Diagnosing the Weakest Link in WSN Testbeds: A Reliability and Cost Analysis of the USB Backchannel

Pablo E. Guerrero, Iliya Gurov, Alejandro Buchmann
Databases and Distributed Systems Group
Technische Universität Darmstadt
Darmstadt, Germany
{guerrero, gurov, buchmann}@dvs.tu-darmstadt.de

Kristof Van Laerhoven
Embedded Sensing Systems
Technische Universität Darmstadt
Darmstadt, Germany
kristof@ess.tu-darmstadt.de

***Abstract*—This paper highlights and characterizes the main obstacle to deploying a robust wireless sensor network testbed: the USB connections that link each of the nodes via ethernet gateways to the central server. Unfortunately, these connections are also the components that, when properly installed, can reduce testbed costs by attaching multiple nodes per gateway. After illustrating how unreliable current solutions can become (regardless of the used sensor nodes, USB cabling, or gateway setup), a set of experiments led to a list of dos and don'ts in testbed deployment. Furthermore, a simple and cost-effective suggestion is presented that allows to bypass current USB backchannel issues, leading to a more robust testbed that avoids manual maintenance of individual nodes.**

***Index Terms*—experimentation; testbeds; testbed reliability; sensor node reprogramming; universal serial bus**

I. Introduction

Testbeds are a valuable research tool as they both facilitate and speed up the evaluation of sensor network software. Installing and running larger testbeds quickly becomes an expensive and time-consuming endeavor, with costs originating primarily from a) the initial hardware acquisition and installation, and b) the day-to-day testbed maintenance. Interconnecting the sensor nodes under evaluation with a central gateway through a Universal Serial Bus (USB) infrastructure has become the method of choice since USB can provide power to the nodes, be used to reprogram nodes, and act as data-logging backchannel. This is in contrast to other experimental approaches of lower efficacy, including reprogramming nodes over the air (e.g., Deluge [8]) or resorting to a wireless backchannel (e.g., [10], [3]).

However, the design and installation of this sort of USB infrastructures is often an underestimated task with pitfalls that can cause the testbed to become highly unreliable and costly to maintain. First, implementations of the current USB protocols involved at hardware and software level in current WSN components and prototypes are not bug-free: although a testbed health monitoring system or a testbed engineer could troubleshoot these issues, frequent manual intervention to restart and reconnect sensor nodes is required on-site. In an unattended setup, this increases the maintenance costs and additionally reduces the testbed nodes' availability.

Secondly, this issue is exacerbated with larger USB topologies, where cabling quickly reaches longer lengths and contains hubs that fan out to many nodes. Despite employing USB topologies and parameters within the USB specification, even high quality off-the-shelf USB components do not play well with these rather extreme setups, exhibiting considerable instability for power and data lines and thus causing nodes to become unreachable. Although the unreliability of the gateway-nodes USB backchannel is well-known in the WSN testbed community [7] and bypassed by manually resetting individual nodes, the literature falls short in describing this problem in detail.

This paper's contributions are threefold: First, typical USB backchannel failures are reported as observed in TUDμNet [6], a 46-node testbed based on MoteLab [15], monitored over an extended in-use period. Second, a systematic evaluation of performance and reliability is given with varying amounts of nodes per gateway, different types of gateways and nodes, and different cabling topologies, leading to a list of best practices in installing the USB backchannel. Third, a mechanism to increase the backchannel reliability is presented based on port power switching of USB 2.0 hubs. We describe the implementation of the necessary steps to exploit such power control and quantify its performance and added-value.

Section II provides an overview of testbed architectures focusing on the design space of USB backchannels. Section III describes USB failures and characterizes their frequency in TUDμNet. Section IV explains the evaluation methodology and reports the results of a number of USB topologies. In Section V we discuss the usage of power switching to improve backchannel reliability. We conclude in Section VI.

