# FlexiCast: A Structure-Adaptive Protocol for Efficient Data-Sharing in IoT

Madhav Tummula, Manish Kausik H, Sudipta Saha
School of Electrical Sciences, Indian Institute of Technology Bhubaneswar
Email: {*tm15,mkh10,sudipta*}@iitbbs.ac.in

***Abstract*—IoT-technology is gaining a wide popularity over a large range of applications including not only monitoring of structures but also management and control of smart-systems. An IoT-system, in general, is composed of a number of IoT-devices which form a wireless decentralized setting as they get installed over a specific area to serve a particular purpose. The structure of the underlying wireless network depends on the structure of the target where the system gets deployed and hence, widely varies based on the exact application. Such structural variations often have an impact on the performance of the underlying IoT-protocols. Unfortunately most of the network protocols do not take care of such issues explicitly. For instance, although there have been quite significant development in the data-sharing protocols, especially with the advent of *Synchronous-Transmission* (ST), most of them are designed without considering the variation in the structural formation of the base networks. These protocols are tested over either in small scale simulated networks or in testbed settings bearing fixed/homogeneous structures. In this work, we demonstrate that the property of self-adaptability in an IoT-system can enable it not only to run faster but also save substantial energy which is an extremely important issue in the context of low-power system, in general. In particular, we design and implement a flexible and structure-adaptive many-to-many data-sharing protocol *FlexiCast*. Through extensive experiments under emulation-settings and IoT-testbeds we demonstrate that FlexiCast performs upto 49% faster and consumes upto 53% lesser energy compared to the case when it does not adapt to the network structure.**

***Index Terms*—Many-to-Many data sharing, WSN, IoT, Self-adjusting protocol, Concurrent-Transmission, Capture effect, TDMA, Time-varying schedule.**

## I. Introduction

An IoT-system is composed of a number of independent low-power IoT-devices which form an *ad hoc* wireless network as they get installed over a specific area for serving a particular purpose, for example, volcanic earthquake prediction [1], monitoring a bridge [2] etc. The topology of the constructed wireless network is often induced by the shape of the place where the IoT-devices get installed. Figure 1 shows a few such scenarios. Network topology is an important issue from the perspective of the performance of the distributed protocols and algorithms which fundamentally drive an IoT-system. For example when the diameter of the network is quite high with respect to the communication range of the devices it would take larger time for the data originating from one end of the network to propagate to the other end. Thus, when an IoT-system needs to run protocols where participation from a significant fraction of the nodes is very essential (e.g., consensus, aggregation), as well as the nodes have limited capability in terms of energy, communication range etc., the influence of the topology of the underlying network happens to be quite non-trivial.

However, since topology is induced by structure of the target place, it is not well defined or not known a priori. Especially when the nodes are installed in an ad hoc fashion, the design and implementation of the distributed IoT-algorithms/protocols are done in a very generic way without considering the impact of such issues. In this work we emphasize that in the context of low-power IoT-systems, the design of the distributed protocols/algorithms should consider the issue of the network topology. To make it feasible we propose the concept of *structure-adaptive IoT-protocols*. Such protocols are supposed to start operating in a *normal mode* where the network topology is not considered. However, gradually, the protocol infers the topology of the network depending on the *structure* of the place where the network is deployed, and adapts itself accordingly. Sharing of data among each other is one of the fundamental tasks in any IoT-application [3]. In this work to demonstrate the concept of structure-adaptability we select the task of many-to-many data-sharing which is the most complex form of data-sharing where many nodes in the network need to actively participate and act as source of the data.

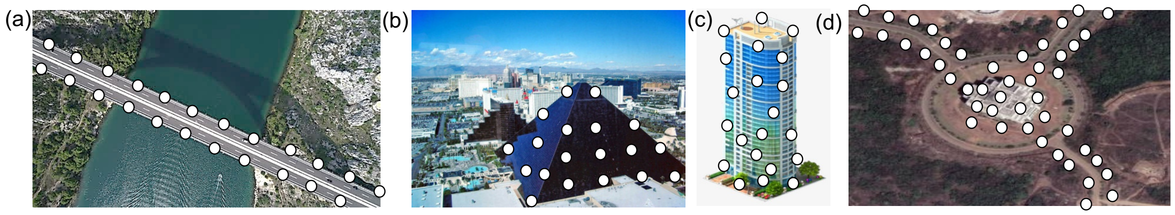


Fig. 1: Example of different types of shape/structure of the network formed in an IoT system.(a) A linear chain/string like structure of the network installed to monitor a bridge. (b) A triangular structure to monitor a pyramid like object. (c) A dense cluster to monitor a multi-storied building. (d) A chain connected with a cluster to monitor an important area and its surrounding roads.

With the advent of the *Synchronous-Transmission* (ST) there has been a huge development in the data-sharing strategies in IoT [4]. ST based protocols have shown their superiority in many recent works in comparison to the traditional *Asynchronous-Transmission* (AT) based protocols. However, unfortunately none of the ST based protocols take any step to mould their design based on the structure of the underlying network. In this work we show that structural information can be appropriately exploited to make an ST based protocol run even faster as well as conserve more energy.

There have been various approaches to achieve many-to-many data sharing using ST. Many of them make use of TDMA [5] as a base as it can be efficiently realized under ST even in a decentralized setting by exploiting the global time-synchronization. Many-to-many data-sharing protocols use TDMA to provide explicit chance to different nodes to disseminate their data in specific time-slots. For example the state-of-the-art protocols LWB [6], Chaos [7], ByteCast [8], and MiniCast [9], all make use of TDMA in some form. LWB uses TDMA in network level, i.e., each node is given independent chance to flood the whole network and therefore, its quite free from any structural influence. However, while LWB is a very stable and efficient protocol, it does not perform as good as the other protocols since it is not very optimized. In contrast, in the other protocols the contribution comes from different nodes in an interspersed manner. Specifically, the floods from the all the source nodes run almost in parallel and hence overall they perform better than LWB. However, since in all these protocols the process starts from a single node designated as the *initiator*, the structure of the network is supposed to substantially affect their performance.

There have been other approaches for many-to-many data-sharing such as Mixer [10], CodeCast [11] etc. They use sophisticated network coding to enhance the throughput in the data-sharing process. However, to demonstrate the concept of structure-adaptability in this work we design and develop a protocol *FlexiCast* where the simplest possible TDMA based protocol MiniCast is used as the base. In summary, the contributions from the work are as follows.

- We introduce the class of network protocols that can self-adjust and optimize its performance based on the specific structure of the underlying network.
- A simple way has been introduced to induce flexibility in ST based protocols through controlled run-time variation of globally set physical layer parameters, e.g., packet-size, length of packet-sequence, etc.
- We build a protocol FlexiCast to carry out many-to-many data sharing in a very time and energy efficient way through the use of time-varying TDMA schedule that are optimized based on the underlying network-structure.
- The proposed system is implemented in Contiki OS for TelosB devices and extensively compared with other ST based state-of-the-art strategies for many-to-many data sharing in both emulated and testbed platforms.

