REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
25-06-2021	Final Report	29-May-2018 - 31-May-2020

4. TITLE AND SUBTITLE
Final Report: Equipment for Secure Coded Cooperative Computations for Internet of Battlefield of Things (IoBTs)

5a. CONTRACT NUMBER
W911NF-18-1-0211

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611103

6. AUTHORS

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
Rutgers, The State University of New Jersey
3 Rutgers Plaza
ASB III, 2nd Floor
New Brunswick, NJ 08901 -8559

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
72233-CS-RIP.1

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Salim El Rouayheb
UU	UU	UU	UU		19b. TELEPHONE NUMBER 848-445-5241

Standard Form 298 (Rev 8/98)
Prescribed by ANSI Std. Z39.18

RPPR Final Report
as of 25-Jun-2021

Agency Code: 21XD

Proposal Number: 72233CSRIP **Agreement Number: W911NF-18-1-0211**
INVESTIGATOR(S):

Name: Hulya Seferoglu
Email: hulya@uic.edu
Phone Number: 3124137573
Principal: N

Name: PhD Salim Y El Rouayheb
Email: salim.elrouayheb@rutgers.edu
Phone Number: 8484455241
Principal: Y

Organization: **Rutgers, The State University of New Jersey - New Brunswick**
Address: 3 Rutgers Plaza, New Brunswick, NJ 089018559
Country: USA
DUNS Number: 001912864 EIN: 226001086

Report Date: 31-Aug-2020 Date Received: 25-Jun-2021
Final Report for Period Beginning 29-May-2018 and Ending 31-May-2020
Title: Equipment for Secure Coded Cooperative Computations for Internet of Battlefield of Things (IoBTs)
Begin Performance Period: 29-May-2018 **End Performance Period:** 31-May-2020
Report Term: 0-Other

Submitted By: PhD Salim El Rouayheb Email: salim.elrouayheb@rutgers.edu
Phone: (848) 445-5241

STEM Degrees: **STEM Participants:**

Major Goals: The proposal is aimed at acquiring equipment to build heterogeneous IoBT network testbeds. The research plan includes two testbeds that will be connected at a later stage: Testbed I at Rutgers University focused on security and Testbed II at UIC focused on dis- tributed resource optimization. The needed equipment (listed in Table 2 and Table 3) includes among others surveillance drones, server racks, laptops, and Android phones. The specific choice of this equipment was dictated by the need to have non-homogeneous devices with widely varying processing, power, and memory capabilities, as well as mobility properties. Specifically, funds are requested to acquire the equipment needed to achieve three main functionalities that we have identified as crucial for IoBT testbeds:

• F1: Heterogeneity, due to the wide spectrum of IoBT devices (sensors, reconnaissance and monitoring cameras, drones, base stations, computing servers, etc.).

• F2: Dynamism and Adaptivity, to reflect the time-varying state of the network such as devices battery levels, wireless channel, etc.

• F3: Mobility, caused by mobile components that are able to move and change their locations, such as drones, driverless vehicles, etc.

Enhancements will include (i) developing a testbed (Testbed II) of different configurations including homogeneous, heterogeneous, and mobile devices, and testing resource allocation algorithms for cooperative computation in these setups to evaluate the performance of our algorithms in a wide variety of conditions, (ii) developing a testbed (Testbed I) of different configurations to evaluate the performance of our low-complexity secure data algorithms against eavesdropping and malicious adversaries for the same configurations described for Tesbed II and (iii) combining the security and resource allocation aspects as well as connecting the two testbeds located at Rutgers and UIC. These new instruments and capabilities will be of immediate value to defense-related projects at both institutions, in particular, ARL W911NF-17-1-0032 on “Coded Cooperative Computation for Internet of Battlefield Things (IoBTs)”. The proposed testbeds capabilities will also be available to faculty and researchers at both institutions and will enable a broad range of experiments across both research and educational domains.

RPPR Final Report
as of 25-Jun-2021

Accomplishments: This DURIP project is a collaboration between Rutgers University and the University of Illinois at Chicago (UIC) to develop and build platforms for testing heterogeneous Internet of Battlefield Things (IoBTs) networks. This enabled testing the two PIs theoretical algorithms and protocols for secure and adaptive cooperative computing that are the product of their ongoing collaborative research funded by the Army Research Lab.

The proposal funded acquiring equipment to build the following testbeds for IoBT.

1. PRAC Testbed: a testbed for private and rateless adaptive coded computation (PRAC) for edge computing on IoBT.

2. ResPipe Testbed: a testbed for distributed Deep Neural Network (DNN) training for IoBT de- ices at edge networks.

Specifically, these testbeds, the algorithms and the experiments that were run were specifically designed to address the following functionalities that were identified in our proposal as crucial for IoBT, namely:
• F1: Heterogeneity, due to the wide spectrum of IoBT devices (sensors, reconnaissance and monitoring cameras, drones, base stations, computing servers, etc.).
• F2: Dynamism and Adaptivity, to reflect the time-varying state of the network such as devices battery levels, wireless channel, etc.
• F3: Mobility, caused by mobile components that are able to move and change their locations, such as drones, driverless vehicles, etc.

More details about the testbeds and experiments and results can be found in the final report and in the referenced publications [1–5].

Training Opportunities: The conducted research supported by this proposal contributed to training PhD students in the engineering field in problems related to security in IoT and IoBT applications. Moreover, due the multidisciplinary nature between the two PIs El Rouayheb (Informatinon and coding theory and Seferoglu (Networking), this created an opportunity for students to collaborate and communicate with researchers from other fields in order to solve multi-faceted problems such as the problem of security in IoBT.

