is wasted owing to the transmission of events that should be discarded at devices locally.

4.3.3 Data Retrieval

The sampling protocol is compatible with DTC and GSC. It is natural for this budget-based sampling mechanism to be compatible with GSC since the sampling algorithm discarding the events half at each round which is essentially removing the existing largest GSC sampling block. As a result, the remaining buffered sample blocks correspond to successive numerical intervals. The data consumer can still fetch any fraction of data that is stored in the cloud. DTC requires a minor modification to support verifiable uniformity when the sampling protocol is performed. Recall how DTC leverages random permutation to guarantee uniformity in Section 4.2.2. The sampling algorithm may discard the events specified by the random permutation. The receiver can still draw events uniformly from the cloud if the receiver can check the existence of every event. Suppose the receiver would like to fetch n events from the cloud. Instead of sending to the receiver the first n events specified by the random permutation, the cloud should reply with the first n **existing events** sorted by the random permutation. To enable the receiver to check the existence of an event locally, the hash function $h(\cdot)$ in the sampling algorithm is based on the event sequence number and epochID, but not the content. If the hash value falls outside the discarded numerical intervals, the receiver instantly knows the existence of the event.

4.4 Security Analysis

Any inconsistency in the verification procedure indicates data in the cloud untrusted. In the sampling protocol, each sensing device maintains a counter to record the number of events that fall in a certain sample block. With the signature, an attacker cannot manipulate, delete or produce fake samples by modifying contents of events and counters without being detected. For the first subsection, it discusses how DTC and GSC guarantee the security by defending the attacks described in the thread model (Section 4.1.2) when the coordinator is not introduced. After that, this section analyzes how to defend against dishonest coordinators by leveraging the local counters from IoT devices.

4.4.1 Defending against Message Forgery Attacks

Both DTC and GSC follow classic Hash-and-Sign Signature paradigm to provide the desirable property of existential unforgeability [72] to defend against message forgery attack. Hash-and-Sign Signature paradigm requires the hash function to be collision resistant.

Tree Chaining (used in DTC) and Star Chaining (used in GSC) are collision resistant if the underlying hash function is collision resistant.

Let function $H : \{0,1\}^{l(n)} \rightarrow \{0,1\}^n$, where $l(n) > n$. $H(\cdot)$ is collision resistant if the following two conditions are satisfied.

- $H(x)$ can be computed in polynomial time.

- $Pr[(x \neq x') \bigwedge (H(x) = H(x'))] \leq negl(n)$, where $negl(\cdot)$ represents a negligible function.

For Tree Chaining, suppose the underlying hash function to compute internal node is $H : \{0,1\}^{2n} \rightarrow \{0,1\}^n$. For simplicity, the tree is full and the height is h. The $i-th$ leaf node is denoted as m_i. Tree Chaining could be viewed as a hash function $H' : \{0,1\}^* \rightarrow \{0,1\}^n$ such that the root is $H'(m_1, m_2, \cdots, m_{2^h})$.

$H'(\cdot)$ is collision resistant. Assume that there is a probabilistic polynomial-time attacker $A(s)$ could output $x = (m_1, m_2, \cdots, m_{2^h}) \neq x' = (m'_1, m'_2, \cdots, m'_{2^h})$ such that $H'(x) = H'(x')$ with probability $\varepsilon(n)$. Collision of $H'(x)$ means that the two root nodes corresponding to x and x' are equal. Suppose one internal node computed from x is equal to its counterpart computed from x'. There are two possible cases for their children nodes:

1) At least one node computed from x differs from its counterpart computed from x'.

2) The nodes for x are identical to those for x'.

If case 1 is true, one collision of $H(\cdot)$ is found which contradicts the initial assumption that $H(\cdot)$ is collision resistant. If case 2 is true, recursively compare nodes of lower levels until case 1 becomes true. At least one time of case 1 should be encountered; otherwise $x = x'$. Therefore, that $H'(\cdot)$ of Tree-chaining is collision resistant is proved by contradiction.

The proof of Star Chaining being collision resistant is similar to the one for Tree Chaining: If an adversary outputs a collision, then we can use the result to find the collision of the underlying hash function.

4.4.2 Defending against Biased Sampling Attacks

For DTC, random permutation is leveraged to sample data from the cloud. The property of uniformity is preserved by random permutation. Random permutation is an implemented of Fisher-Yates shuffle [154]. It has been proved that after random permutation any element can be placed at any position with equal probability. Since the implementation outputs the first m elements of random permutation, the uniformity is preserved.

For GSC, which sample block events belongs to is uniquely determined by a non-cryptographic uniform random hashing function. As a result, any sample block contains uniformly drawn events.

4.4.3 Defending against Dishonest Coordinators

The sampling protocol also defends against dishonest coordinators which do not execute the protocol in a correct way. For example, a dishonest coordinator may stop monitoring sensing devices by intentionally setting a negligible sample rate. The most difficult part is to check the final round that sampling protocol terminates at. **The analysis assumes that any sensing device, whether compromised or not, does not collude with the coordinator.**

Theorem 1. *The counter values from all sensing devices can be used to compute which round the sampling protocol terminates.*

Proof. Suppose the sampling protocol stops at round j at the end of the epoch, which means all counters associated with sample block i $(i \geq j)$ are finally stored in the cloud. $\sum_k l_i^k = l'_i$ $(i \geq j)$ and hence

$$g = \sum_{i}^{i \geq j} l'_i = \sum_{k,i}^{i \geq j} l_i^k \leq B \tag{4.1}$$

On the other hand, the necessary condition for sampling algorithm to promote from round $j-1$ to j is that $\sum_i^{i \geq j-1} l'_i > B$. Since $\forall i, \sum_k l_i^k \geq l'_i$,

$$\sum_{k,i}^{i \geq j-1} l_i^k \geq \sum_{i}^{i \geq j-1} l'_i > B \tag{4.2}$$

Combining Eq. (4.1) and Eq. (4.2), it is not hard to reach

$$j = \underset{\hat{j}}{\operatorname{argmin}} \left\{ \sum_{k,i}^{i \geq \hat{j}} l_i^k \leq B \right\} \tag{4.3}$$

As a result, the final round and therefore the sample blocks should be stored in the cloud can be computed from the counters. □

4.5 Performance Analysis

Merkle tree can be constructed in the form of k-degree tree as well. For the extreme case, Merkle tree authentication is degraded to star chaining when k exceeds the number of events.

4.5.1 K-degree Dynamic Tree Chaining

Suppose the IoT device detects n events in one epoch. Therefore, the height of the k-degree Merkle tree is $O(\log_k n)$. The sibling nodes in the path to the root are needed to compute the root hash value for signature verification. As a result, $O(k \log_k n)$ hash values are attached to every event. For the same reason, the signer maintains at least $O(k \log_k n)$ nodes to dynamically update the authentication tree.

The time to sign a set of events consists of three parts: *authentication tree building time*, *root signing time* and *packet generation time*. The time to build the authentication tree is proportional to the number of nodes in the tree. In the k-degree Merkle tree summarizing n events, there are $\frac{1}{k}n + \frac{1}{k^2}n + \ldots = O(\frac{1}{k-1}n)$ internal nodes in total. The value of each internal node is computed by taking the hashing of the concatenation of all its children. Suppose the time complexity of the hashing function is $O(l)$, where l is the length of the input string. (This is the case for most hash functions such as MD5 [24] and SHA-1 [50].) In this case, computing one internal node takes $O\left(k * O(1)\right) = O(k)$ time units. With $O(\frac{1}{k-1}n)$ internal nodes, the time complexity to compute all internal node values is $O(n)$. For the leaf node, its value equals the hashing of raw data. Denote l_i as the data length of event e_i, the time complexity of computing leaf nodes is $O(\sum_{i=1}^{n} l_i)$. The overall authentication construction time is thus $O(n + \sum_{i=1}^{n} l_i)$.

Signing the root is one public-key encryption operation, which is the same for any degree of Merkle tree.

Packet generation is to append necessary verification information to each event. In DTC, packet generation is offloaded to the cloud. Following what discussed above, $O(k \log_k n)$ time units are required to append all hash values to the event for verification.

If the receiver is granted enough space to cache all the internal nodes, the time complexity to build the authentication tree in the receiver side is identical to that in the sender side. Accordingly, one public-key decryption is applied to the signature of the authentication tree root.

4.5.2 Geometric Star Chaining

Following the notation in the last subsection, the number of events detected in one epoch is presented as n. Recall that hash value of sample block is updated as $D = h\left(h(e_i)||D\right)$ upon a new event e_i. The time complexity to update the sample block is thus $O\left(l + O(1)\right)$. In GSC, an individual event does not go through packet generation phase. Instead, one whole sample block is only associated with one piece of verification information which significantly reduces the packet signing time consumption. In conclusion, the time to sign n events in one epoch for GSC is $O(n + \sum_{i=1}^{n} l_i)$ plus one public-key encryption.

The main advantage of GSC over DTC is its constant space complexity. The memory footprint is only bounded by the number of sample blocks.

4.5.3 Sampling Protocol

Both the sensing device and the coordinator are sensitive to space consumption. Since the space consumption for the events themselves is the same, the analysis concentrates on the space used for different signature schemes. The space complexities of different signature schemes, which are compatible with the sampling protocol, at the sensing device and the coordinator respectively, are shown in Table 4.4, where n is the number of events monitored at one device and n' denotes the maximum value of n among all sensing devices. The space complexity of DTC at the coordinator is $\mathrm{O}(B \log n')$ because there are $\mathrm{O}(B)$ events buffered at the coordinator and in the worse case each event is appended with $\mathrm{O}(\log n')$ hash values for verification. Following the same line of reasoning in Section 4.2.2, the lack of buffer space in the coordinator may significantly degrade the performance of DTC. GSC requires $O(K)$ space at the coordinator because the coordinator maintains the sample block digests for all K devices. Note that space complexity discussed above is for the verification metadata and the space complexity of the raw data is always $\mathrm{O}(B)$ for all signature schemes.

The communication cost of the sampling protocol is the same as the protocol proposed by Cormode et al. [87], which is proved to be optimal in terms of

Table 4.4: Space Complexity of different signature schemes.

Signature Scheme	Device	Coordinator
Sign-each	O(1)	O(B)
DTC	O($\log n$)	O($B \log n'$)
GSC	O(1)	O(K)

communication cost, which is $O(K \log_{K/B} \sigma + B \log \sigma)$ with high probability.

4.6 Evaluation

Two widely used encryption algorithms, RSA-1024 [152] and DSA-512 [10] are from OpenSSL library [52] . MD5 [24], SHA-1 or SHA-256 [50] are leveraged as the message digest function.

4.6.1 Experiment Setup and Methodology

Dataset. The dataset [1] includes a wide variety of 90-day sensing data collected from sensing devices at three homes. 7 sources event data from the dataset selected to represent the event reports generated at sensing devices: environmental information (including temperature and humidity, etc.) about homeA, homeB and homeC respectively; electrical data from dimmable and non-dimmable switches for homeA; two sets of operational data on door and furnace on/off for homeA; the data from the motion detector located at homeA. Each record is encoded into 4 bytes: 2 bytes for timestamp and 2 bytes for sensor reading.

Hardware configuration. The prototype emulation experiments are conducted on a quadcore@3.40GHz Linux desktop with 32GB memory and a Raspberry Pi 3 Model B with a quad-core 64-bit ARM Cortex A53@1.2GHz. For all prototype experiments, only one core is used.

Methodology. The simulation experiment takes the budget limit into consideration. Conducting the simulation experiment servers two purposes: 1) Investigate the sampling protocol. 2) More importantly, the data trace of the simulation experiment will be feed the prototype experiment. For example, the simulation experiment computes which events should be sent to the coordinator and which events should be discarded locally. Only the events sent to the coordinator are feed to the prototype to evaluate signature generation through/speed at full speed. The prototype experiments are conducted on both the Linux desktop and the Raspberry Pi board to represent IoT device. Unless otherwise explicitly expressed, the default hardware platform to conduct prototype experiment is the Linux desktop. The throughput/speed is not tested in real distributed settings, because the data trace (*e.g.* sparse event data) may not be able to stimulate the IoT device to run at full speed. The prototype experiment is not conducted on M3 (the cryptographic operation performances are evaluated on this IoT platform) because it is hard to replay trace on the M3 board. The prototype experiment first tests the signing and verifying performance without sampling protocol involved under varied parameters. Next, prototype experiment is conducted in a setting where sampling protocol is involved. The second prototype experiment focuses more on the potential maximal throughput of tested signature schemes, since other impacting factors are explored in the prior prototype experiment.

4.6.2 Simulation Result

Figure 4.5 illustrates how the sampling protocol proceeds when new events arrive, where the budget limit is fixed to 500 events. The three lines in Figure 4.5 represent the number of events buffered at the coordinator, sent to the coordinator by all the 7 sensing devices and monitored at all sensing devices, respectively. The three lines vary against time in one day (May 1st, 2012). Initially, the number of

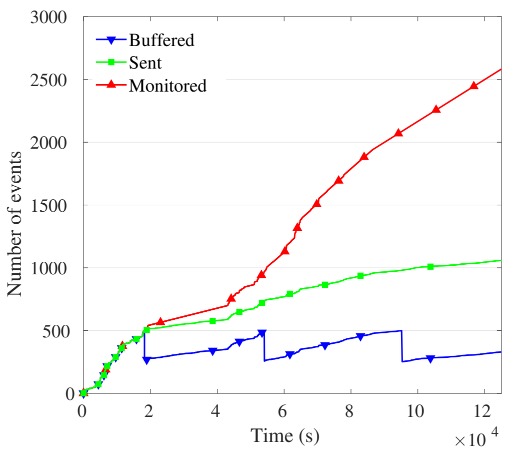


Figure 4.5: One-day micro-scale experiment.

events is the same for the three lines until the number of buffered events at the coordinator reaches the budget limit. At this time, approximately half buffered events are discarded, illustrated as the first vertical drop in Figure 4.5. Events at the coordinator then are accumulated over time until the next sharp decrease. This process repeats down to the end of this epoch. It is evident that the space used at the coordinator never exceeds the budget limit. It is worth mentioning that the total number of events sent to the coordinator grows slower with the time, which is a desirable property since the communication cost stays low even if much more events are monitored. This simulation experiment, to some extent, validates the theoretical analysis on the communication cost in which the communication cost only grow logarithmically. From Figure 4.5, totally 1057 events were sent to the coordinator on May 1st, 2012. On the other hand, there were totally 2572 events monitored on that day.

Next, how different values of budget limit impact the number of events eventually saved to the cloud is investigated. Figure 4.6 presents the number of events saved at the cloud each day from May 1st, 2012 to July 31st, 2012. with different values of budget limit. From the description of the sampling protocol, the

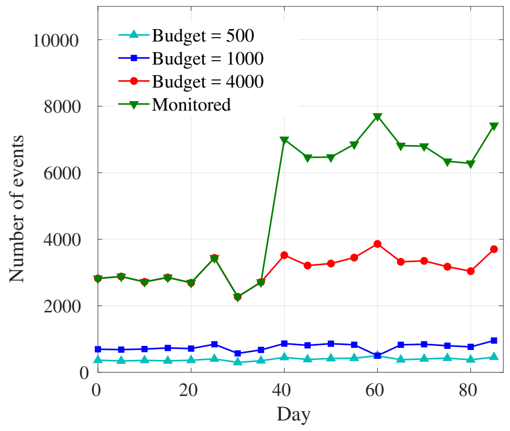
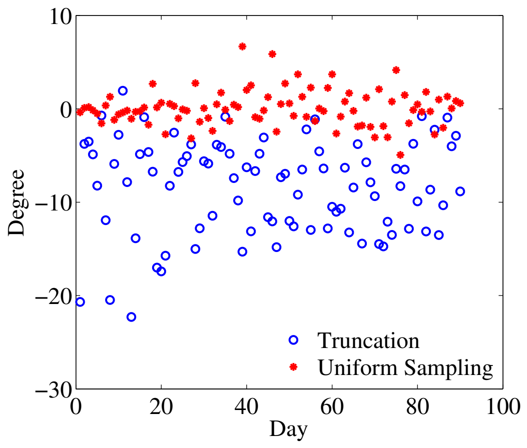


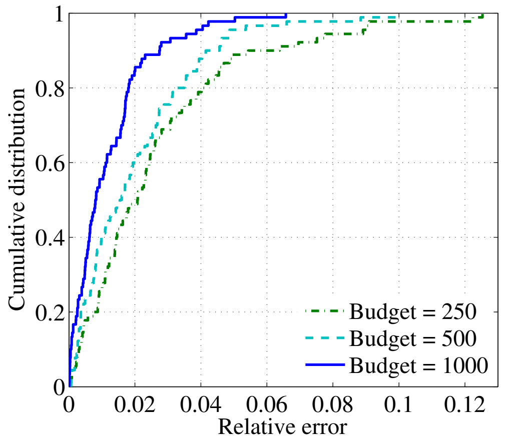
Figure 4.6: #events saved in the cloud.

final number of saved events is not necessarily equal to the budget limit. Figure 4.6 shows that this sampling protocol utilizes approximately 75% of the budget on average for different budget values. Figure 4.6 demonstrates that this sampling protocol works correctly in the presence of drastic changes, as the number of events monitored soars at the 40th day. In this case, the sampling protocol does not violate the budget constraints. On condition that the total number of monitored events is smaller than the budget limit, the sampling protocol saves all of them in the cloud, as is exemplified by the case where $budget = 4000$ in Figure 4.6.

