Secure Embedded Living

Towards A Self-Contained User Data Preserving Framework

Mazumdar, Somnath ; Dreibholz, Thomas

Document Version
Accepted author manuscript

Published in:
IEEE Communications Magazine

DOI:
10.1109/MCOM.001.2200165

Publication date:
2022

License
Unspecified

Citation for published version (APA):
Mazumdar, S., & Dreibholz, T. (2022). Secure Embedded Living: Towards A Self-Contained User Data Preserving Framework. *IEEE Communications Magazine*, *60*(11), 74-80. https://doi.org/10.1109/MCOM.001.2200165

Secure Embedded Living: Towards a Self-contained User Data Preserving Framework

Somnath Mazumdar
Department of Digitalization, Copenhagen Business School,
Solbjerg Plads 3, 2000 Frederiksberg, Denmark
sma.digi@cbs.dk

Thomas Dreibholz
Simula Metropolitan Centre for Digital Engineering,
Pilestredet 52, 0167 Oslo, Norway
dreibh@simula.no

Smart living represents the hardware-software co-inhabiting with humans for better living standards and improved well-being. Here, hardware monitors human activities (by collecting data) specific to a context. Such data can be processed to offer context-specific valuable insights. Such insights can further be used for optimising the well-being, living experience and energy cost of smart homes. This paper proposes a Secure Embedded Living Framework (SELF) that enforces a privacy-preserving data control mechanism by integrating multiple technologies, such as Internet-of-thing, cloud/fog platform, machine learning and blockchain. The primary aim of the SELF is to allow the end-user to retain more control of its data.

I. Introduction

In recent times, technology has seamlessly integrated into our homes. For instance, smart vacuum cleaners become *smarter* thanks to home sensing Internet-of-thing (IoT) devices. A smart living ecosystem consists of various heterogeneous IoTs connected to a local home network for communications. Such IoT devices are further connected to and controlled by software applications. IoT devices can generate data in various forms, such as text and video. Such data can have asset-related information, e.g., media access control addresses or personal features. Collected data can also be divided into semi-structured types and unstructured types. Over time, the amount of collected data may become huge, and managing it in private homes can become complicated. Thus, effective data management should be in place to stop the exploitation of user behaviour and reduce application-usage-related sensitive data (such as browsing histories) collection by third parties or service providers. Su et al. showed that web browsing histories can uniquely be linked to social media profiles using only public auxiliary information [1].

The NIST privacy framework proposes five core functionalities for achieving data privacy [2]. It includes data control, communication, identification, governing data and data protection. We define *privacy* as free from intrusion and having the ability to control one's data, while *security* refers to data protection against unauthorised access to user data. In some cases, privacy and security may overlap. Here, we are focusing on *data protection*. Two primary goals of the proposed Secure Embedded Living Framework (SELF) are to help the user to control (to a certain extent) *i) Which data will be available to third parties (such as healthcare professionals, electricity providers)?* and *ii) How much such data should be delegated?* To manage user data, SELF considers secure cloud data storage and allows access per pre-set rules to legitimate third parties. Blockchain is used to improve trust, transparency and security via its built-in data immutability and tamper-proof features. Blockchain can also strictly enforce data access rules. SELF can offer a mechanism to inform users about their data usage patterns and uses Machine learning (ML) algorithms to detect smart IoT device usage patterns. The main contributions of this article are:

- A user data privacy-aware framework has been proposed. It allows users to add their data privacy-preserving rules to protect their data. The proposed framework aims to stop unwanted third party access to private user data.
- We have explained the SELF architecture and elaborated a remote health monitoring use case to show its usefulness.
- Finally, we have implemented the packet processing part and presented the results while executing it in a multi-cloud research testbed.

II. SELF Overview

Insecure IoT devices and non-secure device pairing and discovery protocols can increase privacy risks by leaking users' private data. Such incidents allow remote attackers to spy further and attack victims. It was part of our motivation behind designing SELF as a platform that aims at improving user data privacy.

A. Core Components

SELF can be decomposed into five primary components consisting of three layers. The components are *i)* IoT devices, *ii)* P4 switch, *iii)* Computational platform, *iv)* ML algorithms and *v)* Blockchain. IoTs and P4 switch constitute the *resource layer*, the computational platform represents the *processing layer*, while the *application layer* consists of ML algorithms and blockchain.

1) IoT Devices

We can find various (in terms of form factor, protocol-based, and application) types of IoT devices in today's smart homes. Popular sets of IoTs include health-related, light, motion, and temperature sensing devices. IoTs are currently equipped with data communication technologies (such as Bluetooth low energy, ZigBee, and WiFi) to collect and receive data at high speeds. Data generated from these IoT devices are pushed to the local communication network. As a next step, data arrives at the input queue of the P4 switch.