Results Dissemination: The research work was documented and published in the journal and conference papers below. In addition, the PIs gave several invited talks related to security, IoT and IoBT in different conferences and special seminars.

[1] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, “Private and rateless adaptive coded matrix-vector multiplication,” EURASIP Journal on Wireless Communications and Networking, vol. 2021, no. 1, pp. 1–25, 2021.

[2] R. Bitar, P. Parag, and S. El Rouayheb, “Minimizing latency for secure distributed computing,” in 2017 IEEE International Symposium on Information Theory (ISIT). IEEE, 2017, pp. 2900–2904.

[3] Y. Keshtkarjahromi, R. Bitar, V. Dasari, S. El Rouayheb, and H. Seferoglu, “Secure coded co-operative computation at the heterogeneous edge against byzantine attacks,” in IEEE Global Communication Conference (GLOBECOM), 2019.

[4] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, “Prac: private and rateless adaptive coded computation at the edge,” in Disruptive Technologies in Information Sciences II, vol. 11013. International Society for Optics and Photonics, 2019, p. 110130T.

[5] P. Li, E. Koyuncu, and H. Seferoglu, “Respipe: Resilient model-distributed DNN training at edge networks,” in ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP). IEEE, 2021, pp. 3660–3664.

[6] P. Li, H. Seferoglu, V. Dasari, E. Koyuncu, "Model-Distributed DNN Training for Memory-Constrained Edge Computing Devices," in IEEE LANMAN, Virtual, July 2021.

RPPR Final Report
as of 25-Jun-2021

Honors and Awards: Nothing to Report

Protocol Activity Status:

Technology Transfer: Nothing to Report

Partners

,

I certify that the information in the report is complete and accurate:
Signature: Salim El Rouayheb
Signature Date: 6/25/21 8:59AM

Final Report: DURIP: Equipment For Secure Coded Cooperative Computations for Internet of Battlefield of Things (IoBTs)

Salim El Rouayheb, sye8@scarletmail.rutgers.edu
Hulya Seferoglu, hulya@uic.edu

1 Introduction

This DURIP project is a collaboration between Rutgers University and the University of Illinois at Chicago (UIC) to develop and and build platforms for testing heterogeneous Internet of Battlefield Things (IoBTs) networks. This enabled testing the two PIs theoretical algorithms and protocols for secure and adaptive cooperative computing that are the product from their on-going collaborative research funded by the Army Research Lab.

The proposal funded acquiring equipment to build the following testbeds for IoBT. The testbeds and experiments and results can be found in the following publications [1–5].

1. PRAC Testbed: a testbed for private and rateless adaptive coded computation (PRAC) for edge computing on IoBT.

2. ResPipe Testbed: a testbed for distributed Deep Neural Network (DNN) training for IoBT deices at edge networks.

Specifically, these testbeds, the algorithms and the experiments that were run were specifically designed to address the following functionalities that were identified in our proposal as crucial for IoBT, namely:

- F1: Heterogeneity, due to the wide spectrum of IoBT devices (sensors, reconnaissance and monitoring cameras, drones, base stations, computing servers, etc.).

- F2: Dynamism and Adaptivity, to reflect the time-varying state of the network such as devices battery levels, wireless channel, etc.

- F3: Mobility, caused by mobile components that are able to move and change their locations, such as drones, driverless vehicles, etc.

Below we give details about the motivation behind these testbeds, the theoretical setup and experimental results.

2 PRAC Testbed

2.1 Background on PRAC

The goal of the PRAC testbed is to run and test new algorithms for edge computing in IoBT settings that can accomodate the three functionalities above. The edge computing paradigm allows

processing data near the edge of the network, where the data is typically generated and collected. This enables computation at the edge in applications such as Internet of Battlefield Things (IoBT), in which an increasing number of devices (sensors, cameras, health monitoring devices, etc.) collect data that needs to be processed through computationally intensive algorithms with stringent reliability, security and latency constraints.

Edge computing advocates that computationally intensive tasks in a device (master) could be offloaded to other edge or end devices (workers) in close proximity. However, offloading tasks to other devices leaves the IoT and the applications it is supporting at the complete mercy of an attacker. Furthermore, exploiting the potential of edge computing is challenging mainly due to the heterogeneous and time-varying nature of the devices at the edge. Thus, our goal for PRAC is to develop a private, dynamic, adaptive, and heterogeneity-aware cooperative computation platform that provides both privacy and computation efficiency guarantees. Note that the application of this work can be extended to involve cloud computing at remote data-centers. However, we focus on edge computing as heterogeneity and time-varying resources are more prevalent at the edge as compared to data-centers.

2.2 PRAC Platform

We design PRAC for heterogeneous and time-varying private coded computing with colluding workers. In particular, PRAC codes sub-tasks using fountain codes, and determines how many coded packets and keys each worker should compute dynamically over time. We provide theoretical analysis of PRAC and show that it (i) guarantees privacy conditions, (ii) uses minimum number of keys to satisfy privacy requirements, and (iii) maintains the desired rateless property of non-private fountain codes. Furthermore, we provide a closed form task completion delay analysis of PRAC. Finally, we evaluate the performance of PRAC via simulations as well as in a testbed consisting of real Android-based smartphones as compared to baselines.

Our PRAC platform includes the following. A master device referred to as master wishes to offload intensive computations to n helpers (IoBT devices) referred to as workers. The workers cannot be trusted with the privacy of the master's data but will run the required computations. The workers have different computation and network resources that change with time which creates a time-varying and heterogeneous computing environment. Workers that are slow or unresponsive are termed as stragglers. The goal of the master is to assign tasks that are proportional to the overall speed of the workers and tolerate the presence of straggling workers. This will be done by assigning a new task to a worker when it is expected to have finished its previous task.