II. Background

Sensor network testbeds typically have an architecture that is essentially a tiered model (Fig. 1). Testbed users create the software under test (SUT, or program image) to evaluate and upload it to a server. The server in turn offers a web interface

978-1-4673-2129-7/12/$31.00 ©2012 IEEE

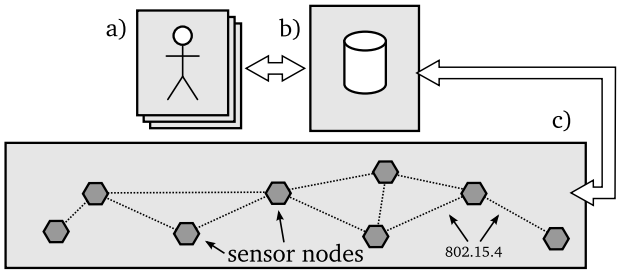


Fig. 1: Generic testbed model: users supply software (a) to a server (b), which forwards these to the right nodes at the right time, and captures debug messages via the backchannel (c).

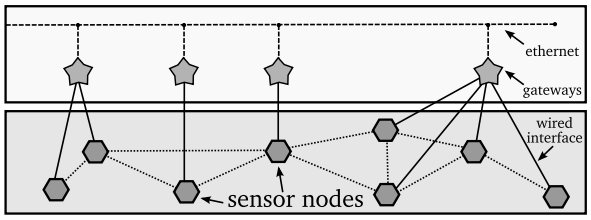


Fig. 2: Example of a wired testbed with gateway (upper) tier connected via a wired backchannel to one or more WSN nodes.

or a set of scripts to define the experiment's properties (amount of nodes required, location) and scheduling information (time, duration and order).

At the experiment's start time, the server allocates the necessary resources and distributes the images to the respective sensor nodes via the so-called backchannel (Fig. 1.c). During the experiment, this communication line is also typically used to send the captured debug messages to the server for any posterior SUT debugging and analysis.

Depending on how the sensor nodes are interconnected to the server, we can classify testbeds into *wireless* or *wired*:

1) Wireless Testbeds: These resort to the wireless channel to distribute software images and to transmit debug messages. Deluge [8], for instance, is a mechanism to distribute TinyOS programs over the air, which can be used for this purpose. Reported image dissemination performance places a considerable overhead on a testbed that must be shared with many users (for instance, 90 bytes/second implies >5 minutes to distribute a 30kB image). In addition, Deluge's service messages themselves might interfere with the SUT's network traffic, making it tough to debug networking issues.

The approach employed by Greenlab [10] consists in splitting the testbed's state in two: one for doing service tasks (e.g., reprogramming) and one for running experiments. Across states, data is stored on external flash. However, using the wireless channel to convey all control messages is reportedly not robust to node failures, blocking the entire testbed for further experiments until a reconnection occurs.

A third wireless approach is to use an additional tier between sensor nodes and the server. An example of this approach is a *deployment support network* [3], which consists of sensor nodes that carry out the management tasks and connect each to one sensor node through a wired interface. Although running the support tier out-of-band effectively eliminates disturbing the experiment results, resource constraints of the support nodes limit the information that can be transported to the server, which requires modifying the SUT to produce a reduced amount of debug messages.

2) Wired Testbeds: For obtaining an architecture that is both faster and more robust for the task at hand, using a *wired* (e.g., ethernet) infrastructure between the server and the support tier is traditionally favored, resorting to more powerful devices as gateways (Fig. 2). The benefits of the wired backchannel come at the cost of running cables through the environment, although these costs are often amortized over testbed deployment time. Several platforms have been used in testbeds as gateways, including an ethernet port and a serial port to connect to sensor nodes. The most widely used of these serial connections between gateway and nodes allows routing power and a high-rate data flexibly to several sensor nodes via hubs and cables, and will be the focus in the remainder of this paper.

A. USB as Node Interface

All sensor nodes (except perhaps for final products) require a programming and debugging interface. Early platforms like the Mica2 made use of a specialized programming board [11], to which they attached via a 51-pin connector. This connector would typically wear out after a number of reconnections. JTAG is another interface broadly adopted for device reprogramming and debugging (e.g., the EyesIFX node). To the best of our knowledge, USB was first used to interface the widely available Telos sensor node [12].

