Providing Data Integrity in Intermittently Connected Wireless Sensor Networks

Igor Talzi
Computer Science Dept
University of Basel
CH-4056 Basel, Switzerland
Igor.Talzi@unibas.ch

Sandro Schönborn
Computer Science Dept
University of Basel
CH-4056 Basel, Switzerland
Sandro.Schoenborn@stud.unibas.ch

Christian Tschudin
Computer Science Dept
University of Basel
CH-4056 Basel, Switzerland
Christian.Tschudin@unibas.ch

***Abstract*— Intermittently connected Wireless Sensor Networks (WSN) for scientific environmental monitoring raise the issue for reliable data gathering which is a key factor for having a consistent data stream as a result. First, measurements must be taken and securely saved on a node. Per node local repository for measurements is a basic mechanism for making possible to retransmit any block of data towards a data sink if it has been lost or damaged. Second, high reliable data communication channels are established between nodes, including MAC-layer and application-layer ”end-to-end” acknowledgments. The last, and the most important aspect is using a multi-level data buffering mechanism based on an ”aging”-factor for reducing power consumption taken by radio transmissions. We describe our solution, combining all these techniques, which is intended for use in a permafrost monitoring project. In our case where half-/hourly measurements are collected over a whole year and nodes can become disconnected for several months, we still manage to provide loss-free and power efficient data delivery.** [1]

Keywords— **Sensor network, data gathering, energy effciency**

I. Introduction

We intend to use our data gathering scheme in the PermaSense project ([1], [2]) whose main objective is to build and customize a set of wireless measurement units for use in remote rock areas with very harsh environmental monitoring conditions. From the natural science's point of view the main goal of our system consists in gathering the environmental data (temperature, conductivity) that helps to understand the processes that connect climate change and rock fall in permafrost areas. During 7 to 8 months out of a full year, the WSN infrastructure is unreachable and the lack of connectivity between nodes is very probable due to extreme weather conditions. The system still must operate unattended, though.

In terms of WSN data gathering assumes using a set of protocols at different levels in order to deliver a packet from one end-point called a measuring node to the other end-point called a data sink (or root/base) node. In turn, the root node itself is attached to a front-end server either via wired or wireless link where data is accumulated for further post-processing and analysis. Note, in this architecture a one does not consider node-to-node communication as a designated purpose but as a way to move data further towards the root node. The set of protocols includes MAC-layer, time synchronization, routing, etc. Over the past several years a lot of solutions were proposed in these domains. Nevertheless, reliable and still **power-aware** data delivery has not been considered as an object yet, which is actually the main task of most WSNs running or being developed.

Talking about reliability we have to consider the major trade-off: communication intensity vs. power consumption. Also, we have a third player on the stage which is called ”intermittent connectivity” meaning situations when the physical connection cannot be established between nodes, and when it can last for a long time. The latter makes even communication-intensive techniques helpless for maintaining network structure and, consequently for trouble-free data draining. As far as we cannot prevent the lack of connectivity, we focus on techniques which enable extraction any piece of data from a network even after long periods of silence. The need for that is dictated by the requirement that measurements are taken very rarely (half-/hourly) and each measurement is valuable for post-analysis.

The remainder of the paper is structured as follows: After giving a brief description of the system architecture in the next section, we describe data flow, memory zones and data gathering scheme in Section III. In Section IV we consider basic means used in our system to control data flow. Section V describes the implementation details where we also provide experimental evaluation results. After relating to similar research, we conclude this paper with the future plans.

II. System overview

Our measurement system is of an Internet-attached architecture (Figure 1). Data is collected in battery-operated sensor nodes. Due to the fact that system is intended to be installed in remote areas where wired connections are not available, one of nodes (also battery operated) has a GPRS extension which makes possible to exchange data with a database side and establish connection with a time server over Internet using GPRS channel. The TinyNode-584 hardware platform from Shockfish is used ([3], [4]). We made this choice, because the

[1]The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. We also acknowledge substantial support from the Swiss Federal Office for the Environment (FOEN).

platform was specifically designed for low-power operation and long distance range radio communications.

