Reprogramming wireless sensor nodes

Helen C. Leligou, Christos Massouros, Eleftherios Tsampasis, Theodore Zahariadis, Dimitrios Bargiotas, Konstantinos Papadopoulos, Stamatis Voliotis

Dept. of Electrical Engineering, Technological Educational Institute of Chalkis, Psahna, Greece

***Abstract*— As the applications of Wireless Sensor Networks increase rapidly, the number of deployed sensor devices proliferates, which prompts the research community to work towards their integration in the so-called "Internet of Things" to gather real time information and make the maximum out of their use towards enhancing the user experience. The capability to reconfigure/reprogram them remotely not only enables easy maintenance and code updates, which is mandatory in large sensor network deployments, but also provides an unprecedented flexibility regarding the use of all available resources of different types. However, the design of a reliable dissemination protocol is a real challenge and the reason is threefold: the desired reprogramming requirements differ from use case to use case (e.g. tolerated reprogramming time, affordable overhead), the wireless medium is characterized by low reliability, and the devices are severely resource constrained. For this reason, in this paper we first explore the reprogramming requirements and the intricacies of WSNs and then review the already proposed network protocols for reprogramming wireless sensor networks placing emphasis on the their features to guide both prospect users and designers efforts.**

***Keywords*— Wireless sensor networks, reprogramming, data dissemination protocols.**

I. INTRODUCTION

As the applications of Wireless Sensor Networks (WSNs) proliferate, the population of sensor nodes increases and the need to manage them remotely becomes more prominent. The need for reprogramming the nodes stems from the fact that such systems must often operate for extended periods of time unattended, while adjustments to the environment after deployment as well as code maintenance and update are needed (e.g. to improve security or robustness). Remote management is also required to fulfill the application requirements which may change in time and space [1]. In emerging WSNs which consist of hundreds or even thousands of nodes, reprogramming them one-by-one requires both physical access to each of them (which is not always feasible) and consists an extremely time-consuming procedure, impeding any real-time reprogramming without human intervention. The ability to add new functionality or perform software maintenance without having to physically reach each individual node is already an essential service.

Although solutions for remotely programming communication devices attached to specific infrastructure exist, programming resource constrained nodes over the wireless medium imposes different challenges than traditional network programming approaches. Transmitting the code that the node will execute over the air represents a real challenge due to the enhanced reliability required (since in this case the complete image must reach all the nodes) while the wireless medium is inherently unreliable. Moreover, due to limited memory, the involved nodes cannot store large files of programming code and due to limited processing resources the complexity of the employed code dissemination schemes has to be carefully assessed to ensure feasibility. If the image cannot fit into a single packet, it must be placed in stable storage until the transfer is complete, at which point the node can be safely reprogrammed. Another concern is the low throughput of WSNs: while the broadcast nature of the wireless medium can be exploited to compensate for the low communication throughput, special actions need to be taken in case of lost segments (e.g. due to collisions or errors) either when all or just one of the receiving nodes has not successfully received a packet. Hence, the design of a reliable data dissemination protocol for propagating large data objects from one or more source nodes to many other nodes over a multi-hop, wireless sensor network is a real challenge.

Unlike the unicast case where requirements for reliable, sequenced data delivery are fairly general, different multicast applications have widely different requirements for reliability. For example, some applications require that delivery obey a total ordering while many others do not. Some applications have many or all the members sending data while others have only one data source. Some deployments have all the network nodes executing the same applications and hence code dissemination is relevant to all nodes, while other consist of groups of nodes supporting different applications. Mission critical applications impose severe time-constrains while in other applications prolonged network lifetime is a top priority requirement. In any case, the dissemination process should ensure that no service interruptions to a deployed application and the debugging and testing cycle will occur. Another great challenge is imposed by the dynamic network membership and thus it must be ensured that all nodes receive the newest code since network membership is not static: nodes come and go. And while handling all these intricacies, the dissemination protocol designer should keep in mind that the dissemination must tolerate node densities which can vary by factors of a thousand or more. Such differences affect the design of a reliable multicast protocol with respect to the considered optimization metric and desired functionality. Given also the limited processing and memory requirement it becomes clear that it is very difficult (if possible at all) to design a reliable multicast delivery scheme that optimally meets the functionality, scalability, and efficiency requirements of all applications.

For this reason, this paper aims to define the functionality that any such protocol needs to fulfill and to explore the

implications of the available design choices based on state-of-the-art network protocols for reprogramming purposes. We anticipate that this work will help both protocol designers providing them guidelines and prosper system designers/administrators to choose the appropriate solution. The procedure of node reprogramming / re-tasking can be split in two steps: first, decide when reprogramming is needed and second, disseminate the program. The schemes addressing the former are usually referred to as Code Consistency Maintenance Protocols (CCMP) and the latter are usually referred to as data dissemination protocols. (Integrated solutions addressing both issues have also been proposed.) Thus, the rest of the paper is organized as follows: in section II code consistency protocols are discussed while section III is devoted to dissemination protocols. Finally in section IV, an assessment and designer guidelines conclude the paper.