The key tool behind our algorithms to be tested is the theory of coded computation, which advocates mixing data in computationally intensive tasks by employing erasure codes and offloading these tasks to other devices for computation [6–18].

We developed in [1,4] algorithms for PRAC that are (i) private as it is secure against eavesdropper adversary, (ii) rateless, because it uses fountain codes [19–21] instead of Maximum Distance Separable (MDS) codes[1] [22, 23], and (iii) adaptive as the master device offloads tasks to workers by taking into account their heterogeneous and time-varying resources. Next, we illustrate the main idea of PRAC through an illustrative example. As such, our algorithms tested on PRAC enjoy the three functionalities highlighted above and required for IoBT.

2.3 PRAC System Model

Setup. We consider a master/workers setup at the edge of the network, where the master device

[1]Our security scheme can be applied to any linear code. However, we choose fountain code since they are a better fit for edge computing characterized by heterogeneous and time-varying behavior.

offloads its computationally intensive tasks to workers $w_i,\ i \in [n]$, via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. The master device divides a task into smaller sub-tasks, and offloads them to workers that process these sub-tasks in parallel.

Task Model. We focus on the computation of linear functions, matrix-vector multiplication. We suppose the master wants to compute the matrix vector product $A\mathbf{x}$, where $A \in \mathbb{F}_q^{m\times\ell}$ can be thought of as the data matrix and $\mathbf{x} \in \mathbb{F}_q^{\ell}$ can be thought of as an attribute vector. We assume that the entries of A and $\mathbf{x}$ are drawn independently and uniformly at random[2] from $\mathbb{F}_q$. The motivation stems from machine learning applications where computing linear functions is a building block of several iterative algorithms [24,25]. For instance, the main computation of a gradient descent algorithm with squared error loss function is

$$\mathbf{x}^+ = \mathbf{x} - \alpha A^T(A\mathbf{x} - \mathbf{y}), \tag{1}$$

where $\mathbf{x}$ is the value of the attribute vector at a given iteration, $\mathbf{x}^+$ is the updated value of $\mathbf{x}$ at this iteration and the learning rate α is a parameter of the algorithm. Equation (1) consists of computing two linear functions $A\mathbf{x}$ and $A^T\mathbf{w} \triangleq A^T(A\mathbf{x} - \mathbf{y})$.

Worker and Attack Model. The workers incur random delays while executing the task assigned to them by the master device. The workers have different computation and communication specifications resulting in a heterogeneous environment which includes workers that are significantly slower than others, known as stragglers. Moreover, the workers cannot be trusted with the master's data. We consider an *eavesdropper adversary*, where one or more of workers are compromised by an adversary who wants to spy on the coded data sent to these devices for computations. In a comprehensive solution for IoT networks, we assume that standard security protocols (such as datagram transport layer security - DTLS) will be in place, in addition to our proposed scheme, in order to protect the wireless links from eavesdropping from nodes other than the intended workers in the master-worker links. We assume that up to z, $z < n$, workers can collude, z workers can share the data they received from the master in order to obtain information about A. The parameter z can be chosen based on the desired privacy level; a larger z means a higher privacy level and vice versa. One would want to set z to the largest possible value for maximum, $z = n - 1$ security purposes. However, this has the drawback of increasing the complexity and the runtime of the algorithm. In our setup we assume that z is a fixed and given system parameter.

Coding & Secret Keys. The matrix A can be divided into b row blocks (we assume that b divides m, otherwise all-zero rows can be added to the matrix to satisfy this property) denoted by A_i, $i = 1, \ldots, b$. The master applies fountain coding [19–21] across row blocks to create information packets $\nu_j \triangleq \sum_{i=1}^{m} c_{i,j}A_i$, $j = 1, 2, \ldots,$ where the $c_{i,j} \in \{0, 1\}$. An information packet is a matrix of dimension $m/b \times \ell$, i.e., $\nu_j \in \mathbb{F}_q^{m/b\times\ell}$. Note that information packets can be encoded using any linear code. Fountain codes enable a fluid encoding of the information and allow the master to obtain $\mathbf{x}$ as long as enough packets are collected from the workers, irrespective of their origin, which makes them a better fit for the heterogeneous and time-varying setting we consider [6]. In order to maintain privacy of the data, the master device generates random matrices R_i of dimension $m/b \times \ell$ called *keys*. The entries of the $_i$ matrices are drawn uniformly at random from the same field as the entries of A. Each information packet ν_j is *padded* with a linear combination of z keys $f_j(R_{i,1}, \ldots, R_{i,z})$ to create a secure packet $s_j \in \mathbb{F}_q^{m/b\times\ell}$ defined as $s_j \triangleq \nu_j + f_j(R_{i,1}, \ldots, R_{i,z})$. We show that encoding the random keys using an MDS code is necessary to guarantee the perfect privacy of the data. Therefore, even though information packets are encoded using Fountain codes, the linear combinations of the random matrices are created using MDS codes.

The master device sends $\mathbf{x}$ to all workers, then it sends the keys and the s_j's to the workers according to our PRAC scheme described later. Each worker multiplies the received packet by $\mathbf{x}$ and

[2]We abuse notation and denote both the random matrix representing the data and its realization by A. We do the same for $\mathbf{x}$.

sends the result back to the master. Since the encoding is rateless, the master keeps sending packets to the workers until it can decode $A\mathbf{x}$. The master then sends a stop message to all the workers.

Privacy Conditions. Our primary requirement is that any collection of z (or less) workers will not be able to obtain any information about A, in an information theoretic sense.