The underlying foundation of the sampling protocol is that uniformly sampling is ensured. The importance of uniformity is demonstrated by one real application. The temperature sensor periodically measures the environmental temperature and sends the sensing data to the cloud for archiving purpose. The ground truth is the mean of all temperate sensing data from the temperature sensor. In order to illustrate the need for uniformity, the average temperature by the first 40 truncated sensing data (which is greater than the number of saved data in the cloud under most circumstances in this simulation experiment) is shown in Figure 4.7a.



(a) Deviation from the ground truth (budget = 500)



(b) Impact of budget limit

Figure 4.7: Computing average temperature from data saved in the cloud.

Figure 4.7a demonstrates how estimated average temperature deviates from the true one with respect to using the sampling protocol and using naive truncation, when the budget limit is fixed to 500. In this example, the truncated data are measured in the morning. Thus, the average temperatures calculated by truncated data are smaller than real average temperatures in nearly all days (only one day is an exception). How the value of budget limit affects the accuracy of estimated average temperature is evaluated. As expected, a greater value of budget limit yields more accurate results, as illustrated in Figure 4.7b.

4.6.3 Prototype Emulation Experiment Without Budget Limit

The efficiency of the signature scheme used greatly impacts the adoption of sensing devices, since most sensing devices are resource-constraint. As an indirect measurement of power consumption, the speed of signing/verifying under different parameter settings is evaluated.

The 7 data sources each divided into 90 epochs are the input to the singing phase of the signature scheme, whose output feeds the verifying phase afterwards. No budget constraints are involved and therefore all signed events are stored in the cloud. The parameter space consists of the space available at the signer/verifier as well as the sampling rate of the data application. The space cost at both the signer and the verifier to host the events themselves is orthogonal to the choice of signature scheme. Thus, the space cost in this subsection is in particular referred to the memory footprint of the signature scheme. It can be implied from the algorithm descriptions in Section 5.4.5 that the memory usage for all signature schemes mentioned in this chapter is a multiple of the length of the message digest function. In order to simplify the presentation, one unit of space cost is referred as the memory space used for storing one message digest. DSA is applied to the public encryption/decryption and MD5 is utilized as the message digest function in this subsection.

First of all, quantitatively comparison on the performance of the signature schemes is shown in Table 4.2. In this set of experiments, no limits are placed onto the signer/receiver space usage and the data consumer's sampling rate is set to be 50%. The encryption algorithm is RSA-1024 and the hashing function to compute digest is MD5. Especially, the amortized receiver communication cost is computed as the $\frac{\#received\ bytes}{\#records\ of\ interest}$. The results are listed in Table 4.5. As

Table 4.5: Quantitative comparison of different signature schemes.

Signature Scheme	Encryption time	Sender communication cost	Receiver communication cost
Sign-each	2.82 ms	132 bytes	132 bytes
Concatenate	0.59 ms	4.43 bytes	8.89 bytes
Hash chaining	0.57 ms	4.43 bytes	8.89 bytes
DTC	0.55 ms	4.45 bytes	4.52 bytes
GSC	0.41 ms	4.43 bytes	4.43 bytes

expected, the amortized time to sign one record much larger for Sign-each method. Since the signature length of RSA-1024 is 128 bytes, the amortized sender/receiver communication cost is $128 + 4 = 132$ bytes. For all other signature schemes, their amortized encryption time is similar. For Concatenate and Hash chaining, the receiver's communication cost is approximately twice as large as the sender's because only half of the received records are of interest.

To focus on the impact of the space issues at the signer side, enough free space is allocated to the verification process and the receiver takes all data stored in the cloud. If DTC is used, once the buffer in the signer is full, the root node in the authentication tree is signed and the remaining nodes are flushed to the cloud to spare space for upcoming events. Thus, lacking space in the signer may lead to multiple expensive encryption operations in one epoch. Furthermore, the same number of decryption operations are also needed at the verifier side. On the other hand, the available space affects the resolution of the sample blocks, rather than signing speed. Only one encryption operation is performed in a single epoch. Figure 4.8 illustrates the signing/verifying performance comparison between GSC and DTC under varied space available at the signer. Both signing and verifying performance of DTC are capped by available memory at the signer, whereas GSC runs at full speed all the time.

The sampling rate at the receiver side also affects the verifying performance

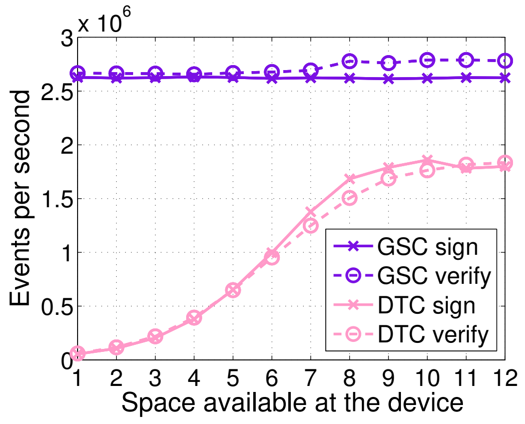


Figure 4.8: Throughput comparison

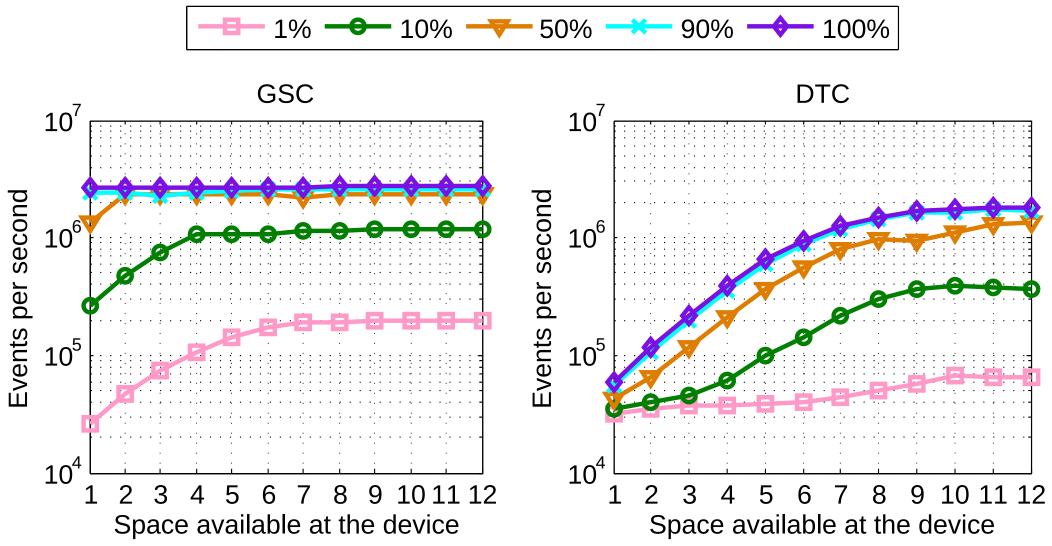


Figure 4.9: Verifying throughput comparison with different sample rate.

for both GSC and DTC, because it directly determines the number of events to share the cost of encryption/decryption, as depicted in Figure 4.9, where higher sampling rate yields better verifying throughput. Another observation from Figure 4.9 is that sampling rate also impacts GSC in terms of the space needed in the signer to achieve the maximal verifying throughput. Recall that the available space in the signer defines the resolution of sample blocks. The unused but verified events decrease as the resolution of the sample block improves. Suppose the receiver asks for 10% data in the cloud. In the case where there are 2 units of space in the signer, the receiver fetches and verifies 30% unused data because the finest sample block contains 50% data. If the available space increases to 3 units, the smallest sample block consists 25% data and thus the unused data shrinks to 15%. The time wasted for unused data becomes increasingly prominent when the sampling rate decreases. Therefore, smaller the sampling rate, more space required at the signer. Nevertheless, as small as 7 units of space are enough to support maximal verifying throughput when the sampling rate is 1%.

Moreover, the verifying throughput varies with the space allocated to cache verified nodes in the authentication tree for DSC. In the current prototype implementation, the verifier stops caching new verified nodes if the buffer is full. As expected, the performance acceleration is more evident with more cached verified nodes, as illustrated in Figure 4.10. Before any of the three lines in Figure 4.10 reaches full speed, for a given fixed space at the verifier, the verifying throughput is higher when there is less space available in the signer. This is because the locality of the cache nodes favors higher refreshing frequency. When smaller space is available at the signer, the number of jointly signed events is less and thus the cached nodes refresh quicker.

This chapter also measures time to build the Merkle tree and to generate

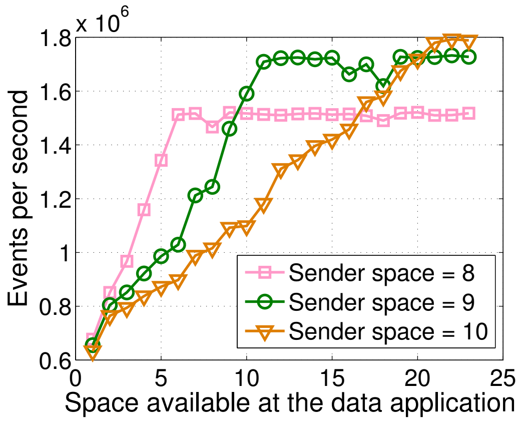


Figure 4.10: DTC reveiving throughput

packets for K-degree DTC. In this set of experiments, various synthesized data traces feed the machine running K-degree DTC. MD5 is used as the cryptographic hash function. For each trace, the experiment runs for 10 times. Figure 4.11 depicts the medians of per-packet amortized time consumption for Merkle tree building and packet generation. In terms of per-packet time to build the Merkle tree of a certain degree, it remains stable with the varying number of events, which conforms the time complexity analysis in Section 4.5.1. In Figure 4.11, the lines lightly decline because more events share the initial setup overhead. It is faster to build the K-degree Merkle tree when $K = 4$ than when $K = 2$ and $K = 3$, even though they all share one asymptotic time complexity. The performance gap is mainly due to the asymptotic time complexity of hash functions cannot describe the time consumption in K-degree Merkle tree building accurately. In Section 4.5.1, it is assumed that the time complexity of the cryptographic hashing function is $O(l)$, where l is the length of the input string. However, the actual time consumption is proportional to the number blocks the input data are chopped into. For MD5, the block size is 512 bits which can host 4 digests. As a result, computing an internal node of a 4-degree Merkle tree is **not** more expensive than computing a 2-degree Merkle tree internal node in terms of performing the

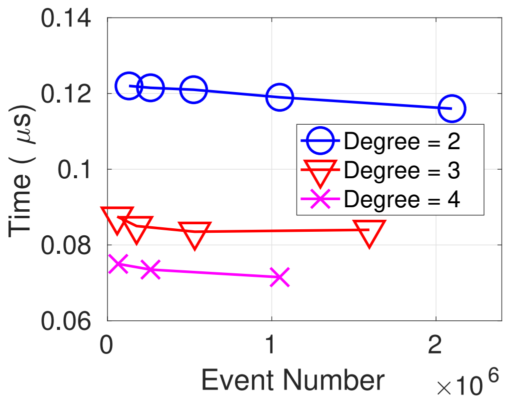
Table 4.6: Per-packet Generation Time with Fixed Height $h = 3$

Degree (K)	500	600	700	800	900	1000
DTC (μs)	4.219	5.035	5.933	7.078	7.925	8.800

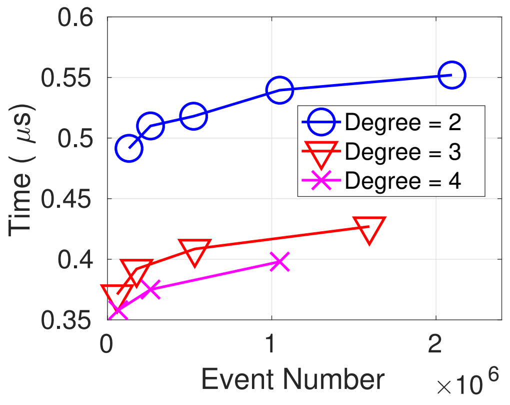
hash function. Building a 4-degree Merkle tree requires less hashing operations compared to the other two, hence smaller per-packet amortized time to build the Merkle tree. The per-packet generation time grows with more events as illustrated in Figure 4.11. This is expected from the theoretical analysis which indicates that the time complexity is $O(\log_k n)$, where n is the number of events signed under a same Merkle tree. From Figure 4.11, 4-degree Merkle tree is more efficient in generating the packet due to the fact that appending multiple hashes from the same level in the Merkle tree in batch is efficient. The per-packet generation time with fixed height $h = 3$ is evaluated to validate its asymptotic time complexity. As shown in Table 4.6, the packet generation overhead is more prominent with larger tree degree.

4.6.4 Prototype Experiment in Raspberry Pi

Experiments are conducted on a Raspberry Pi 3 Model B board to compare the power and time consumption of GSC and DTC. The two metrics are especially important for resource-constrained IoT devices. The experiment setup is exactly the same as the PC experiment which has been described in Section 4.6.3, except that the program is running in one Raspberry Pi instead of a more powerful PC. An inline power meter measures he power consumption overhead to finish the asigning process. The voltage remains at 5.1V all the time but the current jumps from 0.23A in the idle state to 0.37A when the signing program starts. In the



(a) Amortized building time



(b) Amortized packet generation time

Figure 4.11: K-degree DTC Merkle tree exp.

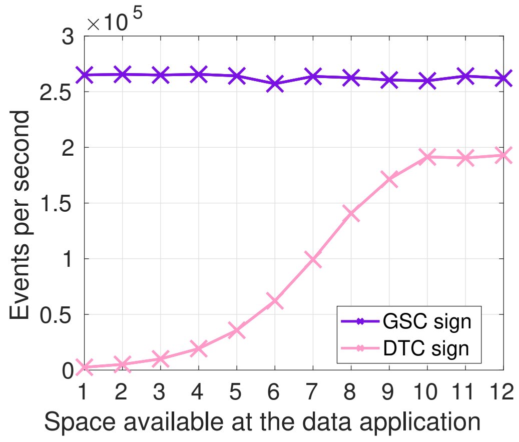


Figure 4.12: Signing throughput at RPi

experiment, the energy consumption due to the signing program is calculate as

$$E_{sign} = E_{total} - P_{idle} \times time \tag{4.4}$$

where E_{total} and $time$ can be read from the power meter. P_{idle} is computed as $5.1V \times 0.23A = 1.18W$. As shown in Section 4.6.3, the signing speed for DTC is limited by the available memory space. The situation is also true on Raspberry Pi where the signing speeds for both GSC and DTC are about 10 times slower than those in the PC experiment. Figure 4.12 illustrates the signing speed. When the program is running, the voltage and current remain stable, hence the power. Therefore, the power consumption is proportional to the program running time. For DTC, the limited memory space elongates the programming running time, leading to higher power consumption, as depicted in Figure 4.13.

4.6.5 Prototype Experiment with Sampling Protocol

From the prototype experiment without budget limit, it seems that the space requirement, $\log n$ units, at the signer is trivial, where n is the number of event reports generated in one sensing device. If the sampling protocol is utilized, the

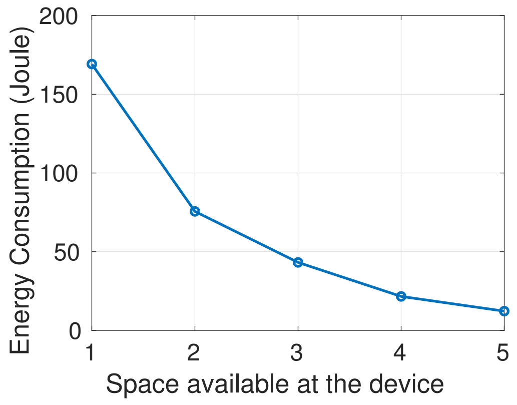


Figure 4.13: DTC energy comsumption

Budget	500	1000
Signature	115821	186619
Verify	35038	70296

Table 4.7: Simulation results regards the number of events.

signing/verifying performance is likely to be limited by the space available at the coordinator. The spacial cost to host auxiliary authentication information is $B \log n$, where B could be very large. Suppose the space available at the coordinator is C. It is equivalent to the situation where there are $\frac{C}{B}$ units of space in the signer. Since Figure 4.8 has already illustrated the impact of space in one single signer, how signing/verifying throughput changes with varied available space in the coordinator is not shown for brevity.

The evaluation focuses more on the potential maximal throughput of tested signature schemes. Suppose there is enough space at both the signer and verifier sides. The events sent to the coordinator are used for signing performance evaluation whereas the verification algorithm is fed by the events saved at the cloud. The number of events involved is listed in Table 4.6.5.