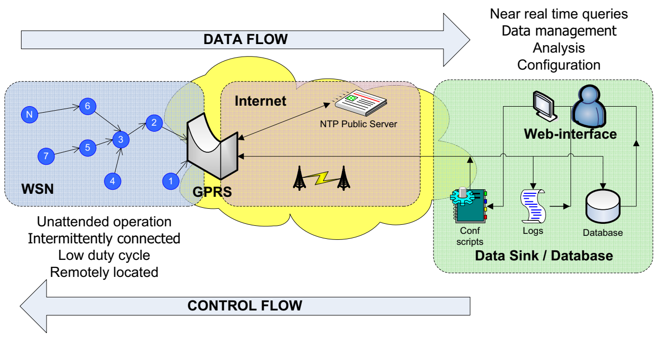

Fig. 1. PermaSense system architecture

As it was mentioned above, measurements are typically taken half-/hourly. Synchronization is done more often - approximately every 5 minutes. For the purpose of synchronizing nodes we use our own TDMA-like protocol (see [5] for more details). This time sync protocol is capable to keep nodes synced even over long periods of disconnectivity, allowing nodes to quickly catch up the network. The protocol was specifically design for small-scale networks (up to 50 nodes). Nodes periodically exchange sync messages, which contain not only info needed for performing synchronization, but also routing info, commands, etc. General fields of these messages are shown in Table I.

TABLE I
SKEW MESSAGE GENERAL FIELDS

Skew Message Field	Description
Scaling factors	Used for changing time scale on-the-fly (see Section IV-B)
Routing info	Used for building routing tree (see Section IV-A)
Commands	Used for delivery commands to nodes (see Section IV-C)
Sync info	Used for sync purposes

Besides sync messages, there are also log messages - data blocks and special system events (e.g. reboot, new sensor attached, etc). The format of these messages is shown in Table II. In contrast to sync messages which are broadcast, log messages are unicast from filial to parental nodes.

TABLE II
LOG MESSAGE FIELDS

Log Message Field	Description
type	Message type
timestamp	Local node's event timestamp
node ID	Event generator's ID
message ID	Message ID (sequence number)
payload	Packet payload

All messages have four major fields: <message type, node ID, message ID (sequence number), local node's event timestamp>. Node ID and message ID fields are used to uniquely identify each message in the system. Additionally, some customized sensors attached have unique 48 bits wide identification numbers provided by using the on-board serial number chip (see [6]). This makes possible to distinguish between different sensors in case of any collisions during post-processing. Having each message timestamped and applying post-hoc reconstruction procedure (as it is described in [5]) it is possible to build a whole precise picture of events and measurements over time.

On ordinary (non-GPRS) nodes measurements, system events and clock skew events are numbered, recorded at the source in the on-board flash memory and queued for transmission towards the GPRS node, and further to the data sink server. The GPRS-equipped node periodically establishes connection with an external NTP server and transmits packets received from other nodes to the database.

III. DATA FLOW

In PermaSense we mark out four memory transfer zones where data is flowing from or to:

– source sensor node, with local (flash) storage,

– buffer (RAM) of transit nodes along a corridor (data packets being forwarded),

– buffer (GPRS module's flash) of the root node,

– sink UNIX system with a database.

Data flow control in computer networks is the problem of controlling when data is transferred from one (memory) place to the other. For example, a sender should not send data too quickly in order to not overwhelm intermediate nodes or the destination sink with data. Also, as soon as memory space has become available again somewhere in the forwarding chain, the source should be informed about this fact. The full data flow used in our system is shown in Figure 2.

