CIDER: *C*ooperative *I*n-home *D*istributed *E*fficient *R*esource Allocation Protocol

Pujan Dharmendra Dave, Ranganadh Karella, Roshini Saravanakumar*
{pdave7, karella2, roshini 2}@illinois.edu

ABSTRACT

Modern households in the U.S. contain a wide variety of underutilized powerful personal devices and an increasing number of specialized IoT devices. These IoT devices constantly collect sensitive data from homes to provide smarter experiences for their inhabitants. Since most IoT devices are resource-constrained, the data they collect is often sent to the cloud for processing, even if the required computational resources can be provisioned on local user devices. In such cases, sensitive data is unnecessarily shared with cloud service providers (CSPs) and their third-party clients, where it is vulnerable to malicious attacks on the cloud and can be used by the CSP for purposes that are not in the end user's best interest. We propose CIDER, a protocol that efficiently allocates idle resources on personal devices within a household to locally process the data collected by IoT devices. Our experimental results show that CIDER effectively distributes load across multiple user devices.

1. INTRODUCTION

As of 2020, there are around 20 billion IoT devices connected to the internet [10]. Cisco projects around 3.6 connected devices per person by 2023, with up to 10 connected devices per household [13]. In the coming years, these devices will enable countless real-time applications from low-latency recommendations for increased user engagement, to live analytics on street camera feeds to reduce traffic fatalities [3]. To handle these workloads, researchers are developing resilient edge computing architectures that provide fault tolerance [4] and adaptive data replication [5] to meet application-specific requirements for data loss, latency, and bandwidth usage. However, many of these architectures rely on the assumption that data needs to be processed on the IoT device itself, or sent to the cloud.

Motivation

The primary motivation for CIDER is privacy. In 2017, almost 1000 conference publications on IEEE and 13000 books on Springer discussed IoT privacy and security concerns [20]. Such concerns are amplified in smart homes, where internet-connected smart-home devices constantly monitor user behaviors to facilitate smart experiences for users. When this sensitive data is sent to the cloud for processing, it is only as secure as the cloud and the intentions of the cloud provider. Often, users are left in the dark on what their data is being used for, due to over-complicated end-user agreements [11].

Existing smart-home technologies also fail to consider and respect private moments that occur in homes [16]. Even though it is evident that consumers do not want to reveal their private routines, ISPs and CSPs constantly monitor encrypted traffic flows to

* Author last names are in alphabetical order.

infer potentially-sensitive user behaviors [17]. Additionally, smart-home device vendors (like Samsung) often host Automatic Content Recognition (ACR) services to fingerprint and identify private browsing for advertising purposes [18]. If vendors can do this, similar strategies (e.g., Fingerprint and Timing-based Snooping) can be employed by any attacker to snoop on private in-home activities [19].

Instead of exposing sensitive user data by processing it in the cloud, we can guarantee data privacy by using existing compute resources on trusted devices connected to the home network. We propose CIDER (***C***ooperative ***I***n-home ***D***istributed ***E***fficient ***R***esource Allocation Protocol) to prioritize data privacy in smart-home applications. CIDER efficiently allocates the idle resources on trusted devices connected to the home network to perform computations on data collected from smart-home devices.

CIDER is written in Golang and only uses one external dependency (Chi, a lightweight HTTP API router). Chi adds less than 1000 Lines of Code (LOC) to our existing source of approximately 1100 LOC. Our team has open sourced CIDER[†] and we welcome any contributions or ideas for improvement.

2. SYSTEM ARCHITECTURE

Throughout this paper, we will consistently use the terms below to describe the components of our system. *CIDER nodes* refer to user devices (e.g., laptops, phones, tablets) and Wi-Fi enabled smart-home devices (e.g., the smart home hub, smart doorbells, thermostats, voice assistants etc.) that run the CIDER client. *IoT Sensors* communicate with other smart home devices via a low-power protocol (e.g., Z-Wave, Zigbee), and do not run CIDER. Since CIDER nodes serve users in a single household, we assume that all nodes have an inherent incentive to cooperate [6, 7, 8] and form a powerful *local cluster* with the ability to run idempotent computations locally.