In particular, let P_i, $i = 1 \ldots, n$, denote the collection of packets sent to worker w_i. For any set $\mathcal{B} \subseteq \{1, \ldots, n\}$, let $P_{\mathcal{B}} \triangleq \{P_i, i \in \mathcal{B}\}$ denote the collection of packets given to worker w_i for all $i \in \mathcal{B}$. The privacy requirement[3] can be expressed as

$$H(A|P_{\mathcal{Z}}) = H(A), \quad \forall \mathcal{Z} \subseteq \{1, \ldots, n\} \text{ s.t. } |\mathcal{Z}| \leq z. \tag{2}$$

$H(A)$ denotes the entropy, or uncertainty, about A and $H(A|P_{\mathcal{Z}})$ denotes the uncertainty about A after observing $P_{\mathcal{Z}}$.

Delay Model. We focus on the delays incurred by distributing the tasks to the workers and overlook the time spent on encoding and decoding the tasks[4] at the master. Each packet transmitted from the master to a worker w_i, $i = 1, 2, ..., n$, experiences the following delays: (i) transmission delay for sending the packet from the master to the worker, (ii) computation delay for computing the multiplication of the packet by the vector $\mathbf{x}$, and (iii) transmission delay for sending the computed packet from the worker w_i back to the master. We denote by $\beta_{t,i}$ the computation time of the t^{th} packet at worker w_i and RT_i denotes the average round-trip time spent to send and receive a packet from worker w_i.

2.4 PRAC Experiments

Setup. The master device is a Nexus 5 Android-based smartphone running 6.0.1. The worker devices are Nexus 6Ps running Android 8.1.0. The master device connects to worker devices via Wi-Fi Direct links and the master is the group owner of Wi-Fi Direct group. The implementation of all the algorithms is done using an Android application written in Java. The master and the workers form a star topology where the master device is the center. The devices are placed approximately 10 inches away from each other. The devices communicate via TCP sockets. TCP sockets are wrapped by data output and input stream to send and receive different data types such as integer, double and bytes. A simple communication protocol is built between the master and the workers. The master firstly sends the size of the data followed by the actual data for multiplication. The worker receives the data, process it and sends the acknowledgement and the results back to the master. The master retrieves the results and estimates the processing delay. To allow parallel processing, the master device utilize multi-threading and deals with each worker by independent threads. We denote this type of thread as "worker thread". A main thread at the master controls all the worker threads. For PRAC, the main thread records the round of each worker. All the results from each worker thread are reported to the main thread as well.

The master device is required to complete one matrix multiplication ($y = Ax$) where A is of dimensions $60 \times 10,000$ and x is a $10,000 \times 1$ vector. We also take $m = b$ each packet is a row of A. In our implementations the workers are dedicated to the master and do not run background applications or other computing tasks. Therefore, we introduced an artificial delay at the workers following an exponential distribution. The introduced delays serves to emulate real scenarios in which workers would be running other applications in the background. We manipulate the delays in the experiment to analyze the performance of PRAC and other baselines algorithms in the presence

[3]In some cases the vector $\mathbf{x}$ may contain information about A and therefore must not be revealed to the workers. Our scheme can be easily generalized to account for such cases.

[4]It is worth noting that fountain codes enjoy a linear time encoding and decoding complexity. Other codes such Reed-Solomon codes require inverting a $k \times k$ matrix and have decoding complexity of at least $\mathcal{O}(k \log k)$ if the generator matrix is a Vandermonde matrix and $\mathcal{O}(k^2)$ in general.

of stragglers and validate our theoretical findings. A worker device sends the result to the master after it is done calculating and the introduced delay has passed. Furthermore, we assume that $z = 1$ there is one unknown worker that is adversarial among all the workers. The experiments are conducted in a lab environment where there are other Wi-Fi networks operating in the background.

Baselines. Our PRAC algorithm is compared to three baseline algorithms: (i) Staircase codes that preallocate the tasks based on n, the number of workers, k, the minimum number of workers required to reconstruct the information, and z, the number of colluding workers; (ii) GC3P in which we assume the adversarial worker is known and excluded during the task allocation; (iii) Non secure C3P in which the security problem is ignored and the master device will utilize every resource without randomness. In this setup we run C3P on $n - z$ workers.

Results. Figure 1 presents the task completion time with increasing number of workers for the homogeneous setup, when all the workers have similar computing times. Computing delay for each packet follows an exponential distribution with mean $\mu = 1/\lambda = 3$ seconds in all workers. C3P performs the best in terms of completion time, but C3P does not provide any privacy guarantees. PRAC outperforms Staircase codes when the number of workers is 5. The reason is that PRAC performs better than Staircase codes in heterogeneous setup, and when the number of workers increases, the system becomes a bit more heterogeneous. GC3P significantly outperforms PRAC in terms of completion time. Yet, it requires a prior knowledge of which worker is adversarial, which is often not available in real world scenarios.

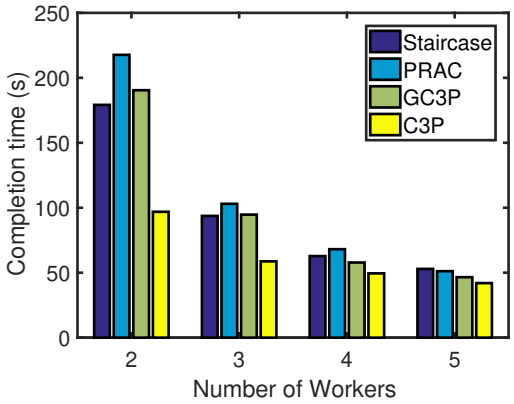


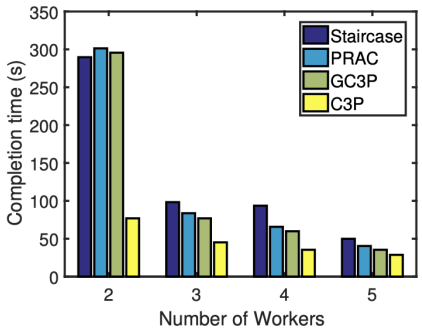
Figure 1: Completion time as function of the number of workers in homogeneous setup.