2) P4 Switch

Programming Protocol-independent Packet Processors (P4) make full programmability on switches possible [3]. P4 is a domain-specific language for realising the data plane of a programmable switch. The switch itself is an off-the-shelf hardware or can be a software implementation. P4 applies to standard P4 hardware/software (refer to Figure 1). It avoids vendor lock-in while allowing full flexibility for customising the network configuration.

A P4 program defines rules to parse a packet into fields of different protocol layers. The field information may then be used with match-action tables to trigger custom actions, for instance, dropping or modifying the packet at its egress port(s). In egress, the packet data is deparsed into a packet, and the resulting packet is sent over to the selected egress port(s). A P4 compiler compiles the P4 program into a data plane implementation for a specific target switch and control plane data. As the name suggests, control plane data (p4info) defines the interaction between the data and the control plane. It is typically a control plane instance (P4Runtime Client) communicating with the switch's data plane instance (P4Runtime Server) over a defined application programming interface (API). The client may run on the switch hardware or an external system. The control plane is responsible for modifying the match-action tables of the data plane. The data plane may also forward (or parts of) a packet to the control plane for special handling.

P4 can identify flows by parsing header fields of multiple protocols. Such protocols are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Stream Control Transmission Protocol (SCTP), and Datagram Congestion Control Protocol (DCCP). In a similar case, Febro et al. developed a virtualised network function written in P4 and running on a programmable data plane [4]. P4 allows handling packets of arbitrary new or custom protocols. To do that, P4 mark packets by rewriting information at the *DiffServ Code Point* (DSCP) bits of the Type of Service (TOS)/Traffic Class fields, or by adding additional headers to IPv4 options field, or IPv6 extension header. In general, most work on dissecting packet headers and making forwarding decisions is made in the data plane, based on the match-action tables set by the control plane. The data plane may forward information from a packet to the control plane for further analysis and possible modification of the match-action tables.

3) Computational Platform

Now, a user can select a data processing and storage platform, essentially cloud-based services, at lower costs. The cloud platform has an enormous computational and storage capacity. Its capability can be further extended by introducing another resource-level abstraction called fog [5]. Fog sits between the end-user and the cloud. Fog offers low computational capabilities with a faster response time. End-users can submit applications together with relevant data sets. If the number of connected IoT devices and the generated data is growing very quickly, then fog resources backed by a cloud platform can also be one viable solution. Each home can generate different amounts of data. The data creation process is directly impacted by the number of connected smart devices and the data collection frequency. SELF aims to support multi-cloud for latency-tolerant applications and fog for latency-sensitive applications. The user can opt for a cloud/fog service or a serverless computing service based on requirements. In a serverless model, the code executes on-demand in a stateless container, which can be ideal for smart home applications. Due to the wide adoption of microservices and containers, seamless transfer from a container to a dedicated virtual machine (VM) is also possible. A customised P4 switch can send data directly to cloud storage via relevant REpresentational State Transfer (REST) API calls.

4) ML Algorithms

At the application level, various data analysis tasks, such as outlier detection, prediction, IoT devices usage pattern extractions and pattern matching, can be performed on the collected data. Next, the relevant outcome or recommendations can be sent back. Some of the application-specific ML algorithms are [6]:

- Classification-based algorithms are Random Forests, K-Nearest Neighbors, and Naive Bayes.
- Support vector machine can be used for data classification and outlier detection.
- Regression-supported algorithms are linear regression, support vector regression and random forests.
- K-means algorithm is popular for data clustering.
- Principal component analysis is used for feature extraction and dimension reduction.

Now, computation has increasingly become an off-campus activity. Preserving in-house data privacy while processing it on a public cloud is challenging. Data privacy can be further enhanced by applying ML to the user data stored in the cloud, with added encryption features by paying extra cost [7]. In this process, the encrypted data will be moved to the cloud for applying ML algorithms (such as Neural Networks) to the encrypted data to make encrypted predictions. Finally, it returns the output to the data owner, who can decrypt it. Such scenarios are very relevant for health-related applications. Training feed-forward neural networks on encrypted data are computationally expensive due to activation and loss functions. Furthermore, Homomorphic encryption is a cryptographic mechanism that preserves the message structure and primarily supports addition and multiplication operations without data decryption. However, the efficiency and practicality of homomorphic encryption are always open research questions.

