Timed Loops for Distributed Storage in Wireless Networks

Anandarup Mukherjee, *Graduate Student Member, IEEE*, Pallav Kumar Deb, *Graduate Student Member, IEEE*, and Sudip Misra, *Senior Member, IEEE*

***Abstract*—IoT deployments that have limited memories lack sustained computation power and have limited connectivity to the Internet due to intermittent last-mile connectivity, particularly in rural and remote locations. For maintaining congestion-free operations, most of the collected data from these networks are discarded, instead of being transmitted remotely for further processing. In this paper, we propose the paradigm *Timed Loop Storage* to distribute the data and use the underutilized bandwidth of local network links for sequentially queuing packets of computational data that are being operated-on in parts in one of the IoT nodes, similar to a FIFO. While the sequenced packets are executed sequentially on the target IoT device, the remaining packets, which are currently not being operated on, distribute and keep looping over the network links until they are required for processing. A time-synchronized packet deflection mechanism on each node handles data transfer and looping of individual packets. In our implementation, although we observe that the proposed approach requires data rates of 6 Mbps, it incurs only 45 Kb usage of primary storage systems even for sizeable data, ensuring scalability of the connected IoT devices' temporary storage capabilities, thereby making it useful for real-life applications.**

***Index Terms*—Wireless networks, Internet of Things, Resource Allocation, Connectivity, Distributed Storage Networks**

I. Introduction

Most low-power IoT devices are characterized by highly constrained computing power and storage capabilities, which results from an effort to have massively deployable IoT units with increased lifetime. Presently, memory-intensive computation and long-term data storage options for such systems are typically dependent on the use of Cloud and Fog technologies to overcome the inherent constraints in resources. However, such solutions have a significant dependence on the availability and quality of the network, which dictates the frequency and volume of data or computation to be offloaded by the constrained IoT devices. Solutions with such dependencies fail to make inferences and decisions from computations on the data on loss of last mile connectivity. Methods for performing computations without effecting regular operations, irrespective of such network bereaves is necessary.

IoT applications are widely adopted in domains such as agriculture [1] and environmental monitoring [2]. These IoT deployments often face challenges and loss of last-mile connectivity in IoT systems is one of the typical ones, particularly in remote locations and noisy channels. Although the local ground-level IoT networks, within the purview of a gateway, may be well connected, the last-mile connectivity from the gateway to the cellular service provider's infrastructure is often constrained [3] or absent (as shown in Fig. 1). In such cases, transmission of the entire data solely to the gateway (may be acting as a fog node) for storage and processing is challenging due to low configurations. In this manuscript, we take agriculture and environment monitoring system as an example for motivating our implementation. Interestingly, the ground-level systems developed for utilizing IoT in such domains have the following uniform characteristics:

A. Mukherjee, P. K. Deb and S. Misra are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India. e-mail: anandarupmukherjee@ieee.org, (pallv.deb, sudipm)@iitkgp.ac.in

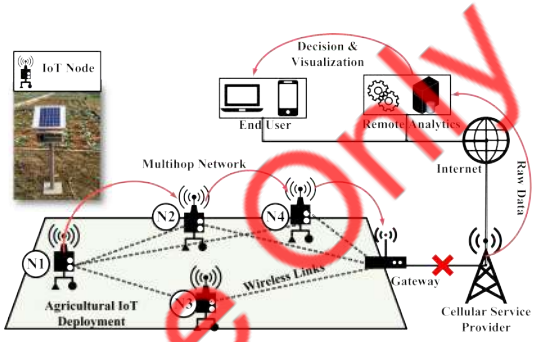


Figure 1: Motivating scenario: agricultural IoT deployment.

1) **Low-intensity data logging.** The requirement of low temporal resolution of sensed agricultural and environmental parameters keeps most of the IoT nodes non-operational for prolonged periods.
2) **Unused bandwidth.** Due to long periods of inactivity in majority of these IoT nodes, most of the local network's bandwidth remains unused [4].
3) **Temporally unused computation.** The prolonged inactivity of these IoT nodes allows for a significant availability of collaborative computation powers within the local network.
4) **Renewable energy sources.** Due to the high possibility of deployment in remote areas, with no promise of manual energy replenishment, almost all agricultural and environmental monitoring IoT systems are designed to harvest energy or rely on renewable energy.

WHAT WE PROPOSE. In this work, we introduce the proposed paradigm of "Timed Loop Storage Networks (TSN)" as a two-pronged approach: 1) It attempts to perform the computations on-site in the absence of last mile connectivity and 2) It loops the data in the network for deflecting data

among the devices in the network, where *deflection* is a commonly used term in Network-on-Chips (NoC) for routing packets within the chip [5]. We propose the use of optimized loop time for the data packets in the network until returning it back to the designated device. We attempt to study the feasibility of using the massively omnipresent local network links at the edge of IoT deployments for timed offloading of data. This paradigm aims to utilize the intra-network links between IoT devices in a subnet or a local network for locally offloading temporary computational variables from memory-constrained IoT devices in the absence of a feasible offloading path to a more computationally powerful fog or cloud infrastructure. TSN is a Time-Division Multiple Access (TDMA) based scheme – the network links facilitate the TDMA of computational data, instead of communication packets. We consider interconnected constrained IoT devices, which 1) perform mutually exclusive tasks, and 2) operates/computes on these tasks that use intermediate and independent variables. Upon satisfying these two conditions, the TSN loops the unused variables over the network links in a timed manner, such that the variables return to the source node after a predefined interval of time (or network-induced delay). The TSN-based approach frees a constrained IoT device from storing temporary variables within its already limited memory by offloading it on the network links for a pre-calculated duration, after the expiry of which, the variable returns to the source device and is used-up for computation.

First, we estimate the delays for looping the offloaded variables on the network by calculating the clock cycles required for each mathematical operation – addition, subtraction, multiplication, and division – on each device type. Before assigning delays (loop time) to the variables, the system estimates the number of mathematical operations required from the present time instant, before which the variable remains unused.

Second, we implement the proposed scheme, where the devices first synchronize their internal clocks using the network time and then offload their variables onto the network, based on the estimated delay. Once the system decides the looping delay for a variable, which we decide using Round-Trip Time (RTT), from and to the source IoT device, via deflections from multiple network-connected IoT devices, the system offloads the variable to the network. The intermediate network devices, upon receiving the data packets (which are the offloaded looping variables), check the remaining looping time and deflect them further to the next device. Looping occurs until just enough looping time remains, which allows the packet to return to the source device.

WHAT THIS WORK IS NOT. Owing to the similarity of some of its operations with various offloading schemes – whether it is data, task, or computation – our work might seem similar to offloading in Edge, Fog, or Cloud. However, the proposed solution is not an offloading scheme of either conventional nor machine learning-based computational tasks to the mentioned external platforms; rather, it is a scheme for sequencing tasks over network links with a variation of resource allocation, which usually would be done in an individual device's temporary memory. In continuation, our work dynamically chooses network routes for deflection, unlike schemes with pre-defined routes. Finally, the proposed solution is not universally applicable to all IoT domains and application areas, but is suited for those with low data generation rate.