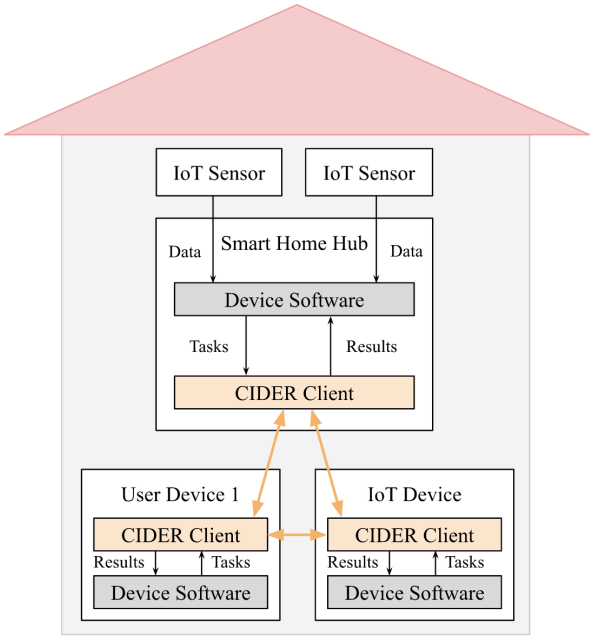

Figure 2.1 CIDER System Architecture

Each CIDER node exposes a single API to other CIDER nodes. This API provides an interface to deploy *tasks* that invoke *functions* (analogous to serverless functions in the cloud) with the provided data. This "serverless" execution platform is compared to a functionally-equivalent cloud deployment on AWS Lambda in Section 5.

[†] https://cider-io.github.io

3. CHALLENGES

As we designed CIDER, we identified several challenges that are unique to providing a reliable task execution service on top of an inherently unreliable cluster of user devices. In this section, we explore these challenges, and briefly discuss how CIDER addresses them.

Handling Churn in the Local Cluster

Unlike servers in the cloud, user devices are not dedicated compute devices. Many user devices are also mobile (e.g., phones, tablets), and may join/leave the local cluster at any time. Due to this churn, resource availability in the local cluster changes dynamically, and may also have diurnal patterns that follow the user's daily/weekly routines. It is critical that we consider these aspects to ensure that tasks can be reliably executed on the local cluster.

At any point, every CIDER node has a global view of the total compute and storage capacity available in the local cluster. This view is communicated through the gossiped membership list, and includes the number of available CPUs, total available RAM, and additional load/reputation metrics at each node. These metrics provide a lightweight load balancing mechanism for task deployments, which is discussed below.

Compute Node Selection

Compute nodes refer to CIDER nodes that have sufficient resources to run smart-home tasks. When selecting a compute node for a task, it is important to efficiently balance the load across the available compute nodes (refer to the experiments in Section 5). We define effective load on a node (L_{node}) as

$$L_{node} = \frac{T_{node}}{C_{node}} \qquad \text{(Equation 3.1)}$$

Where T_{node} is the number of tasks currently running on the node and C_{node} is the number of available CPUs (cores) on the node.

We also consider the *reputation* of a node, inspired by worker counters in Prague [25]. Reputation indicates how many tasks a node has completed so far to indicate its availability. Due to this, it is useful to persist it across the node's participations in CIDER. In our current implementation, we are not persisting the reputation but it is planned for future work. Additionally, available RAM on a node also influences how fast a task is processed, and consequently reduces the load on a node.

These metrics yield the following equation for the node score (S_{node}), which is used to estimate the node's ability to be a compute node:

$$S_{node} = \frac{M_{node}}{L_{node}} + R_{node} \qquad \text{(Equation 3.2)}$$

Where M_{node} is the RAM size of the node and R_{node} is the reputation of the node.

4. IMPLEMENTATION

Group Membership

We have implemented a simple gossip-style membership and failure detection protocol [1,2] to maintain a list of active CIDER nodes. Nodes are marked as failed after other nodes haven't received a heartbeat from them for more than $5 * log_2(N)$ seconds, where N in the size of the local cluster. Failed nodes are eventually removed from the membership list after $2 * T_{fail}$ seconds. Each node gossips to $log_2(N)$ other random nodes in the cluster in every gossip round.

CIDER API

CIDER's API supports the endpoints shown in Table 4.1, which enable task deployment on the local cluster. These endpoints communicate using the request and response schemas defined in the cider/api package (partially shown in Figure 4.2).