Figure 4.14 shows the throughput comparison between GSC and DTC. Since each day is one epoch and there are only 7 sensing devices, there are only $90 \times$

$7 = 630$ encryption/decryption operations for both GSC and DTC. For all the experiments conducted in this subsection, GSC is faster than DTC in terms of both signing and verifying. This is because processing an event in GSC is simpler than tree traversal in DTC, but not more hashing operations. This conjecture can be verified by analyzing the performance comparison in Figure 4.14. The performance gap between GSC and DTC becomes more prominent when quicker message digest function is applied. For example, the throughput gap increases from 0.35M events per second to 0.7M events per second if MD5 replaces SHA256 in the prototype emulation experiment using DSA signature and the budget limit is 500. The throughput decreases when the value of budget limit is reduced as implicated in Figure 4.14, because the same number of encryption/decryption operations are amortized to fewer events.

4.7 Conclusion

This chapter first summarizes the new challenges of the IoT data communication framework with authenticity and integrity and argue that existing solutions cannot be easily adopted. This chapter then illustrates the design of a protocol suit to address these challenges. The sub-systems in this protocol suit symbiotically operate together and this protocol suit is efficient in terms of space and time, as is validated by extensive simulation and prototype emulation experiments.

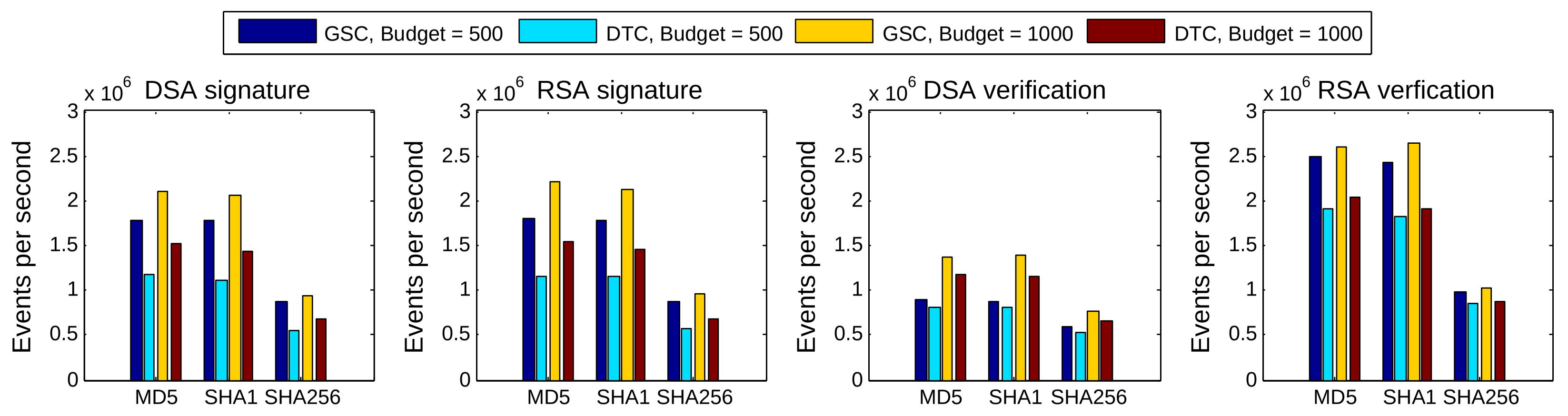


Figure 4.14: Throughput comparison with different parameter settings.

Chapter 5

An IoT Data Management System for Verifiable Range Queries

This chapter presents VERID, a verifiable IoT data management protocol suit that provides data consumers the ability to retrieve data on customized searching criteria. This feature is not supported in both DTC and GSC described in Chapter 4.

VERID supports many SQL-style range selection and aggregation operations including AVG, MAX, MIN, COUNT, SUM and MEDIAN.

1) Selection (σ). $\sigma_C(r) = \{t \in r | C(t)\}$. A selection operation $\sigma_C(r)$ over relation r returns all the tuples in the relation r meeting the condition C. C can be used to specific the range of a range query on indexed attributes (dimensions).

2) Aggregate ($\mathcal{G}$). Aggregate function f maps a set of values into a single value. Common aggregate functions include AVG, MAX, MIN, COUNT, SUM. In addition, VERID supports MEDIAN and its generalization p-th percentile.

To satisfy the requirements of IoT applications, VERID enables a number of SQL-like *ranged selection and aggregate queries* of sensing data while imposing minimal overhead for resource-constraint IoT devices.

The general problem of verifiable database outsourcing has been studied for over a decade [140, 110, 180, 179, 120, 81, 61]. A common approach is that a data publisher also uploads an *Authentication Data Structures* (ADS) to the cloud and keeps updating it. An ADS is a data structure signed by the IoT device using its private key. When a *Data Consumer* (DC) receives the data from the cloud, it also gets a number of *Verification Objects* (VOs) which are constructed by the cloud from the ADS and can be used to verify the data correctness. However, existing solutions are not fully suitable or optimized for the IoT framework. The hardware constraints, deployment features and application requirements of IoT demand a new design of verifiable data management.

Computation efficiency. IoT devices are usually limited in computation power. In addition energy efficiency may be another top concern. Hence solutions based on computation-intensive cryptographic operations [180, 179, 19] are not appropriate for IoT applications.

Memory efficiency. IoT devices are constrained by memory capacity as well. Solutions based on multi-way trees [120, 81] are not efficient on disk-less IoT devices with limited available memory. For dynamic database outsourcing schemes using memory-friendly ADSes [140, 180], these ADSes however grow fast with #update. When the ADS exceeds memory limit, it is signed and then flushed to the cloud. In this case a query may result in a jumbo VO constructed from an overwhelming number of ADSes. vSQL [179] uses an information-theoretic interactive proof system to achieve small memory footprint. However it assumes the data publisher and consumer are the same device hence vSQL is not applicable to IoT environments.

Communication efficiency. Communication of IoT devices is often more power-consuming than computation by orders of magnitude [93, 96]. Therefore, the sizes of updates from IoT devices to the cloud is an important metric. Multiway-tree based schemes such as AAR-tree [120] configure the ADS node size to that of a page. When the IoT device updates its ADS, entire stale nodes (nodes different from the previous version) are transmitted to the cloud even with a small modification, which are in large size.

Multiple data publishers. Multiple homogeneous IoT devices being deployed to collectively monitor the physical environment is an unique feature of IoT. These homogeneous IoT devices form a *task group*. For example, one dataset [55] provides the outdoor temperature of areas in Rome collected by 289 taxicabs over 4 days. Existing data outsourcing methods do not exploit any opportunities from this unique feature, spatial locality.

Multi-dimensional range query. Many IoT devices are equipped with multiple sensors and IoT data analytics often rely on *range queries*, which may be in multi-dimension. Most existing IoT sensing datasets are multi-dimensional, such as those in [45, 1, 43, 18, 9]. For example, querying the data collected during 8-9AM and reporting temperature in 30-40°C is a two-dimensional range query.

Massive data. Massive data are generated over time from multiple devices. Hence storing and retrieving massive data and the corresponding ADSes in the cloud should be scalable. Some proposed schemes [180, 179] do not take scalability into account and the large in-memory ADSes incur both enormous capital expenditure and operating cost.

Monotonically increasing timestamps. Most IoT data carry timestamps (called 'epoch' in this chapter) that are monotonically increasing. The ADS may require careful re-design to avoid performance downgrading under this situation, such as those using unbalancing trees, e.g., VKD-tree [81]. Note the timestamp may not be the time dimension indexed for range queries. For example, analytics may be more interested in the time of 24 hours rather than the timestamps including dates and years.

5.1 Existing Solutions

Verifiable database outsourcing have been studied for over a decade and two broad categories of approaches are exerted towards this goal. General verifiable delegation of computation [83, 99, 67, 88, 160, 19] can handle any query on outsourced data which requires the data owner to compile the entire dataset into an arithmetic circuit. Circuit-based systems incur excessive proof construction overhead. A very recent work vSQL [179] improves the performance of this approach by combining an information-theoretic interactive proof system [86] and a polynomial-delegation protocol [142]. However, the overhead of vSQL is still high for practical uses.

On the other side, numerous prior works aim at verifying one or multiple specific data query types, including range query [180, 141, 140, 61, 81], data aggregation [120, 110], join [176, 181, 140], search over encrypted data [173], *etc.*. VERID falls into this category. Table 5.1 shows an qualitative comparison between representative database outsourcing schemes.

Chained Signature Approach [140]. The ADS is an authenticated and unforgeable linked list ordered by one dimension (such as time, temperature) over the dataset where each node contains the cryptographic hashes of its predecessor and successor. At query execution, all nodes falling in the query range are lined up to form the VO. The data consumer verifies authenticity and completeness of the

Table 5.1: Qualitative comparison of representative database outsourcing schemes. ○: efficient; ●: inefficient; ◐: inefficient in some situations/metics.

Scheme	**Category**	**Memory**	**Computation**	**Log ADS update**	**Multi-dimension**	**Comments**
BAS [140]	Signature-based	◐	○	✓	✗	inefficient for aggregate
APS-tree [110]	Prefix Sum	○	◐	✓	✓	coarse granularity
IntegriDB [180]	Set operation	●	●	✓	✓	crypto heavy
vSQL [179]	Circuit-based	○	●	✓	✓	computational intensive
AAR-tree [120]	Multiway-tree	◐	○	✓	✓	large VO size
VKD-tree [81]	Binary-tree	◐	○	✗	✓	unbalanced tree
CorrectDB [61]	Hardware-aid	○	○	✗	✗	trusted hardware
VERID (this work)	Binary-tree	○	○	✓	✓	holistic design for IoT

results by sequentially checking the signatures of the nodes in the VO. For data publishing, the newly inserted node along with its two neighbors are updated, re-signed and then uploaded to the cloud by the data publisher. Chained signature approaches perform three signing operations per data insertion which would be inefficient on IoT devices. This approach does not support aggregate queries such as SUM or multi-dimensional queries.

Prefix Sum. APS-tree [110] uses Prefix Sum for efficient validation of aggregate operations, specially on SUM. The basic idea is to pre-process the data at the data publisher's side such that the aggregated value could be easily assembled by those pre-processed values. Compared to signature-based approaches, Prefix Sum reduces the communication cost of aggregate queries to $O(1)$. Prefix Sum however suffers from inefficient update: A single update may trigger conducting pre-processing over the whole dataset in the worst case.

Authenticated Set Operations. Some prior works [180, 141] achieve authentication of set operations including *union*, *intersection*, and *set-difference*, which are powerful building blocks to compute multi-dimensional range queries. However they are very inefficient in computation. Each data publisher needs 1) computing q exponents with up to the s^qth powers where s is secret value and q is a big integer, and future ranged queries are limited to those whose results have cardinality less than q; 2) $O(\log |r|)$ encryptions per insertion ($|r|$ denotes #rows of the relation).

Tree-based Approaches. Tree-based approaches employ Merkle Hash Tree (MHT) [128] or its variants (*e.g.* Merle B+-tree [119, 103], Merkle R*-tree [176, 120]) as the core ADSes. During a query phase, the cloud traverses the tree to identify the query results and construct the tree traversal path as the proof to the data consumer. Based on the received proof information, the data consumer reconstructs and replays the path to verify the correctness of the query results. Most tree-based approaches employ disk-based multi-way search tree such as B+ tree and R* tree as the indexing structure. The node size of one Multiway-tree is configured to that of a page whose minimum size is 4KB on most architectures. When the IoT device updates the multi-way tree, the entire stale nodes are transmitted to the cloud even only a small fraction inside a stale node is modified.

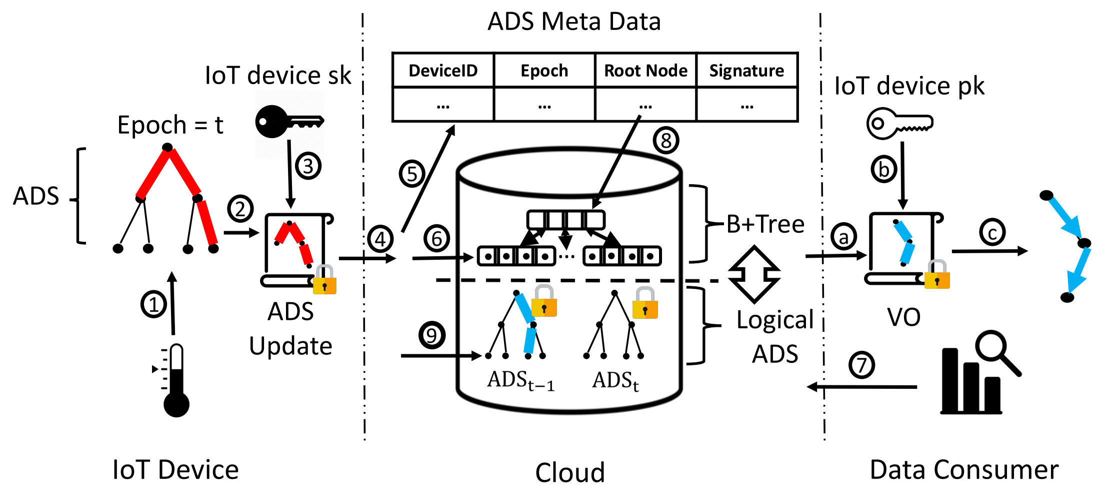
Trusted Hardware Aided Approaches. CorrectDB [61] and EnclaveDB [82] rely on the specialized trusted hardware Intel SGX [51] inside the cloud to conduct the heavy-lifting tasks in database outsourcing. Trusted hardware however can be attested by only one party. For IoT applications, it is impractical to pair every IoT device with one SGX in the cloud.

5.2 System Design

5.2.1 Overview

This chapter first dissects VERID into steps and depicts the design overview in Figure 5.1. Multiple IoT devices are sending data to the cloud simultaneously, but only one IoT device is demonstrated in the picture for ease of presentation. Let all IoT devices collaboratively perform a monitoring task be defined as a *task group*. All data from one single group follow a common schema.

Authentication Data Structure (ADS) is an indexing data structure whose operations can be carried out by an untrusted cloud and the result could be verified by data consumers. Each IoT device maintains and updates one ADS in accordance with the sensor readings during its entire life cycle. In VERID, the ADS is a new data structure PrefixMHT, which is essentially a binary search tree (BST) and can be authenticated similar to the Merkle Hash Tree (MHT) [128] (see de-



Step	Description
1	Sensor insertion
2	Build ADS update
3	Sign with secret key
4	Send signed ADS update to cloud
5	Update ADS metadata
6	Store ADS update in B+ tree
7	Send query to cloud
8	Retrieve root node to start search
9	Traverse logical ADS + build VO
a	Send VO to data consumer
b	Verify VO signature with IoT public key
c	Replay VO

Figure 5.1: VERID design overview

tails in Section 5.2.3). Therefore, VERID falls in the broad category of tree-based method. Upon new sensor readings, the IoT device inserts the value into PrefixMHT as a normal BST insertion except that all visited nodes are marked as *stale*. (**Step 1**). VERID updates the latest ADS to the cloud at a fixed time interval called *epoch*. Therefore, each PrefixMHT node includes an epoch attribute to indicate the epoch when the node is updated. The epoch attribute together with the node value can uniquely identify a PrefixMHT node over time and thus are collectively defined as the *NodeID* which is the key enabler for VERID to perform queries on historical data. At the end of each epoch, the IoT device updates the digests of the PrefixMHT by recomputing the hash of stale nodes from bottom up like all other MHT variants. As such, the hash of the root digest summarizes the whole PrefixMHT and therefore is also referred as the digest of the PrefixMHT. (**Step 2**). Afterwards, the IoT device signs the root node using its own private key (**Step 3**). Thanks to the tree structure of the PrefixMHT, the IoT device sends to the cloud only the stale nodes along with the new signature instead of the entire PrefixMHT (**Step 4**). Since BST insertion starts from the root, the root node for every epoch is always stale and included in every PrefixMHT update. To maintain the tree structure when serializing PrefixMHT nodes, the NodeIDs of both left and right children are explicitly included in each node. Note it is possible that one or two children are attributed to previous epochs.

When the PrefixMHT update is received, the cloud first stores the root NodeID as well as the signature in one structure for maintaining ADS Meta Data table. ADS Meta Data are used at the data query procedure described later. (**Step 5**). All other parts of the ADS update, *i.e.* the stale nodes, are stored at the leaf nodes of one B+ Tree, which indexes PrefixMHT nodes by their NodeIDs. Since the cloud has all the incremental updates history (*i.e.* PrefixMHT updates from all proceeding epochs), it is able to reconstruct the complete PrefixMHT of any epochs hitherto from PrefixMHT nodes stored in the B+ tree. As a result, multi-version *logical PrefixMHTs* embed in the B+ tree to answer data queries.

Steps 1-6 run repetitively during the whole lifecycle of the IoT device. On the other hand, one round of Steps 7-c are stimulated when the data consumer issues an data query to the cloud. The query may span multiple epochs across the IoT sensing devices. The same search criteria will be applied to all IoT devices in the task group (**Step 7**). Upon receiving and parsing the data query request, the cloud uses $(DeviceID, epoch)$ as the key to retrieve from ADS Meta Data table the root NodeIDs and the signatures of interest (**Step 8**). Given the root node, the logical PrefixMHT, basically a binary search tree, is traversed according to the search criteria specified by the data consumer. The PrefixMHT nodes on the search path in the logical PrefixMHTs are assembled as an unforgeable *Verification Object* (VO) used by the data consumer later to verify the query result: The VO consists of one or multiple partial PrefixMHTs which have already signed by the IoT device (**Step 9**). The query result, together with the VO and associated signatures, are returned to the data consumer (**Step a**). Following the signatures verification with the public key of the IoT device (**Step b**), the data consumer replay the searching path embedded in the VO to verify the query result finally (**Step c**).

