Efficient many-to-many data sharing using synchronous transmission and TDMA

Sudipta Saha
Indian Institute of Technology Bhubaneswar
sudipta@iitbbs.ac.in

Olaf Landsiedel
University of Chalmers
olafl@chalmers.se

Mun Choon Chan
National University of Singapore
chanmc@comp.nus.edu.sg

***Abstract*—Achieving fast and efficient many-to-many communication is one of the most complex communication problems, especially in wireless systems. A compact form of many-to-many communication in a distributed system has the potential to bring huge benefit to many distributed algorithms and protocols. Many-to-many communication can be implemented as a sequential instantiations of a network wide one-to-many communication. One limitation of such an approach is that each individual instance of a one-to-many communication has to be given enough time to propagate through the whole network before the next instance. In addition, there is large overhead in generating the schedule for the sequence of individual one-to-many communications. In this work, we show that many-to-many communication can be more efficiently implemented as many parallel many-to-one communications. In this direction, we first develop an efficient TDMA based many-to-one communication module, and then use it in many-to-many setting. Our approach achieves a minimum about 20% to 50% improvements on latency (radio-on time) over the state-of-the-art solutions in a 90-node wireless sensor network testbed.**

I. Introduction

Sharing of data and control information among the entities in a decentralized system is one of its major requirements. In many distributed computing algorithms for system wide consensus, agreement, crash recovery or detection, deadlock detection or recovery, leader election, termination detection, decentralized mutual exclusion and in many others, we find the use of many-to-many and even all-to-all sharing of data or control information. For example, even a simple crash consensus algorithm, requires several rounds of many-to-many communication to detect whether a specific node has crashed.

However, achieving a compact and fast many-to-many communication is itself a non-trivial operation. Fundamentally, an instance of many-to-many communication can be decomposed into either successive initiations of one-to-many communications by each node in the network following some TDMA schedule or as simultaneous initiations of many-to-one communication from multiple initiators and repeating the process in a cascaded fashion hop by hop. The Low Power Wireless Bus (LWB) protocol [1] utilizes the former approach. While LWB performs well, it has two major issues. First, LWB uses multiple (one-to-all) rounds of Glossy [2] based network-wide flooding. Each network-wide flooding operation has to be given enough time so that the data can reach the whole network. This flooding time depends on the size of the network. Consequently, this approach loses compactness. Second, LWB needs to be given a TDMA schedule a-priori which will be used as a sequence for instantiating multiple Glossy flooding cycles. As a result, implementing many-to-many communication using LWB incurs substantial efficiency and overhead.

In ByteCast [3], many-to-many communication is implemented as simultaneous initiations of many-to-one communication. ByteCast exploits a special feature in the hardware that allows one to change the radio transmission power on a per byte basis even during the transmission of a packet. ByteCast uses the largest possible packet size allowed and divides the packet into equal sized small segments and relies on time synchronization among the nodes to perform fine grained time division multiplexing. However, precise time synchronization is very difficult to achieve since there is no good hardware support such as an precise interrupt during the transmission of a packet. ByteCast uses a software based technique to deal with such micro-scale time synchronization. As a result, ByteCast has lower reliability and the protocol needs to perform multiple rounds of transmissions over multiple hops to achieve high reliability.

In this work, we introduce an alternative and more reliable way of achieving many-to-one and many-to-many communication. In contrast to ByteCast that combines transmissions from many senders into a single packet, our approach uses independent but small packets. In particular, different senders put their content in different small packets, instead of packet-segments as in ByteCast. These packet segments are conveyed to the receiving nodes according to a common TDMA schedules. Synchronous transmission and capture effect are used to make the communication as compact as possible. This change makes the transmissions more reliable. However, the implementation of the techniques becomes totally different. The proposed approach is a generic technique that is applicable for any wireless network. In this work, we implement and evaluate the protocol on TelosB and Contiki [4] which are the popular motes and operating system, respectively, used for wireless sensor networks.

The contributions from the work are as follows -

- Exploiting TDMA and capture effect we design a many-to-one communication protocol called PacketSync.
- PacketSync has been implemented on TelosB motes using Contiki OS. Our evaluation shows that compared to existing protocols PacketSync is about 2 to 14 times

faster than the existing asynchronous transmission based techniques.

- Based on PacketSync, we also design a protocol MiniCast for multi-hop many-to-many communication.
- We implement the protocol on TelosB motes using Contiki OS. We compare the performance of MiniCast in a wireless sensor network testbed Indriya [5] and find it to be at minimum about 20% to 50% faster than the state-of-the-art protocols.

The paper is organized as follows. In the next section we provide a brief overview of the state-of-the-art solutions for the many-to-many communications. In section III we describe the details of the design of the protocols PacketSync and MiniCast. Next, in section IV we describe certain implementation details. Finally in section V we compare our protocols with the state-of-the-art protocols.

II. Related Work

A compact and fast solution for many-to-many communication can not only improve the performance of existing distributed computations, but can also encourage the design of new ways to solve many distributed problems. However, in a shared medium such as wireless, due to its inherent broadcast nature, performing many-to-many communication using traditional CSMA based protocols is highly inefficient due to repeated collisions among the uncoordinated transmissions of packets from different nodes [6], [7], [8]. While multi-packet reception (MPR) based techniques [9], [10], [11] have been proposed, these protocols have some limitations. These protocols either need sophisticated hardware support or are too complex to implement in tiny resource limited devices which are very common in large application domain, e.g., wireless sensing and Internet-of-Things. These protocols also cannot be easily applied to multi-hop setting.