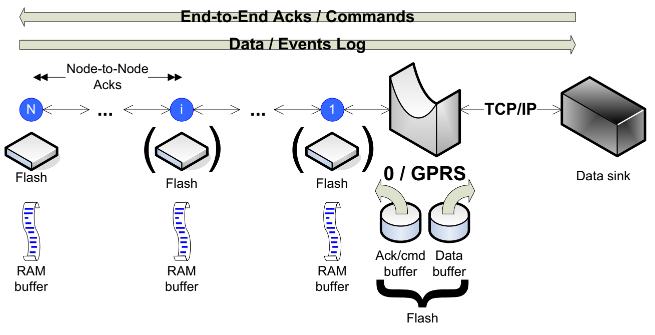

Fig. 2. Complete data flow

Typically some "credit system" is used where the number of available credits corresponds to the maximally available free memory slots one hop downwards, or end-to-end. Sending a data item consumes a credit point, and the sender can only continue sending if it has not exhausted its credit, and/or when it receives new send credit. The ultimate goal is to deliver sensor readings to a "safe place". Because packets can be lost, some scheme supporting acknowledgments and ARQ retransmissions (automatic repeat request) is needed. A simple end-to-end acknowledgment is such a combined flow control and confirmed-delivery system: a sender (zone 1 on the list above) can only erase an item (data block) if it has been acknowledged by a receiver (zone 4). If the receiver makes a decision that a packet is lost, it can send a request for retransmitting this packet to the sender.

In the following subsections we discuss all four memory zones and possible interactions between them in details.

A. Zone 1: On-node flash repository

After taking measurements or registering some event data is wrapped in the so-called ASU (Atomic Storage Unit) structure which is an universal form of representing data in the on-board flash. The ASU format is quite similar to the log message format given in Section II except for the message ID field which is assigned dynamically each time. All ASU blocks are of equal size. The payload part of each ASU contains actual data corresponding to the type field. Some high-usage data types are shown in Table III.

TABLE III
ASU DATA TYPES

ASU data type	Description
Data blocks	Sensor readings; Flash and RAM
Reboot event	Registered when reboot occurs, Flash and RAM
Flash chip reset	Registered when flash chip reset is done; Flash and RAM
Statistics blocks	Various statistics info (current system conf parameters, memory size (RAM and Flash), topology info, MAC layer stat, etc.); Flash and RAM
Sync info messages	Needed for post-hoc time reconstruction on a server; RAM

Although there are existing solutions (see [7] and [8]) implementing data reduction strategies, we do not use any data compression or averaging algorithms in our system (data blocks contain raw data). It is dictated by the fact that measurements must be of very high resolution and data has a high entropy (e.g. temperature must be measured with a calibrated accuracy of 1000 ppm or $0.02\,^{\circ}\mathrm{C}$ at the melting point, temperature variations can be from $-40\,^{\circ}\mathrm{C}$ to $+50\,^{\circ}\mathrm{C}$). Besides analog sensors we also use digital ones which deliver already processed and compressed data to a node, thus there is no need to apply any further reduction techniques to this data.

Beside the flash repository, each node including the root one has an internal message buffer allocated in RAM. It is used for storing locally generated packets and packets received from other nodes. As you can see in Table III, not all locally generated packets are stored into flash (e.g. sync messages). This is done in order to save space in the flash memory. We discuss organization of the RAM buffer in details in Section III-B.

The TinyNode platform provides flash memory for 512 kB of data comprising 2048 pages of 264 Bytes/Page. For storing ASU blocks in the flash we use the logical organization shown in Figure 3.

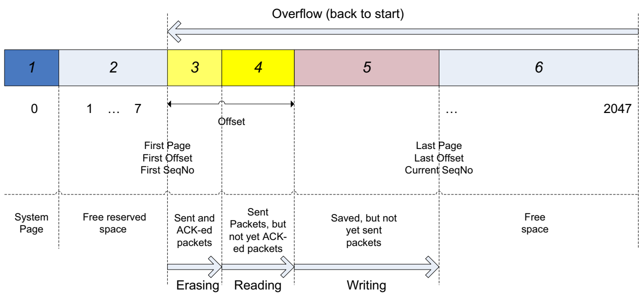

Fig. 3. On-node flash repository

System page (region 1; 1 page) contains various runtime configuration parameters, pointers, counters, etc. After it we reserved some space (region 2; 7 pages) for future use. Regions 3-6, the rest of the flash, contain actual data - ASU blocks. Blocks are written one by one without padding. Basically, regions 3-6 function as one single circular buffer: new blocks are added to a tail, acknowledged blocks are erased from a head releasing flash space.