The paper is organized as follows. Section II and Section III provide a brief study of the related works and the necessary background, respectively. Section IV discusses the basic idea and design of the proposed protocol. Section V describes the performance metrics used in the work while Section VI provides an in-depth evaluation study of the proposed protocol.

## II. Related Works

**Self-adaptability:** Self-adaptability has been studied in various contexts in WSN/IoT. For example, the works [12]–[14] consider mobile networks (e.g., humans/vehicles/robots). The proposed protocols in these works constantly keep on optimizing the routing strategy and the hierarchical organization of the fog-layer with the changes in the network due to the mobility of the nodes. The work [14] does the adaptation explicitly considering the security related issues. Similarly, to maintain QoS requirements the works [15]–[17] proposes dynamic adjustments of the protocol parameters when there are changes in the strengths of the links in the network. Event based adaptation has been considered in the works [15], [18], [19]. They dynamically decide various parameters of the protocols explicitly when important events are detected at one/few nodes. The work [20] proposes an adaptive TDMA protocol for multi-robot systems where the time-slots are decided dynamically when new robots enter the system or existing robots leave. However, to the best of our knowledge self-adaptability with respect to the network structure in a pure low-power resource-constrained systems has not been considered in any of the existing works so far.

**ST-based data-sharing:** Most of the data-sharing protocols under ST, are built considering the protocol Glossy [5] as their base. Glossy achieves a very fast one-to-many data sharing and implicit time synchronisation across all the nodes in large networks. In summary, unlike traditional CSMA based strategies, Glossy avoids inter-packet collision with the help of special physical layer phenomena known as *Constructive-Interference* (CI)/*Capture Effect* (CE). BlueFlood [21], SCIF [22], RedFixHop [23], LiM [24], Splash [25], Pando [26], Ripple [27] etc., are all extensions over Glossy. Glossy has been also used for dynamically updating other MAC layer parameters on-the-fly [28]. One recent work [29] applies run-time learning and updates of the parameters of Glossy itself so that its performance can improve. However, these works do not target any network structure-specific run-time optimization.

**Many-to-many data-sharing**: Many-to-many data sharing [4] protocols serve a significant role in the smart-systems. Under ST, due to an inherent grip over time, TDMA based policy has been quite common in realizing many-to-many data-sharing. The works LWB [6], WTSP [30], Blink [31], Crystal [32] etc., apply TDMA schedule in global level for complete and independent repetition of Glossy floods. In contrast, protocols such as Chaos [7], A2 [33], Mixer [10], CodeCast [11], ByteCast [8] etc., use the schedule in a very granular way inside a packet. The work MiniCast [9], uses TDMA in a novel way for arbitrating the order of the transmissions of the packets by different nodes in system. Use of TDMA schedule in every transmission slot intrinsically allows MiniCast to run multiple units of one-to-many data-sharing in parallel. Because of its simple principle and efficient outcome, MiniCast has been used in several recent works for solving problems using IoT-edge in diverse fields, e.g., efficient charging of EVs [34], Load Management in HAN (Smart Grids) [35], [36], Multi-Party Computation for Privacy [37], Byzantine fault tolerant protocols for IoT [38], intelligent-transportation [39] etc.,

However, granular use of TDMA also makes MiniCast more sensitive to network structure which we explicitly study in this work. The concept of run-time variation of TDMA schedules for network-specific optimisation is introduced in our previous work [40]. It uses custom-made TDMA schedules and simulate the concept over standard multi-hop networks. In the current work, we develop a full system that can automatically build up the TDMA schedules based on the data collected through the execution of the protocol in normal-mode which does not consider structure-specific optimization. The proposed strategy in this work can even dynamically update those schedules as per the changes detected in the network-structure.

## III. Background

In the following, we provide a brief description of MiniCast.

**MiniCast**: MiniCast extends the protocol Glossy to carry out many-to-many data-sharing where TDMA is used to arbitrate the participation of the source nodes. The process starts with the transmission of a probe message by the initiator which triggers the transmission of the data packets from the first-hop nodes. Reception of this packets triggers the transmission from the second-hop nodes and so on until all the nodes complete a predefined number (N-TX) of transmissions. In order to provide equal and fair chances to every source node, the protocol allows all the nodes to transmit at every transmission slot. TDMA is used to define a unique position for every node in every slot. The terms *slot* and *sub-slot* in MiniCast refer to the *transmission of the complete chain of packets* and the *transmission of the packet from a specific node in a chain*, respectively. Note that MiniCast needs to exploit transmission-time SFD interrupts to keep the nodes synchronized during transmission of long chains. This is referred to as *self-synchronisation*. However, to make this possible, a node has to transmit in every sub-slot even if it has not yet acquired the data. These dummy transmissions are done with the lowest possible power so that they do not disturb other transmissions.

**ChainTx**: Transmission of a chain of packets exploiting self-synchronization is a very fundamental unit of MiniCast. We extract out this basic unit from MiniCast and call it *ChainTx*. In principle, ChainTx fundamentally extends Glossy to enable it to disseminate a number of packets together in a very compact form.

## IV. Design and implementation

The dissemination process in an ST based protocol starts from the initiator. In an usual network setting, most of the other nodes are discovered within few hops from the initiator. However, as discussed in Section I, in a cost-optimized IoT-edge deployment over a large area, the connectivity structure of the underlying network may be quite non-uniform with presence of dense components as well as long and narrow string like structures. Use of TDMA in the traditional way allows every node to contribute at every transmission-slot from the beginning itself. As a result of that a significant fraction of the transmission slots in the TDMA schedule remains empty for a considerable amount of time. To optimize the performance, instead of a globally fixed schedule what is necessary is customized TDMA schedule prepared specifically for each transmission-slot considering only those nodes that can contribute upto that slot. However, such run-time variation of schedule as per the network structure needs sufficient degree of flexibility in the underlying base protocol. Unfortunately, in order to accomplish and maintain tight time-synchronization ($\mu$S level), the ST based protocols inherently exhibit strong rigidity in important physical or data link layer parameters such as packet-length, schedule-length etc. In the following we first address this specific issue.

### A. Run-Time Variation of Schedule-Size

In order to exploit the benefit from CI/CE, the ST based protocols maintain rigidity in their design and implementation explicitly to remove any possibility that can hamper the maintenance of the tight time-synchronization. For example, multiple nodes are allowed to transmit packets together but exactly at the same time (with $\mu$S level precision) and the packets should have exactly the same size and content. The work Chaos [7], depends more on CE and hence it relaxes the restriction on having the same content in the packets, but it still follows the restriction on the packet size and timing. In order to allow run-time variation amidst such restrictions, in this work we exploit a special parameter called *Relay-Count* (aka slot-number or RC). The protocol Glossy [5] introduces RC to keep track of the number of slots passed during its execution. RC is initially set to zero in every node. Before transmission, the local value of RC is incremented and placed in the packet. In MiniCast, the same strategy is followed. The value of RC stays the same for an entire slot. Each sub-slot is identified by another parameter *packet-counter*. One very important property of RC is that at any point in time all the packets being transmitted throughout the network bear the same value of RC irrespective of the position of the transmitters. This allows RC to be used as a global reference time. We exploit this feature to realize the time-varying schedule in MiniCast and ChainTx.