Recent works such as Chaos [12] and LWB [1] have successfully exploited synchronous transmission based techniques to achieve efficient solutions for many-to-one and many-to-many communication patterns. In particular, LWB and Chaos are based on Glossy [2] which fundamentally realizes a fast one-to-many data sharing using synchronous transmission. In LWB, Glossy-based flood is repeated by each of the nodes in a TDMA schedule to realize many-to-many communication. On the other hand in Chaos, flooding is executed with heterogeneous data from different nodes. Such flooding operation violates the requirement of constructive interference, which is a major need in Glossy. But the protocol still works due to capture effect [13]. However, although Chaos achieves the goal, the data to be flooded are not coordinated among the different nodes. In this work we exploit capture effect but the operation is performed in a more systematic way to solve the problem of achieving efficient many-to-many communication.

III. Design

We perceive many-to-many communication to be composed of instances of many-to-one communications. Hence, we first focus on improving the basic many-to-one communication protocol. In the following, we first describe the design of the many-to-one communication module PacketSync which is a single hop protocol. Next, we show the design of MiniCast as an extension of PacketSync to multi-hop settings.

A. Design of PacketSync

The basic operation of PacketSync is based on the pioneering synchronous transmission based protocol Glossy. However, Glossy is primarily meant for one-to-many communication, i.e., to share data from one node with all other nodes in the network. In order to comply with the requirements for achieving constructive interference, the transmitting nodes are required to send the same data at (almost) the same time. In order to achieve very efficient operation, the core design of Glossy is very simple. The state transitions are triggered by radio events. Every transmission, except the initial one by the initiator node, is triggered by a reception event. The nodes thus toggle between the states of transmitting a packet and then subsequent reception of a packet. Being a one-to-many communication, all the nodes are supposed to transmit and received the same data. However, in order to support many-to-one communication, where it is essential for the nodes to be able to transmit different data, we incorporated a number of changes to the control-flow of Glossy. In the following, we first describe the main protocol and then describe the state diagram to outline the new control flow.

In general, our many-to-one communication protocol PacketSync works as follows. In a single hop setting, an initiator node first broadcasts a probe message which contains scheduling information. Upon receiving that probe and the schedule, each source node who wants to convey data to the initiator sends its data packet according to the scheduling information. Finally, the initiator may send an acknowledgement based on the status of the reception of the packets from the source nodes. The main problem is to achieve a fruitful solution for the second step where all the source nodes transmits their data simultaneously to the initiator.

In the work [3] we first propose a single-hop many-to-one communication protocol SyncMerge and then use the same for a multi-hop many-to-many communication protocol ByteCast. SyncMerge accomplishes the second step of the many-to-one communication by arranging transmission and transmission power so that multiple nodes can put their content in a single packet but only in the respective segments. However, one big limitation of SyncMerge is the lack of a good technique to get a time synchronization at the level of the segments of a packet. To address this problem of achieving good synchronization to support transmissions by multiple nodes, in this work we use multiple small packets instead of a single large packet. A possible design is as follows.

Once the source nodes receive the INIT packet, they start a timer. According to the scheduling information encoded in the INIT packet, the source nodes schedule their transmissions. Finally, after reception of the final packet, the initiator can send an ACK packet to all the source nodes.

This approach works as long as the solution is confined to a single hop. However, the underlying goal is to achieve a many-to-many communication where the many-to-one communication is used as a basic building block. The design of the many-to-one communication module hence needs to support simultaneous instantiations of multiple such units at the same time from different initiators. Hence, all the simultaneous packet transmissions will have to be perfectly aligned so that at the same time slots multiple nodes can transmit the same data. Moreover, the transmissions should either result in a constructive interference or at least capture effect at the receiving nodes. This requires very tight time synchronization and perfectly aligned packet transmissions. But in order to do so, a node has to depend on its internal clock source which may vary from node to node. In our case we work with TelosB motes having MSP430 micro-controller. The internal digitally controlled RC oscillator DCO of MSP430 has several well-known issues [2] and hence it cannot be used for such accurate time synchronization in our protocol.

Another alternative solution can be when a certain source transmits a packet, the other source nodes can listen to it and use the SFD interrupts to remain synchronized with each other. However, in a general setting, the assumption that all the source nodes can hear each other may not be valid. So, although this could be a very efficient and useful strategy, we need to rule it out.

To solve the problem of tight time synchronization in this work, we used another alternative solution similar to the one mentioned above. Note that SFD interrupts are not only triggered during the receptions but also during the transmissions. In this approach we exploit the SFD interrupts available during the transmission of a packet. The source nodes, after receiving the INIT packet, start continuously transmitting the data packets one by one and thereby remain fully time synchronized during the full slot of transmissions. However, among these packets, it sends its own data only in a single packet. In this description we refer to each single packet transmission as a *sub-slot*. The rest of the transmissions are performed only for synchronization purpose. However, in order to allow other nodes to transmit data in the other sub-slots, we exploit the use of transmission using the lowest possible transmission power. In particular, a single slot begins with the reception of an INIT packet by the source nodes and the source nodes starts to transmit the data packets one-by-one in each sub-slot. Thus, a source node only transmits the data packet that contains it own data with the highest transmission power and in order to allow other transmissions, it transmits the rest of the data packets with the lowest possible power. As a result of either capture effect or constructive interference the initiator receives the data packet in each sub-slot from the respective source node. Note that the use of the lowest transmission power has been used earlier in the work [14] in a totally different context.