As shown in Figure 2.1, CIDER processes TaskRequests which expose the task results and any error messages, once they are available, at the /tasks/{id}/result endpoint. Task input data and results can be of any type (Figure 4.2, Lines 1, 5, and 9).

Assumptions

To build a minimum viable product, we have simplified a few of our implementation details for the time being.

First, we have assumed that tasks can only invoke locally stored functions (currently in the cider/functions package). Additionally, our mechanism for sending control signals to deployed tasks (Table 4.1, PUT /tasks/{id}) currently only supports aborting a task via a boolean channel. Finally, when a CIDER node receives a task, it needs to verify that the request originates from a trusted source. At the moment, this check is trivially implemented, and only accepts task requests from source IP addresses in our membership list. These assumptions are addressed in Section 6, which provides an overview of our planned future work.

/tasks	GET	Get metadata for all deployed
	PUT	Deploy a new task
/tasks/{id}	GET	Get metadata for a specific task
	PUT	Put control updates to a task
	DELETE	Delete a task
/tasks/{id} /result	GET	Get the result of a completed task

Table 4.1 Supported Endpoints

```
1  type Data interface{}
2
3  type TaskRequest struct {
4      Function string
5      Data     Data
6  }
7
8  type TaskResult struct {
9      Result Data
10     Error  string        `
11 }
```

Figure 4.2 API Schema

5. EVALUATION

Here, we evaluate the feasibility and scalability of the CIDER system, and how it performs in comparison to commercially available Function as a Service platforms (e.g., AWS Lambda). We tested our current implementation on a cluster of 10 Linux virtual machines (VMs) provided to us by the university for the duration of this course. We employ several Python and shell scripts to automate building, testing and deploying our system to the VMs.

Our evaluation scripts interact with CIDER's HTTP API to deploy computation tasks, and fetch the results, once they're available. For a particular test run, we can configure data size, number of iterations and the serverless function that is invoked. To simulate an in-home IoT environment, we configure a certain number of nodes in the system to be resource constrained with limited compute capability.

Evaluation Results

We started with a base model that simulates a local cluster with 10 nodes, 9 of which are resource constrained. This setup implies that only 1 of the 10 nodes meets the minimum compute node selection criteria. Each of the 10 nodes uses CIDER's API to deploy tasks that sum 1000 floating point numbers. We ran this experiment for 1000 iterations for all the VMs simultaneously, and observed excellent average task completion times (Table 5.1, Base Model). The total duration presented in Table 5.1 is the end-to-end time from sending the data for computation to receiving the result.

To evaluate the scalability of our system, we increased the input data size to 500,000 floating point numbers (Model 1). With this change, we noticed that the system was overwhelmed due to high bandwidth usage from flooding the one compute node with task requests (Table 5.1, Model 1).

Model	Total Duration (s)		
	Max	Mean	Median
Base	10.03	0.19	0.02
1	17.5	5.6	4.68
2	21.90	2.25	1.42
3	15.83	1.64	1.22
4	6.23	1.08	0.88
AWS λ	5.43	2.23	2.09

Table 5.1 Total Duration Comparisons for various CIDER models

To experiment with improved load balancing, we increased the number of compute nodes to a randomly selected set of size 5. This scenario is more reflective of typical smart-home environments that house at least several capable user devices. When a node needs to deploy a task, it randomly selects one of these compute nodes to deploy the task to. With randomly-selected compute node sets of size 5 (Table 5.1, Model 2), we observed some improvement in overall performance compared to single compute node models (due to improved load balancing of tasks), but we are still

extremely vulnerable to computation request floods and node failures.

To better distribute the load across the cluster, we evaluated Model 3, which randomly selects a compute node per iteration per node in the system (Table 5.1, Model 3). With this more dynamic node selection approach, we see an improvement in performance and greater fault tolerance.

Then, we take Model 3's approach and add wait time of $log_2(N) + r$ seconds where r is a random number in [0, 1) (Table 5.1, Model 4). With this approach, we are able to more effectively distribute the load across the cluster and avoid huge floods of compute requests in most scenarios.

We have also evaluated our system in comparison to a functionally-equivalent cloud deployment on AWS Lambda. We deployed the functions in our cider/functions package to AWS Lambda, and observed that our system performs faster than Lambda in the average case (due to compute locality). However, our maximum latencies are higher than those observed in our Lambda deployment. This is because we have not employed any techniques to reduce tail latencies. From these experiments, we come to a conclusion that effective load balancing across unreliable compute nodes is critical to providing a reliable task execution service.