HIGHLIGHTS. We propose the paradigm of "TSN" for utilizing unused and available network links at the local edge layer of IoT deployments to enhance the computational capabilities. Consequently, we propose TSN as a solution for agricultural and environmental monitoring systems as an example for our implementation, as shown in Fig. 1. We demonstrate and evaluate the feasibility of the proposed approach towards the use of intra-network communication links for temporarily offloading computational variables from a constrained device until the time it is required back at the offloading device. Our scheme not only enhances the in-memory data handling capacity of memory-constrained IoT devices, but also opens up a new paradigm for enhancement of the processing capabilities of edge IoT devices, in the absence of a networked path to a more powerful Fog or Cloud infrastructure. Towards realizing the proposed mechanism, we outline the major implementation highlights of our work, as follows:

- We implement and evaluate the proposed TSN protocol, both as a uni-tiered as well as a two-tiered system to evaluate its scope in multi-tier architectures. Implementations show that TSN does not require any hardware changes and is suitable for the existing networking infrastructures.
- We make use of a mix of Raspberry Pi micro-computers, regular computers, and virtual machines hosted on servers to evaluate the proposed solution.
- We evaluate TSN using mathematical operations of single-valued variables as well as 2D matrices of varying sizes (which is roughly synonymous with images and image-based operations). This evaluation to study the effect of our proposed paradigm with increasing networked devices shows that it can support simple as well as complex arithmetic operations.

LIMITATIONS. Although implemented on real-life systems and evaluated thoroughly, this work is still a proof-of-concept. Owing to this, the following are the limitations of this work:

1) **Lack of dynamicity.** The network delays and assignment of looping duration are not dynamic enough to accommodate the tradeoffs between the network and device parameters. However, this can be addressed using optimization schemes such as *Game Theory* or through Buffered Bandwidth Centric Networks (BBNets).
2) **Processing load at intermediate nodes.** Although, originally aimed for the packets to be restricted within the bottom three layers of the OSI stack (up to the Network layer), our present implementation uses all seven layers of the OSI stack, which uses additional resources.
3) **Delays at intermediate nodes.** Presently, the movement of deflected packets is up to the Application layer of the

intermediate nodes. This incurs additional delays, which can be further optimized by limiting the packets up to the Network layer.

4) **WiFi-based network.** For the sake of evaluating a diverse range of network parameters on resource-constrained IoT devices, the approach is tested using WiFi. However, the approach is implementable with any wired or wireless connectivity standard such LoRa, Zigbee, and others. Note that depending on the communication technology, the performance of TSN will vary.

II. Related Work

The proposed TSN methodology is analogous to bufferless networks in Network on Chips (NoCs), particularly bufferless NoCs. The limited storage capacity on the chips is overcome by continuous deflection of data packets as it does not use buffers in its router ports. Limiting our focus on the problem at hand, we discuss some of the existing approaches and categorize them into two classes, which are functionally related to this work – 1) distributed storage and 2) bufferless networks.

Distributed Storage: He *et al.* [6] proposed a data placement method by considering the server performance, server space, and application I/O pattern. They store only the critical data on the hybrid servers and the others in HDD servers. The authors in [7] proposed a reinforcement learning-based solution to reduce data transmission delay by enabling caching in small base stations. Another popular hardware Solid-State Drives (SSDs), which are multi-tiered, opens the scope for the Dynamic Allocation Problem (DAP). Bayat *et al.* [8] proposed a method for a scalable coded chaching as the number of users increases. The authors based their work as a variant of the Maddah-Ali and Niesen (MAN) scheme. The authors in [9] proposed a distributed storage system by partitioning the metadata namespace tree. They achieved this by developing a history-based strategy and allocating the necessary data to the servers dynamically. On the other hand, Xu and Tao [10] proposed a caching method for small base stations without the prior information about user preference. They used a multi-agent multi-armed bandit method to achieve this.

Bufferless Networks: Shpiner *et al.* [11] comprehended the capacity of bufferless NoCs and proposed scheduling algorithms (*DTNS* and *TNS*) to communicate within deadlines of real-time applications. On similar lines Vishwanath *et al.* [12] contemplated a bufferless core optical network and proposed an edge-to-edge packet-level Forward Error Correction (FEC) scheme to reduce packet losses. They further analyzed the performance of FEC in TCP flows. Mavridopoulos *et al.* [13] proposed 'HopWindows' that allocates bandwidths based on the hop distances. To understand the network links in NoC, the authors in [14] examined data transmission in bufferless networks with a focus on critical metrics of network science, such as deflection times and packet loss rate. They proposed a deflection mechanism based on delivery queues to cope with packet forwarding contentions and pointed out how the network performance varies non-monotonically.

Synthesis: We observe that the current solutions either focus on 1) strategically distributing the data for storage among the participating devices in the network, 2) perform optimization techniques onboard the devices, or 3) introduce new devices in the network. Such methods add overheads on the device CPU for performing the optimization routines as well as the storage peripherals for distributed storage. Further, the control messages add load on the network, leading to packet collisions. On the other hand, hardware design-based approaches towards the creation of new processors, memory, and storage architectures are expensive and involve modifying the legacy infrastructures. The proposed TSN approach overcomes these issues by utilizing the existing network infrastructure for containing the content necessary for processing on low-power IoT edge devices. Additionally, we limit the offload within the local network and subnet only, which enhances the utility of these otherwise constrained devices and removes the congestions on a general network. In summary, the proposed work is an attempt to reduce the dependency on hardware peripherals by exploiting the features of the network links, as opposed to the solutions available in the literature.

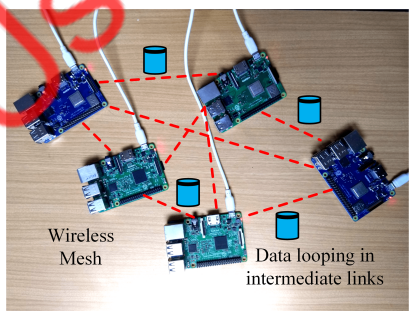


Figure 2: Implementation of the uni-tier TSN architecture.

III. Implementation Design

In this Section, we describe our problem scenario and discuss the various modalities of our work towards utilizing the local network links between IoT devices at the edge for temporarily storing/looping large data variables. These variables arise and are used at various levels of a computational task in IoT devices, a majority of which follow a sequential operation execution pattern. We use Raspberry Pi micro-controller boards for assuming the roles pertaining to the TSN protocol. Additionally, we present evaluations of TSN as both one and two-tier architectures. Fig. 2 depicts a snapshot of our implementation of TSN as a uni-tier architecture. The devices are wirelessly connected to one another over which the data loops until a calculated loop time (T_{loop}) exhausts. It may be noted that the devices are placed close to one another for better representation and are actually randomly placed far apart from one another during implementations. At this point, we define two interrelated terms for discerning the proposed work:

- ***Looping:*** It refers to the condition when the data packets are in the network, either in transmission through the links or are being deflected.

- ***Deflection:*** It refers to the phenomenon when the data packet is deviated towards another device in the network, which is the basis of looping in TSN.

PRIMARY CONSIDERATIONS. We consider a network of resource-constrained IoT devices that are tasked with periodic operations involving basic mathematical operations. These operations – addition, subtraction, multiplication, and division – consist of single-valued variables as well as 2D matrices of varying sizes. We assume that in order to accommodate operations on large entities (matrices, images, and others) locally, the participating devices have a higher tolerance for delays, and have networked neighbors, upon whose links it can offload its data temporarily. Additionally, we represent W_{dev} as the time required by a device to perform arithmetic operations (either single-variable or matrices). In case of single-valued variables, W_{dev} is considered equal to the execution time (T_{ex}). According to the proposed protocol TSN, the variables/data to be offloaded to the network for timed looping have to consider the following – 1) the clock cycle required for each operation on the device, 2) the estimated number of clock cycles that passes before the variable is required for continuation of the operation, 3) available network bandwidth, 4) available network links at the source device, 5) Round Trip Time (RTT) for each link towards routing a packet from the source device, 6) synchronicity between the other participating networked devices, and 7) offload data size.

IV. TSN: The Timed Loop Storage for Wireless Networks

The implementation of single-valued variable-based operations in TSN are quite straight forward. However, it gets complicated in case of large matrices as it is not possible to send a sizeable matrix in a single TCP packet over the network links (as per our implementation). To mitigate this problem, we split the matrix column-wise before injecting them onto the network links. Upon completion of looping, the source device receives these individual columns and performs operations on the matrix stored locally.