Table I summarizes various sensor nodes that employ USB to connect to a host computer. Some sensor nodes' microcontroller units (MCUs), such as the Telos' MSP430, export UART pins for reprogramming and debugging, thus require a USB converter chipset (which can be either *on-board* or on a *separate* device). Other nodes, such as the jNode [13] have MCUs with USB support already *built-in*. In this work we focus the evaluation on a number of nodes based on the very popular MSP430 MCU, with a wide variety of USB chipsets; an exhaustive evaluation of additional WSN node types is deemed out of the scope and is left as future work.

B. USB Topologies, Hubs, and Cabling

A USB topology connects sensor nodes with a gateway. Physically, the USB forms a layered star topology (Fig. 3),

TABLE I: Various sensor nodes with USB interfaces.

sensor node	USB type	USB chipset / MCU
Telos	on-board	FTDI FT232BM
Econotag	on-board	FTDI FT2232HL
jCreate	separated	FTDI FT232RQ
Z1	on-board	Silicon Labs CP 2102
iMote2	built-in	Intel PXA271
SunSPOT	built-in	Atmel AT91RM9200
Egs, Opal	built-in	Atmel SAM3U
jNode	built-in	Atmel ATmega32u4

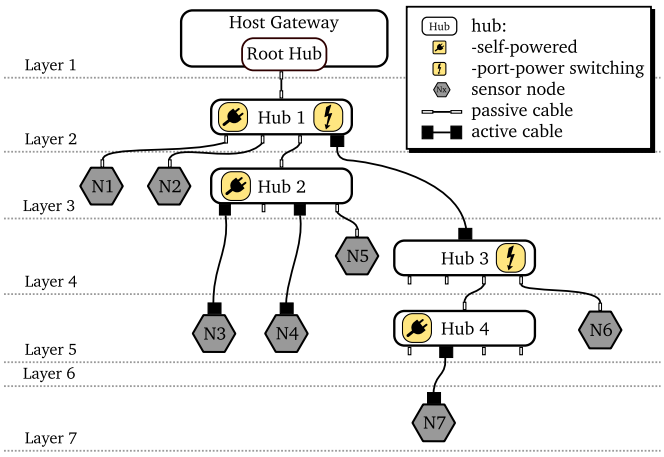


Fig. 3: Key components of a USB topology in several layers: root hub, active and passive hubs and cables, and nodes.

or tree, with hubs at the center of each star, and the *root hub* typically embedded in the host gateway device. Hubs can be passive (bus-powered) or active (self-powered). Due to timing constraints, up to 7 layers are allowed. Nodes and hubs connect to their parent hub via point-to-point USB cables. These cables can be passive (limited by power and timing constraints to a length of 5 meters) or active (extend the length to 10 or 12 meters by using signal repeaters and specialized circuitry). By chaining a sequence of up to 5 passive cables and active USB hubs, the distance can be extended to 30 meters and still conform to the USB standard. As we will show, this is the most robust topology, and comes at the expense of extra power lines for each active hub.

C. Gateways

The literature reports various host platforms being used as gateways. Due to its low-cost hardware, the NSLU2 was adopted early on as gateway for Telos nodes (e.g., [15], [7]). Intel Stargates, featuring a broader set of ports (PCMCIA, CompactFlash, I^2C, etc.) and a faster processor, were used in [4]. Similarly powerful, a number of routers with USB ports, like Buffalo's, have been chosen as gateways and customized with slimmed-down Linux distributions such as OpenWRT, e.g. in [6]. All these platforms require some major involvement in setting up the gateway software due to their incompatibility with x86 architectures. Finally, some testbeds employ general purpose PCs as gateways as well (e.g., in [1] and [2]). Such added flexibility in the testbed software preparation comes at a higher price per gateway. Table II summarizes these, ordered by increasing processing power.

D. Gateway-Node Ratio and Scale

In some testbeds, only one sensor node is connected to each gateway, either directly (e.g., Kansei [4]) or with a very short cable (e.g., MoteLab [15]), thus a large number of gateways is needed. By employing USB cabling between gateways and sensor nodes, the number of design options increases considerably: In TWIST [7], USB hubs are used, enabling up to 7 sensor nodes to be connected to each gateway while achieving a similar amount of nodes in the testbed. There, also a combination of passive and active USB cables is used to extend the distance between gateways and sensor nodes up to 15 meters. Indriya [2] resorts to *high quality* active USB cables which can be daisy-chained to cover a maximum distance of up to 25 meters. This enables covering longer distances with very few gateways. In SignetLab [1], 48 nodes are connected through a two-level USB hub hierarchy to a single gateway (a PC). We summarize these properties in Table III[1].