Now, we focus on heterogeneous setup. We group the workers into two groups; fast workers (per task delay follows exponential delay with mean 2 seconds) and slow workers (per task delay follows exponential distribution with mean 5 seconds). Figure 2 presents the completion time as a function of number of workers. In this setup, for the n-worker scenario, there are $\lceil \frac{n}{2} \rceil$ fast and $\lfloor \frac{n}{2} \rfloor$ slow workers. The difference between the setups of Figure 2(a) and Figure 2(b) is that we remove a fast worker (as adversarial) for GC3P in the former, whereas in the latter, we assume that the eavesdropper is a slow worker. As illustrated in Figure 2, for the 2-worker case, due to the 5% overhead introduced by fountain codes, PRAC performs worse than Staircase code. However, PRAC outperforms Staircase codes in terms of completion time for 3, 4, and 5 worker cases. This is due to the fact that PRAC can utilize results calculated by slow workers more effectively when the number of workers is large. On the other hand, the results computed by slow workers are often discarded in Staircase codes, which is a waste of computation resources. If a fast worker is removed as adversarial for GC3P, the difference between the performance of GC3P and PRAC becomes smaller. This result is intuitive as, in PRAC, the master has to wait for the $(z+1)^{\text{st}}$ fastest worker to decode , which is also the case for GC3P in this setting.

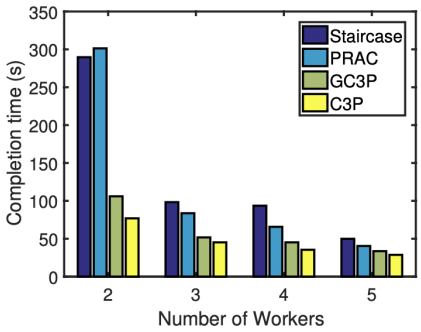
In Figure 3, we consider the same setup with the exception that for the n-worker scenario, there

are $\lceil \frac{n}{2} \rceil$ slow and $\lfloor \frac{n}{2} \rfloor$ fast workers. Staircase codes perform more closely to PRAC in the 3-worker case as compared to Figure 2 since the setup of Fig.6 assumes that the n-z=2 slowest workers are homogeneous, whereas in Fig.5 the n-z=2 slowest workers are heterogeneous. Yet, for 5-worker case, PRAC outperforms Staircase codes when comparing to Figure 2 since PRAC is adaptive to time-varying resources while Staircase codes assigns tasks a priori in a static manner.

Note that in all experiments when $n - z$ slowest workers are homogeneous Staircase codes outperform GC3P and PRAC. This happens because pre-allocating the tasks to the workers avoids the overhead of sub-tasks required by fountain codes and utilizes all the workers to their fullest capacity.

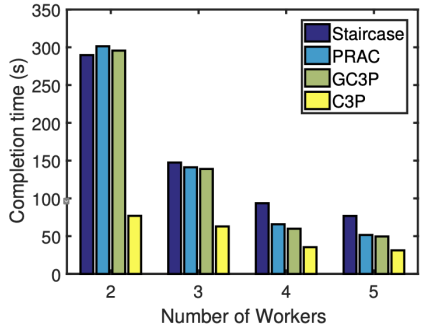


(a) We assume a fast worker is adversarial for GC3P.

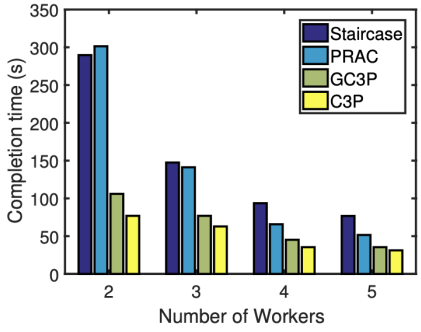


(b) We assume a slow worker is adversarial for GC3P.

Figure 2: Completion time as function of the number of workers in heterogeneous setup.



(a) We assume a fast worker is adversarial for GC3P.



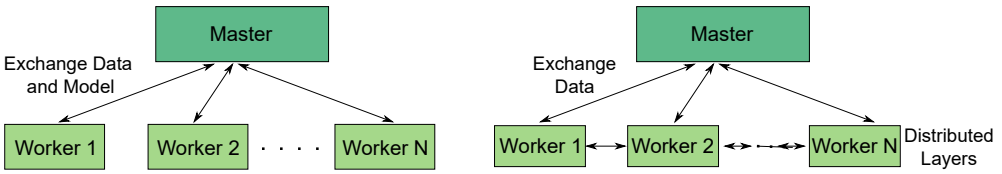
(b) We assume a slow worker is adversarial for GC3P.

Figure 3: Completion time as function of the number of workers in heterogeneous setup.