II. CODE CONSISTENCY MAINTENANCE PROTOCOLS

The first step in reprogramming a sensor node is to decide when reprogramming is needed and which part of the code needs update (if partial reprogramming is supported). Depending on the application, this can be decided and initiated by the system user, or automatically (in a distributed manner) by the nodes themselves. In the former case, the system user issues a reprogram command along with a set of attributes and the nodes operate in a slave-like mode. In the latter, the nodes should realize whether a code update is needed on their own.

An interesting solution for the first case is presented in [2]. The main goal of the Sensor Network Management System (SNMS) is the monitoring and control of the node and network status by humans. SNMS provides two core services: a query system to enable rapid, user-initiated acquisition of network health and performance data and a logging system to enable recording and retrieval of system-generated events. For this reason, a logical tree-topology for reporting the status of the nodes and the network is constructed and each time the system operator decides to check the health of the network, it issues several queries. In response to these queries the nodes provide status information and if a new code version needs to be downloaded, DRIP is used as the dissemination protocol. The command to switch over the new code uses (a different) named dissemination scheme. SNMS provides also naming instructions for the attributes that may need to be reported. The first contribution of the SNMS networking stack is a collection tree construction protocol that minimizes state requirements by not requiring a neighbor table, and minimizes network traffic by requiring explicit initiation of tree construction. However, the cost of maintaining a tree construction can only be afforded in static or semi-static sensor networks. If high mobility has to be supported, the cost of maintaining a tree just for checking the network status becomes high.

Coming to the second case, a straight forward solution would be to periodically announce a profile of the code they run, so that their neighbours can compare the received information with the version of the code they run to figure out whether a code update is needed. As this way overhead is introduced, multiple schemes trying to reduce it have been proposed.

A first attempt in this direction was proposed in 2004 and is widely cited. Trickle [3] is an algorithm for propagating and maintaining code updates in wireless sensor networks. Borrowing techniques from the epidemic/gossip, scalable multicast, and wireless broadcast literature, Trickle uses a “polite gossip” policy, where motes periodically broadcast a code summary to local neighbors but stay quiet if they have recently heard a summary identical to theirs. When a mote hears an older summary than its own, it broadcasts an update. Instead of flooding a network with packets, the algorithm controls the send rate so each mote hears a small trickle of packets, just enough to stay up to date. This simple mechanism can scale to thousand-fold changes in network density, propagate new code in the order of seconds, and impose a maintenance cost on the order of a few sends an hour.

The data discovery and Dissemination Protocol (DIP) proposed in [4] places emphasis on the search of new items that need to be exchanged among nodes. The rationale behind its design is that dissemination protocols have two main performance metrics: detection latency and maintenance cost. Maintenance cost is the rate at which packets to announce the current code version are sent when a network is up-to-date. Traditionally, these two metrics have been tightly coupled. A smaller interval lowers latency but increases the packet transmission rate. A larger interval reduces the transmission rate but increases latency. Trickle addresses part of this tension by dynamically scaling the interval size, so it is smaller when there are updates and larger when the network is stable. While this enables fast dissemination once an update is detected, it does not help with detection itself. DIP uses a hybrid approach of randomized scanning and tree-based directed searches. The result is that for T items, DIP can identify new items with O(log(T)) packets while maintaining a O(1) detection latency. By dynamically selecting which of the two algorithms to use, DIP achieves high performance both in terms of transmissions and speed.

Based on the observations that when nodes have different code versions, the network may not behave as intended, wasting time and energy, DHV has been proposed [5] as an efficient code consistency maintenance protocol to ensure that every node in a network will eventually have the same code. DHV is based on the simple observation that if two code versions are different, their corresponding version numbers often differ in only a few least significant bits of their binary representation. DHV allows nodes to carefully select and transmit only necessary bit level information to detect a newer code version in the network. The name DHV comes from the three steps in the protocol:

- Difference detection: each node broadcasts a hash of all its versions called a SUMMARY message. Upon receiving a hash from a neighbour, a node compares it to its own hash. If they differ, there is at least one code item with a different version number. The next step is identification which is

achieved thourgh the horizontal search and vertical search steps.