The rest of this section elaborates on the design of the PrefixMHT for VERID as well as the concrete cloud storage solution.

5.2.2 Design of Prefix Tree

This subsection introduces a new design called *Prefix Tree* which enables efficient aggregation queries. Prefix Tree are motivated by the idea from Prefix Sum

[110] but are further extended to handle dynamic updates. This subsection then describes an ADS which embeds Prefix Tree in Merkle Hash Tree (MHT) called *PrefixMHT*.

Prefix Sum [110] performs the SUM operation over static dataset with low overhead. Given an integer array nums, Prefix Sum can efficiently find the sum of the elements between any two indices. The basic idea is to pre-process the array such that range sum query could be easily assembled by pre-processed values. The prefix sum PS of array M is an array and each element is: $PS\,[i] = \sum_{j=0}^{i} M\,[j]$. Given the prefix sum, the range sum can be computed as: $Sum([l,h]) = PS[h] - PS[l]$, where l and h denote the exclusive lower bound and inclusive upper bound of the range query respectively. To apply Prefix Sum in IoT applications to answer the question like "show the number of temperature readings between 5 and 13 degree", the sensor readings are bucketized and an array M is initiated to maintain the number of readings in each bucket. Obviously the number of buckets hence space complexity determines the query precision. More significantly, Prefix Sum suffers from inefficient update: A single update may trigger conducting pre-processing over the whole dataset in the worst case. Additionally, Prefix Sum wastes considerable space when the array itself is sparse, which is a common situation in IoT applications if the distribution of some sensor readings are highly skewed and the values are highly concentrated.

Prefix Tree thus is proposed to handle the dynamic updates while achieving the efficiency on aggregation queries. Prefix Tree does not suffer from space-precision dilemma. Each interval node of Prefix Tree maintains four attributes: *key*, *cardinality*, *sub-tree count* and *sub-tree sum*. The key attribute stores the searchable sensor reading value (*e.g.* temperature) and is used as the search key. All data items are sorted along the tree based on the key attribute. All future range and aggregate queries should be on the key attribute dimension. How this method can be extended to support multi-dimensional range queries is presented in Section 5.2.6. Cardinality refers to #elements having the associated value. Sub-tree count and sum summarize the corresponding sub-tree (including the node itself) which are analogous to the element of Prefix Sum array PS. Since the operations of the both aggregated values are similar, the following discussion concentrates on sub-tree count only. Prefix Tree enables efficient *prefix count* queries, like "show the number of temperature readings below 13 degree". Figure 5.2 gives an illustrative example to accomplish the query "SELECT COUNT(*) FROM value <= 13". In Figure 5.2, the two numbers inside each node attribute to the value and its cardinality respectively. For instance, 10(3) inside N_{01} indicates three instances of value 10. The numeric aside the link summarizes the total number of instances under the subtree: the *sub-tree count* attribute of lower node of this link. The prefix-count query starts from the root and traverses the Prefix Tree like a normal BST. *Count* is initialized to 0 in the beginning. On delving into the right child, count is increased by the difference of the sub-tree count of the parent node and that of the right-child node. Take Figure 5.2 for instance. On traversing from N_{root} to N_1 the count is increased from 0 to $14 - 7 = 7$, meaning that number of instances equal or smaller than 11 is exactly 7. On traversing from N_1 to N_{10}, the count does not change. When the leaf node N_{10} is reached, its cardinality is then added to the count if its value equals to the higher searching bound. In particular, the query result of the aforementioned example is $7+2 = 9$. Range sum query can be conducted similarly (*sub-tree sum* is not shown in Figure 5.2).

Upon insertion, only nodes on the path from the newly inserted node to the root are updated. Therefore, the insertion complexity is $\mathcal{O}$ $(\log n)$, where n is

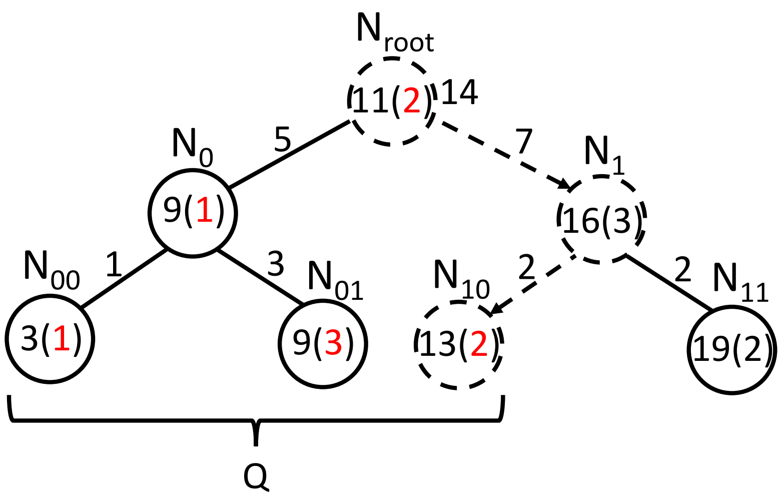


Figure 5.2: Intuition on PrefixTree

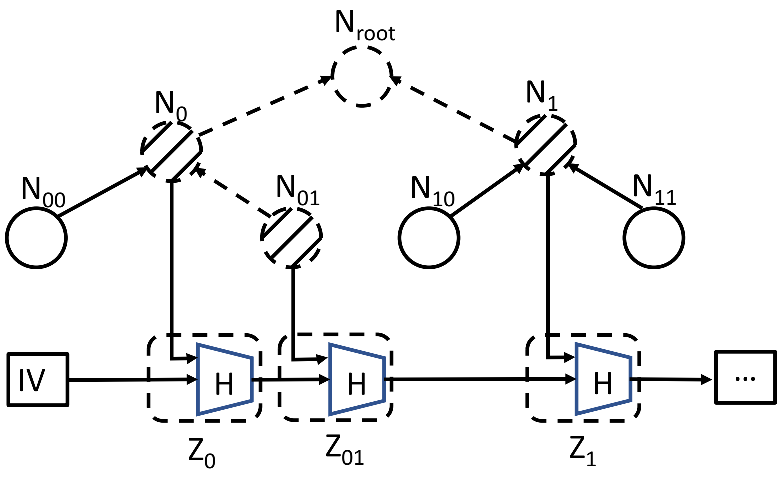


Figure 5.3: Illustration of PrefixMHT

#nodes in the Prefix Tree. Prefix Tree is far more efficient than Sum Prefix in dynamic settings.

5.2.3 Design of PrefixMHT

PrefixMHT is an ADS based on Prefix Tree. PrefixMHT allows a Prefix Tree to be authenticated in a fashion similar to a Merkle Hash Tree (MHT). **PrefixMHT ensures query correctness of both selection and aggregation queries.**

Each PrefixMHT node is composed of the following attributes: *key*, *cardinality*, *sub-tree count*, *sub-tree sum*, *epoch*, *lchild NodeID*, *rchild NodeID*, *hash accumulator*, *lchild hash* and *rchild hash.* The first four attributes are consistent to those of the Prefix Tree. The epoch attribute indicating when the node is updated which is indispensable to reconstructing multi-version ADSes in the cloud. In addition, PrefixMHT node stores the NodeIDs of its two children to enable logical tree traversal in the cloud.

Another data structure included in PrefixMHT is called a hash accumulator. The hash accumulator is associated with the Prefix Tree, which is used for authenticating the non-searable part of a sensor reading for every epoch. For example, an acoustics sensor measuring ambient background noise may send both the raw audio signal and the associated noise level to the cloud for storage. In this case, only the noise level can be indexed and authenticated through the Prefix Tree. Each raw signal is associated with a searchable key indexed by the Prefix Tree. To this end, VERID utilizes Merkle-Damgård construction to compute running digests of non-searchable raw signals for authentication. The running digests will be stored as the *hash accumulator* attribute in the PrefixMHT nodes. At the end of each epoch, the IoT device sorts all raw signals generated at that epoch in search key order and then constructs one *hash chain* in Merkle-Damgård fashion: The hash accumulator is computed by taking the hashing of the concatenation of its intermediate upstream hash accumulator and the hash of corresponding raw signal. The example in Figure 5.3 demonstrates one hash chain consisting of three raw signals, where Initial Vector (IV) is the hash chain starting point that has been hard-coded in the program. Compared to stand-alone hashes for authenticating individual raw signals, hash accumulators curtail *verification objects* (VO s) for verifying range selection queries. Suppose the range selection query for a certain epoch returns raw signals chained consecutively in the Merkle-Damgård construction. The authenticated hash accumulator corresponding to the close upper query bound on the hash chain, accompanied by unauthenticated open lower bound, suffices to verify the query result. For example, in order to authenticate the raw signals S_{01}, S_1 associated with N_{01} and N_1 respectively, the verifier checks $H\Big(H\Big(Z_0||H(S_{01})\Big)||H(S_1)\Big) \overset{?}{=} Z_1$.

In order to explain *lchild* (*rchild*) hash, the concept of *node hash* is introduced. Node hash encompasses all aforementioned attributes in the PrefixMHT node and is used for authentication at node level. The attribute *lchild* (*rchild*) hash is simply the node hash of its left (right) child. Therefore the node hash recursively summarizes the whole subtree. The root node hash is referred as the *digest* of the whole PrefixMHT and is signed by the private key to create the *digital signature*. The authentication of PrefixMHT can therefore be conducted in a top-down approach starting from the root node.

The IoT device does not create a PrefixMHT from scratch for every individual epoch. PrefixMHT incrementally accumulates cardinality, sub-tree count and sum over time. However, the hash accumulator is computed on a per-epoch basis. The directly impacted nodes due to insertion and nodes on their paths to the root are collectively defined as *stale* nodes because they all contribute to the incremental update. The prefix count $PC[k, t]$, for instance, can be interpreted as accumulated #insertions whose key values are smaller than or equal to k from epoch 0 to epoch t. As such, the answer for range count across multiple epochs are modified to:

$$\begin{aligned} &Count([l, h], [t_b, t_e]) \\ =&PC[h, t_e] - PC[l^-, t_e] - PC[h, t_b - 1] + PC[l^-, t_b - 1] \end{aligned} \tag{5.1}$$

The two inclusive key search bounds are denoted as l and h. Let t_b and t_e be the inclusive start and end epochs. These four parameters collectively defines the *search predicate*. The correctness of Eq. (5.1) can be easily checked by a Venn diagram.

Property 1. When a data consumer (DC) queries k, the cloud needs to find the PrefixMHT node hosting the largest *key* value that is smaller than or equal to k. Recall that PrefixMHT was constructed by the data publisher (*i.e.* IoT device in this chapter) and its root was signed using the private key of the publisher. PrefixMHT is a binary search tree such that the cloud can easily find the PrefixMHT node hosting the largest *key* value that is smaller than or equal to k by traversing the PrefixMHT. The path from that node to the root is called the *authentication path*. Given all PrefixMHT nodes on the authentication path, the verifier (*i.e.* data consumer) replays the tree traversal and checks the node hash in a top-down approach. The PrefixMHT signature allows the verifier to detect any tampered node. Hence **the DC can successfully verify that the result of querying k is the PrefixMHT node hosting the largest *key* value that is smaller than or equal to k.**

Property 2. When a data consumer (DC) queries k for a specific epoch, it can verify the position of the first node in the hash accumulator whose *key* is smaller than or equal to k. For a particular epoch, all stale nodes is a subgraph of the complete PrefixMHT which also forms a BST. Therefore the conclusion is implied by Property 1.

Proposition 5.2.1. *PrefixMHT ensures query correctness of both selection and aggregation queries.*

Demonstration of the proposition. For selection queries from a DC, the authenticity, integrity, and correctness of the results returned by the cloud can be verified by the hash accumulators corresponding to the boundary values, which can be authenticated through the Prefix Tree embedded in the PrefixMHT according to Property 2. One selection query spanning multiple epochs can be broken down into multiple per-epoch queries and be verified individually.

For aggregated queries, range COUNT (SUM) can be assembled by four prefix counts (sums) as shown in Eq (5.1). The prefix counts (sums) can be authenticated based on Property 1. COUNT and SUM are authenticated directly by PrefixMHT and calculating AVG requires one additional division operation. The data consumer can validate MAX/MIN result by issuing another COUNT query where the range exceeds the returned MAX/MIN value. If the result of the second query equals 0, the MAX/MIN value is finally validated. Authenticating MEAN and general p-th percentiles follows this two-stage process as well.

5.2.4 Efficient PrefixMHT Update

Upon the end of an epoch, the IoT device builds the hash chain and updates PrefixMHT according to the sensing data generated at that epoch. The root node is always marked with the new epoch even if no insertion occurs in that epoch; otherwise the cloud may discard data without being detected. The hash of the root node summaries a snapshot of the whole PrefixMHT, which is signed at every epoch by the IoT device using its own private key. The signing procedure is discussed in Section 5.3. Stale nodes and the PrefixMHT signature are transmitted in JSON format [16], together with the raw signals if there is any, to the cloud for storage. In IoT settings, communication is more energy-consuming than computation [93, 96]. By leveraging JSON, unassigned attributes such as the hash accumulator can be encoded succinctly to reduce communication cost. AVL tree is chosen for self-balancing among possible methods, according to the results shown in Section 5.4.4.

5.2.5 Storage in the Cloud

As shown in Figure 5.1 PrefixMHT updates are stored at the leaves of a B+ tree in the cloud. Most prior tree-based methods let the cloud create individual B+ trees, one for each IoT device, since different IoT devices host their own private keys. VERID however takes the opposite way: PrefixMHT updates from the a same task group are stored one per-group B+ tree. In the per-group B+ tree, a PrefixMHT node in an update is uniquely identified by a tuple of $(DeviceID, key, Epoch)$, where $DeviceID$ is augmented upon the node arriving at the cloud and *key* is the *key* field of the PrefixMHT node. How the updates are sorted in the B+ tree have a profound impact on the I/O cost. Through excessive experiments that the three attributes prioritized as $Epoch > key > DeviceID$ exhibits the best I/O performance. Basically, *Epoch* preserves least locality because the query can specify arbitrary starting and ending epochs. On the other hand, since the same query range is applied to all IoT devices in the same task group, clustering together all PrefixMHT nodes associated with the same *key* values exploits the spatial locality. Thus, $DeviceID$ possesses the lowest priority among the three constitutional attributes of the node identification.

Meanwhile, raw signals are stored at per-device Binary Large OBject (BLOB) files to preserve locality: Range selection queries demand consecutive sensor readings meeting the search criteria. Since the raw signals have already been sorted by the IoT device before transmission, newly arrived raw signals are directly appended to the BLOB file. In-file offsets of different sensor readings are saved at the corresponding PrefixMHT nodes to index the BLOB file.

Buffer management is an integral part of VERID, which caches the content of disk reads in the memory to reduce I/O cost. The basic management unit is one *page*, a fixed-length contiguous disk block (*e.g.* 4KB). Upon read from the disk, the page is cached at one of the page frames in the buffer pool. Some cache replacement algorithm decides which page frame the page should resides in and writes its former resident back to the disk. In VERID, its buffer management uses Clock replacement algorithm [163] for its simplicity. VERID may leverage buffer management to exploit the spatial locality to significantly reduce I/O cost. For example some PrefixMHT nodes, especially those near the root are repetitively retried to construct VO.

5.2.6 Extending to Multi-dimensional Data

The previously discussed PrefixMHT is a binary search tree which can only handle one-dimensional data. However, most IoT devices carry multiple sensors and as a result, IoT data are mainly high-dimensional. VERID rests on Space-filling Curves (SFCs) to map a high dimensional data point to a one 1-D value. SFC is a bijection from multi-dimensional discrete universe to one-dimensional universe of the same cardinality. Notable examples of SFC include C-curve, Z-curve (a.k.a. Morton order) [132] and Hilbert curve [131]. Therefore, by utilizing SFC, one high-dimensional range is transformed to one or multiple 1-D segments which can be computed efficiently using the divide & conquer strategy [137]. *Clustering number* [131] is proposed to capture #segments during range query processing. From the viewpoint of VERID, smaller clustering number leads to shorter VO size. For IoT applications, the range query usually exhibits some constraints or patterns. For example, in geographical IoT applications, each query zone (*e.g.* from citywide to street block) is bounded to one 2-D query range with fixed length and width. As a result, VERID employs the query-aware QUILTS [135] as the space-filling curve. QUILTS is a framework optimizing clustering number by choosing one SFC from a family of *Bit-Merging Curves* [135] based on the query pattern. IoT devices are also in favor of QUILTS over other SFC such as Hilbert curve [131] due to the low computational overhead to map the multi-dimensional data point to 1-D QUILT point. The QUILT 1-D address is calculated by interweaving the binary representation of values from each dimension following a certain template. Suppose a 2-D point is represented as 2×4 bit-sequences $< x_1x_2x_3x_4, y_1y_2y_3y_4 >$. If the interweaving template is $xyxyxyxy$, the QUILT address in accordance is then $x_1y_1x_2y_2x_3y_3x_4y_4$. Likewise, the QUILT address becomes $y_1y_2y_3y_4x_1x_2x_3x_4$ when the template is $yyyyxxxx$. Due to space limit, the detailed construction of bit-interweaving template from a-priori information on query patterns is skipped.