To determine the duration for which a packet needs to rotate/loop in the network, we base our formulations on standard Queueing Theory models. For any variable or matrix, the IoT device injects packets into the network based on its availability – the device sends only a specific number of packets over a duration of time. Since a specific number of events occurs over a given time, we consider a *Poisson Distribution* to represent the arrival rate on our devices. Along similar lines, we consider that the service rates also follow the same distribution. We further assume that the devices can simultaneously perform computation as well as networking services (which we also refer to as *deflection*). As the IoT devices on TSN are resource-constrained and dedicated for computing tasks, we consider the presence of a single processor core. Accordingly, we model our work as a $M/M/1$ queue.

The precise nature of the offloading, looping and return of the data to the source node mandates that the participating devices in our TSN paradigm be time-synchronized. In our implementation, we use Precision Time Protocol (PTP) [15] to achieve this. Although PTP is a centralized solution, where slave nodes reflect the master node's time, we plan to keep the provision for any device to act as a master upon the unanimous decision of the participating devices in case the previous one ceases to operate. Since the master can be anyone in case of failure, TSN eventually has the capability to synchronize the clock times in a decentralized manner. The selection of a master PTP node is beyond the scope of our present work.

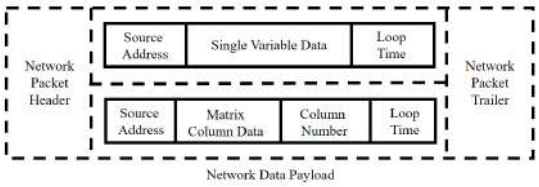


Figure 3: Packet format in TSN for both single (above) and matrix (below) variables.

We design the payload part of the packets to support the inclusion of data in our format, and the associated modalities required for making TSN work over a network (refer Fig. 3). As our implementation language is Python, we fashion the payload as a Python dictionary; however, formats such as JSON will also suffice for this purpose. Each packet for TSN contains the originating source address (A_{src}), so that the packet can be sent back upon completion of the looping duration, the loop time (T_{loop}), which dictates the duration for which the packet needs to stay in the network, and the variable data. We show the calculation of (T_{loop}) later in this Section. In case of matrices, we add an extra parameter into the packet, which represents the column number of the fragmented matrix. We plan to remove the column number in the future.

Algorithm 1 Ping Test

INPUTS:
- Devices in TSN.

OUTPUT: $\mathbb{D}_{min}^{RTT}$: Device with minimum packet transmission time.

```
while True: do
    for Devices in TSN do
        Perform ping test.
        Calculate minimum RTT.
    end for
    D_min^RTT = Device with minimum RTT.
    Wait for 5seconds.
end while
```

Based on the factors mentioned above, the devices in TSN have two modes of operation – either as a *Source* node/device, or as a *Intermediate* device. Both of these modes transfer their data to the next device by selecting the one with minimal network RTT, the details of which are outlined in Algorithm 1. As a consequence of these criteria, more data are pushed into links with higher usable bandwidth in comparison to the ones with lower bandwidths. The offloading device checks the RTT

and the available bandwidth for each of its available network links after every 5 second. This update time is selected to avoid unnecessary delays due to the checking of these network parameters. This network parameter update time is tunable as per requirement. Network links with higher usable bandwidths get injected with more data, while those with lower bandwidth transfer fewer data during the same period.

Source Node: We define a device as a source when it offloads its variables onto the network. The device sets the loop time T_{loop} for its variables in this mode. As the devices in TSN are time-synchronized, T_{loop} is the sum of the current system time of the device T_{clk}, W_{dev}, and W_{net}^{max}, where W_{net}^{max} is the maximum time needed to communicate with any of the devices in the network. W_{net}^{max} is incorporated to add tolerance to a packet's arrival delay. This is mathematically represented as,

$$T_{loop} = T_{clk} + W_{dev} + W_{net}^{max} \tag{1}$$

We inject data onto the network using Algorithm 2. In case of matrices, we use a variation of W_{dev}, (refer Section VI).

Algorithm 2 Data Looper

INPUTS:
- $\mathbb{D}_{min}^{RTT}$ from Algorithm 1.
- Single-valued variable/Matrix M for looping.

OUTPUT: Inject Payload into the network.

for Jobs in device **do**
  Payload: New dictionary.
  Payload['Address'] = A_{self}
  if Single Variable **then**
    Payload['LoopTime'] = Random(3,5).
    Payload['Data'] = Variable value.
    Send Payload to $\mathbb{D}_{min}^{RTT}$.
  else if matrix M **then**
    for Columns in matrix M **do**
      Set Payload['LoopTime'] using equation (1).
      Payload['Data'] = M[Column].
      Payload['ColumnNumber'] = Column Number.
      Send Payload to $\mathbb{D}_{min}^{RTT}$.
    end for
  else
    Do Nothing.
  end if
end for

Intermediate Node: A device participating in the TSN acts as an intermediate node/device, when it does not offload its data onto the network. In this mode, the devices receive packets over network links and compare T_{loop} and T_{clk}, which helps the device to decide on the future course of action based on predefined conditions. If $T_{loop} + W_{net}^{min} > T_{clk}$, the packet is deflected to a device with W_{net}^{min} (minimal one-way delay). If $T_{loop} + W_{net}^{min} < T_{clk}$, the packet is directed towards the originating source via the most direct path available. Finally, in case $T_{loop} = T_{clk}$, and the source matches the device itself ($A_{src} = A_{self}$), the device starts performing its allocated operation. The threshold conditions for network actions are:

$$Action = \begin{cases} Deflect, & \text{if } T_{loop} + W_{net}^{min} > T_{clk} \\ Send\ to\ source, & \text{if } T_{loop} + W_{net}^{min} \leq T_{clk} \\ Execute, & \text{if } T_{loop} \leq T_{clk} \text{ and } \\ & A_{src} = A_{self} \end{cases} \tag{2}$$

We enable the intermediate devices to deflect the packets using Algorithm 3. On exhaustion of the loop time, the packet is returned to the source device where the intended operation with the variable data is resumed (Algorithm 4).

Algorithm 3 Deflector

INPUTS:
- $\mathbb{D}_{min}^{RTT}$ from Algorithm 1.
- T_{min}^{RTT} : Time to transmit packet to $\mathbb{D}_{min}^{RTT}$
- Single-valued variables/Matrix Columns from devices.

OUTPUT: Forward packet to suitable device.

for Payload packets received **do**
  if ActualTime + $T_{min}^{RTT} \geq$ Payload['LoopTime'] **then**
    Send Payload to A_{src}
  else if ActualTime + $T_{min}^{RTT} <$ Payload['LoopTime'] **then**
    Send Payload to $\mathbb{D}_{min}^{RTT}$.
  else if ActualTime + $T_{min}^{RTT} \geq$ Payload['LoopTime'] AND Payload['Address'] = A_{self} **then**
    Execute Algorithm 4.
  else
    Do Nothing.
  end if
end for

To demonstrate the feasibility of TSN, we design the following three experiments to analyze the behavior of the devices when subjected to each. Firstly, we loop single-valued variables into the network to establish the workability and feasibility of TSN. Secondly, we loop square matrices and analyze the delays, as well as network and memory usage. Finally, we extend TSN into a two-tiered architecture to observe its scope in multi-tier architectures like Cloud/Fog computing. Sections IV-A - V outline the experimental details.

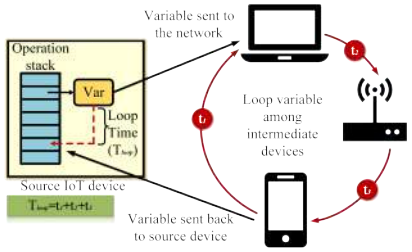


Figure 4: TSN as a uni-tier system where devices act as source as well as intermediates. The variable loops among these devices until exhaustion of the loop time.