5) Blockchain

Blockchain is one of the implementations of distributed ledger technology (DLT) [8]. In a distributed ledger, data can only be appended by applying a common consensus among participating nodes. Blocks in the blockchain are a collection of transactions chained together using hash values of the previous data block. SELF uses blockchain to secure data and bring trust among users.

Figure 2 presents a canonical blockchain architecture, which can be divided into a core layer (based on a peer-to-peer platform), middle layer and application layer. At the bottom, besides standard computing, storage, system management and

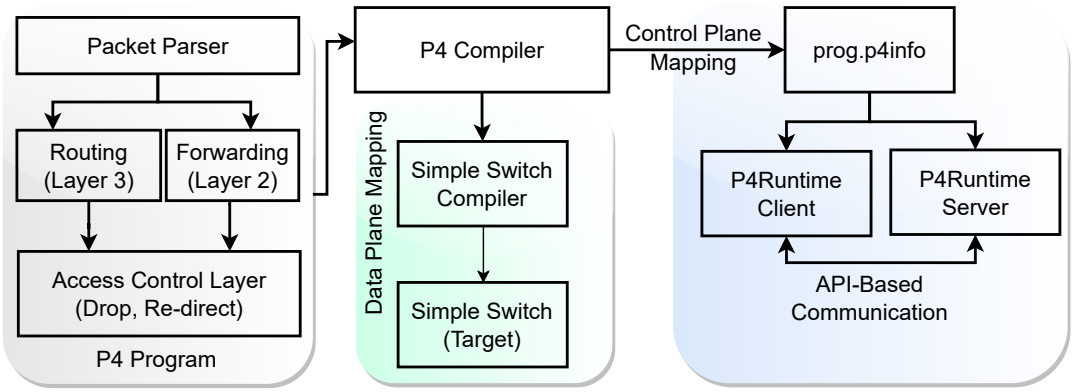

Figure 1: Representation of a standard P4 switch components

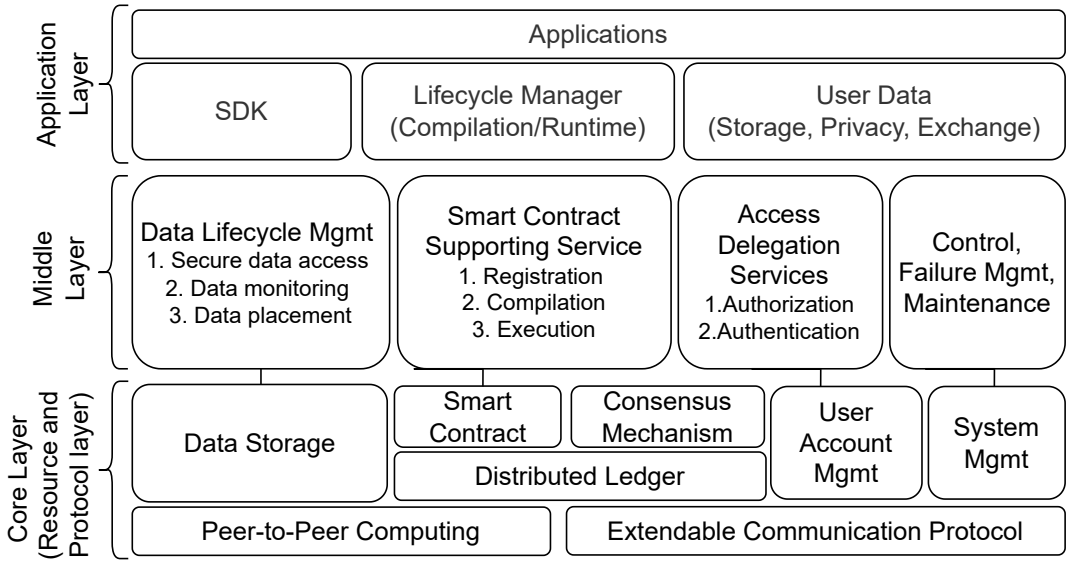

Figure 2: Reference building blocks of Blockchain

user management, smart contracts and consensus protocol features are added for blockchain support. For simplicity, a distributed ledger can be viewed as a distributed database, where each network node will have its synchronised local copy. In the middle layer, a data lifecycle management module might be added to manage data throughout the lifetime. Via smart contract services, legitimate node users add code and can trigger it whenever required. The user access delegation module handles the authentication and authorisation process. The platform is also monitored together with the system state logs. The top-level of Figure 2 primarily faces developers who can develop the code using the platform-compatible software development kit (SDK). Users can also specify data management policies by calling proper REST APIs.