The order in which the nodes will transmit their data packets depends on a TDMA schedule transmitted by the INIT packet. A similar technique is used in [3], but at the level of segments of a packet. The work [3] reports that the use of zero power although is harmful for a receiver to successfully receive packets from the actual sender, but is much less severe than the issue of time synchronization which is solved in a better way by this process.

This overall scheme is depicted in figure 1. The details of the implementation of the strategy on TelosB motes are described in the next section.

B. Design of MiniCast

Once the design of PacketSync is ready we consider it to be a fundamental unit similar to a single broadcast operation. A major advantage is that similar to a single unit of broadcast operation, the boundaries of one single instance of PacketSync are fully known to all the participating nodes. To extend PacketSync to a multi-hop setting it is instantiated in a cascaded fashion in hop-by-hop. However, note that the schedules that are used will have to be globally meaningful.

The basic framework of MiniCast is similar to Glossy. The major issue is to use PacketSync as a basic unit instead of a broadcast operation. The operation of MiniCast is explained in the following in comparison to the operation of Glossy. Figure 2 illustrates the operation.

In a multi-hop setting Glossy operates as follows. Once the immediate neighboring nodes (call them layer 1 nodes) of the initiator complete the reception of the first packet sent by initiator, they start their transmission. Similarly, when the nodes in layer 1 completes their transmission, both layer 2 nodes, i.e., the set of nodes that could successfully hear from at least one node in layer 1, as well as layer 0 node, i.e., the initiator itself, start their transmissions together. This way the wave proceeds and ultimately the whole network receive the data.

Thus, the basic operation of Glossy can be understood as a layer by layer communication where all the nodes in a layer simultaneously transmit the same data together. MiniCast works in a similar way as depicted in figure 2. The basic change is that in each slot, instead of all the nodes transmitting their data at the same time, each node transmits only in their designated sub-slot. Thus, upon completion of a single slot by a certain layer say layer L, the nodes of the two adjacent layers, i.e., layer L-1 and layer L+1 get the data from all the nodes at layer L. The cascade begins exactly at the end of reception from layer L with the start of the transmission of layer L-1 and layer L+1 together.

IV. Implementation

In this section we describe the implementation details of the scheme.

Schedule: The order in which the source nodes transmit the data packets have to be pre-specified. However, it may not be possible for the initiator node to know in advance the exact identities of the source nodes. Generic or default scheduling policies have to be applied in such cases. For example, schedule number 1 may globally imply nodes 1 to 30 or upto a certain value. The first node that is supposed to start transmission under schedule 1 is node id 1 and so on.