A. Single-Valued Variables

We perform this experiment to establish the feasibility of TSN. In this scenario, we consider basic mathematical operations

such as addition, subtraction, multiplication, and division between two single-valued variables. We initiate this experiment by storing one variable within the device and looping the other. The source node sets its address and puts in a single value (floating point values) as payload in the packets. The packet then loops within the network as the devices keep deflecting it as shown in Fig. 4. The packet returns to the source device upon completion of its assigned loop time. The source device incorporates the received variable into an operation.

Algorithm 4 Operator

INPUTS:

- Payload['Data']: Single-valued variables/Matrix Columns from devices.
- Matrix $X[dim][dim]$ of dimension dim within device.
- Column matrix $M[dim][1]$ from Algorithm 3.

OUTPUT: Result matrix $\mathcal{R}$.

```
while True do
    if Payload['Data'] is single-valued variable then
        Perform arithmetic operation.
    else if Payload['Data'] is matrix column then
        M[dim][1] = Payload['Data']
        R^col_mat = X[dim][dim] × M[dim][1].
        Update R[Payload['ColumnNumber']] with R^col_mat.
    else
        Do Nothing.
    end if
end while
Display R
```

B. Matrices as Variables

For the sake of demonstration, we perform only matrix multiplications in this experiment. Similar to the single-valued variables, we store one matrix within the device and loop the other in the network. As it is not possible to fit a sizeable matrix into a single packet, we fragment it column-wise for looping over the network. We discuss the computation of looping time for these columns in subsequent sections. Upon receiving the columns back after the expiry of the designated loop time, the device multiplies it with the matrix contained within to generate the corresponding result column. The device stores this result column to finally create the complete result and displays the resulting matrix.

V. TSN as a Two-Tier Architecture

In this experiment, we split the devices into two tiers. We dedicate devices on the first tier to only inject their packets into the second tier. The devices on the second tier are responsible for deflecting/looping the packets within the network. Fig. 5 shows the devices on the second tier returning the packets to the devices on the first tier upon exhaustion of the looping time in the packet. The principle of TSN for multiplying the matrices remains the same, as described in Section IV-B. We

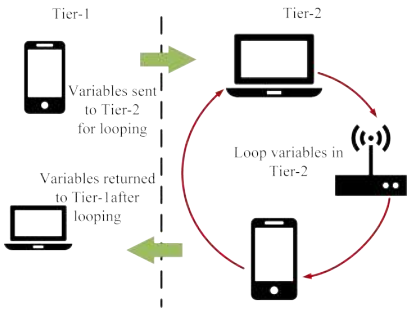


Figure 5: TSN as a two-tier system where devices on Tier-1 act as sources while devices on Tier-2 act as intermediates. The data sent from Tier-1 loops in Tier-2 for the duration of the assigned the loop time. At the end of the loop time, the intermediate node returns the data back to the source in Tier-1.

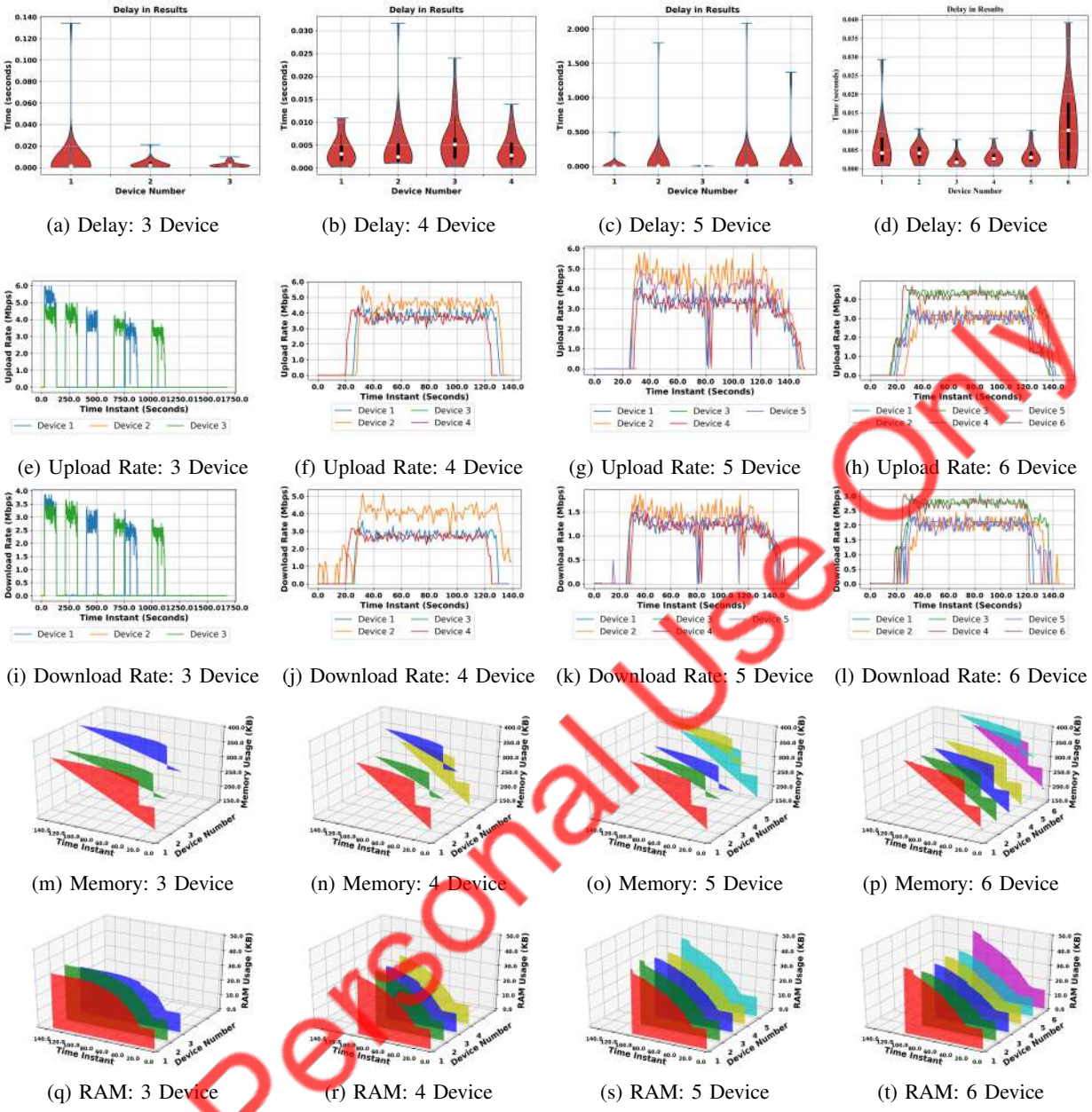
use the terms, variables and fragments, for the matrix columns interchangeably. The duration for which the variable packets need to loop over the network is dependent on two factors – 1) *device processor*, and 2) *network state*.

Device Processor: Let λ be the mean arrival rate of the variables in a device, and μ the mean service rate. We compute the probability of the device processor to be busy due to an operation as $\rho = \lambda/\mu$. The waiting time for a variable in the device is represented as $W_{dev} = W_q + \frac{1}{\mu}$, where W_q is the mean waiting time in the device's queue, and is given by $W_q = L_q/\lambda$. Here, L_q is the number of variables that have already looped and are in the device awaiting a response from the processor, which is computed as $L_q = \rho^2/(1-\rho^2)$.