1) In-place down-sampling: One of the basic principles of our system is that all sensor readings and system events are put into on-board flash memory. Here we meet a first flow control problem: the flash memory might become full after some time. Two ways are foreseen to handle this:

1) Flash memory is constantly drained by sending its content to the sink. This is a normal operational mode.

2) If draining is not possible (for lack of connectivity), and memory is exhausted, we propose a ”thinning” approach (or as we call it - ”in-place down-sampling”): new entries will overwrite old ones resulting in a downsampling[2]. Instead of keeping logs in half-/hour interval, we keep records for each hour. If this also has exhausted, we start keeping values at 2h intervals and overwrite some of the hourly records (but not all[3]).

The scheme works as follows: Assume an array of size 2^n+1, first array position has index 0, first sensor reading has serial number $s = 1$. At first, entries are filled sequentially $(1, 2, ..., 2^n$ at exactly this index position) and the last one $(s = 2^n + 1)$ goes into array position 0. When the store is full, we discard every second sensor readings and store the other fresh values at the positions ”$s \ mod \ (2^n + 1)$”. If we also exhaust this level (after $(2^n + 1)/2$ write ops), we store every forth entry (discarding three out of four), then every 8th etc. Example with $n = 2$, i.e. array of length 5 is shown in Table IV: Time goes downward, each thinning level commented in a block.

[2]Down-sampling is more preferable in our case than any data reduction methods as it has been described in Section III-A. From geo-science point of view for modelling having ”thinning” but more accurate data is better than averaged continuous data stream.

[3]There will be some entries in the store which must NOT be down-sampled. Especially the ”reboot-event” entry must be preserved, otherwise there might be problems in accurately reconstructing global time.

TABLE IV
IN-PLACE DOWN-SAMPLING

a0	a1	a2	a3	a4	Comments
5	1	2	3	4	(we can store all 5 consecutive readings)
		7		9	(we can only store every second reading. the content at the end of this level is: 5 1 7 3 9 which has the last, the first, and in-between every 2nd sensor value)
		17	13		(we can only store every 4th reading. Content at the end: 5 1 17 13 9)
	25		33		(down-sampling now with factor 8. Content at the end: 25 1 17 33 9)
	65			49	(down-sampling now with factor 16. Content at the end: 65 1 17 33 49) etc.

Assuming that connectivity is available, a node always follows a greedy draining policy: as much data as possible (in according to the transmission corridor - see Section III-B) is forwarded towards the sink. As far as each packet is assigned a unique ID, and data can be transmitted in the different order as it is stored into the flash and still identified properly at the end point, there is no special mechanism used for up-sampling. Typical settings (see Section V-B) assume that transmission capacity (the number of packets can be transmitted in each round) is at least greater than generation capacity (the number of packets generated in each round). Therefore, if there is connectivity again, then sampling rate is immediately reversed to an initial value (half-/hour interval) and new data takes free space left by the successfully transmitted packets.

The sink does not implement explicit flow control towards the sources because we assume that the UNIX database can cope with all data collected in the sensor field. Temporarily, the UNIX node can do implicit flow control via back pressure by injecting commands (see Section IV-C) and scaling measurement activities (Section IV-B) or simply by refusing any TCP connections from the GPRS node that attempts to deliver data.

B. Zone 2: RAM buffer on intermediate nodes

After taking a measurement or registering some event first it goes into internal buffer allocated in RAM. The buffer is available on all nodes, including the root. The size of this buffer is as big as possible (in our case 100 slots). The buffer serves two main goals:

1) It accepts all locally taken measurements and generated events. After completing measurement operations this part of the content is flushed into flash memory and erased from the buffer.

2) It accepts packets from neighboring nodes in order to send them further towards the root node. These packets are not saved into flash, instead they are assigned an ”age”.

Logically RAM buffer is divided into two sub-buffers: for local and received packets. The space for locally generated packets ($LOCPACK = 10$) is reserved in order to always have some free space even if the buffer is completely full. The rest is dedicated for packets to be forwarded ($MESBUF - LOCPACK = 90$).