Figure 2 graphically shows the feasibility of the strategy through execution of ChainTx with such variation in the chain-length in a network consisting of three layers w.r.t the initiator. We assume the nodes already have a common predefined map that provides different chain-length for every different RC value and they participates in the ChaiTx process accordingly. It can be seen from the figure that despite this variation, at different time point the length of the schedule remains constant throughout the network which ensures successful execution of ST. Following the same strategy, packet size can be also varied over sub-slots. We modify Glossy and ChainTx to

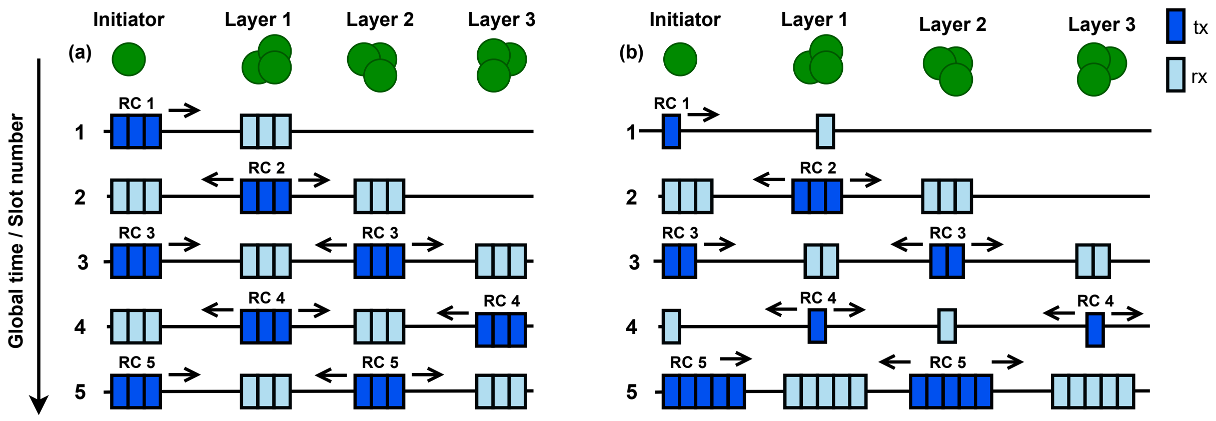


Fig. 2: Illustration of feasibility of ST despite dynamic variation in the chain-length over time (RC) through execution of ChainTx with (a) fixed chain length and (b) variable chain length.

accommodate this flexibility in the chain length and packet size. The modified versions are evaluated rigorously where no change in their reliability is observed (detailed in Section VI-A).

## B. FlexiCast

We use the RC based mechanism to implement the run-time system-wide variation of the schedules. We refer the modified version of MiniCast that uses different schedule at different RC in run-time as FlexiCast. In general, a system is supposed to start its operation with MiniCast which gradually converge to FlexiCast once the schedules to be used for each RC are sorted out. In the following we first talk about the state-diagram of FlexiCast and next describe in details how the time-varying schedules are developed through an automated process.

**State diagram**: The state diagram of FlexiCast is given in Figure 3. It bears a transmission loop and a reception loop to carry out the series of transmission and reception of packets (in a chain). Before the transmission loop, RC is incremented and the schedule for the slot is picked from a predefined map. Transmitters check the availability of the data from a local *data-store* before every packet transmission and receivers save the data after every reception in the same data-store.

## C. Formation of the Time-Varying Schedule

Since the network in an IoT-system gets formed in an ad hoc basic and there can be no prior idea about the exact structure of the target as well as positions of the nodes, the manual formation of the time-varying schedules is quite impossible. Thus, the proposed structure-adaptability is achieved through an automated run-time process that starts after the system actually gets deployed.

Since the dissemination process in MiniCast starts from the initiator, it is quite intuitive that in slot-number $i$ there is no need for the TDMA schedule to reserve the sub-slots for the nodes that are located beyond hop-$i$ or layer-$i$ w.r.t the initiator. For example, during the execution of MiniCast in the network shown in Figure 2(a), in slot-number 2, the schedule does not need to consider the nodes in layer 3 or higher since there is no way in which those nodes can contribute anything at that

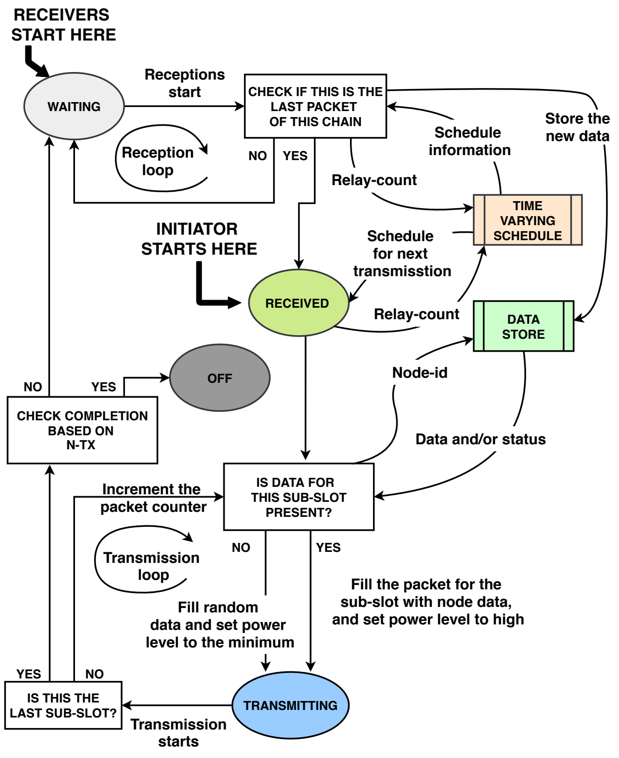
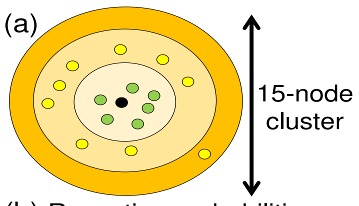

RECEIVERS START HERE
WAITING
Receptions start
CHECK IF THIS IS THE LAST PACKET OF THIS CHAIN
Reception loop
NO
YES
Schedule information
Store the new data
Relay-count
TIME VARYING SCHEDULE
INITIATOR STARTS HERE
Schedule for next transmisstion
RECEIVED
Relay-count
DATA STORE
OFF
NO
YES
Node-id
CHECK COMPLETION BASED ON N-TX
Data and/or status
Increment the packet counter
IS DATA FOR THIS SUB-SLOT PRESENT?
Transmission loop
NO
YES
Fill random data and set power level to the minimum
Fill the packet for the sub-slot with node data, and set power level to high
YES
NO
IS THIS THE LAST SUB-SLOT?
TRANSMITTING
Transmission starts

Fig. 3: State diagram of the proposed protocol FlexiCast.

point. Thus, the target should be to form the TDMA schedules precisely and carefully considering this issue.