Currently, multiple blockchain platforms exist. Some are cryptocurrency-based (such as Ethereum), and some are not (such as Hyperledger Fabric). However, they primarily support DLT features, with some variations relevant to their use cases. It is also worth noting that network setup can be without permission (such as Ethereum) or with permission (such as Hyperledger Fabric). For SELF, a permission network is preferred. Here, only specific node users can access data. Hyperledger Fabric offers *channels* to support a private communication tunnel between specific node users [8]. The service provider can create one channel for each smart home user, where third parties can access user data as per their privilege level. It is worth noting that third parties will not be SELF blockchain network members.

B. Proposed Architecture

Figure 3 represents the SELF architecture. It aims at users to control and protect their data. Multiple technologies converge to achieve this goal. The framework has four broad working stages. The stages are described below:

1) Data Generation

It generates raw data, which is controlled by user-defined rules together with the *smartness* of the IoT devices. The data generation frequency of the IoTs is either synchronous or asynchronous. Generated data can be represented in JavaScript object notation (JSON) format and either be non-personal or personal. Each device has different data collection and filtering rules based on the implementations. SELF runs the data filtering rules inside the customised P4 switch.

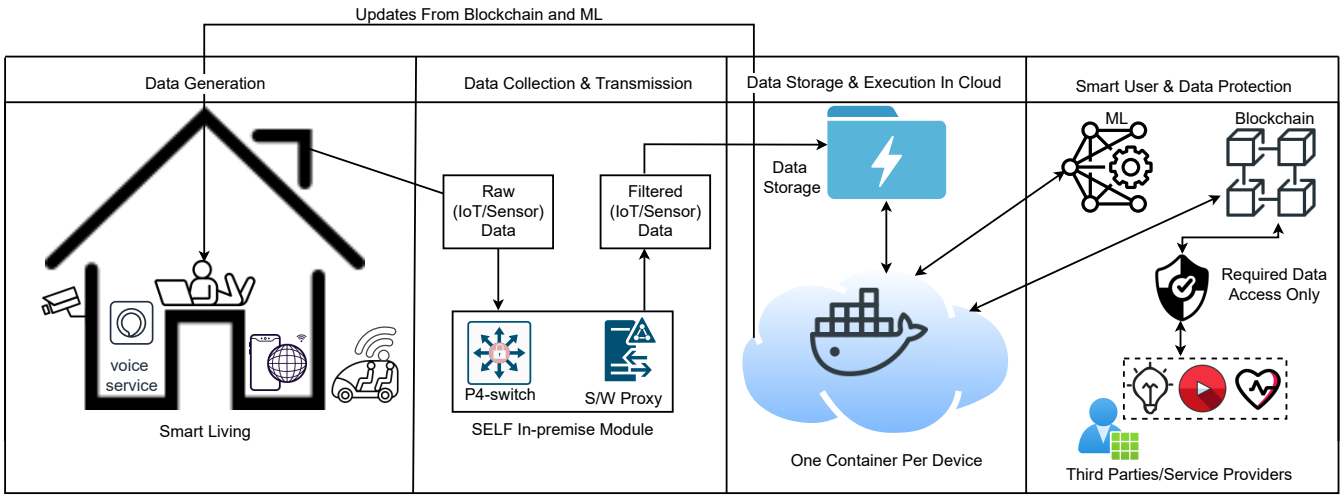

Figure 3: Block diagrammatic representation of SELF architecture

2) Data Collection and Transmission

This stage starts when all the connected devices push their data to the shared communication link. Next, it arrives at the exit point, a customised P4 switch programmed with user-specified privacy-preserving rules. Such a custom switch is not the same as an off-the-self device but an extended one by adding a customised P4 runtime. This runtime is based on the P4 programming language, developed as portable switch architecture. The primary reason for using P4 runtime is that it can efficiently work on the data plane. User-defined data filtering rules can process and forward the required data to the container (hosted in the cloud). Data is passed to the cloud using REST API supporting JSON data format. The custom logic in the P4 runtime can forward/redirect packets of specific flows to an application-specific proxy (can watch flows, process data, and transmit sensitive information safely and securely), which can then perform packet aggregations. Such a proxy can run in a container on the software switch itself.

3) Data Storage and Execution

It happens outside the user's premises. The data is now stored in the cloud. Such data can move forward and backwards between the storage and the containers. For simplicity, we consider both the code and the data hosted at the same cloud. As time passes, the data size will increase, so a simple 'one container per device' rule can be applied. Overall, ML algorithms and also blockchain will be running in the cloud. SELF primarily employs ML algorithms to receive more insights related to IoT devices. Blockchain can ensure user data privacy via a private communication channel and apply customised data access control policies [8]. Primarily, all the required data is pushed to the cloud, while IoTs are responsible only for data collection.