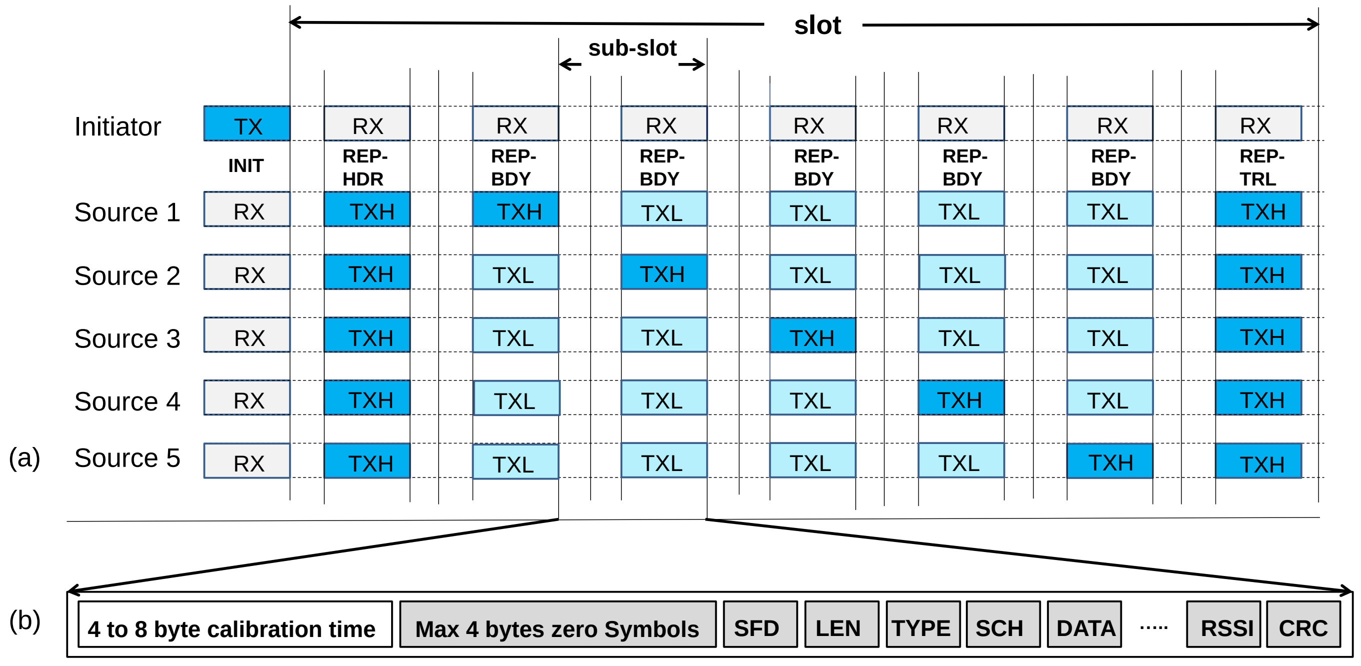

Fig. 1. Part (a) shows the details of a single unit of PacketSync (without the ACK phase). The control packets REP-HDR and REP-TRL, which are more meaningful in a multi-hop setting (such as in MiniCast) are also shown. TXH and TXL denote the transmission of a packet by a source node in a sub-slot with highest and lowest possible transmission powers, respectively. RX denote the reception of a packet in a sub-slot by the initiator. Part (b) shows different parts of a sub-slot in details.

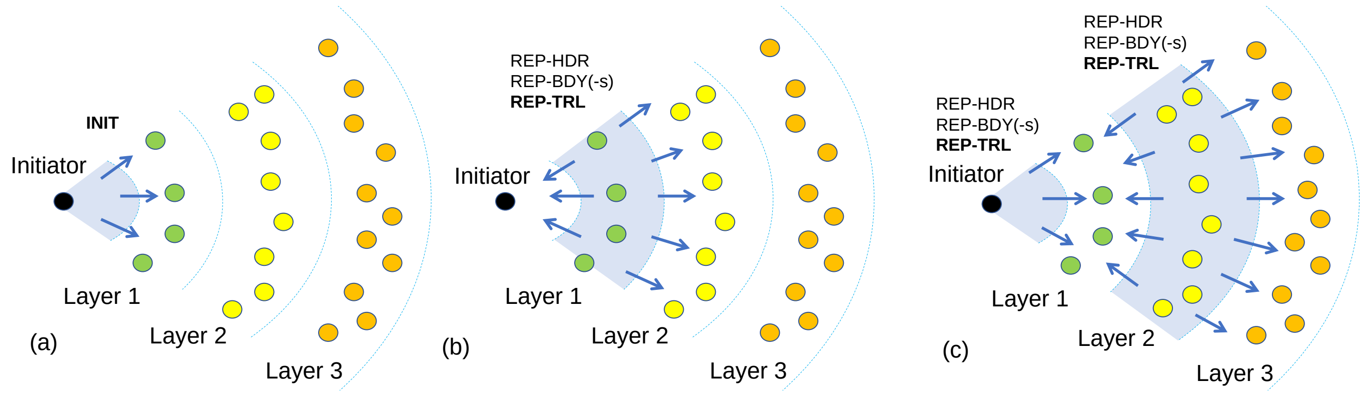

Fig. 2. Execution of MiniCast in a typical multi-hop setting. It begins with transmission of an INIT packet from the initiator. Transmissions of the nodes in a layer start upon reception of the REP-TRL packet (or an INIT packet) from the nodes in the adjacent layers (or the initiator).

Beginning and end of a slot: In a given sub-slot which node transmits its data packet with the highest power is decided by a schedule. If there is no node to transmit in a given sub-slot (i.e., no one's id matches with the one that is mentioned in the schedule), the sub-slot will remain empty. However, if there is no transmission in the last sub-slot, the initiator will not be able to detect the precise point of time when the current slot ends. As a result, when the PacketSync is extended to in a multi-hop setting (e.g., in MiniCast as shown in figure 2) the nodes in the two adjacent layers will fail to remain synchronized about when the current slot ends. Similar problems may occur when a slot begins. To solve this problem, we indicate the start and end of a slot by two special control packets - REP-HDR and REP-TRL, respectively. Once the source nodes receive the INIT packet from the initiator, they all synchronously transmit the REP-HDR with the highest power to indicate the beginning of a slot. Similarly, to indicate the end of a slot, all the nodes synchronously transmit a REP-TRL packet. In the current implementation, the size and structure of the REP-HDR and REP-TRL packets are the same as the data packet, except the packet type field indicating the particular type.