We plan to use the initial runs of MiniCast itself to form the time-varying schedules. To have a clear understanding, we run MiniCast for 100 iterations in a small 15-node sub-network of the IoT-testbed DCube [41] as shown in Figure 4(a). Figure 4(b) shows the average cumulative probabilities of acquiring data by the initiator from each of the nodes at the first six receptions-slots in MiniCast. These are referred to as the *Reception-Probabilities* (RP). The values of the RPs show a sharp transition from low to high at a certain reception-slot in most of the nodes. The reception-slot number where this transition happens can be perceived as the slot-number from which the data from the node starts being available at the

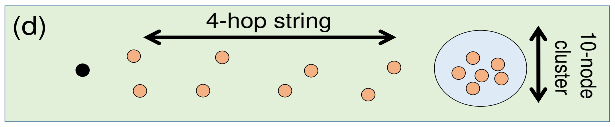


(b) Reception probabilities

| Node id | Hop | Rx-1 | Rx-2 | Rx-3 | Rx-4 | Rx-5 | Rx-6 |
|---|---|---|---|---|---|---|---|
| 224 | **0** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 203 | **1** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 209 | **1** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 217 | **1** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 218 | **1** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 223 | **1** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 216 | **1** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 215 | **1** | **0.57** | 0.63 | 0.79 | 0.86 | 0.92 | 0.96 |
| 222 | **2** | 0.02 | **0.88** | 0.90 | 0.90 | 0.90 | 0.90 |
| 201 | **2** | 0.00 | **0.47** | 0.74 | 0.82 | 0.90 | 0.94 |
| 202 | **2** | 0.00 | **0.50** | 0.74 | 0.85 | 0.92 | 0.94 |
| 204 | **2** | 0.00 | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 |
| 205 | **2** | 0.00 | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 |
| 208 | **2** | 0.00 | **0.89** | 0.91 | 0.92 | 0.94 | 0.95 |
| 206 | **3** | 0.00 | 0.00 | **0.89** | 0.90 | 0.93 | 0.95 |

(c) Time-varying schedule

| Time Step | Schedule |
|---|---|
| 1 | [224] |
| 2 | [224, 203, 209, 217, 218, 223, 216, 215] |
| 3 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208] |
| 4 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208, 206] |
| 5 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208, 206] |
| 6 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208, 206] |
| 7 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208, 206] |
| 8 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208, 206] |
| 9 | [224, 203, 209, 217, 218, 223, 216, 215, 222, 201, 202, 204, 205, 208, 206] |



(e) Reception probabilities

| Node Id | Hop | Rx-1 | Rx-2 | Rx-3 | Rx-4 | Rx-5 | Rx-6 |
|---|---|---|---|---|---|---|---|
| 1 | **0** | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 2 | **1** | **0.93** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| 3 | **1** | **0.83** | 0.98 | 1.00 | 1.00 | 1.00 | 1.00 |
| 4 | **2** | 0.00 | **0.93** | 1.00 | 1.00 | 1.00 | 1.00 |
| 5 | **2** | 0.02 | **0.98** | 1.00 | 1.00 | 1.00 | 1.00 |
| 6 | **3** | 0.00 | 0.09 | **0.92** | 1.00 | 1.00 | 1.00 |
| 7 | **3** | 0.00 | 0.11 | **0.97** | 1.00 | 1.00 | 1.00 |
| 8 | **4** | 0.00 | 0.00 | 0.12 | **0.93** | 1.00 | 1.00 |
| 9 | **4** | 0.00 | 0.00 | 0.11 | **0.96** | 1.00 | 1.00 |
| 10 | **5** | 0.00 | 0.00 | 0.00 | 0.11 | **0.98** | 1.00 |
| 11 | **5** | 0.00 | 0.00 | 0.00 | 0.02 | **0.86** | 1.00 |
| 12 | **5** | 0.00 | 0.00 | 0.00 | 0.08 | **0.92** | 1.00 |
| 13 | **5** | 0.00 | 0.00 | 0.00 | 0.08 | **0.95** | 1.00 |
| 14 | **5** | 0.00 | 0.00 | 0.00 | 0.05 | **0.98** | 1.00 |
| 15 | **5** | 0.00 | 0.00 | 0.00 | 0.12 | **0.98** | 1.00 |
| 16 | **5** | 0.00 | 0.00 | 0.00 | 0.00 | **0.92** | 1.00 |
| 17 | **5** | 0.00 | 0.00 | 0.00 | 0.06 | **0.95** | 1.00 |
| 18 | **5** | 0.00 | 0.00 | 0.00 | 0.11 | **0.98** | 1.00 |
| 19 | **5** | 0.00 | 0.00 | 0.00 | 0.00 | **0.86** | 1.00 |

(f) Time-varying schedule

| Time Step | Schedule |
|---|---|
| 1 | [1] |
| 2 | [1,2,3] |
| 3 | [1,2,3,4,5] |
| 4 | [1,2,3,4,5,6,7] |
| 5 | [1,2,3,4,5,6,7,8,9] |
| 6 | [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19] |
| 7 | [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19] |
| 8 | [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19] |
| 9 | [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19] |

Fig. 4: (a) The 15-node sub-network of testbed DCube. (b) RPs for each node at each reception-slot in the initiator for the network shown in (a). (c) Time-varying schedules composed from the RPs in (b) setting $Th_p$ = 0.4. (d) An emulated network (in Cooja) having 19 nodes. (e) RPs while emulating MiniCast in the network shown in (d). (f) Time varying schedules composed from the RPs shown in (e) setting $Th_p$ = 0.4.

initiator. In turn it also reflects the position of the node w.r.t. the initiator in the network which we use as the main resource in the formation of the time-varying schedule. In order to automatically decide an appropriate time-varying schedule, we set a global threshold (referred as $Th_p$) on the RP. A node can be considered to be ready for participating in the dissemination process on and after a slot number at which the RP goes higher than $Th_p$. Consequently, the schedule for a slot number $i$ contains only those node ids which are available by that slot number. Figure 4(c) shows the time-varying schedule for $Th_p$ = 0.4 for a general 15-node cluster from DCube [41].

Once a node is assigned with a particular hop $i$ based on a specific $Th_p$, the process will not allow the node to transmit before that slot number $i$. On the one hand, a higher $Th_p$ can produce a compact time-varying schedule. However, a too high value may delay the convergence of the process. On the other hand, if $Th_p$ is not high enough then the schedule size for each slot may remain almost the same as the one used for MiniCast resulting many empty sub-slots during execution. Hence, a balance has to be established by deciding a correct value of $Th_p$.