4) Smart User and Data Protection

It represents the application layer, where each container executes an ML algorithm on a particular device data to provide better insights related to that IoT device usage patterns or energy management. For instance, specific ML algorithms can detect some disease patterns from health-related data. After successful detection, it not only can update users but also update healthcare professionals. We have presented a remote patient monitoring scenario in Subsection III-A to show the framework's usefulness.

We can set the SELF user-level data access privileges using a smart contract so third parties can get only the relevant data. We employed two privilege levels: one holds generic data, and the other consists of personal data. Here, ML primarily aims to improve user well-being, while blockchain secures processed data, allowing only authorised third parties to access user data. The user selects data access rules, and the smart contract encodes them into the blockchain.

III. Proposed Solution

SELF offers an end-to-end ecosystem that encompasses multiple technologies to preserve user privacy. It is designed to be implemented by a service provider because of technical complexities. The service provider can agree with the user on *which data to keep?* and *what kind of optimisation the user requires?* The service provider can set the rules inside the P4 runtime based on the requirements. The user can be further consulted on *where to host the data?*, *who should access user data?* and also about the privilege levels. Embedding the IoTs with ML at the hardware layer is worthwhile, but due to the limited resource capacity, the performance will also be minimal, together with high energy costs. ML and blockchain will run in containers hosted in the cloud to overcome such issues.

A. Use Case: Remote Patient Monitoring

Now, we will demonstrate how SELF can delegate a user's health-related information based on privilege-based access control to healthcare professionals (e.g. doctors, nurses). Suppose health monitoring devices are already planted into a smart home and employed IoT devices can collect data related to oxygen level, heart rate, and blood pressure (to name a few). We can see that user health-related information can be stored in the cloud (left side of Figure 4). Next, a container running the health-related pre-trained ML algorithm can process the

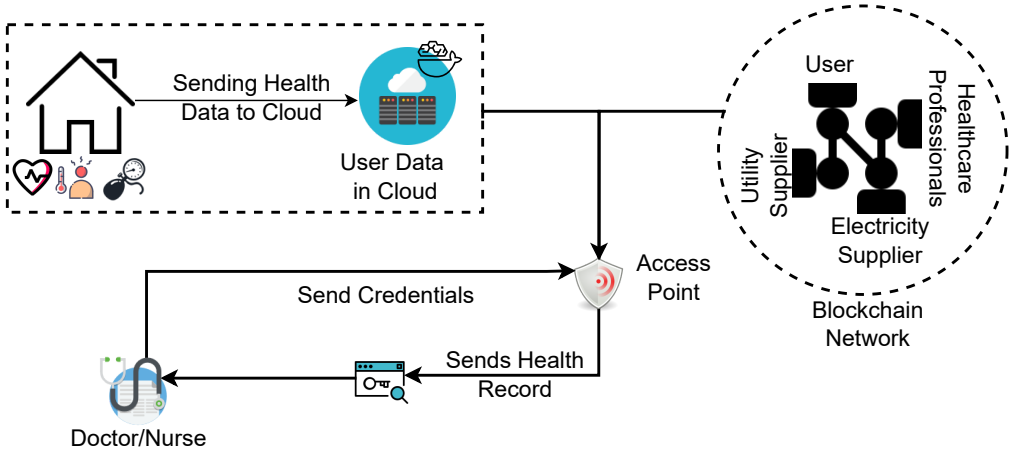

Figure 4: Representation of remote patient monitoring use case

user health data and predict if there is a trend of upcoming health issues. For instance, predicting the blood pressure level can help to detect the plausibility of developing cardiovascular disease, kidney disease, or others. Upon detection of such events, already-encoded rules in smart contracts can be executed, which may inform the healthcare professional and the patients. Smart contracts are a collection of rules and data executed on the blockchain, and the results are recorded on the blockchain securely. While deploying a smart contract, the user must digitally sign it, ensuring that only relevant third parties can access the data.

On the other hand, doctors/nurses can *pull* user health data for a regular check-up by providing their relevant credentials while making relevant REST API calls. After successful validation, a doctor can receive the required health record, and an alert informs the patient. Changes to one health record can cause subsequent records to become invalid because blockchain can easily detect unwanted changes to data blocks.