- Horizontal search: a node broadcasts a checksum of all versions, called a HSUM message. Upon receiving a checksum from a neighbour, the node compares it to its own checksum to identify which bit indices differ and proceeds to the next step.
- Vertical search. In vertical search, the node broadcasts a bit slice, starting at the least significant bit of all versions, called a VBIT message. If the bit indices are similar, but the hashes differ, the node broadcasts a bit slice of index 0 and increases the bit index to find the different locations until the hashes are the same. Upon receiving a VBIT message, a node compares it to its own VBIT. After identifying which (key, version) tuples differ, the node broadcasts these (key, version) tuples in a VECTOR message.

Upon receiving a VECTOR message, a node compares it to its own (key, version) tuple to decide who has the newer version and if it should broadcast its DATA. A node with a newer version broadcasts its DATA to nodes with an older version. DHV can detect and identify version differences in O(1) messages and latency compared to the logarithmic scale of other protocols while in [5] DHV is shown to outperform DIP.

Multicast-based Code redistribution Protocol (MCP) is a stateful protocol for code maintenance that places emphasis on energy efficiency [6] designed to support also the case where nodes implementing different applications may exist in the same sensor networks. Each node in MCP maintains a small table to record the interesting information of known applications. The table enables sending out multicast-based code dissemination requests. MCP employs a gossip-based source node discovery strategy. Each sensor summarizes the application information from overheard advertisement messages and periodically sends out this information. To reprogram a subset of sensors, the sink floods a dissemination command that guides which sensors should switch to run application A. After receiving the command from the sink, each sensor identifies its dissemination role as a) source, if the sensor has the binary of application A; b)requester, if the sensor does not have the binary of A but needs to switch to run A; or c)forwarder, if the sensor is neither a source nor a requester.

A requester periodically sends out requests to its closest source, until it acquires all the pages of application A. Instead of broadcast, the request message is sent to the source via multicast. A requester resends the message until it timeouts. It tries to request data from each source node several times before marking the node as a temporary non-available source. A source node responds with the data (i.e., Data messages) that contain code fragments while a forwarder forwards both request and data packets. Thus, dissemination requests are forwarded to nearby source nodes rather than flooding the network. Compared to broadcasting based schemes, MCP greatly reduces signal collision, saves both the dissemination time and reduces the number of dissemination messages.

III. DATA DISSEMINATION PROTOCOLS

A. The design options

Once one or more nodes have recognized the need for code update, the dissemination protocol is triggered. The design space for data dissemination protocols is large and includes:

- selection of nodes to transmit data. (For example, having multiple nodes transmitting the same data in the same transmission range area is not a wise option in the energy and throughput constrained environment of sensor networks.)
- segment management, to alleviate the limited memory resources problem.
- reliability assurance scheme which should define how lost segments are identified, who is responsible for repairs.

To decide which node will provide the updated program, acting as the source of the updated program different options arise. Even if initially only one node has the updated code, if it broadcasts it, the set of neighbours in its transmission range can become the transmitting nodes in the sequence. (An example is shown in fig. 1a, where nodes 2, 3 and 4 receive all pages of a program image.)

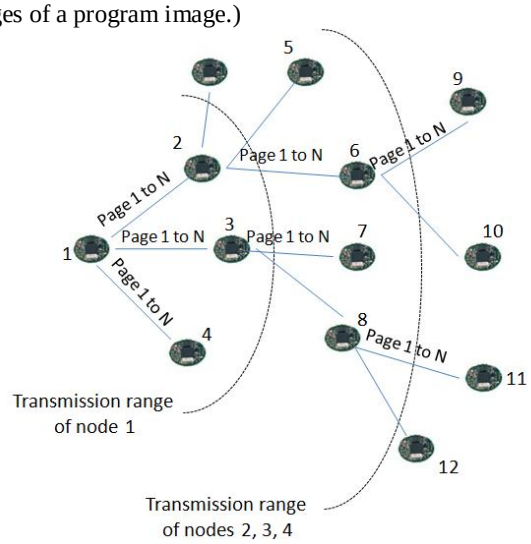

Fig. 1 Code dissemination in a neighbourhood-per neighbourhood manner

In this case where multiple nodes can transmit the code, a specific protocol to choose the source for each neighbour has to be defined. (In fig. 1, it is obvious that node 4 may save energy since node 8 can receive the updated code from node 3 and the same holds for node 7.) Solutions ranging from choosing the latest node that advertised the program to more sophisticated publish-subscribe schemes have been proposed. The source selection scheme is tightly coupled with the way the program is disseminated (e.g. fragmented or not) and with the way that lost segments are retrieved. For example, fragmentation allows for spatial multiplexing which results in reduced time required for the dissemination of the program, as shown in fig. 2, where node 6 disseminates page 1, when node 1 disseminates page 2, assuming that the code has been split in

pages. As regards lost segments retrieval, the use of negative acknowledgements is usually most suited to resource constrained WSNs. If one node has lost a certain packet, it can either request the retransmission of the packet, or wait to hear it again as part of the dissemination of the code to a next round. In the former case, the request can be unicasted to the source, multi-casted to improve the probability of receiving back the packet or even broadcasted, to make sure the packet will be received at the cost of energy waste from multiple nodes and increased congestion probability. Other ways to enhance reliability include the adoption of forward error correction (FEC) to avoid (to the extent possible) retransmissions or use link quality estimates to improve decisions or even use fountain codes to transmit data.