## D. Full System

Figure 5 provides a schematic of the flow of the execution of the full system. The process begins with the initiator executing a number of iterations of MiniCast with the full TDMA schedule. The RPs are calculated and the time-varying schedules are derived accordingly. The RPs are preserved for future references. They are referred to as *Reference Reception-Probabilities* (RRP). Subsequently, the initiator disseminates the schedule to all the nodes in the network using ChainTx. FlexiCast starts from the very next iteration after reception of the time-varying schedule at all the nodes. However, in order to track for possible changes in the network structure, FlexiCast keeps on calculating the RPs and compare with RRPs in the initiator node. A set of possible cases are listed below.

1) *A node gets shifted to a new location in the same hop*: It cannot be detected. However, it will not affect the process or the goal in any way.
2) *A node gets pushed further away from the initiator*: The initiator can detect it with a fair idea about how many hops it got shifted. Initiator recomputes the new schedule and disseminates again.
3) *A node comes closer to the initiator*: Initiator will observe increase in the RPs for this node. It would recompute the schedule and disseminate.
4) *Node failure*: This can be well detected by the initiator. This can happen if there is a certain change in the network, e.g., reorientation etc.
5) *Node addition*: Its not detected automatically, but usually in such cases the initiator is supplied with the id of the new node(s).

Under cases 2 and 4, initiator recomputes the time-varying schedule and disseminates the new schedule to the nodes. Whereas, under cases 3 and 5, the schedule needs to re-formed from scratch through explicit execution of MiniCast with full schedule for certain number of iterations. These cases are referred to in Figure 5 as *small change* (case 2 and 4) and *significant change* (case 3 and 5).

Note that the overall system is supposed to execute in a purely distributed fashion although the initiator has a special role. In fact, any node in the system can act as an initiator and if the current initiator fails there exist strategies to quickly elect a new initiator [42].

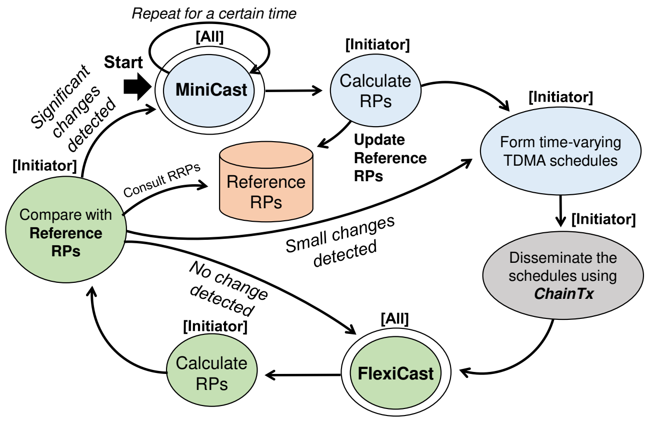


Fig. 5: The overall execution flow of the proposed system.

## V. Metrics

FlexiCast is implemented in Contiki OS for TelosB devices. The performance of FlexiCast is compared with state-of-the-art many-to-many data sharing protocols Chaos (in its default configuration) and LWB in addition to its base protocol MiniCast in both Cooja emulator and testbeds. The following metrics are used for comparison.

**Information Coverage:** We mostly consider all-to-all data sharing where each node has some amount of data to share with all other nodes in the network. The percentage of data collected at every node after each reception-slot is measured and is referred to as information coverage.

**Latency:** Latency is defined as the time taken for a certain percentage of information coverage in each node.

**Radio-on time**: It is the total time the radio in a node is required to remain 'ON' to complete a single iteration of the process. It represents energy consumption.

All results are presented as average over at least 1000 runs of the experiments and over all the nodes. Error bars reflect the standard-deviation.

## VI. Evaluation

In this section, we first show the effectiveness of the RC based solution for implanting flexibility in ST based protocols as described in Section IV. Next, a detailed emulation based study of FlexiCast is provided followed by an evaluation study over publicly available testbeds DCube and Indriya. In all our experiments the value of N-TX is set to 10, and $Th_p$ to 0.4 (see Section IVC). The slot-time in LWB is set to 15 ms unless explicitly specified. To make the comparisons among the protocols easy, we share 1-byte payload from each node unless specified. Increase in data size brings about more improvement in FlexiCast, which we show later in the same section. The protocol can be easily scaled to accommodate sharing of larger data by splitting data among multiple rounds of the protocol, each round taking a few milliseconds. In all our experiments we show the results for all-to-all data-sharing.

### A. Implanting Flexibility in ST

Glossy and ChainTx originally use fixed packet size and fixed chain length. In our experiments, we refer to them as *Glossy with Fixed Packet Size* (**GFPS**) and *ChainTx with Fixed Chain Length* (**CFCL**). To evaluate the proposed RC based run-time variation we modify the base protocols and introduce the following two versions: *Glossy with Varying Packet Size* (**GVPS**), and *ChainTx with Varying Chain Length* (**CVCL**). The metric *Reliability-rx*, described below, is used as the primary metric for comparison here.

**Reliability-rx**: In Glossy [5], a node gets N-TX number of chances to receive a packet throughout the execution of a single round of dissemination process. However, an iteration is considered to be successful (i.e., reliability 100%) if a node receives at least once out of these N-TX chances. For the current set of experiments, we consider each reception to be important and observe how controlled variation in physical layer parameters affects the same. *Reliability-rx* is calculated as a ratio of *the number of the packets a node successfully receives* to *the total number of chances it gets to receive those packets*.

All the four strategies are tested in the testbeds Indriya [43] and DCube [41]. Figure 6 presents the average reliability-rx in DCube. The line graph in 6(a) shows the reliability-rx in GFPS. It can be noticed that reliability-rx drops with the increase in the packet size. It happens due to higher accumulation of clock-drifts with higher packet size resulting in inaccurate time synchronization. However, original reliability still remains 100% regardless of the packet size. In case of GVPS, we generate a random sequence of packet sizes within a *(min,max)* range. Figure 6(a) shows the average reliability-rx in this experiment w.r.t. the mean of this random sequence as bars. Its visible from the figure that the bars match closely with the line graph representing reliability in GFPS (i.e., no considerable drop in GVPS). In CFCL, we fix the chain-length at 20 and vary the packet size. Dashed line graph of Figure 6(a) shows that the reliability-rx do not degrade much with back-to-back chain based transmissions of packets. In CVCL, we vary the chain length in each slot while all packets bear a fixed size. We generate five random sequences of chain lengths in the same way as in GVPS experiments. Reliability-rx is plotted w.r.t. the average chain size in Figure 6(b) for three different packet sizes. Comparing Figure 6(a) and Figure 6(b) it can be understood that the average reliability-rx for a certain packet size in CVCL matches almost perfectly with the same for the same packet size in CFCL. Reliability-rx in CVCL also matches with the cases of similar average packet size in GVPS and GFPS.

In a nutshell, all these results establish the fact that the proposed strategy to incorporate flexibility in ST based methods is quite feasible and effective and does not incur any degradation in reliability.