Network State: Typically, the delays due to networks consist of three basic factors – *Transmission* (T_{trans}), *Propagation* (T_{prop}), and *Processing Delays* (T_{pros}). Consequently, we calculate the delay in the network as sum of all three, i.e., $W_{net} = T_{trans} + T_{prop} + T_{pros}$. We compute each of the parameters as – $T_{trans} = N/B_{rate}$, where (N) is the number of transmission bits and (B_{rate}) is the available rate of transmission; $T_{prop} = D/v$, where D is the distance between the source and destination and v is the speed of packet propagation in the medium; and (T_{pros}) is the time needed to unpack the packet and read the headers, which is dependent on the device's processor speed. However, in our implementation, we do not directly consider W_{net} as the looping time, as the round trip time (RTT) represents the end-to-end delay between the source and destination, and back for 56 Bytes of data in a standard *Network Ping test*. Further, the one-way network delay is given as $W_{net} = RTT/2$, and the transmission rate is calculated as $B_{rate} = 56/W_{net}$. We keep track of the indices sent out along a particular link as $I = \frac{B_{rate} \times T_{dur}}{P_{size} \times N_{chunk}}$, where T_{dur} is the duration of pushing packets into the particular link, P_{size} is the size of each packet, and N_{chunk} represents the number of chunks sent.

VI. Performance Evaluation

In our work, we primarily focus on the evaluation of the processing-intensive matrix multiplication task. As matrix



(a) Delay: 3 Device (b) Delay: 4 Device (c) Delay: 5 Device (d) Delay: 6 Device

(e) Upload Rate: 3 Device (f) Upload Rate: 4 Device (g) Upload Rate: 5 Device (h) Upload Rate: 6 Device

(i) Download Rate: 3 Device (j) Download Rate: 4 Device (k) Download Rate: 5 Device (l) Download Rate: 6 Device

(m) Memory: 3 Device (n) Memory: 4 Device (o) Memory: 5 Device (p) Memory: 6 Device

(q) RAM: 3 Device (r) RAM: 4 Device (s) RAM: 5 Device (t) RAM: 6 Device

Figure 6: Comparison of parameters observed in case of single-valued variables in TSN.

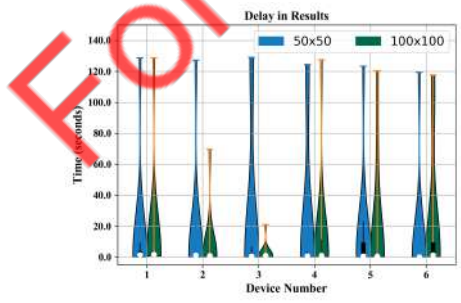
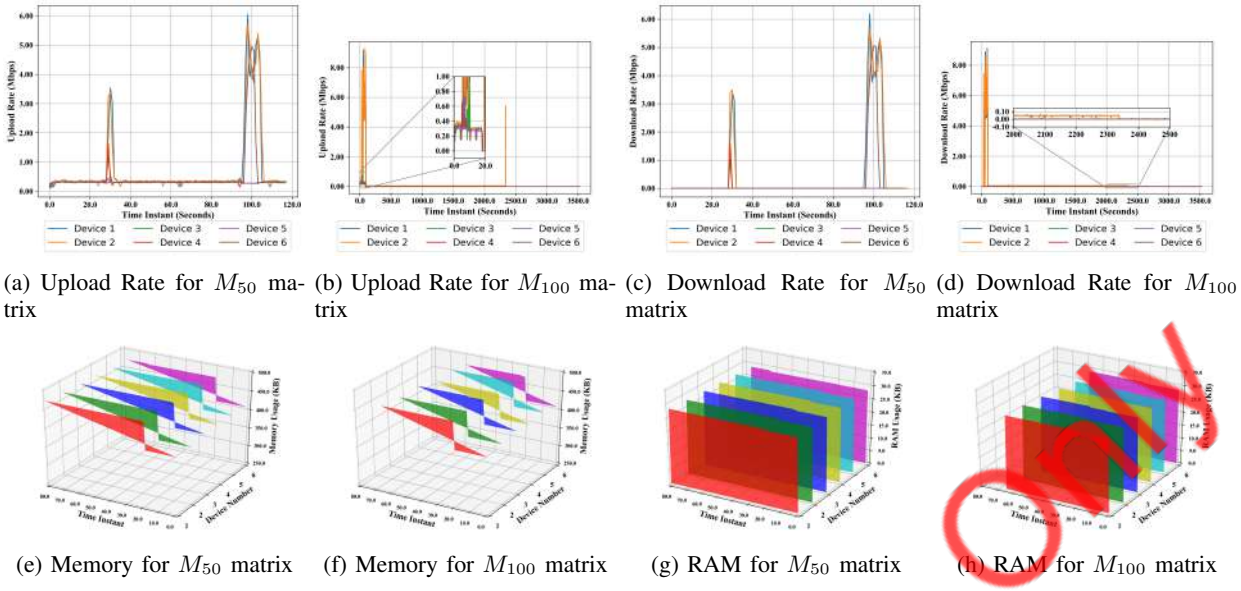


Figure 7: Delay in case of M_{50} and M_{100} matrices.

multiplication operation involves the generation of large intermediate values, which are held in the device memory till the operation finishes, we store one matrix within the device and fragment the other into columns, which loop in the network. We use matrices of three sizes, 50×50, 100×100, and 250×250, and refer to them as M_{50}, M_{100} and M_{250}, respectively. The loop time for these columns is determined according to Equation(1) with a minor modification to W_{dev}, as upon receiving the columns, the devices need to perform the multiplication of the matrix stored in it with the former and generate a result column. Thus, $W_{dev} = T_{ex} \times C_{num}$, where T_{ex} is the time needed to generate one result column and C_{num} is the column number. These result columns are then combined to form the final matrix. In our experiments, we use regular computers and laptops as deflector units and Raspberry Pi as a source unit. We show the results from the experiments mentioned in Section IV and analyze them in



(a) Upload Rate for M_{50} matrix (b) Upload Rate for M_{100} matrix (c) Download Rate for M_{50} matrix (d) Download Rate for M_{100} matrix

(e) Memory for M_{50} matrix (f) Memory for M_{100} matrix (g) RAM for M_{50} matrix (h) RAM for M_{100} matrix

Figure 8: Comparison of parameters in case of M_{50} and M_{100} matrices in TSN.

detail. In the two-tiered system, we assign 6 deflectors in Tier-2, while increasing source devices from 3 to 6 in Tier-1.