QUILT enjoys a property that if two points are close in the high-dimensional space, it is highly likely that their 1-D counterparts are also nearby. With the help of SFC, validation of one range query is converted to authenticating its corresponding end points in the SFC space, which are indexed by the B+ trees in the SFC order. Therefore, VERID also provides the property of locality preserving described in Section 5.2.5. Other multi-dimensional search tree solutions (*e.g.* Merkle R*-tree [176, 120], VKD-tree [81]) do not maintain an unified order across IoT devices and hence have weak locality.

Authenticating one end point requires all nodes on the path to the root, which is named as *authentication path*. The authentication paths of different end points share significant number of common nodes. To save the communication cost, VERID detects and removes duplicated nodes in a VO.

5.3 Signature Scheme for VERID

This signature scheme allows aggregation of multi-user signatures to reduce both computation and communication costs from data consumer's perspective.

Multi-user signatures aggregation should be ubiquitous in VERID: The data query is applied to all IoT devices in a task group. Every device in the task group generates the signature with its own private key. Moreover, the signature scheme exploits the sparse setting of event detection applications where the IoT device generates data only when events of interests occur. These events, by their nature, are rare. As a result, the PrefixMHT of each IoT device is rarely updated except for the monotonically increasing epoch of the root node. Our signature takes advantage of this fact to further optimize the verification phase. The signature scheme introduces little overhead compared to existing solutions such as BGLS [71].

This section first gives the preliminaries to understand the signature scheme. After that, *Hash Fusion Signature* (HFS) generated at the IoT device side is described in Section 5.3.2. The signature aggregation scheme, *Condensed Bilinear Pairing* (CBP) and its verification are discussed in Section 5.3.3. How the signature scheme is optimized in sparse settings are illustrated in Section 5.3.4.

5.3.1 Preliminaries

Notations

Let λ denote the security parameter. $\upsilon(\cdot)$ represents a negligible function, which means $\exists L \in \mathbb{N}$, such that $\upsilon(x) < 1/f(x)$ for any $x > L$ and any polynomial function $f(\cdot)$. $x \leftarrow_R S$ means assigning x a uniformly dawn value from set S. If $A(\cdot)$ is a probabilistic algorithm, $y \leftarrow A(x)$ means A return its output to x. PPT is a shorthand for Probabilistic Polynomial-Time. $\{0,1\}^*$ represents the set of string of any length. Concatenation is written as $||$. $Pr[E]$ means the probability of the occurrence of event E.

Bilinear Pairing

CBP is based on an algebraic structure, namely *bilinear pairing*. Suppose $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic multiplicative groups of prime order p with the cyclic generators g_1 and g_2 respectively. $\mathbb{G}_T$ is another cyclic multiplicative group of the prime order p. A bilinear pairing is a map e: $\mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ satisfying 1) Bilinearity: $\forall u \in \mathbb{G}_1, \forall v \in \mathbb{G}_2, \forall a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u^a, v)^b = e(u, v^b)^a = e(u, v)^{ab}$. 2) Non-degeneracy: $e(g_1, g_2) \neq 1$. 3) Computability: An algorithm exists to compute the mapping efficiently. Bilinearity and non-degeneracy imply another two important property:

$$e(u_1 u_2, v) = e(u_1, v) e(u_2, v) \qquad \forall u_1, u_2 \in \mathbb{G}_1, \forall v \in \mathbb{G}_2 \tag{5.2}$$

$$e(\psi(u), v) = e(\psi(v), u) \qquad \forall u, v \in \mathbb{G}_2 \tag{5.3}$$

Here ψ is an efficient computable isomorphism function $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$ such that $\psi(g_2) = g_1$. The 7-tuple $bp := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$ defines a bilinear pairing.

5.3.2 Hash Fusion Signature

Hash Fusion Signature (HFS) is the building block of the signature scheme. HFS is a variant of BGLS signature scheme [71] which is constructed based on bilinear pairing. HFS is defined as a tuple $(KeyGen, Sig, Ver)$ consisting of three algorithms. Assume an algorithm $BilGen(1^\lambda)$ is available to setup the public parameters of bilinear paring, which is used as a subroutine in $KeyGen$. $bp := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2) \leftarrow BilGen(1^\lambda)$, where λ is the security parameter and p is a λ-bit prime. Let H: $\{0, 1\}^* \to \mathbb{G}_1$ be a full domain hash function modeled as a random oracle [66]. The message to be signed are composed of two sub-strings, *i.e.* $\overrightarrow{m} = m_1||m_2$. $\overrightarrow{m}[i]$ represents the *ith* component of $\overrightarrow{m}$, $i \geq 1$.

Definition 5.3.1 (HFS)**.** *Hash Fusion Signature (HFS) is a tuple of three algorithm $(KeyGen, Sig, Ver)$ as described below.*

KeyGen*$(\boldsymbol{1^\lambda})$: $bp \leftarrow BilGen(1^\lambda)$, pick a random secrete $s \leftarrow_R \mathbb{Z}_p^*$ and compute g_2^s. Set the public key $pk \leftarrow (g_2^s)$ and the private key $sk \leftarrow s$. The public parameter is $pp \leftarrow (bp, pk)$.*

Sig*$(\overrightarrow{\boldsymbol{m}}, \boldsymbol{sk})$: for message $\overrightarrow{m} = m_1||m_2$, let $h \leftarrow H(m_1) * H(m_2)$. The signature is $\sigma \leftarrow h^s$. Return $\alpha \leftarrow (\overrightarrow{m}, \sigma, pk)$.*

Ver*$(\overrightarrow{\boldsymbol{m}}, \boldsymbol{\sigma}, \boldsymbol{pk})$: for message $\overrightarrow{m} = m_1||m_2$, let $h \leftarrow H(m_1) * H(m_2)$. Check $e(g_1^h, g_2^s) \stackrel{?}{=} e(\sigma, g_2)$. If the two terms are equal, return 1; otherwise return $\perp$.*

In the case of VERID, m_2 is the epoch and m_1 represents other content of PrefixMHT root node. The purpose to isolate epoch is to align the signature scheme with the highly structured communication pattern of IoT applications where only the epoch increases and other part stays unchanged most of the time. This design is the key enabler to reduce communication and computation complexity of sparse settings described in Section 5.3.4.

The security of HFS is captured by a standard notation, *Existential Unforgeability under Chosen Message Attack* (EU-CMA) [102]. EU-CMA of HFS is defined by an experiment where the advantage of any PPT adversary $\mathcal{A}$ is negligible. In this experiment, $\mathcal{A}$ is provided with a signing oracles $HFS.Sig_{sk}(\cdot)$. The signing oracle $HFS.Sig_{sk}(\cdot)$ returns the signature of the input under private key sk. $\mathcal{A}$ can adaptively choose the messages as the input to the signing oracle, but it can only query the signing oracles for up to $poly(\lambda)$ times, where $poly(\cdot)$ can be any polynomial function and λ denotes the security parameter. $\mathcal{A}$ finally returns a forgery $(\overrightarrow{m}^*, \sigma^*)$ under pk, where $\mathcal{A}$ did not query the signing oracle on $\overrightarrow{m}^*$ before. $\mathcal{A}$ wins iff $Ver(\sigma^*, \overrightarrow{m}^*, pk) = 1$. EU-CMA of HFS can be expressed formally as:

Definition 5.3.2 (EU-CMA of HFS). *HFS is Existential Unforgeable Secure under Chosen Message Attack if the following formula holds.*

$$Adv_{HFS}^{EU-CMA}(\mathcal{A}) = \Pr\left[\begin{array}{l}(sk, pk) \leftarrow HFS.KeyGen(1^{\lambda}) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{HFS.Sig_{sk}(\cdot)}(pk) \\ 1 \leftarrow HFS.Ver(\sigma^*, m^*, pk)\end{array}\right] < \upsilon(\lambda)$$

Even though a HFS signature can be viewed as the aggregation of two BGLS signatures from a same user, the definition of unforgeability of BGLS is slightly different from that of HFS. EU-CMA of HFS focuses on $m_1||m_2$ as a whole whereas the notion of unforgeability in BGLS captures individual parts. The proof of the unforgeability of HFS is provided in Appendix A.1 under the assumption given in the following theorem.

THEOREM 5.3.1. *Hash Fusion Signature Scheme is EU-CMA secure with the assumption of Computational Co-CDH hardness under random oracle model.*

5.3.3 Condensed Bilinear Pairing

Imagine a fire alarm system with thousands of sensors deployed in the forest. Individual sensor sends an alarm only when detecting hazardous situations. If high temperature never occur in the forest, the sensor periodically generates *default messages* indicating safety, *i.e.* a PrefixMHT node with sub-tree count = 0. Since the content of the default message has been known priori, sending default messages on the network is not necessary. The epoch attribute is the same across sensors. In the case of most sensors sending default messages, Condensed Bilinear Pairing (CBP) can accelerate the verifying speed and save communication cost.

Definition 5.3.3 (CBP). *Condensed Bilinear Pairing (CBP) is a tuple of six algorithms* $(KeyGen, Sig, Ver, PkAgg, SigAgg, VerAgg)$. *The default message is* *denoted as $M||t$ where M is the static content and t is the epoch. $SigAgg$ and $VerAgg$ are only applicable to messages from the same epoch, and hence identical epoch value. $KeyGen$, Sig and Ver have already been formalized in Definition 5.3.1. $SigAgg$ and $VerAgg$ are described below.*

$\boldsymbol{PkAgg(pk_1, pk_2, \cdots, pk_n)}$*: given public keys for every IoT device in a task group, the aggregated public key is* $apk = \prod_{i=1}^{n} pk_i$

$\boldsymbol{SigAgg(\alpha_i, \alpha_j)}$*: for any two HFS signatures,* $\alpha_i = (\overrightarrow{m}_i, \sigma_i, pk_i)$ *and* $\alpha_j = (\overrightarrow{m}_j, \sigma_j, pk_j)$*, if* $\overrightarrow{m}_i[2] \neq \overrightarrow{m}_j[2]$*, return* $\perp$*. We define* $\widetilde{\Gamma_i} = (\overrightarrow{m}_i[1], pk_i)$ *if* $\overrightarrow{m}[1] \neq M$*; otherwise* $\widetilde{\Gamma_i} = \emptyset$*.* $\widetilde{\Gamma_j}$ *is defined similarly. The aggregated signature is* $\alpha = (\widetilde{\Gamma_i} \sqcup \widetilde{\Gamma_j}, \sigma_i * \sigma_j, \overrightarrow{m}_i[2])$*, where* $\sqcup$ *represents the merging operation.*

$\boldsymbol{VerAgg(\alpha, apk)}$*: let the aggregated signature be* $\alpha = (\Gamma, \sigma, t)$ *where* $\Gamma = \{(\overrightarrow{m}_1[1], pk_1), \cdots, (\overrightarrow{m}_\kappa[1], pk_\kappa)\}$*. Define* $\widetilde{apk} = \frac{apk}{\prod_{i=1}^{\kappa} pk_i}$*.*
Check $e(\sigma, g_2) \overset{?}{=} e\left(H(M), \widetilde{apk}\right) * \prod_{i=1}^{\kappa} e\left(H(m_i), pk_i\right) * e\left(H(t), apk\right)$*. If the equation holds, return 1; otherwise return* $\perp$*.*

The following derivation provides the intuition on the correctness of CBP.

$$\begin{aligned} & e\left(H(M), \widetilde{apk}\right) * \prod_{(m_i, pk_i) \in \Gamma} e\left(H(m_i), pk_i\right) * e\left(H(t), apk\right) \\ = & e\left((H(M) * H(t)), \widetilde{apk}\right) * \prod_{(m_i, pk_i) \in \Gamma} e\left((H(m_i) * H(t)), pk_i\right) = e(\sigma, g_2) \end{aligned}$$

THEOREM 5.3.2. *Condensed Bilinear Pairing is EU-CMA secure with the assumption of Computational Co-CDH hardness under random oracle model.*

Appendix A.2 presents associated EU-CMA security definition and the proof for Theorem 5.3.2.

The unforgeability of aggregated signature is different from that of a single message, which requires that an adversary cannot generate a signature indicating the authenticity of an unsigned message, even if all other signers are dishonest.

Compared to the naïve method where the receiver verifies every single timestamped default message separately, our newly proposed scheme reduces both the communication cost and computation cost from $\mathcal{O}(k)$ to $\mathcal{O}(1)$, where k is #sensors. Even if the assumption that most sensors have not detected any events does not hold, CBP can still work but is degenerate into BGLS signature scheme.

5.3.4 Incorperating General Sparse Setting

In the sparse setting, most of roots of PrefixMHT remains untouched except for their monotonically increased epochs. However, applying CBP directly does not work in general because no default message is shared by most sensors in VERID.

If no event is detected between two epochs t^b (begin epoch) and t^e (end epoch), we can reach to:

$$e(\sigma_b/\sigma_e, g_2) = e\left(\frac{(H(M) * H(t_b))^s}{(H(M) * H(t_e))^s}, g_2\right) = e\left(\frac{H(t_b)}{H(t_e)}, g_2^s\right) \tag{5.4}$$

By verifying the above equality, the data consumer is able to ensure no new event being detected without knowing concrete M.

THEOREM 5.3.3. *Given two HFS signatures (σ_b, σ_e) and two epochs (t_b, t_e), $e(\sigma_b/\sigma_e, g_2) = e\left(\frac{H(t_b)}{H(t_e)}, g_2^s\right)$ indicates identical content at t_b and t_e except with negligible probability.*

The formal procedure to prove Theorem 5.3.3 is very similar to the techniques we used to prove Theorem 5.3.1. Intuitively, there is only negligible probability for a PPT adversary to find random oracle collisions.

The roots' signatures could be aggregated even with different M value, because the two instances of $H(M)$ at the denominator and the numerator in Eq. (5.4) are canceled out. Condensed Bilinear Pairing-VERID (CBP-VERID) opportunistically aggregates HFS signatures.

Definition 5.3.4 (CBP-VERID)**.** *Condensed Bilinear Pairing-VERID is a tuple of six algorithm* $(KeyGen, Sig, Ver, PkAgg, SigAgg, VerAgg)$*.* $KeyGen$*,* Sig*,* Ver *and* $SigAgg$ *are exactly the same as those of CBP presented in Definition 5.3.3. Inherent from CBP,* $SigAgg$ *and* $VerAgg$ *in CBP-VERID also require that the operated messages are from two epoches only: the begin and the end. For ease of representation, we define* augmented signature $\hat{\alpha} \triangleq (m_b, t_b, \sigma_b, m_e, t_e, \sigma_e, pk)$ *from a single device, where the subscription* b *and* e *stands for begin epoch and end epoch respectively.* m*,* t *are the two constructional components of the PrefixMHT root: content and epoch value. Aggregated signature is* $\Sigma = (\Gamma, \sigma, t)$*, where* Γ *concatenates changing content,* σ *is the product of individual signatures and* t *represents the epoch.*

$\boldsymbol{SigAgg(\hat{\alpha}_i, \Sigma^b, \Sigma^e)}$*: for the input,* $\hat{\alpha}_i = (m_i^b, t_i^b, \sigma_i^b, m_i^e, t_i^e, \sigma_i^e, pk_i)$*, If* $(t_i^b \neq \Sigma^b.t) \vee (t_i^e \neq \Sigma^e.t)$*, return* $\perp$*. Define* $\hat{\Gamma_i^b} = (m_i^b, pk_i)$*,* $\hat{\Gamma_i^e} = (m_i^e, pk_i)$ *if* $m_i^b \neq m_i^e$*; otherwise* $\hat{\Gamma_i^{b(e)}} = \emptyset$*. The aggregated signature is updated as follows.* $\Sigma^{b(e)}.\Gamma = \Sigma^{b(e)}.\Gamma \sqcup \hat{\Gamma_i^{b(e)}}$*,* $\Sigma^{b(e)}.\sigma = \Sigma^{b(e)}.\sigma * \sigma_i^{b(e)}$*.*

$\boldsymbol{VerAgg(\Sigma^b, \Sigma^e, apk)}$ **:** *if* $(|\Sigma^b.\Gamma| \neq |\Sigma^e.\Gamma|)$*, return* $\perp$*.*
Let $\Sigma^{b(e)}.\Gamma = \{(m_1^{b(e)}, pk_1), \cdots, (m_\kappa^{b(e)}, pk_\kappa)\}$ *Define* $\widetilde{apk} = \frac{apk}{\prod_{i=1}^{\kappa} pk_i}$*. Then check* $e\left(\sigma^b/\sigma^e, g_2\right) \overset{?}{=} e\left(H(t^b)/H(t^e), \widetilde{apk}\right) * \prod_{i=1}^{\kappa} e\left(H(m_i^b)/H(m_i^e), pk_i\right)$*. If the equation holds, return* 1*; otherwise return* $\perp$*.*

To speed up the computation of $\widetilde{apk}$ when more than half devices contribute Σ, $\widetilde{apk}$ can alternatively calculated as the product of their signatures.