Application: We believe such usage of SELF will become paramount, particularly during an epidemic/pandemic time (such as Covid-19), when the healthcare facilities are overloaded with patients. This way, we can provide care to the patients in their homes. Such remote patient monitoring should be considered a standard case, which can further be extended for utility service providers, such as electricity-providing companies, to adjust their services dynamically. The stored data in the blockchain is protected by the inherent immutability feature and underlying cryptography.

B. Vendor Lock-In

Currently, vendor lock-in is an issue while using the cloud. Such issues arise if a cloud service provider changes its pricing model, deploys completely different APIs, or shuts down its old web services. Multi-cloud-supporting frameworks (such as SELF) can mitigate such risks by adding another layer of abstraction. It means the service uses RESTful APIs in the framework instead of using a specific cloud provider directly. Later, changing the underlying cloud provider becomes easy as only the multi-cloud framework (SELF in this case) has to take care of the details of the different cloud/fog service providers. A smart multi-cloud framework can even work as a broker, which allows users to use the most cost-effective cloud service at a specific time. Such a framework can also optimise metrics such as latency or location. In particular, location can be a constraint for preserving privacy. Thus, SELF aims to become an open-source multi-cloud framework that can prevent vendor lock-in, allowing cost-effective cloud resources and becoming more location-aware (concerning privacy) to offload the workload into the cloud/fog ecosystem (as future work).

C. Managing Data at Packet Level

A P4 switch can isolate the insecure IoTs from each other and can control the communications. Unlike modern routers with access control lists, P4 offers complete flexibility to apply customised deep packet inspection. Depending on the user applications, using SELF-customised rules can be applied to handle marking, perform filtering, and create alerts. Such customised rules can then be provided as *modules* and deployed by the SELF controller plane, leading to a switch behaviour tailored to the user's demands. P4 is made in an open, standardised and vendor-independent way on off-the-shelf devices. SELF appropriately configures the physical network into logical networks by defining virtual LANs (VLANs) and virtual private networks (VPNs). SELF can also use the available techniques for the underlying communications infrastructures. Hence, an insecure device cannot be compromised by other devices, and a compromised device cannot attack other devices. Overall, data cannot leak out to arbitrary locations on the Internet. The controller can deploy a network intrusion detection system (IDS) to spot suspicious network activity of a device. After successful detection, the user can be informed (e.g. if a device tries to contact unknown servers), or the device can be blocked (e.g. if a device attempts a denial-of-service (DoS) attack).

Suppose a smart home is fitted with temperature and movement-related IoT devices in rooms for a smart heating/-cooling control. These IoT devices share the same wireless LAN with other heterogeneous devices to connect to the Internet. With SELF, a VLAN can be configured for each IoT devices. A separate per-VLAN VPN to a remote cloud instance can also be configured if needed. So, each IoT device

Table I: Overheads of P4-switch considering both IPv4 and IPv6 while representing RTT of 900 samples

Destination	Linux Bridge						SELF					
	IPv4			IPv6			IPv4			IPv6		
	Mean	$Q_{10\%}$	$Q_{90\%}$	Mean	$Q_{10\%}$	$Q_{90\%}$	Mean	$Q_{10\%}$	$Q_{90\%}$	Mean	$Q_{10\%}$	$Q_{90\%}$
HU/CERNET	341.2	340.4	341.9	341.2	340.4	341.9	352.4	347.4	356.2	349.5	345.2	352.7
HU/China Unicom	202.2	190.6	231.9	272.1	258.2	300.0	205.7	198.1	208.5	269.8	262.9	272.5
KAU/SUNET	32.5	31.7	33.2	32.8	32.0	33.5	42.9	37.9	46.3	39.7	35.7	42.8
NTNU/PowerTech	24.2	21.1	24.3	24.5	21.4	24.1	32.4	26.0	35.9	29.7	24.6	32.8
NTNU/UNINETT	14.7	13.9	15.3	14.7	13.9	15.3	20.2	16.5	22.5	18.7	16.0	21.0
SRL/UNINETT	5.2	4.6	5.8	4.7	4.0	5.4	13.3	8.3	16.8	10.3	5.8	13.7
UDE/DFN	32.2	31.4	32.9	37.9	37.1	38.5	42.9	37.7	46.2	45.4	41.3	48.6
UiA/PowerTech	24.2	22.9	24.7	–	–	–	31.7	27.3	34.8	–	–	–
UiA/UNINETT	12.2	11.5	12.7	12.3	11.5	12.8	17.2	14.0	19.8	15.8	13.3	17.7
UiB/BKK	12.4	11.6	13.2	12.5	11.7	13.2	19.5	15.2	22.2	17.6	14.2	20.3
UiB/UNINETT	12.3	11.6	12.9	12.5	11.8	13.1	16.8	13.7	19.5	15.6	13.2	17.5
UiO/Broadnet	18.6	14.9	19.6	19.3	15.0	19.9	27.2	21.3	30.3	25.1	19.3	28.2
UiO/UNINETT	7.0	6.1	7.4	7.0	6.2	7.4	13.9	9.3	17.3	11.2	7.5	14.3

is *alone* in its logical network, and a security issue in one device does not *trivially* provide an entry point to other IoT devices. Internet access for data processing is also restricted to the VPN connection with the corresponding cloud instance.