Above we have mentioned the idea of ”transmission corridor”. If a node sends a packet towards the sink (root), it must be sure that the next node in line has enough space to store it. In each cycle every node announces how many packets can be forwarded to it from a certain node (credit information) according to the number of free slots in the RAM buffer and the number of neighbors as it is shown in (1). Note that these credits are per hop, and that inside a forwarding corridor we can store many data packets.

$$capacity = \frac{MESBUF - LOCPACK - CurBufSize}{number\ of\ visible\ neighbors} \tag{1}$$

This mechanism is used to prevent overloading nodes with information from its neighbors (also it is used as a criterion for building routing tree - see Section IV-A).

Due to the fact that nodes can get disconnected, or topology can change, each forwarding message is assigned an ”age”. If a message cannot be forwarded further along the tree during N cycles ($N = 10$ in our case), then it is erased from the buffer. At the end of each cycle the buffer is cleaned up by flushing local packets into flash and erasing old forwarding messages.

C. Zone 3: GPRS module flash buffer

For connecting to the database server on Internet we use a GPRS module Siemens TC65 which features 1.7 Mb internal flash memory. In this memory we have two buffers organized: data buffer (for data gathered from the WSN) and acknowledgment/command buffer (see Figure 2 in Section III).

Data retrieved from the network is put into the data buffer and when connection with the UNIX server is established, data is transmitted to the database and the buffer is cleaned. Acknowledgments and commands (see Section IV-C) are encoded on the server side, transmitted during the same session as data to the GPRS module, where they are stored into this buffer. In each network cycle, the next command is fetched and injected into the network, where it starts propagating until it reaches its destination node. Also, GPRS module is responsible for establishing connection with an external NTP server for synchronization purposes.

D. Zone 4: Server-side database

The sink accepts everything. Periodically connection is established by the GPRS module, data is transmitted towards the sink where it is put into database, and encoded acknowledgments/commands/configuration parameters are sent towards the sink. A special scheduler is continuously running on the server side, which maintains per-node seqno tables and network health maps, checks data integrity and generates acknowledgments/commands for nodes.

IV. Flow control

Above we have mentioned already several times data flow control. Let's have a closer look at the means used for providing it in our system: routing, runtime parameters (scale factors) and commands.

A. Multi-hop routing

Routing scheme used in our project is based on the well-known spanning tree approach. A gateway floods the net with reachability info, reverse paths form spanning tree. Forwarding is integrated in the TDMA-like MAC scheme.

The root (the base station) with ID=0 broadcasts routing information in each cycle. Every node positions itself in the spanning tree based on this info. There are two routing-related fields interpreted by a node:

1) **Level.** Current level in the spanning tree: 0 – gateway level, 1 – first level, etc. Each node adds 1 to its parent's level.

2) **Capacity.** The number of messages allowed to be forwarded to a sender of this message; used to limit transmission corridor for each node (see Section III-B).

Each node keeps this information until a better parent is found. If a node has two potential parents at the same level, it will choose the one, which offers a broader transmission corridor.

B. Scale factors

For purposes of fine tuning, control and remote management a set of configuration parameters is used:

– sync interval,

– registering sync info,

– measuring, storing to flash and transmission.

They propagate through the network and interpreted by nodes. Each scale factor is applied to the global cycle counter on a node and form an activity diagram for the corresponding operation (e.g. synchronize every 5 min, register sync info every 10-th cycle, measure/store/transmit data every 2-nd cycle).

The root node, to which a GPRS module is attached, has an extended set of runtime parameters. Extra parameters include:

– sync interval with a NTP server,

– min number of records needed for making a call,

– min number of bytes needed for making a call,

– min number of rounds needed for making a call.

Properly setting shown parameters a one can tune the network for specific conditions and configurations.

C. Commands

Besides configuration parameters our system also supports a number of commands, which can be injected in the network. Commands propagate through the network along with the other parameters, but allow to individually assign some operations to nodes.

Commands can be classified into three major groups: control commands (C), status commands (S) and acknowledgments (A). They are encoded on a front-end software server, transmitted to the base station (GPRS module), then passed to the adjoining wireless node, and from this point are injected in the network. In each round only one command can be injected into the network. If there are a few command pending, they are queued on the base station. Some supported commands are listed in Table V.