THEOREM 5.3.4. *Condensed Bilinear Pairing-VERID is EU-CMA secure with the assumption of Computational Co-CDH hardness under random oracle model.*

The experiment to define EU-CMA and the corresponding proof are nearly the same as those of Condensed Bilinear Pairing in Appendix A.2, except that multiplication is replace with division.

5.4 Evaluation

The prototype of VERID includes all parts: the working programs on sensing devices, cloud, and DCs. All experiments are compiled by gcc 5.4.0. The cloud and DC instances run on one quadcore@3.40GHz Linux desktop with 32GB memory and each sensing device instance runs on a Raspberry Pi 3 Model B [53] equipped with a quad-core 64-bit ARM Cortex A53@1.2GHz as well as one 16GB flash drive. The experiments all are driven by real datasets.

5.4.1 Evaluation Methodology

Two publicly available IoT datasets are used for the experiments:

(1) **IntelLab [45]**: 54 Mica2Dot sensors with weather board deployed in the Intel Berkeley Research Lab collecting timestamped temperature, light, *etc.* once every 31 seconds. In the experiments, timestamp and temperature (in Fahrenheit) information is extracted to represent 1-D dataset. The queries for experiments are synthetic, which are 100 randomly generated aggregation count queries. For the synthetic queries, the lower temperature bound is a float number uniformly drawn from $[0, 200]$. The upper bound is set to always 20 degrees higher than the lower bound.

(2) **Rome [55]**: 289 taxicabs in Rome occasionally report outdoor temperature readings together with the GPS coordinates (*i.e.* longitude and *i.e.* latitude). The dataset spans 4 days and the tuple $(longitude, latidues, temperature)$ forms

Table 5.2: Summary of the Two Datasets

	IntelLab	**Rome**
#Devices	54	289
#Original readings	370667	4992
#Injections	387	110719
#Total readings	371054	115711

multi-dimensional sensing readings. VERID utilizes QUILT to map individual multi-dimensional readings into 1-D points. The temperature is represented by one 16-bit integer. Synthetic count queries are generated to simulate those from geographic information system applications, where the spatial range is displayed in different hierarchical levels: from city-wide to street block level. The experiments have 24 levels of hierarchical spatial query ranges. Given level l, the entire region in a $2^{24} \times 2^{24}$ grid, is partitioned into l^2 square areas with each being $2^{24-l} \times 2^{24-l}$. To synthesize one spatial range query, first uniformly generate one hierarchical level l and then choose a spatial range from the l^2 square areas at random. The temperature range $[0, 2^{16} - 1]$ is evenly partitioned into 16 sub-ranges and each synthetic query specified one sub-range to explore.

For both datasets, one epoch is set to 15 minutes and all experiments driven by the data are from the first 400 epochs, which is approximately 4 days. To make sure every epoch is signed, one *dummy message* are artificially injected indicating an empty epoch into the epochs that do not possess any sensor reading. For the two sets of synthetic queries, the start and end epochs are randomly drawn from $[1, 400]$. The datasets details are summarized in TABLE 5.2. Rome represents one sparse setting as can be inferred from the large injection to reading ratio. IntelLab obviously posits at the opposite side.

Methods to compare with. VERID compares with two other state-of-art works, Authenticated Aggregation R-tree (AAR-tree) [120] and IntegriDB [180].

The two works are selected among a large number of existing methods because 1) they support both aggregation and selection queries; and 2) they are relatively recent and demonstrate good performance compared to other methods. In fact, IntegriDB supports more operations (*e.g.* JOIN) than VERID does at extra cost. This chapter discusses and measures its performance in IoT scenarios for reference. For VERID and AAR, the capacity of the buffer pool is set to host 1000 frames of size 4KB. IntegriDB has a parameter q which determines the largest possible cardinality of verifiable query result. q is set to 1000 in the experiments.

Cryptographic algorithms to use. SHA-256 [49] in OpenSSL [52] are used as the cryptographic hash function for all the three works in the experiments. For both VERID and IntegriDB, the bilinear pairing for signature is Ate-paring [70, 48] on a 254-bit elliptic curve which is estimated to offer 128-bit security. Since the encryption scheme to generate signatures is not specified in the original paper of AAR, in the experiments it is specified as BGLS [71] on a 254-bit elliptic curve rather than the classic RSA [114] for fair comparison. From the measurement in generating signatures, BGLS is nearly $30X$ faster than RSA with 3072-bit keys which also approximately offers 128-bit security.

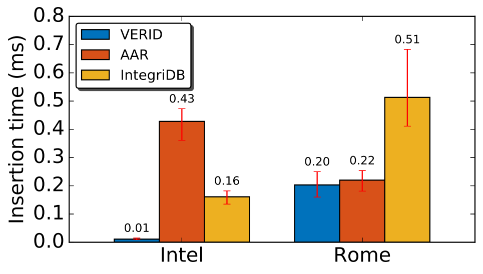
Methodology and Metrics. The prototype experiments on IoT devices are conducted on the Raspberry Pi board. The data trace contributed by each individual device is feed into the VERID prototype program at full speed. The timestamp from the trace drives the program to update the PrefixMHT digest and generate a signature when one epoch ends as indicated by the timestamp. The output of the program, *i.e.* what should be transmitted to the cloud, is stored locally at the flash drive. The evaluation measures the following metrics: **1)** Average time to process one insertion captures the computation costs at the IoT device side, which is denoted as the *insertion time.* **2)** The *ADS update cost* reflects the amortized communication cost which is computed by the size of all updates (excluding original data and signatures) divided by #insertions. **3)** The *memory* is an average memory usage of IoT devices. These three metrics for AAR and IntegriDB are measured on the Raspberry Pi in the same way.

After the Raspberry Pi experiments, the generated ADS updates and signatures along with original data are directly restored in the Linux desktop to study all other parts of VERID. Therefore, the experiments avoid the impact of varying networking environments. The synthetic queries are applied to all devices. VERID (or AAR/IntegriDB) constructs the VO when receiving the query request. The evaluation measures both **4)** the *VO construction time* and **5)** the *VO size*. The verification procedure starts in the same program immediately after the query result and its VO are produced. **6)** The *verification time* captures the computation efficiency at the data consumer's side.

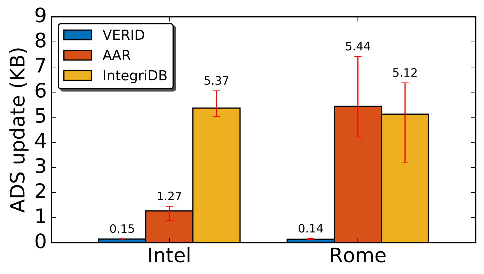
5.4.2 Aggregation Queries

Figure 5.4 demonstrates the aggregation queries performance results for IntelLab and Rome.

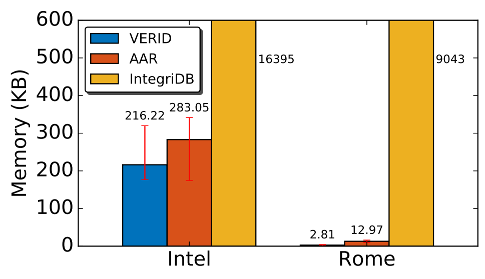
From the IoT devices' perspective, VERID outperforms AAR and IntegriDB in terms of computation (insertion time), communication (ADS update) and memory consumption for both datasets. Particularly, for VERID the insertion time in IntelLab experiments is smaller than that of Rome (Figure 5.4a) because more insertions amortize the signature generation time at each epoch. When the insertion is rare as in Rome dataset, the time to generate signatures dominates the computation cost and this explains the comparable insertion time for VERID and AAR in the experiment using Rome dataset. For AAR, the amortized insertion time increases in the IntelLab experiments due to excessive time spent on computing



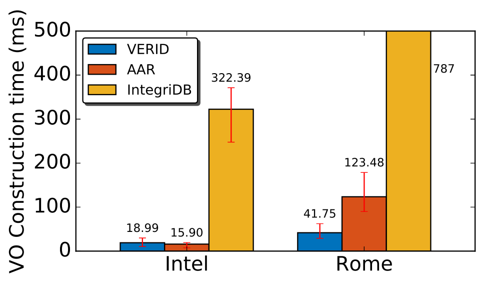
(a) Insertion time



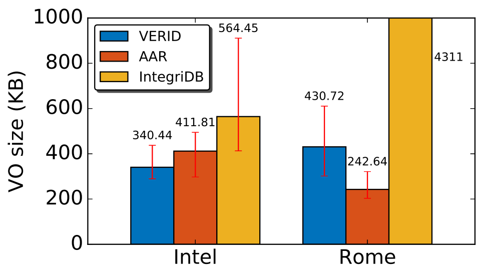
(b) ADS update size



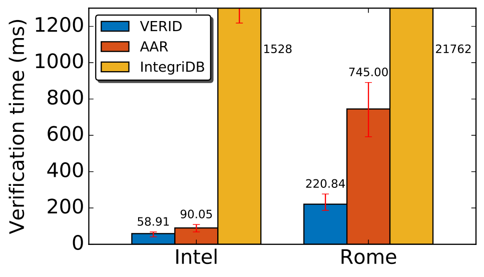
(c) Memory



(d) VO construction time



(e) VO size



(f) Verification time

Figure 5.4: Aggregation query performance results

hashes of 4-KB tree nodes. The long insertion time for IntegriDB is mainly due to excessive cryptographic operations. Since PrefixMHT of VERID is operated at fine grain, the ADS update size is much smaller than that of AAR which needs to update the whole stale 4-KB nodes (Figure 5.4b). The update size of AAR is worse when rare insertions amortize the update cost, as can be validated from the results of Rome. The average ADS size for VERID in the Rome experiments is slightly smaller than in the IntelLab experiments due to the lower PrefixMHT height. VERID is memory efficient and acquires additional memory space when a new PrefixMHT node is inserted (Figure 5.4c). On the other hand, AAR allocates memory at per-page basis. If the dataset is small, no enough readings are available to fill up the allocated memory space. Therefore, the memory usage gap between VERID and AAR is huge for Rome. Even if IntegriDB updates ADS at fine grain like VERID, its ADS node size is much larger hence ADS update size and memory footprint.

The mechanisms to construct VO for VERID and AAR are similar: recording the searching path on the ADS. The experiments conducted at the Linux desktop involves all devices collectively. For example, the VO construction time is the total time to find ADS paths for all 54 (387) devices in the IntelLab (Rome) experiments. VERID spends slightly more time to construct the VO than AAR does in the IntelLab experiments but the situation reverses in the Rome experiments where the data exhibit higher locality hence faster searching speed (Figure 5.4d). The VO size of AAR is smaller than that of VERID for Rome (Figure 5.4e), because the ADS of AAR is extended from R* tree, which is originally designed for GIS applications. VERID outperforms AAR regarding verification time for both datasets (Figure 5.4f) because the AAR data consumers need to compute excessive hashes to verify the query results. Another contributing factor is the signature scheme which avoids some bilinear pairing evaluations. IntegriDB constructs and verifies VO by doing complicated cryptographic operations, which consume considerable time.

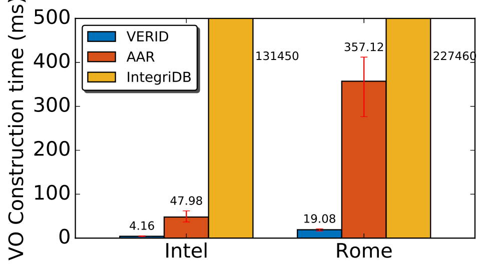
5.4.3 Selection Queries

Each selection query requests for data from a specific epoch, which represents the scenario where the data consumer retrieves data from latest epoch for further analysis. The selection queries are generated by modifying the range queries described in the experiment setup: The range exception for the epoch attribute stays the same and the epoch of interest is set to the upper epoch bound from the corresponding aggregation query.

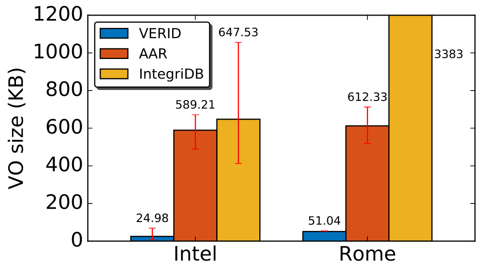
The VO construction time, VO size and verification time are reduced compared to the results from aggregation experiments as shown in Figure 5.5. Each range query requires 4 ADS paths as indicated by Eq.(5.1). On the other hand, the performance of AAR and IntegriDB downgrades significantly. The ADS of AAR relies on the R* tree. The range query within a single epoch can be geometrically viewed as a "long strip". R* tree is not good at processing "long strip" range queries. For IntegriDB, the mechanism of selection query is different from that of count queries.

5.4.4 Choose Self-balancing Tree

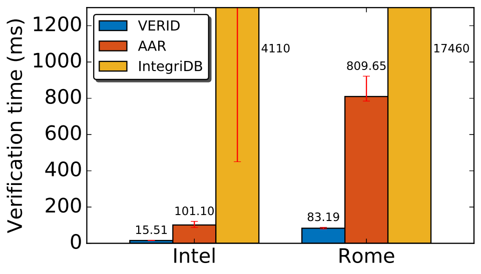
Figure 5.6 shows the communication cost using different BSTs to transmit stale nodes over time where input sensor readings are subject to normal, uniform, or Zipf distributions. The communication cost is conspicuously reduced for AVL Tree, which is used in VERID. The performance of Red-Black Tree (RBT) is even slightly inferior than plain BST without self-balancing. For RBT, the extra



(a) VO construction time



(b) VO size



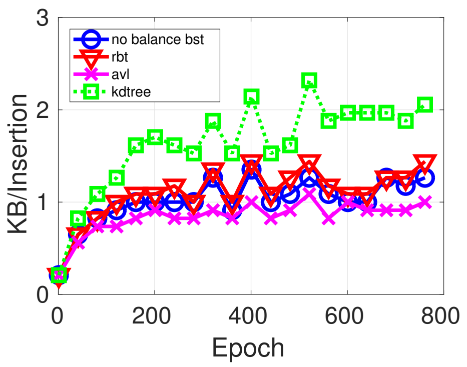
(c) Verification time

Figure 5.5: Selection query performance results

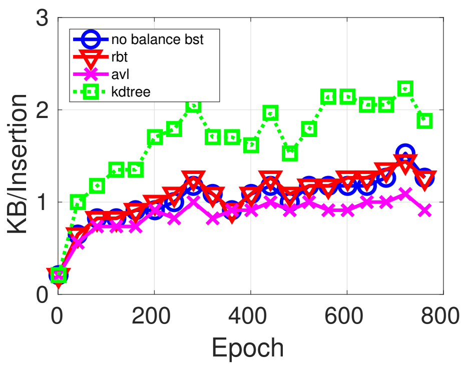
communication cost due to self-balancing such as rotation offsets the lower average height. It is also worth noting that communication cost growth rate declines significantly over time. New data are gradually inserted into the PrefixMHT and thus the later insertions have lower probability to create a new node. For the same reason, the normal distribution case incurs slightly less communication overhead compared to uniform distribution case. The cost of kd-tree is very high especially for Zipf distribution, as shown in Figure 5.6c.

5.4.5 Comparison of Signature Schemes

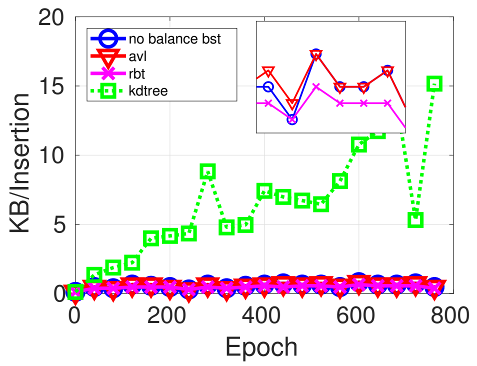
For VERID, validating signatures dominates the verification time at the data consumer's side: 99% and 98% verification time are spent on signature validation for IntelLab and Rome datasets respectively. VERID signature scheme reduces the signature validation time for sparse settings. For Rome dataset, VERID signature



(a) Normal Distribution



(b) Uniform Distribution



(c) Zipf Distribution

Figure 5.6: Update Communication Cost Comparison

Table 5.3: I/O Cost Comparison

#disk I/O times	**IntelLab**	**Rome**
Per-device B+ tree	20051	10125
Per-group B+ tree	5610	1209

scheme totally avoids 8143 out of 38700 PrefixMHT traversals in the cloud and bilinear pairing evaluations at the data consumer's side. This subsection evaluates the performance gain from VERID signature scheme by comparing it with BGLS [71]. The verification time is 307.19ms for BGLS and 220.84ms for VERID, an approximately 28% reduction. The VO construction time is reduced from 48.10ms (BGLS) to 41.75ms (VERID). Similarly, the VO size experiences a 19.8% decrease. In IntelLab dataset, nearly all devices have insertions for all epochs. Only 8/5400 PrefixMHT traversals thus are avoided in this case. As a result, there are no apparent changes on construction and verification time as well as VO size.