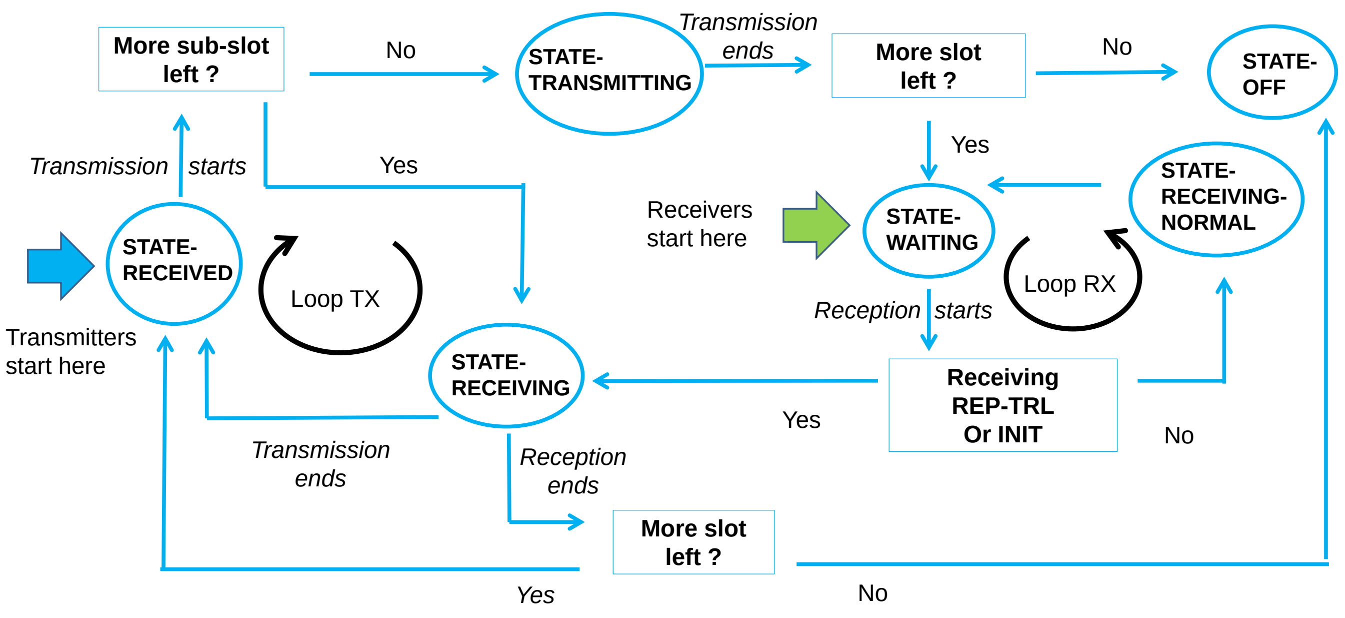

Fig. 3. State diagram of the core part of PacketSync and MiniCast. Radio events are denoted by italics. Loop TX and Loop RX denote the continuous transmissions and receptions, respectively. Once a slot is completed the transmitters and receivers exchange their roles. This process continues for a transmission of a predetermined number of slots (T). Decisions are taken at the start and the end of both transmissions and receptions of packets.

A. *State diagram*

A node may act either as a transmitter when it continuously transmits N number of packets, or as a receiver when it continuously listens for packets. This is unlike most synchronous transmission based approaches that are based on mainly back-to-back reception-transmission phases such as Glossy, Chaos, LWB and ByteCast [3]. Figure 3 illustrates a generic state transition diagram for the nodes acting as transmitter and receiver.

There are two loops in the protocol, loop RX and loop TX - where a node continuously receives and transmits a certain number of packets, respectively. The sequence of transmission ends based on a certain count. As some packets may be missing in the transmissions, the reception sequence ends based on the reception of the REP-TRL.

Note that the state diagram shown in figure 3 is the core part of both PacketSync and MiniCast. Both the protocols start with the initiator transmitting an INIT packet which is not shown in figure 3. After transmitting the INIT packet the initiator node goes to the receiving mode in the state STATE-WAITING and the source nodes start transmitting from the state STATE-RECEIVED.

In MiniCast, the number of times a node needs to repeat the transmission of slots depends on the size and shape of the network. Thus, upon completion of transmitting or receiving a slot, a node needs to decide based on how many slots it has already transmitted.

B. *Time requirement*

Here we show how the time required by MiniCast to disseminate a certain size of data from each node to all other nodes in a network can be approximated. Note that this is not possible in existing many-to-many communication protocols that does not solve the problem in a systematic way. We show a sample calculation for our implementation in TelosB with CC2420 radio.

The time required by MiniCast depends on the time a unit of PacketSync takes. The total time for an PacketSync unit, without considering the INIT or ACK phase (i.e., considering only a single slot in MiniCast) can be computed as,

$$t_{slot} = t_{REP-HDR} + N \times t_{REP-BDY} + t_{REP-TRL}, \quad (1)$$

where N is the total number of nodes supposed to participate in a slot. In order to keep it simple, we use the same packet length for REP-HDR, REP-TRL and REP-BDY. Let the time taken by one sub-slot (which is the total time taken to transmit a full physical layer packet with a certain number of data bytes) be $t_{sub-slot}$.

Thus, t_{slot} can be calculated as,

$$t_{slot} = (N + 2) \times t_{sub-slot}. \quad (2)$$

As depicted in 1(b) a sub-slot comprises of the following parts -

- Calibration time which is equivalent to the time required for transmitting 4 or 8 bytes (t_{cal}, programmable).
- Transmission time for 0 to 4 bytes (programmable) of zero symbols for the preamble (t_{zero}).
- Transmission time of 1 byte SFD, 1 byte length field, 1 byte packet type field, 1 byte schedule information, data bytes (t_{data}) and 2 byte trailer.

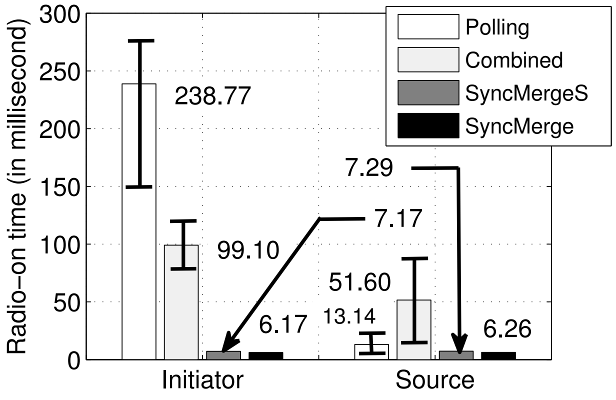

Fig. 4. Comparison of performance of PacketSync with SyncMerge and two other asynchronous transmission based techniques.