For the experiment, we have used a cloud/fog research testbed [9]. All remote cloud VMs, routers, P4 switch and devices are running on Ubuntu Linux. The local private cloud setup, (termed as *Home*), consists of devices, a P4 switch and a router. The P4 switch is based on the Behavioural Model version 2 (BMv2) Simple Switch, running the SELF P4 program in the data plane, with a corresponding Python-based control plane [10]. In Table I, we have presented results to show the overheads of our customised software switch. We examine the round-trip times (RTTs). RTT measurements are performed using the ICMP/ICMPv6 Echo Request/Reply measurement series to record the RTTs (Ping) using HiPerConTracer [11].

We reported the impact of the P4 switch on the RTT between a device in the home network (at SRL in Oslo, Norway) and in four cloud locations. For both IPv4 and IPv6, we have computed the mean values. The presented values in Table I are in milliseconds for the P4 switch-based SELF setup (right side) in comparison to a simple Linux bridge (left side) without any packet marking or filtering (i.e. no added delay overheads). Since IPv4 has a header checksum, differentiating between IPv4 and IPv6 is important. It needs to be verified during ingress and updated during egress (since we changed the DSCP bits in the header for marking packets). IPv6 does not have a checksum, so there is no checksum handling overhead for IPv6 (compared to IPv4, which requires additional 1.5 ms to 2.5 ms). It may be argued that skipping the IPv4 checksum verification at ingress would reduce the overhead. However, this would clearly violate the IPv4 protocol: a damaged IPv4 packet would remain undetected and get a new, valid checksum at egress. Rather than violating the IPv4 protocol, future work will move to a more advanced hardware/software P4 switch implementation, which can perform the checksum computations in hardware. Also, with IPv4 addresses becoming an increasingly scarce resource in the exhausted IPv4 address space, the widespread deployment of IPv6 will make this computation unnecessary.

Overall, the cost of adding the P4 Simple Switch adds around 2 ms to 6 ms of additional RTT to the IPv4 communications. For IPv6, the difference is a bit smaller, with around 1.5 ms to 3.5 ms. It is worth noting that P4 adds complete flexibility to the packet handling by custom P4 programming. Such additional RTT will not impact the performance of latency-insensitive applications (such as blockchain and ML).

IV. Security Challenges and Others

Generally, security issues can appear primarily at the device, communication, and service level. The hardware encryption and the fail-secure device design (e.g., security features) can be installed on the IoT devices, but with higher area and energy costs. While at the communication level, VPN and IDS can be used in the cloud. Finally, encryption can be applied at the service level.

A. IoT-Related Security Issues

Smart home users deploy IoT devices from different vendors with significantly different firmware quality. Ideally, all IoT devices should receive bug fixes and security updates from their vendor over their lifetime. Unfortunately, this is entirely unrealistic. It is fair to assume that many or even most of these devices, once sold, will never receive any updates. A user cannot install custom open-source software either, as most devices are proprietary. It is wise to assume that such devices are insecure. It brings software- and communication-protocols-related technical challenges. Communication-related issues can also lead to unstable communication. DoS and information leakage can happen during data transfer. The IoT device management system should be aware of the user's rules to protect user data from malicious activity because malicious code can also be embedded into the IoTs by attackers. Adding built-in anomaly and intrusion detection units can also be helpful. Standard security mechanisms are now embedded into most IoT devices to protect user data. Similarly, unique hardware signatures can be introduced at the hardware level to stop IoT device cloning.

B. ML-Related Security Issues

Multiple popular ML algorithms are not attack-resilient. For instance, deep neural networks and support vector machines are vulnerable to security attacks. Attacks on such algorithms

lead to decreased performance or failure. It can happen due to the injection of malicious data into the training data sets. Such an attacking method is known as *poisoning*. In such instances, attackers add malicious data with similar features to original data and wrong labels. Sometimes, attackers might know the training data distribution and the applied learning algorithm.