5.4.6 Disk I/O at Cloud Storage

VERID leverages the special query pattern of IoT application to build per-group B+ tree to store PrefixMHT nodes instead of using individual per-device B+ trees. The measurements of I/O cost, *i.e.* #disk reads for both IntelLab and Rome datasets are presented in this subsection. Per-group B+ tree greatly reduce the I/O cost as as illustrated in Table 5.3.

5.5 Conclusion

Due to the lack of studies of verifiable data management for IoT applications, VERID is designed to satisfy the unique properties and requirements. VERID is designed to resolve the unique requirements of IoT systems including computation, memory, and communication efficiency, multi-dimensional data, and monotonically increasing timestamps. The innovations of VERID include a new authentication data structure PrefixMHT and a novel signature aggregation scheme Condensed Bilinear Pairing. Experimental results show that VERID is much more efficient in memory, update, and time cost than prior works on both sensing devices and data consumers.

Chapter 6

Summary and Future Work

6.1 Summary

This dissertation consists of two parts to enhance the security of cloud. The first part focuses on the efficient and correct policy enforcement for network security infrastructure, *i.e.* network functions for security purposes. This dissertation presents one VNF orchestration framework called APPLE which provides interference-free policy enforcement. APPLE applies an optimization engine to determine VNF placement and a flow tagging scheme to reduce TCAM consumption and retain flow affinity. Results from both implementation and simulations using real network topologies and traffic matrices show that APPLE is resource efficient and can quickly react to traffic changes.

The second part targets at authenticity and integrity of emerging IoT applications. This dissertation identifies the hardware constraints, deployment features and application requirements of IoT that demand new designs of IoT communication and data management for authenticity and integrity. The proposed IoT communication framework, GSC, is able to uniformly sample data from sensing devices and then securely store the data in the cloud while respecting resource budget constraint. A distributed sampling protocol that is compatible with GSC is also proposed to incorporate budget limits. Extensive simulations and prototype emulation experiments driven by real IoT data show that the GSC is more efficient than alternative solutions in terms of time and space. This dissertation further introduces VERID, a verifiable IoT Data Management system. VERID provides an expressive interface that enables the data consumers to retrieve IoT data on customized multidimensional searching criteria. The innovations include a new authentication data structure PrefixMHT and a novel signature aggregation scheme Condensed Bilinear Pairing. Experimental results show that VERID is much more efficient in memory, update, and time cost than prior works on both sensing devices and data consumers.

6.2 Future Work

There are several directions can expand this dissertation, including both improvement to the works presented in this dissertation as well as long-term research projects.

6.2.1 Improvement to Current Projects

APPLE: When VNF instances processing packet, they consume multiple hardware resources (e.g. CPU cycles, NIC bandwidth). However, current VM hypervisor's resource scheduler only considers how to statically fairly share CPU and memory [107]. To integrate a max-min fair multi-resource scheduler for policy enforcement would be the future work.

GSC: Different sensing devices may send generated data to the coordinator at vastly different speed. The video surveillance system [75] continuously generates tons of data whereas human-motion detector [174] sends much less data occasionally. In order to avoid starvation of devices, the sampling protocol can be easily generalized to allow weighted items. How to automatically set weights for different devices with little human intervention could be the future work.

VERID: VERID only supports a subset of SQL language but it can integrate with verifiable computation frameworks such as Pantry [73] to conduct arbitrary computation including the rest of SQL operations. The PrefixMHT structure perfectly matches the data structure that Pantry can manage.

6.2.2 Long-term Research Projects

1) **Automatic management of smart-home IoT devices.** The number of smart-home IoT devices is ever increasing. However, to manually configure the variety of smart-home IoT devices to work collaboratively is cumbersome for users. Following the basic idea of APPLE, one future research direction could be designing a modular management system that automatically orchestrate smart-home IoT devices. This system also enforces some policies to provide security and privacy.

2) **Incorporate trusted hardware for IoT data manipulation.** Trusted computation result from the cloud is not provided in this dissertation work. Trusted hardware such as Intel SGX [51] provides trust and privacy on the arbitrary computing results which significantly improves the efficiency of various IoT applications. In my future work, I would discover and resolve the challenges in applying trusted hardware for IoT data manipulation.

Appendix A

Some Proofs on VERID Signature Scheme

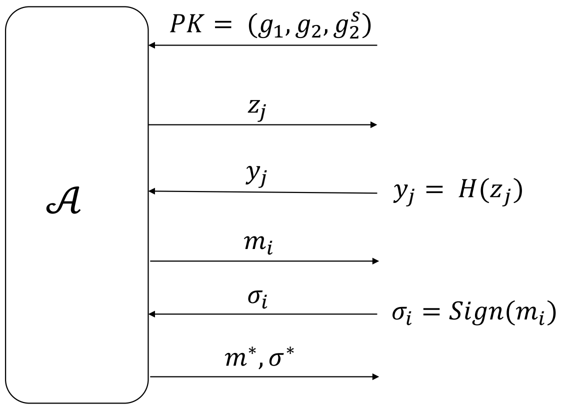
A.1 Proof of Unforgeability of HFS

We assume that there is no PPT adversary can compute $g_1^a \in \mathbb{G}_1$ given $g_2, g_2^a \in \mathbb{G}_2$ and $g_1 \in \mathbb{G}_1$ except with negligible probability $\upsilon(\lambda)$, where λ is the bit length of prime p.

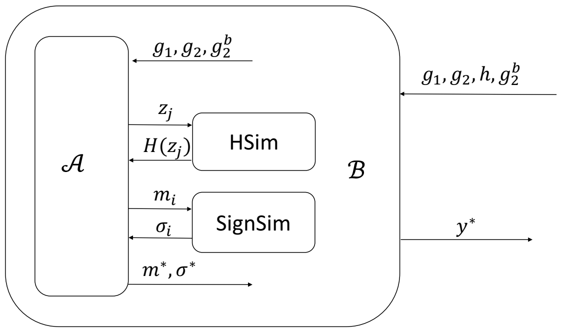
Assumption 1 (Co-CDH Problem)**.** *Let g_1 and g_2 be the generators of two multiplicative cyclic groups of order p respectively. For any PPT adversary, the following probability is negligible.*

$$Adv^{Co-CDH}(\mathcal{A}) = \Pr\left[\mathcal{A}(g_1, g_2, h, g_2^b) = g_1^{ab} : h \leftarrow_R \mathbb{G}_1, b \leftarrow_R \mathbb{Z}_p\right]$$

The pictorial view of HFS EU-CMA experiment (Definition 5.3.2) is given at Figure A.1a, where $H(\cdot)$ and $Sign(\cdot)$ stand for the *Random Oracle* and *Signing Oracle* respectively. To prove the the unforgeability, we construct a *reduction* in which any PPT adversary $\mathcal{A}$ breaking HFS EU-CMU experiment indicating the



(a) Illustration of Adversary $\mathcal{A}$



(b) Illustration of Adversary $\mathcal{B}$

Figure A.1: Pictorial reduction procedure

violation of Assumption 1.

The reduction is through another PPT adversary $\mathcal{B}$ who tries to solve Co-CDH Problem with non-negligible probability $\varepsilon(\lambda)$. $\mathcal{B}$ wins iff the output $y^* = g_1^{ab}$. Adversary $\mathcal{B}$ runs $\mathcal{A}$ with the public key $PK = (g_1, g_2, g_2^b)$ as a subroutine to solve the problem as illustrated in Figure A.1b. The adversary $\mathcal{A}$ is limited to query oracles provided by adversary $\mathcal{B}$. We assume that the forgery output of adversary $\mathcal{A}$ is $(\overrightarrow{m}^*, \sigma^*)$. Let $\overrightarrow{m}^* = m_1^*||m_2^*$. $\mathcal{A}$ should had queried m_1^* and m_2^* on the random oracle during the experiment. Suppose adversary $\mathcal{A}$ queries the random oracle q_H times in total, where q_H is any polynomial of λ. Adversary $\mathcal{B}$ "programs" $HSim$ and $SignSim$ to mimic the random oracle and the signing oracle from adversary $\mathcal{A}$'s view. If $\mathcal{A}$ succeeds in producing a forgery signature σ^*, the probability that it can be transformed to answer for Co-CDH problem is at least $1/q_H^2$. Adversary $\mathcal{A}$ has totally q_H^2 ways to pick up 2 out of q_H $HSim$ query results to construct $(H(m_1^*) * H(m_2^*))^a$. Therefore, the advantage of adversary $\mathcal{B}$ is:

$$Adv^{Co-CDH}(\mathcal{B}) = Pr[y^* = g_1^{ab}] > \frac{\varepsilon(\lambda)}{q_H^2}$$

which implies the violation of Assumption 1.

The formal reduction procedure at adversary $\mathcal{B}$ is presented as follows:

1) Choose $j_1^* \leftarrow \{1, \cdots, q_H\}$, $j_2^* \leftarrow \{1, \cdots, q_H\}$.

Let us assume $j_1^* < j_2^*$ without losing generality.

2) On a Random Oracle query z_j from $\mathcal{A}$:

(a) if $j \neq j_2^*$:

$g_1^{x_j} \leftarrow_R \mathbb{G}_1$; output $y_j = g_1^{x_j}$; add (x_j, z_j) to list L.

(b) otherwise:

output $y_j = h/z_{j_1^*}$

3) On signing query $m_i = m_{i1}||m_{i2}$, $m_{i1} = z_{j1}$ and $m_{i2} = z_{j2}$:

(a) Lookup z_{j1} and z_{j2} in list L. If neither is found, abort.

(b) if $(j1, j2 \neq j_1^*) \vee (j1, j2 \neq j_2^*)$:

output $\psi\left((g_2^b)^{x_{j1}} * (g_2^b)^{x_{j2}}\right) = (g_1^{x_{j1}} * g_1^{x_{j2}})^b$

(c) otherwise:

abort

4) Output σ^*

A.2 Proof of Unforgeability of CBP

The unforgeability is aggregated signature is different from that of a single message, which requires that an adversary cannot generate a signature indicating the authenticity of an unsigned message, even if all other singers are dishonest. Since the dishonest signers can manipulate (pk, sk) pairs, PPT adversary $\mathcal{A}$ is only provided with public key pk_1 chosen at random. Adversary $\mathcal{A}$ can adaptively choose the messages to query the singing oracle with pk_1. The forgery should contain pk_* and other $\kappa - 1$ (pk, sk) pairs as well as the messages to be authenticated. CBP is EU-CMA secure if the advantage of any PPT adversary $\mathcal{A}$ is negligible.

The reduction procedure is similar to what presented in Appendix A.1 except for how adversary $\mathcal{B}$ constructs y^* from the output of adversary $\mathcal{A}$. On receiving the output from $\mathcal{A}$, adversary $\mathcal{B}$ first checks that $h^{sk_i} = pk_i$ for $2 \leq i \leq \kappa$. Assume that adversary $\mathcal{A}$ succeeds in producing a forgery. If it is the case, adversary $\mathcal{B}$ recorded the discrete logarithms of $H(M)$, $H(m_i)$ and $H(t)$ in the list L, which are denoted as r, r_i and r_t respectively. If the forged message m^* belongs to the explicitly transmitted messages, we can compute h^b from the following equation with probability $1/q_H^2$:

$$e(\sigma, g_2) = e\left(H(M), \widetilde{apk}\right) * \prod_{(m_i, pk_i) \in \Gamma} e\left(H(m_i), pk_i\right) * e\left(H(t), apk\right)$$

$$= e(\psi(\widetilde{apk})^r, g_2) *$$

$$\prod_{(m_i, pk_i) \in \Gamma, m_i \neq m^*} e(\psi(pk_i)^{r_i}, g_2) * e(h^a, g_2) * e(\psi(apk)^{r_t}, g_2)$$

Therefore, h^a can be computed as:

$$\sigma * \left(\psi(\widetilde{apk})^r * \prod_{(m_i, pk_i) \in \Gamma, m_i \neq m^*} \psi(pk_i)^{r_i} * \psi(apk)^{r_t}\right)^{-1}$$

In the case that $m*$ is the static M and M was never queried against the signing oracle. Again, if adversary $\mathcal{A}$ produced a successful forgery, adversary $\mathcal{B}$ have access to the discrete logarithms r_i of $H(m_i)$. The derivation is as follows:

$$e(\sigma, g_2) = e\left(\prod_{\substack{i=1 \\ (m_i, pk_i) \notin \Gamma}}^{\kappa} h^{sk_i}, g_2\right) * e(h^a, g_2) * \prod_{(m_i, pk_i) \in \Gamma} e(\psi(pk)^{r_i}, g_2)$$

Thus, h^a in this case is:

$$\sigma * \left(\prod_{\substack{i=1 \\ (m_i, pk_i) \notin \Gamma}}^{\kappa} h^{sk_i} * \prod_{(m_i, pk_i) \in \Gamma} \psi(pk)^{r_i}\right)^{-1}$$

Bibliography

[1] http://traces.cs.umass.edu/index.php/Smart/Smart.

[2] Amazon S3 Pricing. https://aws.amazon.com/s3/pricing/.

[3] AWS Cloud Hacked by Bitcoin Miners. https://www.enterprisetech.com/2017/10/09/aws-cloud-hacked-bitcoin-miners/.

[4] AWS Cloud Hacked by Bitcoin Miners . https://www.enterprisetech.com/2017/10/09/aws-cloud-hacked-bitcoin-miners/.

[5] AWS customer success. https://aws.amazon.com/solutions/case-studies/.

[6] Broadband Network Gateway and Network Function Virtualization. https://www.broadband-forum.org/technical/download/TR-345.pdf.

[7] The caida ucsd anonymized internet traces 2013 - 2014. mar. http://www.caida.org/data/passive/passive_2013_dataset.xml.

[8] Computer hackers take to the cloud. https://www.sciencenewsforstudents.org/article/computer-hackers-take-cloud.

[9] Crawdad. https://crawdad.org/.

[10] Dsa. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[11] eHealth. http://www.who.int/topics/ehealth/en/.

[12] FIT IOT-Lab. https://www.iot-lab.info.

[13] Google cloud customers. https://cloud.google.com/customers/.

[14] HVAC Monitoring System. https://www.sensaphone.com/industries/hvac.php.

[15] IBM ILOG CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[16] Introducing json. https://www.json.org/.

[17] Iperf. https://iperf.fr/.

[18] Kaggle datasets. https://www.kaggle.com/datasets.

[19] libsnark. https://github.com/scipr-lab/libsnark.

[20] Libvirt virtualization API. http://libvirt.org/.

[21] Linux bridge. http://www.linuxfoundation.org/collaborate/workgroups/networking/bridge.

[22] Linux network namespace. http://man7.org/linux/man-pages/man8/ip-netns.8.html.

[23] M3 open node. https://www.iot-lab.info/hardware/m3/.

[24] The md5 message-digest algorithm. https://tools.ietf.org/html/rfc1321.

[25] Navigating a cloudy sky: Practical guidance and the state of cloud security. https://www.mcafee.com/enterprise/en-us/assets/executive-summaries/es-navigating-cloudy-sky.pdf.

[26] Nest. https://nest.com.

[27] Netcat: the tcp/ip swiss army. http://nc110.sourceforge.net/.

[28] Netfilter. http://www.netfilter.org/.

[29] Open vswitch. http://openvswitch.org/.

[30] Opendaylight. https://www.opendaylight.org/.

[31] OpenSensors. https://www.opensensors.io/.

[32] OpenStack. http://www.openstack.org/.

[33] P4. http://www.p4.org.

[34] Sampling for big data. www.kdd.org/kdd2014/tutorials/t10_part1.pptx.

[35] Samsung SSD 850 EVO $120GB, 250GB, 500GB\&1TB$ Review. http://www.anandtech.com/show/8747/samsung-ssd-850-evo-review/8.

[36] See the amazing things people are doing with Azure. https://azure.microsoft.com/en-us/case-studies/.

[37] Service Function Chaining Use Cases In Data Centers. https://datatracker.ietf.org/doc/draft-ietf-sfc-dc-use-cases/.

[38] Service function chaining use cases in data centers. http://datatracker.ietf.org/doc/draft-ietf-sfc-dc-use-cases/.

[39] Service Function Chaining Use Cases in Mobile Networks. https://datatracker.ietf.org/doc/draft-ietf-sfc-use-case-mobility/.

[40] The 10 Biggest Cloud Outages Of 2015. https://tinyurl.com/y7sxjf8m.

[41] The Abilene Observatory Data Collections. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/.

[42] The Seagate 600 & 600 Pro SSD Review. http://www.anandtech.com/show/6935/seagate-600-ssd-review/5.

[43] UCI machine learning repository. https://archive.ics.uci.edu/ml/datasets.html.

[44] xxHash. http://www.xxhash.com/.

[45] Intel lab data. http://db.csail.mit.edu/labdata/labdata.html, 2004.

[46] Telosb datasheet. http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf, 2005.

[47] Z1 datasheet. http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf, 2010.

[48] https://github.com/herumi/ate-pairing, 2015.

[49] Secure hash standard. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, 2015.

[50] Sha-1. http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, 2015.

[51] Intel SGX. https://software.intel.com/en-us/sgx, 2018.

[52] Openssl. https://www.openssl.org/, 2018.