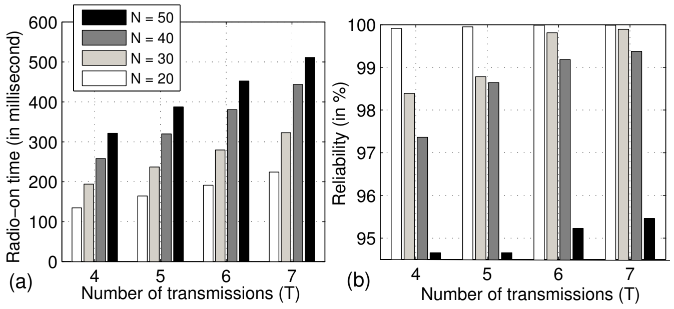

Fig. 5. Performance of MiniCast under various parameter setting in Indriya.

Assuming 4 bytes for calibration, 4 bytes of preamble, D number of data bytes, and transmission time of a single byte as t_{byte}, $t_{sub-slot}$ can be represented as -

$$t_{sub-slot} = (15 + D) \times t_{bytes}. \quad (3)$$

Thus,

$$t_{slot} = (N + 2) \times (15 + D) \times t_{bytes}. \quad (4)$$

The actual execution time of MiniCast depends on the structure of the network, in particular, on the arrangement of the nodes in layers. A layer here implies essentially a hop in the execution of Glossy. This reflects the radius of the network with respect to a certain initiator node. As is explained in the work [3], the time required to complete one round of ByteCast is roughly $3 \times (L-1) \times t_{slot}$ where L is the maximum number of layers in the system. The same argument is true for MiniCast as well.

V. Evaluation

In this section we provide a detailed report of our evaluation of PacketSync and MiniCast.

A. Comparison with asynchronous transmission based techniques

It is obvious that performance of PacketSync would be much better than asynchronous communication and CSMA based medium sharing techniques. However, for the sake of completeness we do a detailed comparison as described below.

The performance of SyncMerge is reported in [3]. We compare PacketSync in an exactly similar setting. We design experiments considering 10 TelosB motes as source nodes and one as the initiator node where each of the sources have 1 byte of data that they want to convey to the initiator. To do the comparison, we perform the job in four different ways. Two among them, i.e., *SyncMerge*, PacketSync are based on synchronous transmission while the other two, i.e., *polling* and *combined request* - are based on asynchronous transmission. In polling, each of the source nodes are individually requested by the initiator for sending the data while in the combined request based technique the initiator sends a single request message and the source nodes start to try to send the data at the same time. For the asynchronous transmission based techniques, we used the default options for medium sharing and duty-cycling protocols in Contiki [4], i.e., CSMA and ContikiMAC [15], respectively. All the experiments are done over a table under the same setting. In each run of an experiment we measure the total radio-on time required by the initiator node to successfully receive the data from all the source nodes as well as the same required by a source node to successfully send the data to the initiator and receive an acknowledgement from it. Each experiment is repeated 2000 times.

Figure 4 shows the average, the 10th and the 90th percentile of the measured values in both source and the initiator nodes. As is shown in the figure, the radio-on time requirement in PacketSync is at least 14 times smaller in the initiator and at least 1.8 times smaller in the source nodes than polling and combined request based techniques. It can be also seen from the figure that for 10 nodes the performance of PacketSync and SyncMerge is almost similar, although its quite obvious that SyncMerge will perform faster as the number of source nodes increases since it uses the most compact form possible. However, one crucial point to note here is that the reliability in PacketSync is 99.97% where the same in SyncMerge is about 98.4% and the same of the asynchronous transmission based techniques is about 98.6%. This difference in reliability is significant because in practical scenarios, PacketSync is used as a unit of operation in bigger protocols and thus, failure of a single unit results in more failure in a multi-hop context. Note that the total time required is different from the total radio-on time in the asynchronous transmission based techniques, while are same for the synchronous transmission based techniques. The figure shows only the radio-on time. The improvement in the total time is much more in both SyncMerge and PacketSync.

B. Effects of the parameters of MiniCast

As described in section IV, the number of times a node transmits a full slot, i.e., T and the number of individual nodes

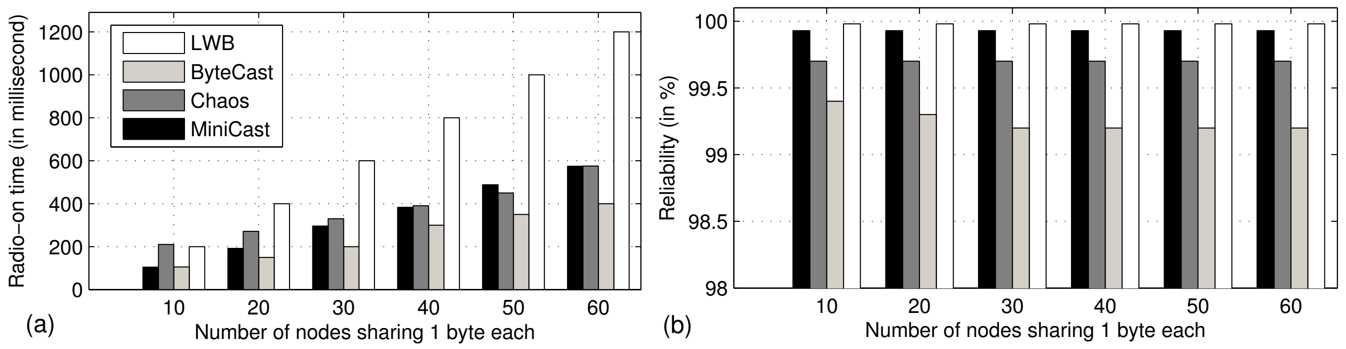

Fig. 6. Part (a) and (b) show the comparison of radio-on time and reliability of MiniCast, respectively, with other protocols in Indriya with varying number of source nodes.