TABLE V
SUPPORTED COMMANDS

Command	**Operation**	Class
ACK	Acknowledgment request (acknowledges all data blocks specified seqno and requests for retransmission of max 16 following consecutive blocks	A
PCP	Retransmission request (requests retransmission of 1 block with the specified seqno)	A/C
SSA	Enable/disable taking measurements	C
SLX	Enable/disable data transmission	C
GSL	Generate status messages	S
FRS	Flash chip reset	C

V. Implementation and experimental results

Our system was implemented for the TinyNode ([3], [4]) platform under TinyOS-1.x. So far, we have made two test outdoor deployments (two sensor fields in the Swiss Alps: 11 sensor nodes on Jungfraujoch (3'500 m above sea level) and 16 sensor nodes on Matterhorn (3'300 m above sea level)). Before installing sensor nodes in the field we test the system on the Deployment Support Network (DSN) [9] testbed provided by the TIK group at ETHZ. For the time being, nodes from both sites are taken away for further testing and upgrade.

A. Network topology

Care must be taken when setting up initial configuration parameters for the network. A one must be sure that the size of the internal message buffer (100 in our case) or the number of slots for data exchange is enough to drain the network properly. Nevertheless, for the testbed we had to reduce the buffer's size down to 80 items otherwise the image would not fit the memory scopes.

Most MAC-layer parameters highly depend on the topology expected. As far as we cannot not define it before (deployment plan is usually set up right in the field) we always have to consider the worst case. Two boundary cases are "all-to-one" and "all-in-line". As long as it is hard to build such topologies in lab conditions we tried to build as complex one as possible. The topology used in our tests is shown in Figure 4.

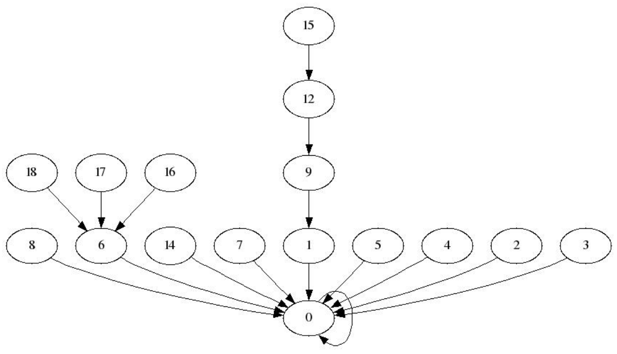

Fig. 4. Testbed topology

B. Choosing configuration parameters

In Section V-A we referred to the point that data transport related parameters must be chosen carefully beforehand. There are three main parameters which must be specified: maximum number of nodes, the number of data slots for each node and internal buffer capacity.

Maximum number of data slots. This parameter affects the slot distribution on the TDMA scale. Our installations typically consist of 15-20 nodes. So the setting for this is $N = 20$.

The number of data slots for each node. It specifies how many packets each node is capable to transmit during its own slot or to accept from each of its children. The following ratio must be met:

$$\sum_{i}^{i \to j} N_i + N_j \leq C_j$$

Where C_j - internal buffer capacity of node j, N_j - number of packets generated by node j, N_i - number of packets generated by node i which is a child of j. For each node we have $N_i = N_{data} + \delta_e$, where N_{data} is the number of data packets periodically generated by the node and δ_e is the number of extra non-periodical system events (e.g. reboot event). Therefore, in order to drain out the network properly each node must support at least up to $(N_{data} + \delta_e) * N$ slots.

Internal buffer capacity. The ratio stated above for the number of data slots also includes the internal buffer capacity variable. It shows why we chose value 100 for our tests. For the root node it is not a strong limitation because in each round data is transferred to the GPRS module which also has an internal flash storage of 1.7 Mb.

C. Performance

We were running the test for 25 hours which is enough to reach a stable state (synchronization may take about 20 rounds - see [5]). Obviously, some data packets can be lost at the beginning when nodes are not perfectly synchronized yet and radio collisions occur. The delivery rate for the whole run is presented in Table VI.