3 ResPipe Testbed

3.1 Background on ResPipe

The traditional approach to distributed deep neural network (DNN) training is *data-distributed* learning, which partitions and distributes data to workers as illustrated in a master/worker setup in Fig. 4(a). In this setup, the data is partitioned as many times as there are workers in the system and all workers train multiple instances of the same model on different subsets of the training dataset. The workers apply the same algorithm to different datasets. The same model is available to all



(a) Data-Distributed Learning (b) Model-Distributed Learning

Figure 4: Data- and Model-Distributed Learning

workers after updated at the master device. Data-distributed learning (also called data parallelism) has been frequently used as a de-facto distributed learning mechanism in practical systems, especially in multi-GPU platforms. This approach, although has good convergence properties, has high communication and storage costs, which puts a strain especially on IoBT devices at edge networks. Excessive communication among computing entities due to frequent weight synchronization in data-distributed learning increases transmission delay and overall training delay. Furthermore, computing entities should receive and store all the weights, which may not be feasible for large models when storage is limited.

Model-distributed learning is an emerging approach, where parts/layers of a training model are distributed across workers, Fig. 4(b). In this setup, no single worker has a complete model, but it is distributed across all workers. Thanks to distributing model across multiple devices, the whole model does not need to be exchanged among master and workers, which is promising to reduce communication and storage costs. This approach has been recently investigated in multi-GPU platforms, [26–28].

Distributed DNN training in edge systems has unique challenges as compared to multi-GPU platforms due to the heterogeneous and dynamic nature of edge computing systems and resources as well as unreliable and delayed computing entities. Furthermore, model-distributed learning is more vulnerable to delayed and failing workers as compared to data-distributed learning, because workers depend on each other to collectively train the distributed model as seen in Fig. 4.

In this project, we focus on model-distributed DNN training at edge networks where edge devices may be delayed or fail during training. We design *ResPipe*, a novel resilient model-distributed DNN training algorithm, where layers of a DNN training model instead of data are distributed across workers. ResPipe is resilient against delayed/failing workers thanks to proactively exchanging weights of the model among workers. We analyze the communication cost of ResPipe, and show that there is a trade-off between communication cost and resiliency. We implement ResPipe in a testbed consisting of Android-based smartphones, and show that it improves the convergence rate and accuracy as compared to baselines; data-distributed training and PipeDream [26] for convolutional neural networks (CNNs).

3.2 ResPipe System Model

Setup. We consider a distributed computing system formed of connected computing entities; end devices, edge servers, and cloud. We divide these computing entities into *(i) masters* who want to perform intensive computations on their collected data; or *(ii) workers* who are willing to dedicate some of their resources to help in the computations. There could be multiple masters and workers in the system, which may overlap. We consider a multi-hop network where each device in the network is responsible with relaying and computing offloaded tasks.

Master/Worker Model. We focus on a master/worker setup at the edge network, where the master device offloads its computationally intensive tasks to Worker $n \in$ (where $\triangleq \{1, \ldots, N\}$) via device-to-device (D2D) links such as Wi-Fi Direct and/or Bluetooth. Computationally intensive tasks are determined by DNNs.

The workers have the following properties: *(i) Failures:* Workers may fail or "sleep/die" or leave the network before finishing their assigned computational tasks. *(ii) Stragglers:* Workers incur probabilistic delays in responding to the master, where delay has two components; transmission delay for exchanging data, model, and parameters, and computation delay.

Learning Model. We denote the training dataset by $A^T \in \mathbb{R}^{m \times d}$, the label vector by $\mathbf{z} \in \{0,1\}^m$, and the weight vector of the model by $\mathbf{x}$. The weight vector of a DNN is obtained by minimizing the loss function $C(\mathbf{x})$, which is determined by the learning model, via gradient descent [29]; $\mathbf{x}^{j+1} = \mathbf{x}^j - \eta \nabla(C(\mathbf{x}^j))$, where $\mathbf{x}^j$ is the weight vector at the j^{th} iteration, η is the learning rate, $\nabla(C(\mathbf{x}^j))$ is the gradient of $C(\mathbf{x}^j)$.

3.3 ResPipe Platform

3.3.1 Model-Distributed Learning via Pipelining

In model-distributed learning, model $\mathbf{x}$ is partitioned and distributed across workers. Each worker stores and updates a subset of the model. Workers process data A, calculates activation vector in the forward propagation phase and error vector for back-propagation phase. These vectors are exchanged among workers, which reduces communication cost as compared to data-distributed learning. The main idea of model-distributed learning is provided next via an illustrative example.

Example 1. *Let us consider that we train a 4-layer fully connected neural network with dataset $A_{(1)}, \ldots A_{(m)}$. There are K neurons in each layer. The goal is to learn the model $\mathbf{x} = \{\mathbf{x}_l\}_{l=1}^4$, where $\mathbf{x}_l$ is a $K \times K$ matrix. There are 4 workers in this setup, and each layer is assigned to one worker, i.e., layer l is assigned to Worker l, so Worker l keeps and updates the weight matrix $\mathbf{x}_l$.*

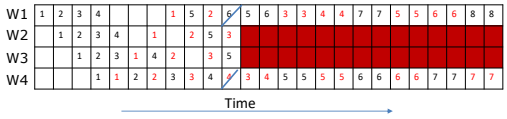
In model-distributed learning, Worker 1 receives data $A_{(j)}$ from the master device (or already has the data), calculates $\mathbf{a}_1^j = A_{(j)}$ and $\mathbf{u}_1^j = \mathbf{x}_1^j \mathbf{a}_1^j$. It transmits $\mathbf{u}_1^j$ to Worker 2. Similarly, Worker 2 and Worker 3 calculate $\mathbf{a}_n^j = g(\mathbf{u}_{n-1}^j)$ and $\mathbf{u}_n^j = \mathbf{x}_n^j \mathbf{a}_n^j$, where $n \in \{2,3\}$ and g is an activation function. Worker 2 sends $\mathbf{u}_2^j$ to Worker 3 and Worker 3 sends $\mathbf{u}_3^j$ to Worker 4, which calculates $\mathbf{a}_4^j = g(\mathbf{u}_3^j)$. This completes the forward-propagation.