To evaluate the effectiveness of our node scoring system, we compare two system models (each with three compute nodes – nodes 8-10). The first model simply selects the first node in its membership list that has sufficient capacity to deploy the task. The second model uses the node score, S_{node}, to select the compute node. With both of these models, each node in the cluster generates task requests and waits for some random time between 10-20 seconds. With the first model, we see disproportionately high load on the compute node 8 (Figure 5.1), while the second model distributes the load evenly across all three compute nodes (nodes 8-10, Figure 5.2).

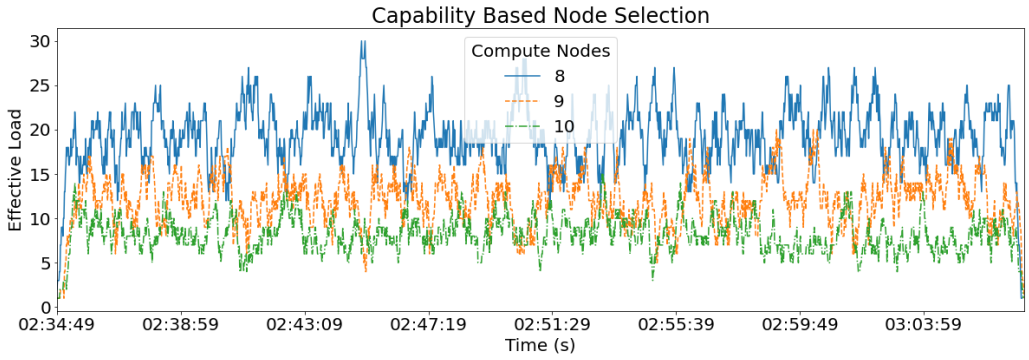

Figure 5.1
Average Load on Node-8: 19.17, Node-9: 12.06, Node-10: 8.18

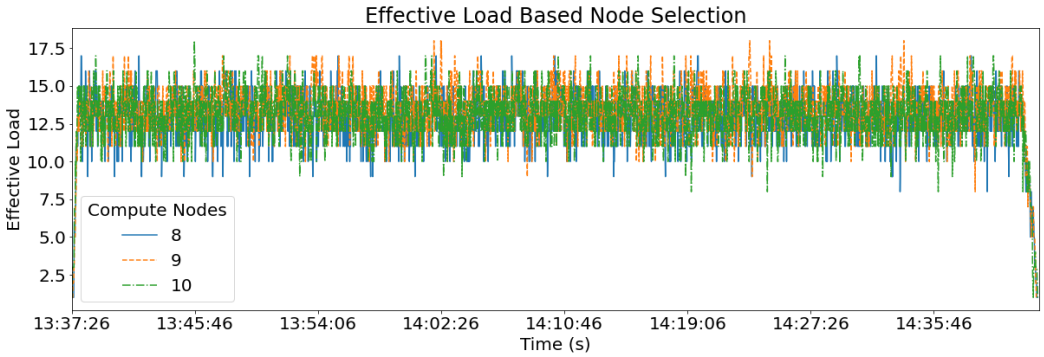

Figure 5.2
Average Load on Node 8: 12.85, Node 9: 13.46, Node 10: 13.09

6. FUTURE WORK

Given that we are working under tight deadlines, there are many features that we are unable to implement in the short timeframe of a semester. The problems listed below are quite relevant to our project, and can greatly improve its usability and future impact.

Survey of Smart-Home Computations
CIDER's functionality heavily relies on the assumption that the local cluster has sufficient available capacity to run typical smart-home computations. We were unable to find any studies that survey the computational requirements of serverless functions that are used by typical smart-home/IoT devices. Such data would further inform whether we need to consider surge computing to overflow temporarily heavy workloads to the cloud, at the cost of data privacy [21].

CIDER API
CIDER's API will be expanded to support a /functions endpoint that allows authorized devices to deploy custom functions that can be invoked by tasks. This feature will allow us to lift the limitation on what functions can be run by CIDER (which is currently just those provided in the cider/functions package). Additionally, we will update CIDER's PUT /task/{id} endpoint to allow other control signals to be sent to the task (e.g., pause/resume a task's execution).