### B. Emulation Based Study of FlexiCast

For usual single-hop and multi-hop network structures, FlexiCast performs at least as good as MiniCast with some improvement in multi-hop networks depending on the distribution of nodes among hops [40]. For modelling skewed networks, we use network structures as shown in Figure 7.

**Networks composed of string and cluster, with varying string length**: We take a network having a single-hop 30-node

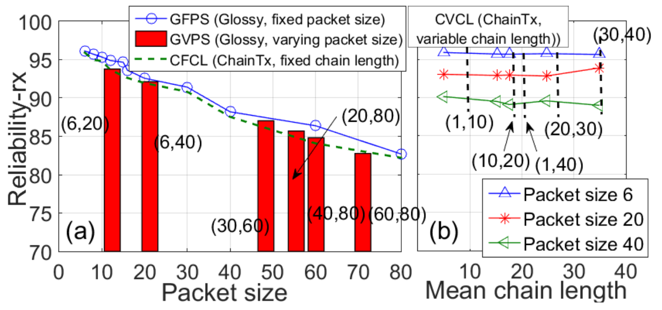


Fig. 6: (a) Reliability-rx in GFPS, GVPS and CFCL (b) Reliability-rx in CVCL as described in Section VI-A.

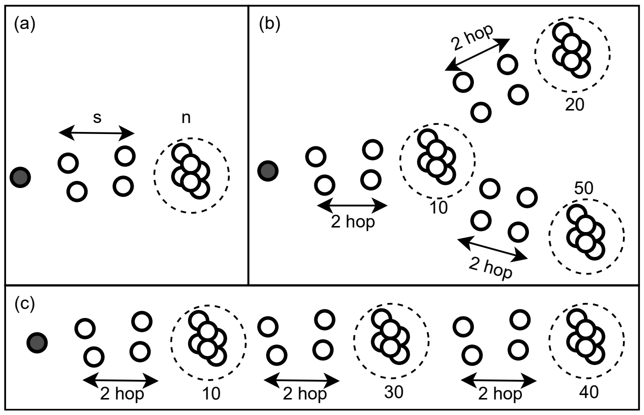


Fig. 7: Network structures modeled as three different combinations of strings and clusters used for emulation. The dark node is the initiator.

(n=30) cluster attached with a linear string of nodes (s) of size 3, 6 and 9 hops. The results are shown in Figures 8(a,b), (c,d), (e,f) for 3, 6, and 9 hop strings, respectively. For the network with 9 hops, we set LWB slot time to 20 ms to accommodate for larger diameter.

In these configurations, increase in the diameter causes the most adverse effects. LWB's performance highly depends on slot-time which in turn depends on the network diameter causing a sudden increase in latency and radio-on time for the network having a 9-hop string. The network becomes more sparse due to the increase in the string length, which benefits Chaos. But increase in the number of nodes leads to an increase in packet size also which degrades its reliability. Overall, Chaos degrades slightly. MiniCast degrades with the string length since the protocol takes more full-size chain

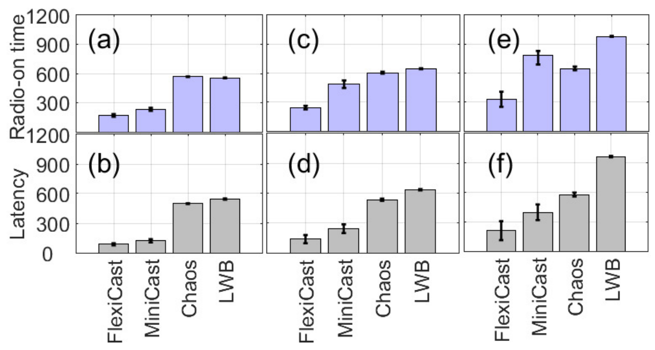


Fig. 8: Radio-on time and latency of FlexiCast, MiniCast, Chaos and LWB in networks of type 7(a) with n=30 and s=3,6,9.

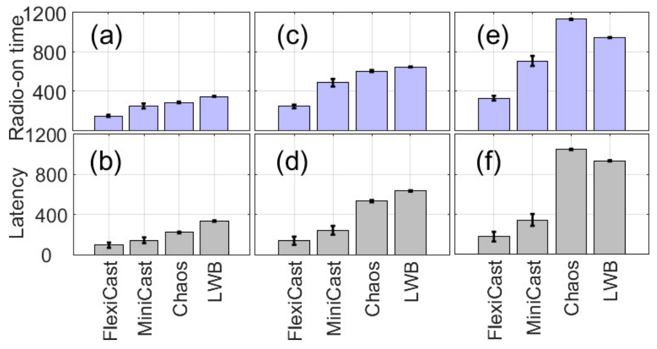


Fig. 9: Radio-on time and latency of FlexiCast, Minicast, Chaos and LWB on networks of type 7(a) with a s=6 and n=10,30,50.

receptions to complete. FlexiCast, through the use of its optimised schedules, saves a lot of time and energy. In particular, it takes about 30%, 43% and 47% lesser latency as well as 26%, 49% and 58% lesser radio-on time compared to MiniCast to achieve above 99% all-to-all dissemination with string lengths 3, 6, and 9 hops, respectively. FlexiCast also defeats Chaos by completing all-to-all dissemination 82%, 74% and 64% faster while consuming 70%, 60% and 50% lesser radio-on time, respectively.

**Networks composed of string and cluster, with varying cluster size**: Next we fix the string length at 6 hops, and vary cluster sizes to 10, 30 and 50 nodes. Results are shown in Figures 8(a,b), (c,d), and (e,f), for these three different cluster sizes, respectively.[1]

As the size of the cluster grows, the number of nodes increases while the diameter is kept constant. In LWB, slots take the same time for all cluster sizes but more slots are needed for more nodes. Hence, it increases linearly with respect to the no of nodes. In Chaos, proliferation in dense cluster weakens CE. Thus, with the growth in the cluster-size, the performance of Chaos degrades heavily. In case of MiniCast, a larger global schedule size to accommodate all nodes is used for every transmission. In FlexiCast, schedule size increases only after the cluster is discovered and the initial transmissions are done with much smaller schedules. Therefore, FlexiCast takes 34%, 43% and 49% lesser latency, and 40%, 49% and 53% lesser radio-on time, compared to MiniCast, in the configurations having cluster sizes as 10, 30, and 50, respectively.

**Varying data length**: To demonstrate the effect of increasing the length of data shared by each node, we use a similar network structure with a string length of 3 and vary cluster size between 10, 20, 40 and 80. We also vary data lengths from 1 byte to 16 bytes and calculate the Latency and Radio-on time for information coverage above 99%. The results are shown in Figure 10. It can be observed that to achieve information coverage above 99% latency in MiniCast is upto 5 times higher than FlexiCast. [2] Its already shown that higher packet-size (for higher data-length) results in lower-reliability in general (see

[1]For the sake of continuity and clarity, Figures 8(c,d) are repeated in Figures 9(c,d)

[2]Results are shown for a selected number of data-sizes. Values in the x-axis are not linear.