In the back-propagation phase, Worker 4 calculates $\mathbf{v}_4^j = \mathbf{a}_4^j - \mathbf{y}^j$, where $\mathbf{y}_j$ is the output vector, and sends $\mathbf{v}_4^j$ to Worker 3. Workers 2 and 3 calculate $\mathbf{v}_n^j = (\mathbf{x}_n^j)^T \mathbf{v}_{n+1}^j . * g'(\mathbf{u}_{n-1}^j)$, where $n \in \{2,3\}$, $.*$ is an element-wise multiplication, and g' is the derivative of g. Worker 3 sends $\mathbf{v}_3^j$ to Worker 2, and Worker 2 sends $\mathbf{v}_2^j$ to Worker 1. Then, Workers 1, 2, and 3 calculate gradient vectors as $\Delta_n^j = \Delta_n^j + \mathbf{v}_{n+1}^j (\mathbf{a}_n^j)^T$, where $n \in \{1,2,3\}$, and $\nabla C(\mathbf{x}_l^j) = \Delta_l^j$. This completes the back-propagation phase.

Finally, each worker updates its model according to $\mathbf{x}_l^{j+1} = \mathbf{x}_l^j - \eta \nabla C(\mathbf{x}_l^j)$, $l \in \{1 \ldots 4\}$. The model is updated in a distributed manner across all workers. As seen, only vectors $\mathbf{u}_l^j$ and $\mathbf{v}_l^j$ are exchanged among workers, not the model itself. □

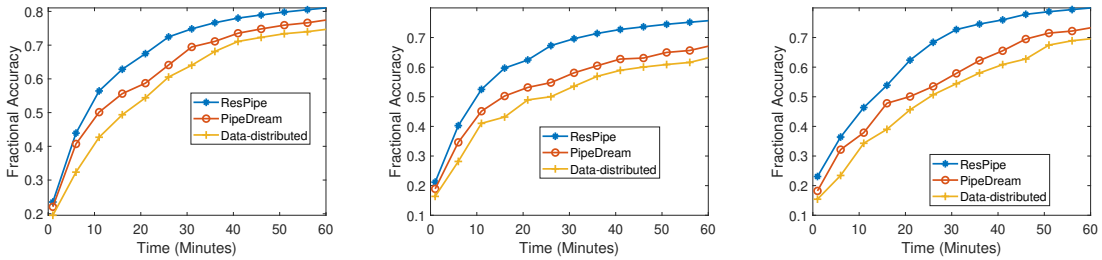
Pipelining is crucial in model-distributed learning so that workers should not be idle waiting for the updates of other workers. For example, Worker 1 in Example 1 stays idle between transmitting $\mathbf{u}_1^j$ and receiving $\mathbf{v}_2^j$, which is not efficient. Pipelining addresses this issue by keeping workers busy all the time. An example pipelining procedure based on PipeDream [26] is demonstrated in Fig. 5(a) for Example 1. We note that the gradients are computed with delayed weights due to pipelining [26]. A crucial challenge of the model-distributed DNN training is the heterogeneous and time-varying resources of edge devices including computing power, storage, battery, bandwidth, etc. Furthermore, straggling (delayed) workers or failures potentially affect the performance of model-distributed learning. For example, if Worker 2 in Example 1 is delayed (i.e., if it is a straggler), this delays all the calculations. Thus, it is imperative to develop a resilient model-distributed DNN training framework.

(a) Pipelining based on PipeDream [26]



(b) Pipelining of ResPipe

Figure 5: Pipelining for (a) PipeDream [26] and (b) ResPipe. Wi is the ith worker in the system. Numbers in the boxes indicate the data/minibatch ID. The black color represents forward propagation, while red color represents back-propagation. We assume that forward and backward works take one time unit. Red boxes in (b) indicate unresponsive workers (2nd and 3rd workers). The crossed out boxes in blue color corresponds to lost weights due to failed workers.



(a) Three workers. 5 min ON, 1.5 min OFF

(b) Three workers. 5 min ON, 5 min OFF

(c) Five workers. 5 min ON, 5 min OFF

Figure 6: Distributed training of a CNN for three and five worker scenarios. One worker fails in both scenarios.

3.3.2 ResPipe

ResPipe provides resiliency by introducing redundant weight exchanges among workers. The main idea of ResPipe is provided in the next example.

Example 2. *Let us consider the same setup in Example 1. To address the issue of failing workers, more weights are exchanged among workers. Assume that only one worker fails, but we do not know which worker. In this scenario, each worker sends its weight matrix to its neighboring worker periodically (less frequently than forward propagation updates). In particular, ith worker sends $\mathbf{x}_i$ to $i+1$th worker, $i+1 \leq 4$. In this scenario, even if one worker fails, the system keeps training the DNN model. Fig. 5(b) demonstrates how ResPipe recovers from two failed workers. As seen, 1st and 4th workers keep training the model (although it takes 2 time units to finish each forward and backward task).*

Let us assume that z workers out of N could be delayed or failed during DNN training. Each worker in ResPipe keeps track of their z immediate neighbor workers, and send their weights to these z workers at every T time units. In our Android testbed, the master device keeps track of workers, and informs each worker about their z-immediate neighbors. Peer discovery mechanisms can be applied in larger and decentralized systems.

The communication cost of ResPipe is lower than data-distributed learning. The communication cost of ResPipe depends on the number of delayed/failing workers (z) as well as the period of exchanging redundant weights (T). Noting that z and T are the parameters that determines the resiliency of the system, we can conclude that there is a tradeoff between resiliency and communication cost. Also, ResPipe introduces higher communication overhead as compared to model-distributed

learning, but it provides resiliency, which is crucial for the convergence and accuracy of the training model as demonstrated next.