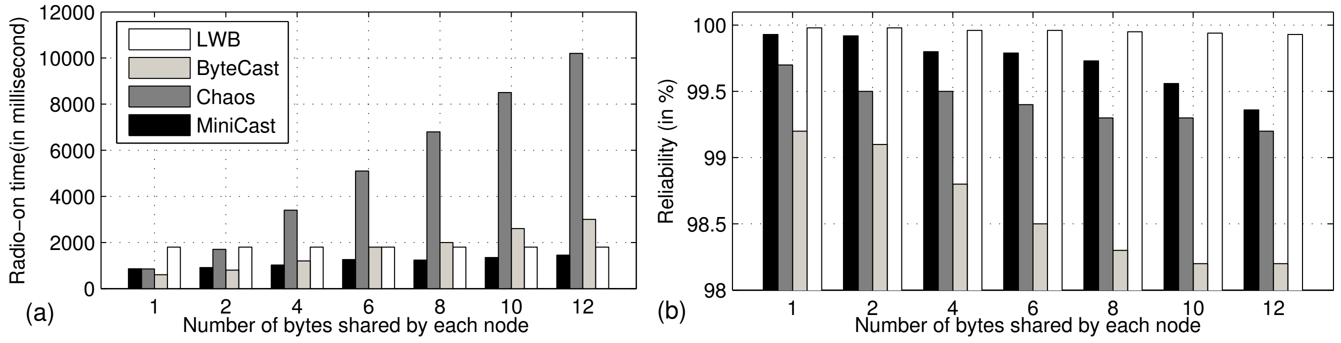

Fig. 7. Part (a) and (b) show the comparison of radio-on time and reliability of MiniCast, respectively, with other protocols in Indriya with varying number of bytes shared by each of the 90 source nodes.

that are accommodated in a single slot, in other words number of sub-slots in a single slot, i.e., N are the two parameters in the protocol MiniCast. These two parameters can be tuned for better performance depending on the network size. To understand the effect of these parameters we run experiments on Indriya.

Indriya at present has 90 active TelosB motes. We set the target of conveying 1 byte of data from N randomly selected nodes to all the other using MiniCast through T transmissions by each layer of nodes (see section IV for details). For each parameter combination we repeated the experiment for more than 10000 times. In each run each node in Indriya received data from how many of the N source nodes were measured as the reliability value. We also measured the time required to complete the operation. Figure 5 shows the average values of these two metrics for different combination of N and T.

As is shown in figure 5, the time requirement increases with T and N. These values can also be calculated approximately from the formulas provided in section IV if the structure of the network is fully known. However, it is to be noted that as we increase the value of N, the reliability decreases. This is because with large N, the length of a single slot also becomes large. Now, as is described in IV, the correct reception of REP-TRL is very crucial at the end of each of the slot in MiniCast. If a node fails to receive the REP-TRL properly, it misses the entire next slot and it can join only in the subsequent slot. Now, since all the nodes transmits simultaneously in during the transmission of REP-TRL, either constructive interference or capture effect must work for correct reception of the REP-TRL. However, as we increase the value of N and thereby allow more nodes to particulate in a single slot, the slot length increases. As a result the synchronization among the nodes may get hampered due to possible drifts in the internals clocks.

C. Comparison with LWB and Chaos

LWB and Chaos are the two state-of-the-art solutions for many-to-many communication using synchronous transmission based approach. In this section we report the performance comparison of MiniCast with these two protocols as well as ByteCast [3] which is the many-to-many extension of SyncMerge.

In order to ensure the best outcome from MiniCast we fix the parameters N at 20 and T at 9. It takes about 200 ms to complete dissemination of 1 byte of data from 20 randomly selected nodes to all the rest of the 90 nodes in Indriya. In Chaos and LWB, we used the default setting. We set the default slot size of 20 ms in LWB. The number of transmission in ByteCast is kept fixed at 18.

We run two sets of experiments on many-to-many communication with these four protocols in Indriya. We designate a set of nodes as the source nodes each of which wants to disseminate data to all the 90 nodes in the network. In the first set of experiments we vary the number of source nodes from 10 to 90, while keeping the size of the data from each node to all other nodes to 1 byte. In the second set of experiments, all nodes disseminate 1 to 12 bytes to all other nodes. Channel 26 is used in all experiments. Note that LWB takes a significant amount of time for bootstrapping to determine the scheduling among the nodes. However, the other protocols do not have such overhead. In the comparison we do not consider the bootstrapping time required by LWB.

Each individual experiment with a certain number of source nodes and data size shared by each source node is repeated 10,000 times. For each such run we measure the total radio-on time taken by each node in each of the protocols. Figure 6 and figure 7 show the average radio-on time as well as the average reliability achieved in each set of the experiments.