[53] Raspberry pi. https://www.raspberrypi.org/, 2018.

[54] A. Abujoda and P. Papadimitriou. Midas: Middlebox discovery and selection for on-path flow processing. 2015.

[55] Mohannad A. Alswailim, Hossam S. Hassanein, and Mohammad Zulkernine. CRAWDAD dataset queensu/crowd_temperature (v. 2015-11-20): derived from roma/taxi (v. 2014-07-17). `https://crawdad.org/queensu/crowd_temperature/20151120`, 2015.

[56] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A. Sadeghi, and M. Schunter. Sana: secure and scalable aggregate network attestation. In *Proc. of ACM CCS*, 2016.

[57] A. Anand, V. Sekar, and A. Akella. SmartRE: an architecture for coordinated network-wide redundancy elimination. In *Proc. of ACM SIGCOMM*, 2009.

[58] B. Anwer, T. Benson, N. Feamster, and D. Levin. Programming slick network functions. In *Proc. of ACM SOSR*, 2015.

[59] A. AuYoung, Y. Ma, S. Banerjee, J. Lee, P. Sharma, Y. Turner, C. Liang, and J. C. Mogul. Democratic resolution of resource conflicts between sdn control programs. In *Proc. of ACM CoNext*, 2014.

[60] A. Bairley and G. G. Xie. Orchestrating network control functions via comprehensive trade-off exploration. In *Proc. of IEEE NFV-SDN*, 2016.

[61] S. Bajaj and R. Sion. CorrectDB: SQL engine with practical query authentication. 2013.

[62] H. Ballani, P. Costa, C. Gkantsidis, M. P. Grosvenor, T. Karagiannis, L. Koromilas, and G. O'Shea. Enabling end-host network functions. In *Proc. of ACM SIGCOMM*, 2015.

[63] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM SOSP*, 2003.

[64] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. On orchestrating virtual network functions. In *Proc. of IEEE CNSM*, 2015.

[65] B. A. Bash, J. W. Byers, and J. Considine. Approximately uniform random sampling in sensor networks. In *Proc. of ACM DMSN*, 2004.

[66] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proc. of ACM CCS*, 1993.

[67] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*. 2013.

[68] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. of ACM IMC*, 2010.

[69] T. Benson, A. Akella, and D. A. Maltz. Mining policies from enterprise network configuration. In *Proc. of ACM IMC*, 2009.

[70] J. Beuchat, J. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-speed software implementation of the optimal ate pairing over barreto–naehrig curves. In *International Conference on Pairing-Based Cryptography*, 2010.

[71] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *In Proc. of EUROCRYPT*, 2003.

[72] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational diffie-hellman. In *Proc. of PKC*, 2006.

[73] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *Proc. of the ACM SOSP*, 2013.

[74] A. Bremler-Barr, Y. Harchol, and D. Hay. OpenBox: a software-defined framework for developing, deploying, and managing network functions. In *Proc. of ACM SIGCOMM*, 2016.

[75] A. J. Brush, J. Jung, R. Mahajan, and F. Martinez. Digital neighborhood watch: Investigating the sharing of camera data amongst neighbors. In *Proc. of ACM CSCW*, 2013.

[76] G. R. Cantieni, G. Iannaccone, C. Barakat, C. Diot, and P. Thiran. Reformulating the monitor placement problem: Optimal network-wide sampling. In *Proc. of ACM CoNEXT*, 2006.

[77] D. D. Chamberlin and R. F. Boyce. SEQUEL: A structured english query language. In *Proc. of ACM SIGMOD workshop on Data description, access and control*, 1974.

[78] M. Charikar, Y. Naamad, J. Rexford, and K. Zou. Multi-commodity flow with in-network processing. `www.cs.princeton.edu/~jrex/papers/mopt14.pdf`.

[79] C. Chaudet, E. Fleury, I. G. Lassous, H. Rivano, and M. Voge. Optimal positioning of active and passive monitoring devices. In *Proc. of ACM CoNEXT*. ACM, 2005.

[80] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *Proc. of ACM SIGMOD*, 1998.

[81] W. Cheng, H. Pang, and K. Tan. Authenticating multi-dimensional query results in data publishing. In *IFIP Annual Conference on Data and Applications Security and Privacy*, 2006.

[82] Manuel C. Christian P., Kapil V. EnclaveDB: A secure database using sgx. In *Proc. of IEEE S&P*, 2018.

[83] K. Chung, Y. T. Kalai, F. Liu, and R. Raz. Memory delegation. In *Crypto*, 2011.

[84] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proc. of the USENIX NSDI*, 2005.

[85] E. Cohen and H. Kaplan. Summarizing data using bottom-k sketches. In *Proc. of ACM PODC*, 2007.

[86] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *Proc. of ACM ITCS*, 2012.

[87] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Continuous sampling from distributed streams. *JACM*, 59(2), 2012.

[88] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *Proc. of IEEE S&P*, 2015.

[89] J. Cropper, J. Ullrich, P. Frühwirt, and E. Weippl. The role and security of firewalls in iaas cloud computing. In *Proc. of IEEE ICARES*, pages 70–79, 2015.

[90] I Dinur and D Steurer. Analytical approach to parallel repetition. In *Proc. of ACM STOC*, 2014.

[91] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. ETTM: a scalable fault tolerant network manager. In *Proc. of USENIX NSDI*, 2011.

[92] D. Evan. The Internet of Things, Cisco White Paper. `https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf`.

[93] K. Fan, S. Liu, and P. Sinha. Scalable data aggregation for dynamic events in sensor networks. In *Proc. of the ACM SenSys*, 2006.

[94] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proc. of USENIX NSDI*, 2014.

[95] A. Feldmann et al. Deriving traffic demands for operational ip networks: Methodology and experience. In *Proc. of ACM SIGCOMM*, 2000.

[96] J. Gao, L. Guibas, N. Milosavljevic, and J. Hershberger. Sparse data aggregation in sensor networks. In *Proc. of ACM IPSN*, 2007.

[97] A. Gember, A. Krishnamurthy, S. St. John, R. Grandl, X. Gao, A. Anand, T. Benson, V. Sekar, and A. Akella. Stratos: A network-aware orchestration layer for virtual middleboxes in clouds. *arXiv preprint arXiv:1305.0209*, 2013.

[98] A. Gember, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella. OpenNF: Enabling innovation in network function control. In *Proc. of ACM SIGCOMM*, 2014.

[99] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Cryptology*, 2010.

[100] R. Gennaro and P. Rohatgi. How to sign digital streams. In *Crypto*, 1997.

[101] M. Gerla, E. Lee, G. Pau, and U. Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Proc. of IEEE WF-IoT*, 2014.

[102] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2), 1988.

[103] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-efficient verification of dynamic outsourced databases. In *CT-RSA*. 2008.

[104] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy. Flow processing and the rise of commodity network hardware. *ACM SIGCOMM CCR*, 39(2), 2009.

[105] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 2013.

[106] R. Guerzoni et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In *SDN and OpenFlow World Congress*, 2012.

[107] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Middleware*. 2006.

[108] T. Gupta, R. P. Singh, A. Phanishayee, J. Jung, and R. Mahajan. Bolt: Data management for connected homes. In *Proc. of USEIX NSDI*, 2014.

[109] J. He, C. Tang, Y. Yang, Y. Qiao, and C. Liu. 3d-ids: Iaas user-oriented intrusion detection system. In *Proc. of IEEE ISISE*, 2012.

[110] C. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in olap data cubes. In *Proc. of ACM SIGMOD*, 1997.

[111] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proc. of the ACM CoNEXT*, 2013.

[112] X. Jin, L. E. Li, L. Vanbever, and J. Rexford. Softcell: Scalable and flexible cellular core network architecture. In *Proc. of ACM CoNext*, 2013.

[113] X. Jin, H. H. Liu, R. Gandhi, S. Kandula, R. Mahajan, M. Zhang, J. Rexford, and R. Wattenhofer. Dynamic scheduling of network updates. In *Proc. of ACM SIGCOMM*, 2014.

[114] J. Jonsson, K. Moriarty, B. Kaliski, and A. Rusch. Pkcs# 1: Rsa cryptography specifications version 2.2. 2016.

[115] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *Proc. of ACM SIGCOMM*, 2008.

[116] S. Kandula et al. The nature of data center traffic: measurements & analysis. In *Proc. of the ACM IMC*, 2009.

[117] Y. Kim, J. Kang, D. Kim, E. Kim, P. K. Chong, and S. Seo. Design of a fence surveillance system based on wireless sensor networks. In *Proc. of the Autonomics*, 2008.

[118] Leslie Lamport. Paxos made simple. 2001.

[119] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *ProcACM SIGMOD*, 2006.

[120] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Authenticated index structures for aggregation queries. *ACM TISSEC*, 13(4):32, 2010.

[121] L. E. Li, V. Liaghat, H. Zhao, M. Hajiaghay, D. Li, G. Wilfong, Y. R. Yang, and C. Guo. Pace: policy-aware application cloud embedding. In *Proc. of IEEE INFOCOM*, 2013.

[122] X. Li and C. Qian. Traffic and failure aware vm placement for multi-tenant cloud computing. In *Proc. of IEEE/ACM IWQoS*, 2015.

[123] Prince Mahajan, Srinath Setty, Sangmin Lee, Allen Clement, Lorenzo Alvisi, Mike Dahlin, and Michael Walfish. Depot: Cloud Storage with Minimal Trust. *ACM Trans. Comput. Syst.*, 29, 2011.

[124] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proc. of ACM SIGMOD*, 1998.

[125] J. Martins et al. Clickos and the art of network function virtualization. In *Proc. of USENIX NSDI*, 2014.

[126] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 2008.

[127] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proc. of IEEE INFOCOM*, 2010.

[128] R. C. Merkle. A digital signature based on a conventional encryption function. In *Proc. of CRYPTO*, 1987.

[129] H. Moens and F. De Turck. Vnf-p: A model for efficient placement of virtualized network functions. In *Proc. of IEEE CNSM*, 2014.

[130] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. In *Proc. of ACM SIGCOMM*, 2011.

[131] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE TKDE*, 13(1), 2001.

[132] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. 1966.

[133] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and Integrity in Outsourced Databases. In *Proc. of NDSS*, 2004.

[134] S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. *ACM TOSN*, 4(2), 2008.

[135] S. Nishimura and H. Yokota. Quilts: Multidimensional data partitioning framework based on query-aware and skew-tolerant space-filling curves. In *Proc. of ACM SIGMOD*, 2017.

[136] R. Olsson. Pktgen the linux packet generator. In *Proc. of the Linux Symposium*, 2005.

[137] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proc. of ACM PODS*, 1984.

[138] S. Palkar, C. Lan, S. Han, K. Jang, A. Panda, S. Ratnasamy, L. Rizzo, and S. Shenker. E2: A framework for nfv applications. In *Proc. of ACM SOSP*, 2015.

[139] A. Panda, S. Han, K. Jang, M. Walls, S. Ratnasamy, and S. Shenker. NetBricks: Taking the V out of NFV. In *Proc. of USENIX OSDI*, 2016.

[140] H. Pang, J. Zhang, and K. Mouratidis. Scalable verification for outsourced dynamic databases. In *Proc. of the VLDB Endowment*, 2009.

[141] D. Papadopoulos, S. Papadopoulos, and N. Triandopoulos. Taking authenticated range queries to arbitrary dimensions. In *Proc. of ACM CCS*, 2014.

[142] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *Theory of Cryptography*. 2013.

[143] B. Pfaff and Ed. B. Davie. The open vswitch database management protocol. Technical report, RFC 7047, December, 2013.

[144] K. Piotrowski, P. Langendoerfer, and S. Peter. How public key cryptography influences wireless sensor node lifetime. In *Proc of ACM SASN*, 2006.

[145] R. Ada Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: protecting confidentiality with encrypted query processing. In *Proc. of ACM SOSP*, 2011.

[146] P. Porras, S. Shin, V. Yegneswaran, M. Fong, M. Tyson, and G. Gu. A security enforcement kernel for openflow networks. In *Proc. of ACM HotSDN*, 2012.

[147] C. Prakash, J. Lee, Y. Turner, J. M. Kang, A. Akella, S. Banerjee, C. Clark, Y. Ma, P. Sharma, and Y. Zhang. PGA: Using Graphs to Express and Automatically Reconcile Network Policies. In *Proc. of ACM SIGCOMM*, 2015.

[148] Z. A. Qazi, C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu. Simple-fying middlebox policy enforcement using sdn. In *Proc. of the ACM SIGCOMM*, 2013.

[149] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield. Split/Merge: System support for elastic execution in virtual middleboxes. In *Proc. of USENIX NSDI*, 2013.

[150] S. Raza, G. Huang, C. N. Chuah, S. Seetharaman, and J. P. Singh. Measurouting: a framework for routing assisted traffic monitoring. *IEEE/ACM ToN*, 20(1), 2012.

[151] World Health Organization. mHealth: New horizons for health through mobile technologies. `http://www.who.int/goe/publications/goe_mhealth_web.pdf`.

[152] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *CACM*, 21(2), 1978.

[153] L. Saino, C. Cocora, and G. Pavlou. A toolchain for simplifying network simulation setup. In *Proc. of ICST SIMUTOOLS*, 2013.

[154] S. Sattolo. An algorithm to generate a random cyclic permutation. *Information processing letters*, 22(6), 1986.

[155] J. Scott, Bernheim B., J. Krumm, B. Meyers, M. Hazas, S. Hodges, and N. Villar. Preheat: controlling home heating using occupancy prediction. In *Proc. of ACM Ubicomp*, 2011.

[156] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *Proc. of USENIX NSDI*, 2012.

[157] V. Sekar et al. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proc. of ACM HotNets*, 2011.

[158] V. Sekar, R. Krishnaswamy, A. Gupta, and M. K. Reiter. Network-wide deployment of intrusion detection and prevention systems. 2010.

[159] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen. cSamp: A system for network-wide flow monitoring. In *Proc. of USENIX NSDI*, 2008.

[160] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proc. of ACM Eurosys*, 2013.

[161] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. Making middleboxes someone else's problem: network processing as a cloud service. 2012.

[162] J. Sherry and S. Ratnasamy. A Survey of Enterprise Middlebox Deployments. Technical report, EECS, UC Berkeley, 2012.

[163] A. J. Smith. Sequentiality and prefetching in database systems. *ACM TODS*, 3(3), 1978.

[164] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. 2002.

[165] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating network monitors: complexity, heuristics, and coverage. *Computer Communications*, 29(10), 2006.

[166] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon. Providing public intradomain traffic matrices to the research community. *ACM SIGCOMM CCR*, 36(1), 2006.

[167] J. S. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, 11(1), 1985.

[168] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for interactive verifiable computation. In *Proc. of IEEE S&P*, 2013.

[169] C. Wang, Q. Wang, K. Ren, and W. Lou. Ensuring data storage security in cloud computing. In *Proc. of IEEE IWQoS*, 2009.

[170] H. Wang, C. Qian, Y. Yu, H. Yang, and S. Lam. Practical network-wide packet behavior identification by ap classifier. In *Proc. of ACM CoNext*, 2015.

[171] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang. An untold story of middleboxes in cellular networks. In *Proc. of ACM SIGCOMM*, 2011.

[172] C. K. Wong and S. S. Lam. Digital signatures for flows and multicasts. In *Proc. of IEEE ICNP*, 1998.

[173] Z. Xia, X. Wang, X. Sun, and Q. Wang. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE TPDS*, 27(2), 2016.

[174] T. Yamada, Y. Hayamizu, Y. Yamamoto, Y. Yomogida, A. IzadiNajafabadi, D. N. Futaba, and K. Hata. A stretchable carbon nanotube strain sensor for human-motion detection. *Nature nanotechnology*, 6(5), 2011.

[175] H. Yang and S. S. Lam. Real-time verification of network properties using atomic predicates. In *Proc. of IEEE ICNP*, 2013.

[176] Y. Yang, D. Papadias, S. Papadopoulos, and P. Kalnis. Authenticated join processing in outsourced databases. In *Proc. of ACM SIGMOD*, 2009.

[177] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, et al. Steering: A software-defined networking for inline service chaining. In *Proc. of IEEE ICNP*, 2013.

[178] Y. Zhang, L. Duan, and J. L. Chen. Event-Driven SOA for IoT Services. In *Proc. of IEEE SCC*, 2014.

[179] Y. Zhang, J. Katz, D. Genkin, D. Papadopoulos, and C. Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proc. of IEEE S&P*, 2017.

[180] Y. Zhang, J. Katz, and C. Papamanthou. Integridb: Verifiable sql for outsourced databases. In *ACM CCS*, 2015.

[181] Q. Zheng, S. Xu, and G. Ateniese. Efficient query integrity for outsourced dynamic databases. In *Proc. of ACM CCSW*, 2012.

[182] D. Zhou, B. Fan, H. Lim, D. G. Andersen, M. Kaminsky, M. Mitzenmacher, R. Wang, and A. Singh. Scaling Up Clustered Network Appliances with ScaleBricks. In *Proc. of ACM SIGCOMM*, 2015.

[183] Z. Zhou. Ensemble learning. *Encyclopedia of biometrics*, pages 411–416, 2015.