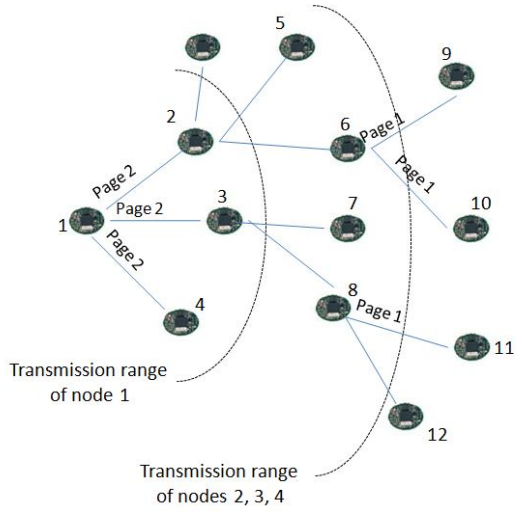

Fig. 2 spatial multiplexing: when node 6 disseminates page 1, node 1 disseminates page 2, reducing the total code dissemination time

The performance metrics reflecting the target of the dissemination protocols [7] include efficiency in terms of overhead (its reduction directly translates to better utilization of the low available throughput) and of energy (which affects the network lifetime, an important parameter in this mainly battery operated environment) as well the time required to detect that an update is needed and the time required to disseminate the update throughout the network.

.

B. Dissemination protocol examples

As the core service is the multicast dissemination of the code, the roots of such protocols lie back in 1997, when the Scalable Reliable Multicast protocol was proposed [8]. It was the first attempt to design a reliable transport protocol suitable for multicasting cases and cases where single/multiple sources initiate sessions with multiple destinations. SRM was designed to meet only the minimal definition of reliable multicast, i.e., eventual delivery of all the data to all the group members, without enforcing any particular delivery order. Its inventors state that if the need arises, machinery to enforce a particular delivery order can be easily added on top of this reliable delivery service. SRM is also heavily based on the group delivery model that is the centerpiece of the IP multicast protocol where data sources simply send to the group' s multicast address (a normal IP address chosen from a reserved range of addresses) without needing any advance knowledge of the group membership. To receive any data sent to the group, receivers simply announce that they are interested (via a "join" message multicast on the local subnet); no knowledge of the group membership or active senders is required. Each receiver joins and leaves the group individually, without affecting the data transmission to any other member. SRM further enhances the multicast group concept by maximizing information and data sharing among all the members, and strengthens the individuality of membership by making each member responsible for its own correct reception of all the data. SRM attempts to follow the core design principles of TCP/IP requiring only the basic IP best effort delivery model and builds reliability on an end-to-end basis. No change or special support is required from the underlying IP network. In a fashion similar to TCP adaptively setting timers or congestion control windows, the algorithms in SRM dynamically adjust their control parameters based on the observed performance within a session. This allows applications using the SRM framework to adapt to a wide range of group sizes, topologies and link bandwidths while maintaining robust and high performance.

However, keeping state information per session and executing the TCP/IP functionality is extremely demanding for sensor nodes. Additionally, when a new code has to be disseminated to all nodes in the network, SRM does not exploit the fact that broadcast costs less in wireless sensor networks.

Another approach to the dissemination problem answering these issues is to transfer the data in a neighborhood-by-neighborhood basis. This implies a single-hop mechanism that can be recursively extended to multi-hop. Each node that receives the code can then start transmitting (broadcasting) it thus becoming the source node. In this case, it is the number of source nodes that need to be handled, since there is no need for two one-hop neighbours to both behave as sources using for example a publish-subscribe interface. Sources with no subscribers should remain silent.

In an attempt to reduce the time required for the code to reach all the network nodes, Multi-hop, Over-the-Air code distribution Protocol- MOAP [9] uses the store-and-forward approach, providing a 'ripple' pattern of updates. While it adopts the previous approach of broadcasting the code in a neighbour-per-neighbour way, it allows each node having received the code to start further disseminating it (announcing the version of the code it has) thus reducing the latency. A link-statistics mechanism is used to try to avoid unreliable links. After waiting a period to receive all subscriptions, the sender then starts the data transfer. As regards lost segments, to alleviate the sender from monitoring multiple sessions with the recipients of the code, in MOAP the receivers are responsible for identifying any lost segments. Once this happens, the request for the segment is not broadcasted, to

avoid duplications, but instead is requested by a single node; a keep-alive timer is used to recover from unanswered unicast retransmission requests – when it expires a broadcast request is sent.