3.4 ResPipe Experiments

We demonstrate the performance of ResPipe in a real testbed consisting of Android-based smartphones. We consider a topology with one master and three and five workers in two different scenarios. The master device is a Google Nexus 5, and the worker devices are all Google Nexus 6Ps. Since, for this set of experiments, the master node will not need do perform intensive computations, we have utilized the relatively weaker Google Nexus 5 as the master node. Master and worker devices are connected to each other via WiFi Direct. We demonstrate the performance of ResPipe as compared to model-distibuted (specifically PipeDream [26]) and data-distributed learning.

Fig. 6(a) shows the accuracy of classification of the trained model versus the time elapsed in minutes during training for the distributed training of a CNN over the MNIST dataset. The input is a 28×28 grayscale image. Beginning with the first layer, the CNN layers are given by the sequence C_8, M_2, C_{16}, M_2, C_{32}, M_2, F_{128}, F_{10}, where C_i represents a convolution layer with i $3 \times 3 \times k$ filters applied to all k channels of the preceding layer, M_2 is a 2×2 max-pooling layer, and F_j represent a fully connected layer with j neurons. All convolution layers as well as the fully connected layers are followed by sigmoid activations. In this setup, the second worker operates at its full performance (ON mode) for 5min and fails for 1.5 min (OFF mode). This ON/OFF mode repeats itself. Initially CNN Layers C_8 and M_2 are located at Worker 1, Layers C_{16} and M_2 are located at Worker 2, and Layers C_{32}, M_2, F_{128}, and F_{10} are located at Worker 3. The redundant weight exchange period of ResPipe is $T = 30$sec. As seen, ResPipe improves over data-distributed training thanks to reducing communication cost by distributing model itself rather than data. ResPipe also improves over PipeDream thanks to providing resiliency (even though the communication cost of ResPipe is higher). The improvement of ResPipe increases when OFF duration of the failing worker increases from 1.5 min to 5 min as seen in Fig. 6(b).

Fig. 6(c) shows accuracy versus time graph for five worker scenario. The CNN model is C_{16}, M_2, C_{16}, C_{32}, M_2, C_{64}, F_{256}, F_{100}, F_{10}. The second worker fails for 5 min at every 10min. ResPipe improves both convergence time and accuracy as compared to PipeDream and data-distributed learning thanks to providing resiliency.

References

[1] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Private and rateless adaptive coded matrix-vector multiplication," *EURASIP Journal on Wireless Communications and Networking*, vol. 2021, no. 1, pp. 1–25, 2021.

[2] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2900–2904.

[3] Y. Keshtkarjahromi, R. Bitar, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Secure coded cooperative computation at the heterogeneous edge against byzantine attacks," in *IEEE Global Communication Conference (GLOBECOM)*, 2019.

[4] R. Bitar, Y. Xing, Y. Keshtkarjahromi, V. Dasari, S. El Rouayheb, and H. Seferoglu, "Prac: private and rateless adaptive coded computation at the edge," in *Disruptive Technologies in Information Sciences II*, vol. 11013. International Society for Optics and Photonics, 2019, p. 110130T.

[5] P. Li, E. Koyuncu, and H. Seferoglu, "Respipe: Resilient model-distributed dnn training at edge networks," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 3660–3664.

[6] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, Sept 2018.

[7] R. Bitar, P. Parag, and S. El Rouayheb, "Minimizing latency for secure distributed computing," in *Information Theory (ISIT), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 2900–2904.

[8] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Globecom Workshops (GC Wkshps), 2016 IEEE*. IEEE, 2016, pp. 1–6.

[9] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," *arXiv preprint arXiv:1705.03875*, 2017.

[10] Y. Yang, P. Grover, and S. Kar, "Computing linear transformations with unreliable components," *IEEE Transactions on Information Theory*, 2017.

[11] W. Halbawi, N. Azizan-Ruhi, F. Salehi, and B. Hassibi, "Improving distributed gradient descent using reed-solomon codes," *arXiv preprint arXiv:1706.05436*, 2017.

[12] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems*, 2017, pp. 4403–4413.

[13] S. Dutta, V. Cadambe, and P. Grover, "Short-dot: Computing large linear transforms distributedly using coded short dot products," in *29th Annual Conference on Neural Information Processing Systems (NIPS)*, 2016, pp. 2092–2100.

[14] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *International Conference on Machine Learning*, 2017, pp. 3368–3376.

[15] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Fundamental tradeoff between computation and communication in distributed computing," in *IEEE International Symposium on Information Theory (ISIT)*, 2016.

[16] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[17] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, "Straggler mitigation in distributed optimization through data encoding," in *Advances in Neural Information Processing Systems*, 2017, pp. 5434–5442.

[18] M. F. Aktas, P. Peng, and E. Soljanin, "Effective straggler mitigation: Which clones should attack and when?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 2, pp. 12–14, 2017.

[19] M. Luby, "Lt codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002, pp. 271–280.

[20] A. Shokrollahi, "Raptor codes," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2551–2567, 2006.

[21] D. J. MacKay, "Fountain codes," *IEE Proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.

[22] S. Lin and D. Costello, *Error-Correcting Codes*. Prentice-Hall, Inc, 1983.

[23] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Elsevier, 1977.

[24] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 329.

[25] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[26] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," *arXiv preprint arXiv:1806.03377*, 2018.

[27] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in Neural Information Processing Systems*, 2019, pp. 103–112.

[28] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.