Security
Our implementation currently serves the CIDER API over HTTP. Even though data processed on CIDER will never leave the home network, we still want to enforce confidentiality and integrity in our communications, since we cannot easily determine whether a user device has been compromised [12]. To support these properties and any future access control mechanisms, we will serve our API over HTTPS and employ a distributed authorization protocol. We plan on providing a similar access control mechanism to that provided by AWS Lambda [15]. This will allow us to ensure that only authorized devices can invoke functions, deploy tasks, and view task results on CIDER's platform.

Fault Tolerance
Currently, we don't handle cases where a node fails while executing a task. In this case, it is expected that the IoT device/gateway is capable of retransmitting recent task requests that have failed. This is not ideal, so we would like to abstract this functionality into our system.

Smart Load Balancing
We would like our design to dynamically choose the size of the data being transferred based on the cluster resource availability. Furthermore, our system would also benefit from restricting the number of nodes that can participate in a computation round to avoid overwhelming the compute nodes with a flood of task requests. We should also avoid scheduling jobs on failed nodes until they have recovered. AWS Lambda limits request payloads to 6MB by default, when synchronously invoking a function and waiting for the response [14]. Meanwhile, CIDER can handle significantly larger loads, by chunking data and processing it in batches.

Persistent Reputation
It is useful to persist the reputation of a node, or some fraction of it, across its participations in CIDER to provide historical information on the availability of the node. This can further guide the compute node selection algorithm based on the requirements of a task being deployed.

Power and Efficiency Considerations
IoT devices and smaller user devices usually have limited resources, and are especially power-constrained since transmitting data to the network via antenna transmitters consumes significant power [22]. In the future, we want to expand our compute node selection algorithm to consider additional node characteristics, like sleep/wakeup times, power consumption patterns, etc.

7. RELATED WORK

A highly related work that we found cites recent trends of bringing the compute to the data, which initially appears similar to CIDER's approach [24]. The researchers propose a technique to offload workloads to local smart devices and IoT gateways [24]. Their technique focuses on improving the gateway's bandwidth utilization, while decreasing power consumption on edge devices [24]. This is orthogonal to our proposal, since CIDER prioritizes data privacy. We also focus on using available compute power on all user devices in a smart home, not just gateways.

In addition to the challenges that CIDER addresses, there are other relevant security aspects in smart home ecosystems. IoT devices often lack sufficient resources to run existing IP-based security protocols [13], so researchers have designed lightweight communication [21], authentication, and authorization [11] protocols for IoT. In the area of distributed machine learning, we see great interest in designing federated learning systems that leverage secure aggregation to collaboratively train a model on the user devices [5] or silos [6] themselves, instead of sending potentially sensitive data to the cloud. There has also been recent interest in verifying the trustworthiness of the IoT

device itself [12] and securing the device boot process [23].

Quality of Service Requirements

The devices in our home cluster are not dedicated compute devices, so they also need to handle user workloads, leading to variable resource availability. Additionally, in clouds, computation can easily be monitored and fully managed, which is not the case when it comes to running an application on a local cluster of user devices. It is an open research problem to meet Quality of Service (QoS) requirements in edge computing [9].

8. CONCLUSION

We believe that processing sensitive data within the home has powerful implications for the future of smart homes. CIDER enables secure and efficient processing of smart home data by scheduling tasks onto underutilized, trusted user devices. We encountered many challenges in providing a reliable task execution service on top of an inherently unreliable local cluster, which we have addressed in our implementation and in our planned future work. Our evaluation results are consistent with our expectations, and certainly highlight areas for further improvement. Our team has open sourced CIDER[†] and we welcome any contributions or feedback.

REFERENCES

[1] Van Renesse, R., Minsky, Y. and Hayden, M., 1998. A gossip-style failure detection service. In Middleware'98 (pp. 55-70). Springer, London.

[2] Das, A., Gupta, I. and Motivala, A., 2002, June. Swim: Scalable weakly consistent infection-style process group membership protocol. In Proceedings International Conference on Dependable Systems and Networks (pp. 303-312). IEEE.

[3] Bahl, V., 2020. Edge Computing for the (Telecom) Infrastructure. San Francisco, Future of Information and Communication Conference

[4] Ardekani, M.S., Singh, R.P., Agrawal, N., Terry, D.B. and Suminto, R.O., 2017, December. Rivulet: A fault-tolerant platform for smart-home applications. In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (pp. 41-54).