While further optimizations are possible, MOAP is a dissemination protocol that was feasible to implement in motes (as reported in [9]) and has been shown to reduce the overhead by 60% compared to the simple flooding case.

Just one year later, Deluge a widely used and recognized protocol has been proposed [10]. It builds on prior work in density-aware, epidemic maintenance protocols and includes several optimisations. Firstly, it adopts Trickle for the advertisement of code versions which reduces the messages needed for the nodes to realize a new code version is available and should be propagated. A second contribution of Deluge is that it splits the code into a set of fixed-size pages thus it provides a manageable unit of transfer which allows for spatial multiplexing, i.e. pages are dealt with as independent transfer objects. This way the time required for the propagation of a large program is reduced and at the same time incremental upgrades are supported.

Based on Trickle, each node occasionally advertises (ADV) its most recent object profile to whatever nodes that can hear its local broadcast. From the object profile, the receiving node R determines which portions of the data need updating and requests (REQ) them from any neighbour that advertises the availability of the needed data, and finally this node provides the data (three way handshake). Deluge simply requests data from the node which most recently advertised the needed page, which is an easy-to-implement scheme and ensures no duplication of pages. Nodes receiving requests then broadcast any requested data. A node requests from a single node (neighbour) updated code or exploits a request issued by another neighbour. Nodes then advertise newly received data in order to propagate it further.

The major advantages of Deluge include: a) Deluge's three-phase handshaking protocol helps ensure that a bi-directional link exists before transferring data, b)Representing the data object as a set of fixed-size pages provides a manageable unit of transfer which allows for spatial multiplexing, c) Deluge advertises the availability of complete pages even before all pages in the object are complete allowing the further propagation of newly received pages, d)supports efficient incremental upgrades, e)Deluge attempts to minimize the set of nodes concurrently broadcasting data within a given cell and f) it adopts lost segment recovery based on negative acknowledgement which reduces the exchanged overhead.

Using both a real-world deployment and simulation, Deluge has been shown to reliably disseminate data to all nodes and characterize its overall performance. On Mica2-dot nodes, Deluge can push nearly 90 bytes/second, one ninth the maximum transmission rate of the radio supported under TinyOS. Control messages are limited to 18% of all transmissions. At scale, the protocol exposes interesting propagation dynamics only hinted at by previous dissemination work. On average a node receives about 3.35 times the minimum number of required data packets, due to the single-channel, broadcast network.

Stream [11] builds on Deluge and optimizes what is actually sent over the channel. Common intuition would be to transfer only what actually needed, i.e., the program image. However, Deluge disseminates the image of the programming protocol together with that of the program to be transferred. This considerably inflates the amount of data to be disseminated (up to 20 folds for the transmission of a program image consisting of a single page). Stream obviates this problem by pre-installing in each sensor node, before its actual deployment, the re-programming application. This is done through the segmentation of the flash memory into multiple partitions so that the re-programming protocol and the program to be transferred are stored in different image areas. Hence, at dissemination time Stream transmits over the channel the minimal support (about one page) needed for the activation of the re-programming image together with the actual program image.

DRIP [2] is the unnamed reliable dissemination protocol of Sensor Network Management System (SNMS).The SNMS dissemination protocol, named Drip, provides a transport layer interface to multiple channels of reliable message dissemination. Implemented as a TinyOS component, Drip provides a standard message reception interface. Each component wishing to use Drip, registers a specific identifier, which represents a reliable dissemination channel. Messages received on that channel will be delivered directly to the component. Each node is responsible for caching the data extracted from the most recent message received on each channel to which it subscribes, and returning it in response to periodic rebroadcast requests. In the implementation reported in [2], space for this cache is allocated by the subscribing component, and data is retrieved from the cache in response to an upcall. The Drip protocol uses a sequence number with half-space wraparound to determine whether a received message is new, and upon receipt of a new message, the data is delivered to the subscribing component for required caching and optional action.

The Drip protocol uses the message as the unit of reliability, and the component as the unit of caching. This design allows Drip to function as a standard transport layer protocol. But, it does introduce extra complexity for a component that has several independent variables which must be reliably synchronized among all nodes in the network. To solve this problem, the component must collect the current value of each variable into a single reliably disseminated message. This method will produce independent reliability for each variable, as long as every node stores the same value of every variable. This problem could also be solved by selecting the variable as the unit of reliability and caching, and by associating a unique key with each variable instead of associating a channel with each message. However, this approach would require a significantly larger key-space, and would blur the boundary between protocol and data storage.