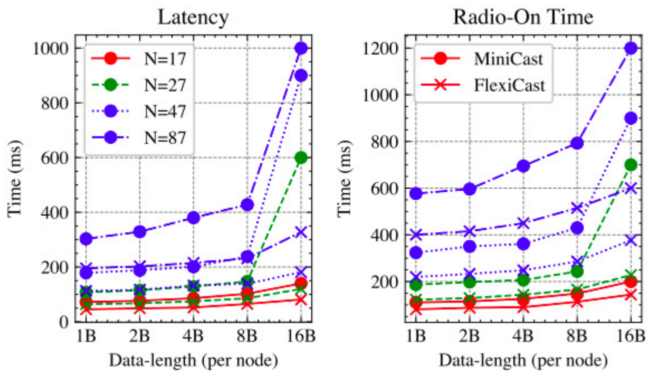


Fig. 10: Latency and radio-on time in the execution of FlexiCast and MiniCast for varying data-length (per node) in the networks of type Figure 7(a) with s=4 and n=10, 20, 40, and 80. In legend, N denotes total number of nodes (string and cluster together).

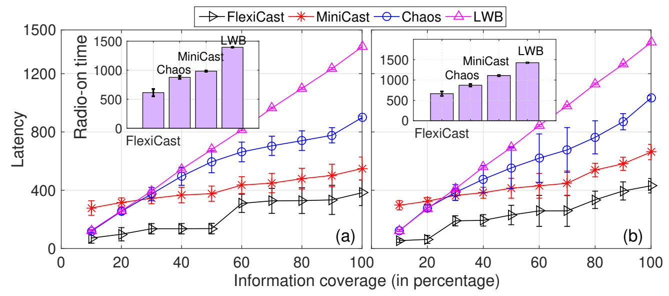


Fig. 11: Latency with respect to information coverage. Inset shows the radio-on time for achieving above 99% information coverage on emulated networks shown in Figure 7(b) and Figure 7(c).

Section VI-A and Figure 6). Therefore, to achieve coverage over 99% with larger packet size, Latency and Radio-on time go little more than linear behavior which is quite visible in the presented results. However, the said reliability issue with larger packet-size affects FlexiCast lesser than MiniCast. For instance, even in a large network of 87 nodes, FlexiCast takes only 30% of Latency and 50% of Radio-on time that MiniCast takes for many-to-many data sharing in the same setting. In addition, note that in MiniCast when a node misses a certain data item, it requires to wait for the next chance after the completion of the reception of the current chain which follows the full TDMA schedule. Whereas in FlexiCast this waiting time is minimal due to the use of optimized TDMA schedules at each time step.

**Networks composed of more than one cluster and string**: Real life IoT/WSN deployments might have multiple clusters with different sizes, multiple strings, junctions, etc (see Figure 1(d)). Due to space limitations, here we discuss only two such special structures as depicted in Figure 7(c) referred to as SP1, and Figure 7(b) referred to as SP2. Results for both are shown in Figure 11(a) and Figure 11(b), respectively. We show latency to achieve different percentages of information coverage and the radio-on time necessary for 100% information coverage (in the inset). In both SP1 and SP2, FlexiCast shows consistently better performance. Specifically, it shows a latency and radio-on time improvement of about 30% (both) in SP1 and around 34% and 23% in SP2 on average compared to MiniCast.

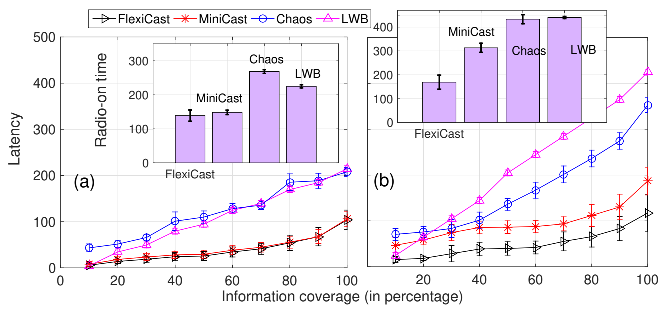


Fig. 12: Latency and radio-on time (inset) in two sub-networks of testbed DCube. Sub-network in (a) is a multi-hop cluster following a standard network model where FlexiCast performs as good as MiniCast. Sub-network in (b) is of type 7(b) where FlexiCast shows significant improvement over MiniCast and all other protocols.

## C. *Evaluation of FlexiCast in testbed*

In general testbed networks are mostly homogeneous and follows standard patterns. For appropriate evaluation we select specific nodes within the testbed networks and define sub-networks with non-uniform/skewed structure. Two such sub-networks of DCube are selected. The first sub-network is an usual multi-hop cluster where FlexiCast works similar to MiniCast. The second one is a carefully selected 22-node sub-network of the type shown in Figure 7(a) with a 6-hop string consisting of one node per-hop and a three-hop cluster of 16-nodes. Figure 12(a,b) show the latency for different percentages of information coverage and radio-on times for information coverage above 99% in FlexiCast. Huge improvements in FlexiCast can be seen in this sub-network owing to the use of the time varying schedule. Specifically, FlexiCast is found to perform 37% faster and consume 45% lesser radio-on time compared to MiniCast. Moreover, FlexiCast performs 66% and 72% faster and consumes 60% and 62% lesser radio-on time compared to Chaos and LWB, respectively.

Similarly, in the testbed Indriya, we select a 16-node sub-network, having structure similar to the one shown in Figure 7(b) without any cluster in the junction point. Even under this small setting, FlexiCast exhibits an improvements of 23.8% in latency and 45.3% in radio-on time, compared to MiniCast.

We also experiment with many other parameters such as cluster positions, strings-widths, overall topology, etc. In addition, we also evaluate the full system where MiniCast gradually converges to FlexiCast, along with adaptation of FlexiCast with possible changes in the network structure in both emulation as well as testbed. However, due to space limitation we cannot include them in this paper.

## VII. Conclusion

Network structure significantly influences the performance of network protocols. However, actual structure of a system gets defined only after it's deployment. Therefore, prior design optimization is almost impossible. To mitigate this issue in the context of low-power IoT systems, in this work we introduce the concept of structure-adaptive protocols that can adapt themselves as per the structure of the network in run-time. We design and implement a protocol FlexiCast and a supporting system to demonstrate the concept. Through extensive evaluation of FlexiCast in simulation and testbeds we show that structure-adaptability in FlexiCast makes it perform considerably better than several state-of-the-art data-sharing strategies. As a future step, we aim to further improve the performance of FlexiCast with clever application of multiple-frequencies [44] and multiple Start-of-Field-Delimiters [45], [46] for in parallel operations in different segments of the underlying structure.

## References

[1] R. Tan, G. Xing, J. Chen, W.-Z. Song, and R. Huang, "Quality-driven volcanic earthquake detection using wireless sensor networks," in *IEEE RTSS*, 2010.

[2] P. K. Patil and S. Patil, "Structural health monitoring system using wsn for bridges," in *ICICCS*, 2017.