A. Delay

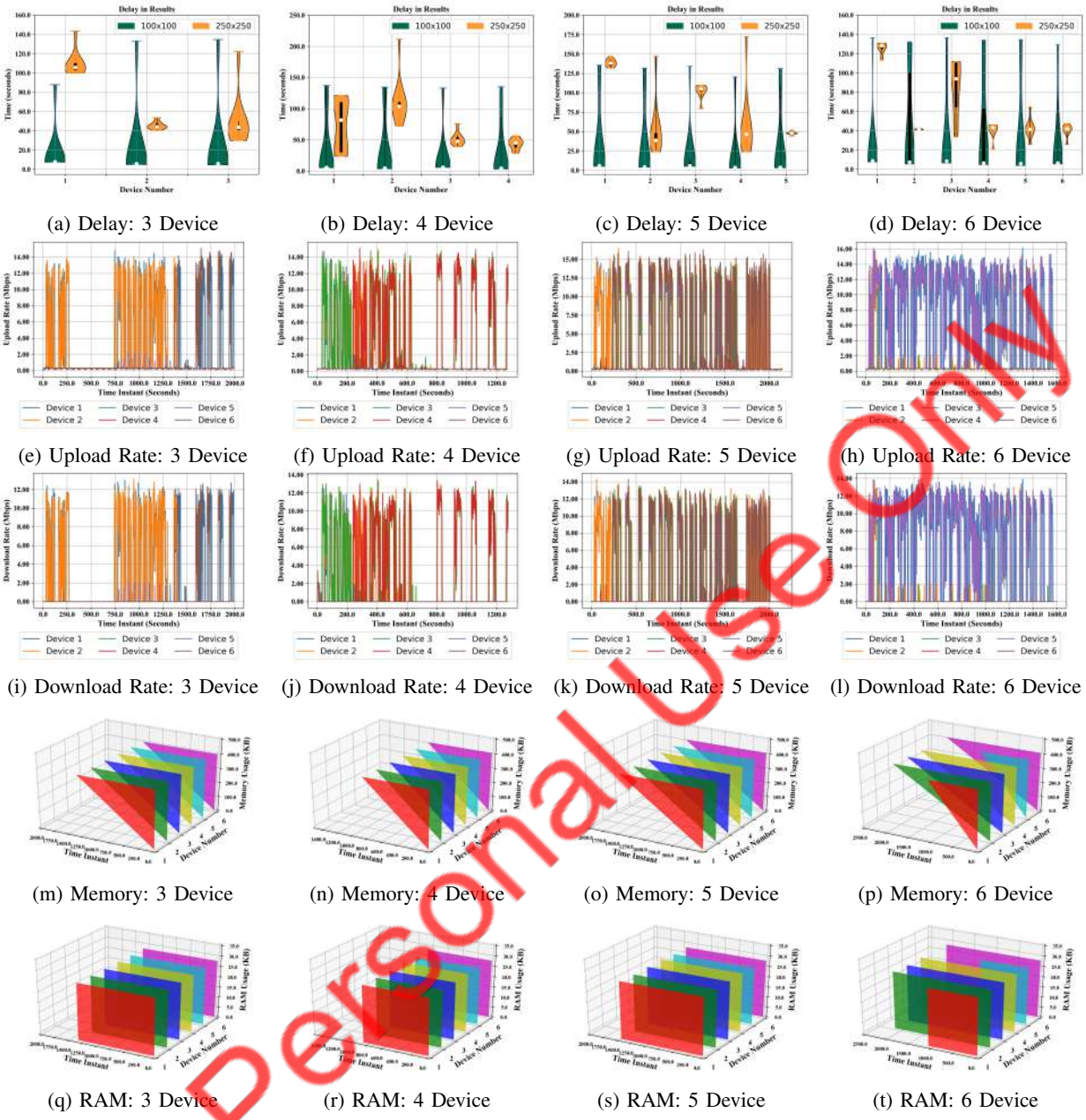
Figs. 6a – 6d show the observed delays, averaged over 30 iterations, where each device performs any one of the basic mathematical operations for single-valued variables. We increase the number of devices and observe the behavior of the delays in Figs. 6a – 6d. We observe that the delay reduces as the number of devices in TSN increases. However, in Fig. 6c, we see a significant increase in delay in Devices 2 and 4. We attribute this to the intermittent congestion in the device's buffer due to which the returning variable gets queued before the operation. Another reason for this behavior is the need for faster context switching. Since we make use of the same processor core for performing computations as well as networking, this delay is expected. However, the delay is more concentrated towards the lower-end and lies in the range of milliseconds. Similarly, Fig. 7 depicts the delays observed in case of looping the M_{50} and M_{100} matrices. We run the experiment on matrices averaged over 10 iterations and observe that TSN performs much faster in case of the larger matrix as the larger matrix has more time to loop than in case of the smaller matrix. However, we observe delays of more than 100 seconds in both cases, which is not desirable in real-time application scenarios. However, the delays are more concentrated at 10 seconds. The additional lag is due to the lack of a faster context switching method between the two modes in Section IV, thereby resulting in undesirable delays.

Figs. 9a – 9d show the delay in the case of the two-tiered architecture, for which we gradually increase the number of devices in Tier-1, whereas Tier-2 has 6 devices acting as deflectors. We observe a similar spread of delays in Fig. 7. We attribute such delays to the change in architecture and device configurations. The lag in accessing two different networks also plays an important part in increasing the delay. Adjustments to the loop time are likely to improve the results. However, as we increase the dimension of the matrix from M_{100} to M_{250}, we observe an improvement in the delay spread. As mentioned earlier, larger matrices get more time to loop in the network, which leads to better synchronization. We comment that TSN is feasible for non-real-time IoT applications and performs better as the size of the payload increases.

B. Upload Rate

Figs. 6e – 6h show the upload rate as the number of devices increases for single-valued variable operations. Since the number of data packets to be pushed to the network remains constant, the upload rate remains almost similar in all of the cases ($< 6Mbps$). We only observe an increase in network utilization, which is the objective behind proposing TSN. Extending this to our matrix scenario, to get an in-depth understanding of the behavior of the devices, we show the records of a single iteration in Figs. 8a and 8b, while looping M_{50} and M_{100} matrices, respectively. We observe that in the case of the M_{50} matrix, the device sends data in a bursty manner. Some of the columns are sent in the initial 40 seconds, and then some of them again at around 100 seconds. On the other hand, all of the columns are sent together in case of M_{100}. How the devices offload their data to the network significantly depends on the availability of the network. Due to the increase in data content, we observe an increase in the overall upload rate for M_{100}, as compared to M_{50}.

Figs. 9e – 9h show the upload rates through all of the iterations on applying TSN as a two-tier system for a M_{250} matrix. Due to the increase in the size of the matrix, the overall rate increases from $6Mbps$ to $16Mbps$. As the number of devices in Tier-1 increases from 3 to 6, the upload activity in the six Tier-2 devices increases, which signifies improved network utilization and scalability.



(a) Delay: 3 Device (b) Delay: 4 Device (c) Delay: 5 Device (d) Delay: 6 Device

(e) Upload Rate: 3 Device (f) Upload Rate: 4 Device (g) Upload Rate: 5 Device (h) Upload Rate: 6 Device

(i) Download Rate: 3 Device (j) Download Rate: 4 Device (k) Download Rate: 5 Device (l) Download Rate: 6 Device

(m) Memory: 3 Device (n) Memory: 4 Device (o) Memory: 5 Device (p) Memory: 6 Device

(q) RAM: 3 Device (r) RAM: 4 Device (s) RAM: 5 Device (t) RAM: 6 Device

Figure 9: Comparison of parameters observed in case of M_{100} and M_{250} matrices in TSN.

C. *Download Rate*

Figs. 6i – 6l show the download rate for each of the devices in a single-valued variable operation. We observe that, as the number of devices increases, the download rate does not vary much, even though the content of data pushed into the network increases. The pattern of the upload and download rates are similar as, when one device uploads, a corresponding device accepts the packets. A pattern similar to the upload rate is observed for the download rate for M_{50} and M_{100} matrices, as shown in Fig. 8c and Fig. 8d, respectively. Since the devices offloaded their data in a bursty manner (refer to Fig 8a), the download rate is also bursty (refer to Fig. 8c). On the other hand, the data rate in case of M_{100} reflects the same behavior as that in Fig. 8b. Additionally, due to the increase in data size, we observe a rise in the download rate from $6Mbps$ in case of M_{50} to $8Mbps$ in case of M_{100}.

The download rates in case of TSN as a two-tier system while looping a M_{250} are shown in Figs. 9i – 9l. As the Tier-1 network gets denser due to the increase in devices, we observe that a deflector in Tier-2 with dominating download rate has a dominating upload rate also (Figs. 9e – 9h), owing to the increased availability of usable bandwidth. TSN significantly increases network utilization, as the size of the matrix increases, which, in-turn, indicates good scalability.

D. Virtual Memory

Figs. 6m – 6p show the virtual memory/ secondary memory utilization for each of the TSN devices during a single-valued variable operation. As single-valued operations consume negligible memory, we estimate that the trend of virtual memory usage is similar throughout all the figures, and we attribute this behavior to the functional code space only. Along similar lines, for operations involving M_{50} and M_{100} matrices, we observe the same memory consumption in both Figs. 8e and 8f. However, they are larger than those in Figs. 6m – 6p by $100KB$. We do not observe additional changes in case of M_{50} and M_{100} matrices. We attribute this increase of virtual memory usage to the increase in lines of code, as well as the size of the data within the device.