SYNAPSE [12] is an original reprogramming system for WSNs designed to improve the efficiency of the error

recovery phase. Synapse implements a hop-by-hop data dissemination protocol in which data blocks are sent during so-called dissemination rounds and one node at a time is allowed to send. SYNAPSE features a Hybrid Automatic Repeat reQuest (HARQ) solution where data are encoded prior to transmission and incremental redundancy is used to recover from losses, thus considerably reducing the transmission overhead. For the coding, digital Fountain Codes (FCs) were selected as they are rate-less and allow for lightweight implementations. Special Fountain Codes were used at the heart of SYNAPSE to provide high performance while meeting the requirements of WSNs. FCs are rateless and have a low computational complexity, as encoding and decoding are performed efficiently through XOR operations. While others approaches mainly concentrated their study upon devising smart algorithms (i.e., modified epidemic schemes) for sender selection, sleeping modes etc., SYNAPSE's focus is on extremely efficient solutions for the local delivery of the data (i.e., between the senders and their neighbors), as well as their proper integration with previous techniques. It uses three way handshakes as the paradigm introduced for Deluge above. It implements randomization when sending advertisements. It exploits broadcast transmissions for the code and Negative Acknowledgments (NACK) to request missing data and it implements the method proposed in Stream.

Considering that resource-awareness, time-efficiency, and the integration of appropriate security solutions are keys to the success and acceptance of a code update mechanism, a dependable data dissemination protocol for time-efficient and secure code updates in large-scale wireless sensor networks has been proposed in [13]. The multi-hop propagation scheme is based on security-enhanced fountain codes and means from fuzzy control theory. The basic idea of a digital fountain is the following: the fountain, i.e. the sender, generates a stream of water drops, the encoded packets. Every receiver, in turn, holds a bucket under the fountain until a sufficiently high number of drops could be collected. The receiver can recover the source data from any subset of encoded packets in which the number of packets is equal to or only slightly higher than the number of source packets. If some fraction of the initial encoded packets is erased, it is not necessary to retransmit the very same packets but rather yet another random linear combination is sent. That is why fountain codes improve the efficiency of wireless broadcast channels, i.e. one and the same packet allows different receivers to extract complementary information that is relevant to them. Furthermore, a digital fountain supersedes the need for the sender to guess the quality of the channel and therefore enables the design of scalable transmission of data in a broadcast and multicast setting over arbitrary channels.

To decrease the number of packet collisions and mitigate the hidden terminal problem, means from fuzzy control were used to dynamically adapt the send rates of sending nodes in accordance with the local congestion of the radio channel. Fuzzy control theory was recognized as a promising approach to control the level of channel utilization as it is well suited for resource constrained sensors. Furthermore, fuzzy control systems were reported to be effective in making real time decisions from incomplete information. The congestion level of the channel and the demand of neighboring nodes for missing data packets are characterized by the numbers of overheard encoded and NACK packets. To reduce data overhead, the output of the fuzzy controller is used to define a time interval during which the next packet will be sent randomly.

The use of fountain codes or random linear codes has been also adopted in ReXOR [14], which is a light-weight and density-aware reprogramming protocol for wireless sensor networks that employs XOR encoding in the retransmission phase to reduce the communication cost. ReXOR places emphasis on the lightweight implementation as well as on ability to adapt on the network density. Regarding the lightweight implementation, results for the TinyOS platform show that it requires less resources than other coding-based schemes. As regards the adaptation of the network density, this is achieved by adapting the inter-page waiting time.

MELETE [15] is a code dissemination protocol designed to support multiple concurrent applications in a WSN, and thus assumes that the network is split in groups of nodes executing different tasks. MELETE employs a group-keyed method to selectively distribute application code to only interested sensor nodes, and reactively distribute code only when it is required. This way only interested node receive the code update while the second design choice delays the time of code transportation until the exact moment the code is required at the cost of higher delay in code transportation. A passive code dissemination policy is proposed with active advertisements. Specifically, version information of all groups is disseminated throughout the network and maintained by all sensor nodes, while code is passively disseminated only when it is requested by certain nodes. Since version packets are usually smaller than code packets, this policy aims to minimize network traffic overhead while keeping all sensor nodes up to date without large delay. Specifically, each node maintains the version information of all applications that it has heard of. Each node advertises its version information for all groups in a round-robin fashion. Whenever a node receives newer version information about a group, it updates its local data, and sets its version timer to the highest rate, similarly to Trickle. If the received information is for an associated group, the node switches to the REQUEST state, and advertises its request for the new code. The key difference between Melete and Trickle is that Trickle allows nodes to advertise version information only after receiving the code, while Melete allows the propagation of version information without sending the actual code. MELETE constructs a multi-hop region between the requesting node (or requester) and potential responding nodes (or responders) so that both requests and responses can be transmitted across the region, referred to as a forwarding region. To do so, one more state, FORWARD, is added into Trickle, as also happens in MCP protocol. A sensor node switches to the FORWARD state if it "believes" that its neighbors need help to get their code, and stays in the FORWARD state until one of the following three events:

timeout, the requests are fulfilled, or there is a need to switch to the REQUEST or RESPOND states. This code chunk in its forwarding cache. A forwarder broadcasts the cached request using the Trickle protocol, as if it is a normal requester. When the forwarding region expands to a responder, the responses are transmitted throughout the forwarding region. Each forwarder caches the most recently received chunk and forwards it using Trickle, as if it is a normal responder. Nodes outside the forwarding region discard any received responses. To reduce traffic, the bitmap of a node is also updated upon reception of forwarded responses in the neighborhood. When the bitmap indicates no more chunks to forward, the forwarder switches to the MAINTAIN state. However, neighbors of a requester may blindly start forwarding even though some other neighbors of the requester can resolve the request and the forwarding region can unnecessarily expand to the entire network even though a nearby responder is discovered. To solve these problems: lazy forwarding and progressive flooding are adopted. From a prospective forwarder's perspective, it should offer help to the requester only when it was truly needed (i.e., one hop neighbors of the requester did not resolve the request).This is difficult because either the requests or responses can be lost or corrupted over the wireless channel. To handle this problem, lazy forwarding was proposed. The main idea is to allow enough time for neighbors of a requester to respond before starting a forwarding process. Specifically, each time a node receives a request, it switches to the FORWARD state with a certain probability, Pf. Pf increases with the number of received requests. It is similar to a human life scenario: after hearing multiple shouts for help, a person is more assured that someone is in real trouble. To prevent a forwarding region from expanding to the entire network, a progressive flooding technique is proposed, a special n-ring model tailored to the periodic broadcasting of Trickle. Each forwarded request is associated with a time-to live (TTL), in terms of hop counts.

IV. ASSESSMENT AND CONCLUDING REMARKS

The problem of reliable data dissemination has been addressed with multiple diverse approaches from the research community. The reason is that, while a common set of performance metrics could be agreed upon, the priority assigned to each of them depends on the specific use case. Additionally, the limited node and network resources prohibit the realization of a sophisticated, possibly even modular scheme that would be activated on demand, i.e. activate the most suitable component based on the realized application. So, any system designer should firstly clearly define the top performance metrics of interest and then choose the appropriate code maintenance and dissemination protocol.

Time-critical applications: when a system to support time critical applications is designed, it is absolutely necessary to avoid any service disruption (during code updates) and to switch to the new code in minimal time. In this case, it is the completion time (the time required for all the nodes to received the updated code correctly) that counts most, and should guide the selection of the dissemination protocol. Moreover, to cope with the rapid code updates, code consistency maintenance protocols that minimize the time required to identify a new version should be used. Such applications could be met at military missions or terrorist attack defense, where a sensor system may undertake the responsibility for gunshot localization and thus rapid dissemination of information to combatants is critical. To achieve low completion times and low version detection times, higher overhead is required which increases the energy consumption and reduces the bandwidth utilization.

Frequent code updates: When reprogramming is used to change the functionality of the sensor network in the day or between the days of the week, then the latency and the completion time no longer represent the top requirement. Instead, the energy consumption is more important in this case, since lower energy consumption means longer network lifetime. For example, reprogramming a sensor network to execute a stronger encryption scheme may change twice every day or to change specific parameters of the application (how often and what types of sensed data are sent to the sink) does not impose latency constraints (e.g. in agricultural monitoring where nodes operate in unattended node in wide areas and may need to monitor the temperature once every 5 or ten minutes in spring but more often in winter to prevent the fruit freeze). To safeguard the network lifetime, the energy consumption which is tightly coupled with the overhead of the code consistency and dissemination protocols should be the basic decision factor.

Lost segment recovery: The scheme employed to recover from lost segments affects both the completion time and the overhead produced by the dissemination protocol. As such, it could be seen as a parameter already addressed previously. However, the use of fountain codes or random linear codes has been shown to require light-weight implementations and to also reduce both completion time and overhead. The research community seems to find a common direction on this issue.