TABLE VI
DELIVERY RATE

Node ID	Received	Lost	SeqNo range
17	454	38	0 – 475
4	539	3	0 – 490
18	499	6	0 – 483
5	535	5	0 – 493
6	65	136	0 – 200
7	523	3	0 – 470
8	406	3	0 – 359
9	346	0	0 – 286
12	114	67	0 – 177
14	537	2	0 – 488
1	534	0	0 – 483
15	481	5	0 – 480
2	539	3	0 – 490
16	459	8	0 – 450
3	539	0	0 – 489

From the analysis of this data we discovered a big loss rate for node 6 was caused by the fact of applying not very optimal mechanism for calculating the announced capacity. In our routing scheme all visible neighbors are taken into account for computing a transmission corridor for each node as far as synchronization and building a routing tree are done at the same time. Instead only children nodes must be taken into account. This improvement is still pending. For other nodes losses are caused by either initial synchronization phase characterized by a big number of collisions or by bad links. Nevertheless, any lost packet can be retransmitted later using our embedded acknowledgment scheme.

As far as the topology in Figure 4 shows only the very last established connections, the best way to monitor how topology was changing over time (due to bad links or finding better routing paths) is to look at the spanning tree level graph presented in Figure 5 [4].

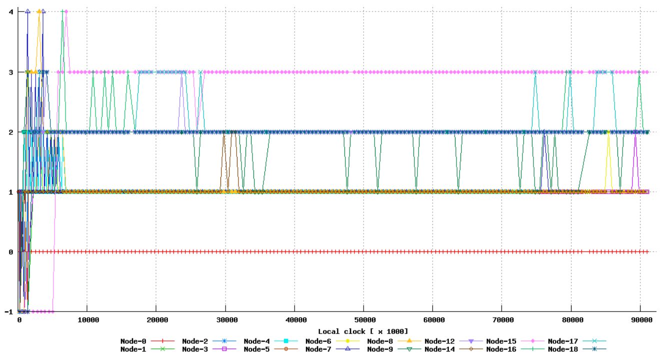

Fig. 5. Spanning tree level vs Local clock

As it was mentioned in Section III-B each node has the RAM buffer for messages to be forwarded. The distribution of the message buffer's capacity is shown in Figure 6.

[4]"-1" along the level axis in Figure 5 means no level assigned yet.

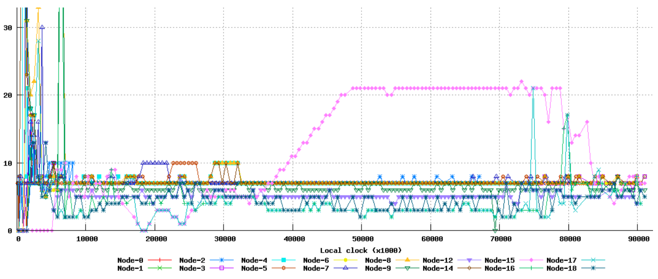

Fig. 6. Capacity of the message buffer vs Local clock

These tests showed that for low-populated networks (up to 20 nodes) the system behaves as expected.

D. Power consumption considerations

As it was stated before, our protocol is specifically designed for very power-critical applications which must run for at least one year without exchanging power elements. Instead of making theoretical estimations of energy consumption we made full power profiling of the node using tools provided by the DSN testbed. We investigated the typical configuration used in our outdoors installations: 20 nodes, 256 seconds sync interval, 60 minutes sensing interval. The average power consumption is 0.124 mA. Although for this profiling a fixed power supply was used, we made estimations for lithium batteries with capacity of 8.2 Ah used on our nodes (one battery per node). Therefore the lifetime can be up to 7.5 years. Taking into account temperature and degradation factors we can expect at least 5 years normal operation.

VI. Related work

Recently, a number of data gathering techniques were proposed for using in environmental monitoring which are aimed at the periodic data collection and ultra-low power consumption. The most interesting solution is called Dozer [10] and is used in the WaterSense [11] project. Dozer is a data gathering framework incorporating a MAC-layer, a routing scheme and a limited set of network control abilities. In contrast to our solution where almost all runtime parameters can be tuned and the data flow can be controlled in each part in run time, Dozer does not provide such level of flexibility. Moreover, using Dozer it is impossible to finely scale measuring/transmitting phases on a per-node basis. Our system natively supports this mechanism.