[5] Wang, C., Gill, C. and Lu, C., 2020, April. Adaptive Data Replication in RealTime Reliable Edge Computing for Internet of Things. In 2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI) (pp. 128-134). IEEE.

[6] Jun, S. and Ahamad, M., 2005, August. Incentives in BitTorrent induce free riding. In Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems(pp. 116-121).

[7] S. Lin and Y.-C. Cheng. A barter-based incentive mechanism for peer-to-peer media streaming. In Digest of Technical Papers —

[†] https://cider-io.github.io

IEEE International Conference on Consumer Electronics, pages 871–875, 2009.

[8] A. Habib and J. Chuang, “Service differentiated peer selection: an incentive mechanism for peer-to-peer media streaming,” Multimedia, IEEE Transactions on, vol. 8, no. 3, pp. 610–621, June 2006.

[9] Srirama, S.N., Dick, F.M.S. and Adhikari, M., 2021. Akka framework based on the Actor model for executing distributed Fog Computing applications. Future Generation Computer Systems, 117, pp.439-452.

[10] Hung M., 2017. Leading the IoT: Gartner Insights on How to Lead in a Connected World. [Online] Available at: https://www.gartner.com/imagesrv/books/iot/iotEbook_digital.pdf [Accessed 28 Feb[1]ruary 2021].

[11] Noto La Diega, G. and Walden, I., 2016. Contracting for the ‘Internet of Things’: Looking into the Nest. Queen Mary School of Law Legal Studies Research Paper, (219).

[12] Hernandez, G., Arias, O., Buentello, D. and Jin, Y., 2014. Smart nest thermostat: A smart spy in your home. Black Hat USA, (2015).

[13] Cisco, 2020. New Cisco Annual Internet Report Forecasts 5G to Support More Than 10% of Global Mobile Connections by 2023. [Online] Available at: https://newsroom.cisco.com/press-releasecontent?type=webcontent&articleId=2055169 [Accessed 28 February 2021].

[14] AWS Lambda, Lambda Quotas, [Online] Available at: https://docs.aws.amazon.com/lambda/latest/dg/gettingstarted-limits.html [Accessed 4 April 2021].

[15] Lambda, A.W.S., 2019. Security Overview of AWS Lambda.

[16] Choe, E.K., Consolvo, S., Jung, J., Harrison, B. and Kientz, J.A., 2011, September. Living in a glass house: a survey of private moments in the home. In *Proceedings of the 13th international conference on Ubiquitous computing* (pp. 41-44).

[17] Apthorpe, N., Reisman, D., Sundaresan, S., Narayanan, A. and Feamster, N., 2017. Spying on the smart home: Privacy attacks and defenses on encrypted iot traffic. *arXiv preprint arXiv:1708.05044*.

[18] Mazhar, M.H. and Shafiq, Z., 2020, April. Characterizing Smart Home IoT Traffic in the Wild. In *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)* (pp. 203-215). IEEE.

[19] Srinivasan, V., Stankovic, J. and Whitehouse, K., 2008, September. Protecting your daily in-home activity information from a wireless snooping attack. In *Proceedings of the 10th international*

conference on Ubiquitous computing (pp. 202-211).

[20] Neeli, J. and Patil, S., 2021. Insight to Security Paradigm, Research Trend & Statistics in Internet of Things (IoT). *Global Transitions Proceedings*.

[21] Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M., 2010. A view of cloud computing. *Communications of the ACM*, *53*(4), pp.50-58.

[22] Anajemba, J.H., Tang, Y., Iwendi, C., Ohwoekevwo, A., Srivastava, G., Jo, O. 2020. Realizing Efficient Security and Privacy in IoT Networks. Sensors 2020, 20, 2609.

[23] Hernandez, G., Arias, O., Buentello, D. and Jin, Y., 2014. Smart nest thermostat: A smart spy in your home. Black Hat USA, (2015).

[24] Samie, F., Tsoutsouras, V., Bauer, L., Xydis, S., Soudris, D. and Henkel, J., 2016, December. Computation offloading and resource allocation for low-power IoT edge devices. In 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT) (pp. 7-12). IEEE.

[25] Luo, Q., He, J., Zhuo, Y. and Qian, X., 2020, March. Prague: High-performance heterogeneity-aware asynchronous decentralized training. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 401-416).