Support of multiple applications in the sensor network: As emerging sensor networks comprise of nodes executing multiple applications, the need to support their remote reprogramming becomes essential. In this case, each node should be able to differentiate its role with respect to the different supported applications and become either source/destination of an updated code of interest or just a forwarder for other nodes. This need slightly increases the processing requirements, and thus the implementation of such a scheme should be restricted to systems where needed, leaving more hardware resources for the execution of more sophisticated application or security schemes. The evolution of sensor node hardware platforms is expected to alleviate such problems and enable the wide implementation of dissemination protocols supporting multiple applications concurrently.

To this end, with the current sensor platforms a code consistency and dissemination protocol there is no one-fits-all cases solutions. For this reason, the prospect protocol designer or system implementer has to carefully consider and prioritize

the requirements of the addressed use-case and then proceed to the proper design choices.

ACKNOWLEDGMENT

The work presented in this paper was partially supported by the ARTEMIS Project 100032 SMART.

REFERENCES

[1] E. Ladis, I. Papaefstathiou, R. Marchesani, K. Tuinenbreijer, P. Langendörfer, T. Zahariadis, H. C. Leligou, L. Redondo, T. Riesgo, P. Kannegiesser, M. Berekovic, C. J. M. van Rijn , "SMART: Secure, Mobile visual sensor networks ArchiTecture", *IEEE SECON2009*, 22-26 June, 2009 Rome Italy

[2] G. Tolle, D. Culler, "Design of an application-cooperative management system for wireless sensor networks", *European Workshop on Wireless Sensor Networks*, Feb. 2005

[3] P. Levis, N. Patel, D. Culler, S. Shenker "Trickle: a self-regulating algorithm for code propagation and maintenance in wireless sensor networks" Proceedings of the *1st conference on Symposium on Networked Systems Design and Implementation* - Volume 1 San Francisco, California, 2004

[4] K. Lin, P. Levis, "Data discovery and dissemination with dip", *International Conference on Information Processing in Sensor Networks* (IPSN 2008), Washington, DC, USA, IEEE Computer Society (2008) 433–444

[5] T. Dang, N. Bulusu, W. Feng, S. Park, "DHV: A Code Consistency Maintenance Protocol for Wireless Sensor Networks", In Proc. of the 6th *European Conference on Wireless Sensor Networks* (EWSN 2009), Cork, Ireland, Feb 2009

[6] W. Li, Y. Zhang, and B. Childers, "MCP: an Energy-Efficient Code Distribution Protocol for Multi-Application WSNs", 5th *IEEE International Conference on Distributed Computing in Sensor Systems*, LNCS 5516, Springer-Verlag, pages 259-272, Marina Del Rey, California, June 2009

[7] M. Horsman, M. Marin-Perianu, P.G. Jansen, and P.J.M. Havinga, "A Simulation Framework for Evaluating Complete Reprogramming Solutions in Wireless Sensor Networks", 3rd *International Symposium on Wireless Pervasive Computing*, 7-9 May 2008, Greece. pp. 6-10

[8] S. Floyd, V. Jacobson, C. Liu, S. McCanne, L. Zhang, "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing", *IEEE/ACM Transactions on Networking*, December 1997, Volume 5, Number 6, pp. 784-803.

[9] T. Stathopoulos, J. Heidemann, D. Estrin, "A remote code update mechanism for wireless sensor networks", Technical Report CENS Technical Report 30, 2003.

[10] J. W. Hui, D. Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale", *Sensys 2004*

[11] R.K. Panta, I. Khalil, and S. Bagchi, "Stream: Low Overhead Wireless Reprogramming for Sensor Networks," *IEEE Conference on Computer Communications* (Infocom), 2007.

[12] M. Rossi, G. Zanca, L. Stabellini, R. Crepaldi, A. F. Harris III, and M. Zorzi, "SYNAPSE: A Network Reprogramming Protocol for Wireless Sensor Networks using Fountain Codes", *IEEE SECON 2008*, San Francisco, California, US. June 16-20, 2008

[13] K. Maier, A. Hessler, O. Ugus, J. Keller, D. Westhoff, "Multi-Hop Over-The-Air Reprogramming of Wireless Sensor Networks using Fuzzy Control and Fountain Codes" *SOMSED'09*, Self-Organising Wireless Sensor and Communication Networks Hamburg, Germany 8 - 9. October 2009

[14] W. Dong, C. Chen, X. Liu, J. Bu, Y. Gao, "A Light-Weight and Density-Aware Reprogramming Protocol for Wireless Sensor Networks", *IEEE Transactions on Mobile Computing*, Dec. 2010

[15] Y. Yu, L. J. Rittle, V. Bhandari, J. B. Lebrun, "Supporting concurrent applications in wireless sensor networks". 4th int. conference on *Embedded Networked Sensor systems*, SenSys 2006 , pp. 139-152.