From 6(a) it can be seen that MiniCast, Chaos and ByteCast all take almost similar times, all substantially less than LWB. ByteCast is still the best among all these four protocols in terms of radio-on time. MiniCast naturally always takes more time than ByteCast because it has much more overhead than ByteCast. To be specific, one segment with 1 byte of data in ByteCast is only of 3 byte length. But in MiniCast it is 16 byte. However, segment synchronization is much more accurate in MiniCast than in ByteCast. Therefore, MiniCast has better reliability over ByteCast as shown in figure 6(b). Among the four protocols LWB achieves the best reliability and MiniCast achieves similar performance. In summary, as the number of nodes sharing 1 byte each in Indriya is varied from 10 to 90, MiniCast, although performs almost similar to Chaos, brings an improvement from 47% to 51% over LWB with similar reliability which is better than ByteCast.

Improvement in radio-on time in MiniCast is more observable in the second set of experiments. The achieved improvements in radio-on time in MiniCast over LWB and Chaos vary from 19% to 52% and 46% to 85%, respectively, as shown in 7(a). Note that Chaos is fundamentally an aggregation protocol. However, it can be modified to work for data sharing purpose by reserving space for each node in a single large packet. We used a 101 byte packet for disseminating 1 byte data from every node to all other nodes in Indriya using Chaos. However, to disseminate X bytes of data from each node we had to repeat the whole execution of the protocol X times. On the other hand, in case of LWB, each slot can support upto 15 bytes of data from a certain node. So, with the change in the number of bytes from 1 to 12 there is no change in the radio-on time. In case of both ByteCast and MiniCast with the increase in the number of bytes to be shared by each node, the size of the segment increases. In ByteCast this results in an increase in the number of necessary schedules. However, in MiniCast the number of schedules remain the same but the size of each sub-slot and hence the slot length increases. Thus in both ByteCast and MiniCast the time requirement increases slowly.

From figure 7(a) it can be also seen that in comparison to ByteCast, MiniCast consumes 14% to 51% less radio-on time when the number of bytes shared by each node varies from 4 to 12 bytes. However, due to the higher overhead in MiniCast, it takes more time than ByteCast for lower number of bytes shared by each node. The reliability of MiniCast as depicted in 7(b) is similar to that of LWB and is much better than both Chaos and ByteCast.

VI. Conclusion

In this paper we propose a novel strategy for achieving compact and fast many-to-many data sharing in wireless system. We implement the proposed protocol in off-the-shelf devices. Evaluation of the protocol on a 90-node testbed for wireless sensor network shows at least 20% to 50% improvement over two state-of-the-art solutions for many-to-many data sharing.

Acknowledgment

This work was partly funded by VR, the Swedish Research Council, and by SSF, the Swedish Foundation for Strategic Research.

References

[1] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele, "Low-power wireless bus," in *Proceedings of SenSys*, 2012, pp. 1–14.

[2] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Proceedings of IPSN*, April 2011, pp. 73–84.

[3] S. Saha and M. C. Chan, "Design and application of a many-to-one communication protocol," *Accepted for publication in IEEE INFOCOM, 2017*, (Manuscript available on request).

[4] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, 2004, pp. 455–462.

[5] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda, "Indriya: A low-cost, 3d wireless sensor network testbed," in *Proceedings of TRIDENT-COM*, 2011.

[6] M. Garetto, T. Salonidis, and E. W. Knightly, "Modeling per-flow throughput and capturing starvation in csma multi-hop wireless networks," *IEEE/ACM ToN*, vol. 16, no. 4, pp. 864–877, 2008.

[7] Y. Tay, K. Jamieson, and H. Balakrishnan, "Collision-minimizing csma and its applications to wireless sensor networks," *IEEE JSAC*, vol. 22, no. 6, pp. 1048–1057, 2004.

[8] F. Österlind, L. Mottola, T. Voigt, N. Tsiftes, and A. Dunkels, "Strawman: resolving collisions in bursty low-power wireless networks," in *Proceedings of the IPSN*. ACM, 2012, pp. 161–172.

[9] P. Patel and J. Holtzman, "Analysis of a simple successive interference cancellation scheme in a ds/cdma system," *IEEE JSAC*, vol. 12, no. 5, pp. 796–807, 1994.

[10] S. Gollakota and D. Katabi, "Zigzag decoding: Combating hidden terminals in wireless networks," in *Proceedings of the ACM SIGCOMM 2008*, 2008, pp. 159–170.

[11] L. Kong and X. Liu, "mzig: Enabling multi-packet reception in zigbee," in *Proceedings of MobiCom*. ACM, 2015, pp. 552–565.

[12] O. Landsiedel, F. Ferrari, and M. Zimmerling, "Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale," in *Proceedings of SenSys*, 2013, pp. 1:1–1:14.

[13] K. Leentvaar and J. Flint, "The capture effect in fm receivers," *IEEE Transactions on Communications*, vol. 24, no. 5, pp. 531–539, 1976.

[14] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. Zúñiga, "Jamlab: Augmenting sensornet testbeds with realistic and controlled interference generation," in *Proceedings of IPSN*. IEEE, 2011, pp. 175–186.

[15] A. Dunkels, "The contikimac radio duty cycling protocol," 2011.