C. Standardisation and Benchmarking Issues

The ongoing standardising and benchmarking process of ML [12] and blockchain [13] is very slow. Ming et al. showed how people's trust in the ML algorithm changes based on the algorithm's stated accuracy and observed accuracy [14]. The effect of stated accuracy can change based on the algorithm's observed accuracy. Although few benchmark suites exist for ML algorithms, this is not true for blockchain platforms. There are no widely accepted standards for developing ML and blockchain-based applications. Although multiple blockchain platforms exist, unfortunately, they are incompatible. One of the reasons behind such incompatibility is the non-existence of global standards. However, some works are in progress in this direction.

V. Conclusion and Future Work

The proposed framework, SELF, let the user decide which data should be stored and how third parties will access it. IoT devices collect data here, and a custom P4 switch ensures that relevant data is passed on to cloud storage. In the cloud, ML algorithms are running to provide more insights, including recommendations about user living behaviours to improve the quality of life and device usage. Furthermore, blockchain helps to authenticate entities to access user data and also to secure SELF from outside attacks. We have selected the P4 Simple Switch environment for implementation, which provides all P4 features but lacks performance. Apart from that, P4Runtime is also vulnerable to man-in-the-middle attacks and channel flooding [15]. For SELF, we have selected P4 because P4Runtime is a *feature-complete* and well-known P4 implementation. In future work, we will change SELF to a high-performance software or hardware-based P4 implementation such as Open vSwitch.

References

[1] J. Su, A. Shukla, S. Goel, and A. Narayanan, "De-Anonymizing Web-Browsing Data with Social Networks," in *Proc. of 26th Int'l Conf on World Wide Web*, 2017, pp. 1261–1269.

[2] K. Boeckl and N. Lefkovitz, "Nist privacy framework: An overview," 2020, accessed on 2022-09-30. [Online]. Available: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=930470

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87–95, 2014.

[4] A. Febro, H. Xiao, J. Spring, and B. Christianson, "Edge Security for SIP-enabled IoT Devices with P4," *Computer Networks*, vol. 203, pp. 1–25, 2022.

[5] M. Mukherjee, L. Shu, and D. Wang, "Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, pp. 1826–1857, 2018.

[6] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine Learning for Internet of Things Data Analysis: A Survey," *Digital Communications and Networks*, vol. 4, pp. 161–175, 2018.

[7] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Int'l Conf. on Machine Learning*. JMLR.org, 2016, pp. 201–210.

[8] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proc. of 13th EuroSys Conf.*, 2018, pp. 1–15.

[9] E. G. Gran, T. Dreibholz, and A. Kvalbein, "NorNet Core – A Multi-Homed Research Testbed," *Computer Networks*, vol. 61, pp. 75–87, 2014.

[10] P4.Org, "Behavioral Model Version 2," 2022, accessed on 2022-09-30. [Online]. Available: https://github.com/p4lang/behavioral-model

[11] T. Dreibholz, "HiPerConTracer – A Versatile Tool for IP Connectivity Tracing in Multi-Path Setups," in *Proc. of 28th IEEE Int'l Conf. on Software, Telecommunications and Computer Networks*. IEEE, 2020, pp. 1–6.

[12] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking Machine Learning Methods for Performance Modeling of Scientific Applications," in *IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. IEEE, 2018, pp. 33–44.

[13] N. Drljevic, D. A. Aranda, and V. Stantchev, "Perspectives on Risks and Standards that affect the Requirements Engineering of Blockchain Technology," *Computer Standards & Interfaces*, vol. 69, pp. 1–7, 2020.

[14] M. Yin, J. W. Vaughan, and H. Wallach, "Understanding the Effect of Accuracy on Trust in Machine-Learning Models," in *Proc. of the CHI Conf. on Human Factors in Computing Systems*. ACM, 2019, pp. 1–12.

[15] A.-A. Agape, M. C. Danceanu, R. R. Hansen, and S. Schmid, "Charting the Security Landscape of Programmable Dataplanes," *arXiv preprint*, 2018, accessed on 2022-09-30.

Somnath Mazumdar is an assistant professor at the Copenhagen Business School. His research interests focus on HPC, Blockchain, and Machine Learning. He holds a PhD in Computing Systems from the University of Siena, Italy, and an M.Sc. in Distributed Computing from Polytech Nice Sophia Antipolis, France. Somnath has also worked on multiple international research projects.

Thomas Dreibholz received his Dipl.-Inform. degree in Computer Science from the University of Bonn, Germany in 2001. Later, he received his PhD degree in 2007, and his Habilitation degree in 2012 from the University of Duisburg-Essen, Germany. Now, he is a Chief Research Engineer for the SimulaMet in Oslo, Norway.