Also a set of protocols and techniques describing different aspects of reliable data transport were proposed in the past. For example, [12] gives a comprehensive description of the data transport methodology including TDMA framework, routing scheme, congestion detection, etc. Still the domain of data recovery strategies is not fully covered.

Another interesting solution is used in the SensorScope project [13]. It is a communication stack which includes ACKs, multi-hopping and a synchronized duty-cycling MAC layer. The biggest drawback with it could be that it is a fully parameterized stack containing a big number of configuration parameters which you have to set up at compilation time.

VII. Future work

A number of test cases of our system are upcoming. These test cases mainly concern system stability and performance over the network topology with multiple partitions and a large scale network structure (up to 50 nodes).

As it was stated in Section V-C some optimization steps to the routing scheme are needed in order to increase delivery rate further.

VIII. Conclusions

Mainly, all research works carried out so far in the community had an aim to fill in one specific niche: time synchronization, routing, MAC-layer, etc. Any wireless sensor network application dictates its own requirements. That is why building a full framework for data gathering is still a big challenge. Unfortunately, common solutions usually cannot be applied for those particular needs. Basically there are two ways to follow: either develop you own software from scratch, or try to adapt some existing solution. From the description given above a one can understand that our system also has a limited number of application domains it can be applied for: little-scale (up to 50 nodes), non-mobile, relatively low data traffic wireless sensor networks.

Unlike existing data gathering protocols we propose a highly flexible and configurable scheme specially design for intermittently connected wireless networks with high probability of data losses and the lack of connectivity. As far as measurements are taken very rarely (half-/hourly) and data stream must be consistent for successful post-analysis, the scheme supports an acknowledgment mechanism which allows to retransmit any piece of data if it has not been delivered to the database side yet. The scheme works despite intermittent connectivity and has been implemented for our permafrost monitoring project.

Acknowledgments

We would like to thank the members of the TIK group at the ETHZ for all of their software testing efforts, and especially Jan Beutel, Mustafa Yücel and Roman Lim for making the second generation of PermaSense feasible.

References

[1] I. Talzi, A. Hasler, S. Gruber, and C. Tschudin, “PermaSense: Investigating Permafrost with a WSN in the Swiss Alps,” Basel-Zürich, Switzerland, 2007.

[2] *PermaSense Project*, University of Basel, http://cn.cs.unibas.ch/projects/permasense/.

[3] *TinyNode Wireless Sensor Modules*, Shockfish SA, http://www.tinynode.com/.

[4] H. Dubois-Ferriere, R. Meier, L. Fabre, and P. Metrailler, “TinyNode: A Comprehensive Platform for Wireless Sensor Network Applications,” Lausanne, Switzerland, 2006.

[5] I. Talzi, S. Schönborn, and C. Tschudin, “Power-Aware Synchronization in Intermittently Connected Wireless Sensor Networks,” Basel, Switzerland, 2007.

[6] *DS2401 Silicon Serial Number Chip*, Dallas Semiconductor, http://www.maxim-ic.com/.

[7] S. Santini and K. Römer, “An Adaptive Strategy for Quality-Based Data Reduction in Wireless Sensor Networks,” Institute for Pervasive Computing, ETH Zürich, 2006.

[8] E.-O. Blass, L. Tiede, and M. Zitterbart, “An Energy-Efficient and Reliable Mechanism for Data Transport in Wireless Sensor Networks,” University of Karlsruhe, Germany, 2006.

[9] *Deployment Support Network Project*, University of Zürich, http://www.btnode.ethz.ch/Projects/Jaws.

[10] N. Burri, P. von Rickenbach, and R. Wattenhofer, “Dozer: Ultra-Low Power Data Gathering in Sensor Networks,” Computer Engineering and Networks Laboratory, ETH Zürich.

[11] *WaterSense Project*, EPFL, http://www.nccr-mics.ch/micsCluster.php?groupName=CL3&action=projects.

[12] V. Turau and C. Weyer, “Long-Term Reliable Data Gathering Using Wireless Sensor Networks,” Hamburg University of Technology, Germany, 2007.

[13] *SensorScope Project*, EPFL, http://sensorscope.epfl.ch/.