In the case of TSN as a two-tiered system, we separate the two modes and place the code for looping on devices in one layer and put the other set of code (computation) on devices on the other layer. Due to this reason, we see reduced memory consumption as compared to other configurations of TSN. Also, since we run the same program throughout, we observe the same memory consumption in Figs. 9m – 9p. We infer that virtual memory consumption in TSN is mostly dependent on the executing code and optimized routines will consume much less memory, resulting in savings.

E. Primary Memory

Figs. 6q – 6s show the primary memory usage for each of the devices during a single-valued variable operation. At the inception, the primary memory usage by TSN is low. As the exchange of data proceeds, we observe a steady rise in the usage of primary memory, which eventually stabilizes after a little over 30 seconds. We again observe that, with an increasing number of devices, the consumption of primary memory remains the same throughout. The sizes of the matrices M_{50} and M_{100} are significantly larger than single-valued variables. Irrespective of the size of data to be transferred and computation size, we observe the same consumption of primary memory in Figs. 6q – 6s, 8g, and 8h.

Again, in Figs. 9q – 9t, the consumption of primary memory in the two-tier system using M_{250} matrices remains the same as that in case of M_{50} and M_{100} matrices and single-valued variables. We infer that TSN mitigates the need for excessive primary memory for performing operations, even on higher dimension matrices, and handling larger data, even on constrained devices (which is the main objective of TSN).

In summary, we observe that a gradual increase in data size, results in increased utilization of the network. However, we do not observe any increase in the secondary as well as primary memory consumption, signifying that, TSN is not only feasible, but is also scalable and conserves memory.

F. Energy Consumption

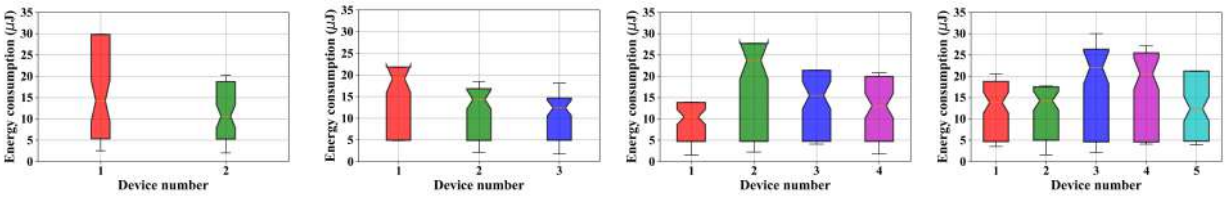
In comparison to existing works, the proposed TSN mandates looping the data in the network, leading to additional energy consumption due to repeated transmissions. We account for this condition and present the additional energy consumptions in Fig. 10, which we calculate for the M_{250} matrix by considering that each device requires 20 nJ for transferring 1 byte. We gradually increase the number of devices and observe a stable energy consumption along with a subtle decreasing trend. For instance, we observe the maximum consumption (30 μJ) in the case of 2 devices (Fig. 10a). On average, each device consumes 20 μJ, irrespective of the number of participating devices. We observe that in each experiment, one or two devices have higher consumptions than the others (device 2 in Fig. 10c and devices 3 and 4 in Fig. 10d). Intuitively, this is because the devices keep track of the free channels as a background process and typically forwards the packets towards these devices. Since all communications occur simultaneously, one or two devices in the network has to manage multiple packets, which is yet lower than the energy consumption in Fig. 10a. In summary, we infer that although all devices offload their own data for looping, the increasing number of participating devices helps in stabilizing the energy consumption on each and in reducing the average value.

Table I: Communication overheads of the TSN system.

No. of devices	CPU usage	RAM usage	Upload rate (Mbps)	Download rate (Mbps)
1	3.09%	5.41%	0.01	0.51
2	7.61%	5.34%	0.51	0.97
3	12.63%	5.39%	1.02	1.50
4	14.58%	5.41%	1.46	1.91
5	19.52%	5.37%	1.93	2.40

G. Scalability

To demonstrate scalability, we connect the devices to the general Internet through a WiFi router and use a matrix of size 3840×2160 among all the devices, and present our observations with respect to an arbitrarily chosen Raspberry pi device in Table I. Since we do not make any changes to the working methodologies of the layers in the OSI reference model, we rely on them and the TCP protocol for smooth and reliable data transfers through implicit segmentation and fragmentation routines. As the number of devices increases, the number of TSN processes running on the network also increases, which leads to an increase in the number of data packets. In the case of connecting the devices through the general Internet, we observe a maximum download rate of 2.5 Mbps, as opposed to 16 Mbps in the private network (refer Fig. 9). The upload rate also shows a similar trend. However, irrespective of the network constraints, the data transfer, and the necessary task execution occurs reliably and successfully in all cases. Interestingly, the RAM usage does not vary in any case., which is the main objective of TSN. On average, we consistently observe the usage of only 5.4%. The CPU usage on the other hand shows a rising trend. This is because of the increasing number of iterative execution of the packet reading and deflecting routines. Although optimizing hardware usage is beyond the scope of this work, some of the possible solutions for overcoming such constraints may be 1) increasing



(a) Energy: 2 Devices (b) Energy: 3 Devices (c) Energy: 4 Devices (d) Energy: 5 Devices

Figure 10: Energy consumption in the devices participating in the TSN (apart from master).

Table II: Comparison of TSN with existing solutions in terms of ability of storing in network (SN), distributed processing (DP), redundancy of packets, necessary data rate (DR), and processing delay (PD).

Category	SN	DP	Redundancy	DR	PD
SDN storage [16]	✓	✗	✓	> TSN	None
Federated learning [17]	✗	✓	✗	523 Kbps	4.32 s
TSN (proposed)	✓	✓	✗	16 Mbps	3000 s

the configuration of the devices (opposed to the proposed work) or 2) devise mechanisms for not deflecting the packets towards the devices who's hardware consumption is beyond a certain threshold. Further, on adding new devices to the network, the number of links also increases, which increases the scope for TSN. In summary, the size of the matrices does not affect performance, and Table I demonstrates the scalability in terms of RAM and network rates.

H. TSN in Comparison to Existing Solutions

We present a comparison of TSN with existing solutions in literature in Table II. Wang *et al.* [16] proposed a similar looping technique in SDN for storing in the network links. However, it requires looping of duplicate packets, resulting in redundancy, which increases the required data rate (compared to TSN). Moreover, it does not support data processing on the SDN switches. We also compared our work with a federated learning approach in [17] using MNIST dataset. While it supports distributed processing, it does not offer storage in the network. Due to the data splitting, it requires 4.32 s to train the local model on a Raspberry Pi device. It requires a data rate of 523 Kbps as it transfers only necessary meta data. On the contrary, since TSN stores data on the network by looping, it requires 16 Mbps data rate with a training time of almost 3000 s (approximately 1 hour). Such increase in delays is due to the delays in networking activities and its proceedings, which necessitates the need for optimized solutions, which we plan to address in our extended work.

VII. Discussion

For implementing TSN, we recommend that it is best suitable for scenarios that satisfy the following conditions:

- C1: Non real-time applications or those that has sufficient tolerable thresholds. We observe the same in Figs. 6a-6d, 7, and 9a-9d.
- C2: In Figs. 6, 8 and 9, we observe data rates of almost 6 Mbps, which suggests the need for reliable communication links. Failure to comply with the network condition, the devices will start queueing data in its buffer, which is in contradiction of TSN's motive.
- C3: In continuation to C1, since we consider delay tolerant applications, devices with some processing capacity (1.1 GHz) will suffice. The observations confirm the independence of TSN from memory peripherals.