[3] T. Qiu, N. Chen, K. Li, M. Atiquzzaman, and W. Zhao, "How can heterogeneous internet of things build our future: A survey," in *IEEE Communications Surveys & Tutorials*, 2018.

[4] M. Zimmerling, L. Mottola, and S. Santini, "Synchronous transmissions in low-power wireless: A survey of communication protocols and network services," in *arXiv preprint*, 2020.

[5] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *IPSN*, 2011.

[6] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *SenSys*, 2012.

[7] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *SenSys*, 2013.

[8] S. Saha and M. C. Chan, "Design and application of a many-to-one communication protocol," in *INFOCOM*, 2017.

[9] S. Saha, O. Landsiedel, and M. C. Chan, "Efficient many-to-many data sharing using synchronous transmission and tdma," in *IEEE DCOSS*, 2017.

[10] C. Herrmann, F. Mager, and M. Zimmerling, "Mixer: Efficient many-to-all broadcast in dynamic wireless mesh networks," in *SenSys*, 2018.

[11] M. Mohammad and M. C. Chan, "Codecast: Supporting data driven in-network processing for low-power wireless sensor networks," in *IPSN*, 2018.

[12] D. Kimovski, H. Ijaz, N. Surabh, and R. Prodan, "Adaptive nature-inspired fog architecture," in *IEEE ICFEC*, 2018.

[13] B. Lorenzo, J. García-Rois, X. Li, F. J. González-Castaño, and Y. Fang, "A robust dynamic edge network architecture for the internet of things," in *IEEE Network*, 2018.

[14] D. Montero and R. Serral-Gracià, "Offloading personal security applications to the network edge: A mobile user case scenario," in *IEEE IWCMC*, 2016.

[15] R. Young, S. Fallon, and P. Jacob, "A governance architecture for self-adaption & control in iot applications," in *IEEE CoDIT*, 2018.

[16] G. Deng and K. Wang, "An application-aware qos routing algorithm for sdn-based iot networking," in *IEEE ISCC*, 2018.

[17] X. Li, D. Li, J. Wan, C. Liu, and M. Imran, "Adaptive transmission optimization in sdn-based industrial internet of things with edge computing," in *IEEE Internet of Things Journal*, 2018.

[18] J. Wang, J. Pan, and F. Esposito, "Elastic urban video surveillance system using edge computing," in *Workshop on Smart Internet of Things*, 2017.

[19] B. Cheng, A. Papageorgiou, F. Cirillo, and E. Kovacs, "Geelytics: Geo-distributed edge analytics for large scale iot systems based on dynamic topology," in *WF-IoT*, 2015.

[20] F. Santos, L. Almeida, and L. Seabra Lopes, "Self-configuration of an adaptive tdma wireless communication protocol for teams of mobile robots," in *IEEE ETFA*, 2008.

[21] B. A. Nahas, S. Duquennoy, and O. Landsiedel, "Concurrent transmissions for multi-hop bluetooth 5," in *EWSN*, 2019.

[22] Y. Wang, Y. He, X. Mao, Y. Liu, and X. Li, "Exploiting constructive interference for scalable flooding in wireless networks," in *IEEE/ACM Transactions on Networking*, 2013.

[23] "Redfixhop: Efficient ultra-low-latency network flooding," in *SECON*, 2016.

[24] P. Zhang, A. Y. Gao, and O. Theel, "Less is more: Learning more with concurrent transmissions for energy-efficient flooding," in *MobiQuitous*, 2017.

[25] M. Doddavenkatappa, M. C. Chan, and B. Leong, "Splash: Fast data dissemination with constructive interference in wireless sensor networks," in *NSDI*, 2013.

[26] W. Du, J. Liando, and H. Zhang, "Pando: Fountain-enabled fast data dissemination with constructive interference," in *IEEE/ACM Transactions on Networking*, 2016.

[27] D. Yuan and M. Hollick, "Ripple: High-throughput, reliable and energy-efficient network flooding in wireless sensor networks," in *WoWMoM*, 2015.

[28] M. Zlmmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "ptunes: Runtime parameter adaptation for low-power mac protocols," in *IPSN*, 2012.

[29] V. Poirot and O. Landsiedel, "Poster: Learning to shine – optimizing glossy at runtime with reinforcement learning," in *EWSN*, 2019.

[30] M. Suzuki, C.-H. Liao, S. Ohara, K. Jinno, and H. Morikawa, "Wireless-transparent sensing," in *EWSN*, 2017.

[31] M. Zimmerling, L. Mottola, P. Kumar, F. Ferrari, and L. Thiele, "Adaptive real-time communication for wireless cyber-physical systems," in *ACM Trans. Cyber-Phys. Syst.*, 2017.

[32] T. Istomin, A. L. Murphy, G. P. Picco, and U. Raza, "Data prediction + synchronous transmissions = ultra-low power wireless sensor networks," in *SenSys*, 2016.

[33] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Network-wide consensus utilizing the capture effect in low-power wireless networks," in *SenSys*, 2017.

[34] J. Debadarshini and S. Saha, "Efficient coordination among electrical vehicles: An iot-assisted approach," in *IEEE INFOCOM WKSHPS*, 2022.

[35] J.Debadarshini and S. Saha, "Collaborative load management in smart home area network," in *IEEE ICDCS*, 2022 [To appear].

[36] J. Debadarshini, S. Saha, and S. Samantaray, "Decentralized load management in han: An iot-assisted approach," in *IEEE SmartGridComm*, 2022 [To appear].

[37] H. Goyal and S. Saha, "Multi-party computation in iot for privacy-preservation," in *IEEE ICDCS*, 2022 [To appear].

[38] H. Goyal, H. Manish Kausik, and S. Saha, "Reli: Real-time lightweight byzantine consensus in low-power iot-systems," in *CNSM*, 2022 [To appear].

[39] C. Shekhar and S. Saha, "Iot-assisted low-cost traffic volume measurement and control," in *COMSNETS*, 2022.

[40] M. Tummala and S. Saha, "Concurrent transmission based data sharing with run-time variation of tdma schedule," in *LCN*, 2020.

[41] M. Schuß, C. A. Boano, M. Weber, and K. Römer, "A competition to push the dependability of low-power wireless protocols to the edge," in *EWSN*, 2017.

[42] B. Al Nahas, S. Duquennoy, and O. Landsiedel, "Network bootstrapping and leader election utilizing the capture effect in low-power wireless networks," in *SenSys*, 2017.

[43] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3d wireless sensor network testbed," in *TridentCom*, 2012.

[44] J. Debadarshini, C. Shekhar, and S. Saha, "Fine-grained frequencies for simultaneous intra-group one-to-all dissemination," in *IEEE MASS*, 2020.

[45] J. Debadarshini, S. Saha, O. Landsiedel, and M. Choon Chan, "Start of frame delimiters (sfds) for simultaneous intra-group one-to-all dissemination," in *IEEE LCN*, 2020.

[46] J. Debadarshini and S. Saha, "Simultaneous intra-group communication: Understanding the problem space," in *IEEE INFOCOM*, 2022.