In summary, TSN is suitable for applications that do not mandate real-time results. While TSN does not impose any constraint on the device configurations, the network links should be of good quality. The performance will vary proportionately with the communication technology.

VIII. Conclusion

In this work, we propose the paradigm of Timed Loop Storage Networks (TSN), and implement and analyze the feasibility of using network links as temporary storage entities. We introduce two configurations of the TSN – single tiered, and two-tiered – on which we evaluate the performance of processing-intensive tasks such as matrix multiplication using M_{50}, M_{100}, and M_{250} matrices. Our real-life implementation results show that TSN successfully executes the intended operations on the devices and the network, accommodates mathematical operations involving more extensive variables without incurring significant overheads to primary and secondary memories of the devices, and facilitates scalability of deployment. The increased delays due to network-based operations and context switching on the devices is a significant tradeoff of this approach. These delays can be further reduced by adjusting the loop times and improving the latencies, to make TSN feasible for real-time applications. Currently, the use of TSN is plausible for local networks and for environments that have low data generation rates, which is the primary objective of this work. This work does not raise any ethical issues.

In the future, we plan to address the prolonged latencies observed in Section VI by adopting methods similar to BBNets. Machine learning-based methods for detecting the quality of the network and then deflecting the data packets may reduce the latencies and optimize resource usage as random traffic effects the wireless networks [18]. With the

reduction in latencies, TSN may find applications in real-time environments. Further, we also plan to restrict our packets and its payload within the bottom three layers of the OSI stack. Such a reduction in packet processing delays will thereby reduce the overall operational delay. Summarizing, we plan to bring the proposed storage over the network to real-time environments, as a contingency for network loss in IoT.

REFERENCES

[1] M. S. Farooq, S. Riaz, A. Abid, K. Abid, and M. A. Naeem, "A survey on the role of IoT in agriculture for the implementation of smart farming," *IEEE Access*, vol. 7, pp. 156 237–156 271, Oct. 2019.

[2] X. Geng, Q. Zhang, Q. Wei, T. Zhang, Y. Cai, Y. Liang, and X. Sun, "A mobile greenhouse environment monitoring system based on the Internet of Things," *IEEE Access*, vol. 7, pp. 135 832–135 844, Sep. 2019.

[3] K. Yang, T. Jiang, Y. Shi, and Z. Ding, "Federated learning via over-the-air computation," *IEEE Transactions on Wireless Communications*, vol. 19, no. 3, pp. 2022–2035, Jan. 2020.

[4] B. Feng, C. Zhang, J. Liu, and Y. Fang, "Turning waste into wealth: Free control message transmissions in indoor WiFi networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, Jun. 2019.

[5] X. Xiang, P. Sigdel, and N.-F. Tzeng, "Bufferless network-on-chips with bridged multiple subnetworks for deflection reduction and energy savings," *IEEE Transactions on Computers*, vol. 69, no. 4, pp. 577–590, 2020.

[6] S. He, Z. Li, J. Zhou, Y. Yin, X. Xu, Y. Chen, and X. Sun, "A holistic heterogeneity-aware data placement scheme for hybrid parallel I/O systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 830–842, Apr. 2020.

[7] X. Xu, M. Tao, and C. Shen, "Collaborative multi-agent multi-armed bandit learning for small-cell caching," *IEEE Transactions on Wireless Communications*, vol. 19, no. 4, pp. 2570–2585, Apr. 2020.

[8] M. Bayat, R. K. Mungara, and G. Caire, "Achieving spatial scalability for coded caching via coded multipoint multicasting," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 227–240, Jan. 2019.

[9] Y. Gao, X. Gao, X. Yang, J. Liu, and G. Chen, "An efficient ring-based metadata management policy for large-scale distributed file systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 1962–1974, Feb. 2019.

[10] X. Xu and M. Tao, "Decentralized multi-agent multi-armed bandit learning with calibration for multi-cell caching," *IEEE Transactions on Communications*, vol. 69, no. 4, pp. 2457–2472, 2021.

[11] A. Shpiner, E. Kantor, P. Li, I. Cidon, and I. Keslassy, "On the capacity of bufferless networks-on-chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 492–506, Feb. 2015.

[12] A. Vishwanath, V. Sivaraman, M. Thottan, and C. Dovrolis, "Enabling a bufferless core optical network using edge-to-edge packet-level FEC," *IEEE Transactions on Communications*, vol. 61, no. 2, pp. 690–699, Feb. 2013.

[13] S. B. Mavridopoulos, G. Beletsioti, P. Nicopolitidis, G. I. Papadimitriou, and E. Varvarigos, "Hop distancebased bandwidth allocation technique for elastic optical networks," *International Journal of Communication Systems*, vol. 33, no. 8, p. e4360, Feb. 2020.

[14] C. Pu, W. Cui, J. Wu, and J. Yang, "Bufferless transmission in complex networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 65, no. 7, pp. 893–897, Jul. 2018.

[15] J. Kannisto, T. Vanhatupa, M. Hännikäinen, and T. D. Hämäläinen, "Precision time protocol prototype on wireless LAN," in *Proceedings of Telecommunications and Networking - ICT*, J. N. de Souza, P. Dini, and P. Lorenz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, Aug. 2004, pp. 1236–1245.

[16] M. Wang, P. Chi, J. Guo, and C. Lei, "SDN storage: A stream-based storage system over software-defined networks," in *Proceedings of IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 598–599.

[17] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S. A. Camtep, H. Kim, and S. Nepal, "End-to-end evaluation of federated learning and split learning for internet of things," in *Proceedings of International Symposium on Reliable Distributed Systems (SRDS)*, Sep. 2020, pp. 91–100.

[18] P. Panagoulias, I. Moscholios, P. Sarigiannidis, M. Piechowiak, and M. Logothetis, "Performance metrics in OFDM wireless networks supporting quasi-random traffic," *Bulletin of the Polish Academy of Sciences: Technical Sciences*, vol. 68, no. No. 2 April (Special Section on Computational Intelligence in Communications), pp. 215–223, Apr. 2020.

Anandarup Mukherjee is currently a Senior Research Fellow and Ph.D. Scholar in Engineering at the Department of Computer Science and Engineering at Indian Institute of Technology, Kharagpur. He finished his M.Tech and B.Tech from West Bengal University of Technology in the years 2012 and 2010, respectively. His research interests include, but are not limited to, networked robots, unmanned aerial vehicle swarms, Internet of Things, Industry 4.0, 6G and THz Networks, and enabling deep learning for these platforms for controls and communications. His detailed profile can be accessed at http://www.anandarup.in

Pallav Kr. Deb is a Ph.D. Research Scholar in the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India. He received his M.Tech degree in Information Technology from Tezpur University, India in 2017. Prior to that, he has completed the B. Tech degree in Computer Science from the Gauhati University, India in 2014. The current research interests of Mr. Deb include UAV swarms, THz Communications, Internet of Things, Cloud Computing, Fog Computing, and Wireless Body Area Networks. Further details are available in https://pallvdeb.github.io/

Dr. Sudip Misra (M09SM11) is a Professor with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur. He received his Ph.D. degree in Computer Science from Carleton University, in Ottawa, Canada, and the masters and bachelor's degrees, respectively, from the University of New Brunswick, Fredericton, Canada, and the Indian Institute of Technology, Kharagpur, India. Dr. Misra is the Associate Editor of the IEEE Transactions Mobile Computing and IEEE Systems Journal, IEEE Transactions on Sustainable Computing, IEEE Network, and Editor of the IEEE Transactions on Vehicular Computing. His current research interests include algorithm design for emerging communication networks and Internet of Things. Further details about him are available at http://cse.iitkgp.ac.in/ smisra/.