TABLE II: Common gateway options: the Linksys NSLU2 (Slug), the Buffalo WZR-HP-AG300NH (Buffalo), and a PC.

Platform	CPU type, speed	RAM / ROM	USB	Price ($)
Slug	Intel IXP42x, 266 MHz	8MB / 32MB	2x2.0	90
Buffalo	Atheros AR9132, 400MHz	32MB / 64MB	1x2.0	100
PC	Intel Dual Core, 2.5GHz	1GB / 80GB	4x2.0	500

TABLE III: Some well known WSN testbeds with gateway (gw.)-node ratio comparison for their USB backchannels.

Testbed	No. gws.	No. nodes	Ratio gw:nodes	Distance gw.↔node (m)
Kansei	210	210	1:1	0
Motelab	90	130	1:{1..2}	0.5
TUDμNet	15	46	1:{2..6}	{2..15}
KanseiGenie	112	432	1:4	0.5
TWIST	90	204	1:{4..7}	<15
NetEye	15	130	1:{6..12}	<10
Indriya	6	127	1:22	<25
SignetLab	1	48	1:48	<15

The choice of gateway-node ratio is a precarious one: From a costs perspective, more nodes connected per gateway implies a lower setup investment and lower gateway maintenance efforts for the duration the testbed will be deployed and active. However, reducing the number of gateways while covering the same area requires more USB cabling, with choices of topology and, as we will see, risk of failures rapidly increasing, as more complex USB infrastructure between gateway and nodes is deployed. Ultimately, up to 127 devices (including nodes and hubs) can be connected to a single USB host according to the USB standard [14], yet gateways will eventually require too much processing power as well as storage capacity to cope with the management of the attached devices. Furthermore, power variations and transmission timing errors are likely to become a major obstacle to increasing the ratio to such high scale.

III. USB Backchannel Issues

Testbed USB backchannels are exposed to several types of failures, which is why there is no guarantee that a topology will work reliably, even when adhering well within the specifications. Variations in the input power of a sensor node (or its USB converter chip) can cause clock synchronization failures. Software glitches in the USB stack and the bootstrap loader, and increased bit error rates can lead to an inconsistent

[1]Published details were not always specific; in this case, the respective authors were consulted and figures adjusted, so table data might be different.

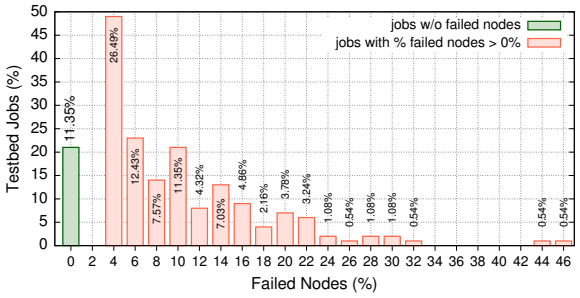
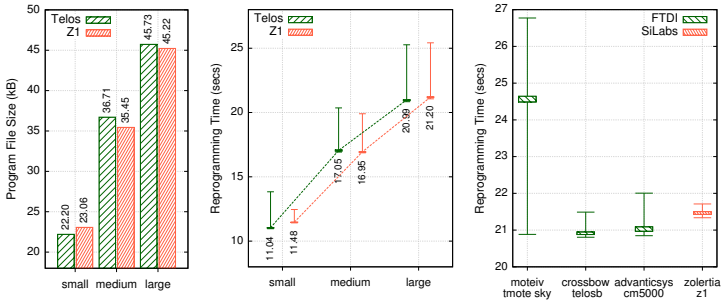


Fig. 4: Histogram characterizing USB reprogramming failures in a WSN testbed over 10 months: 11.35% of the testbed jobs deployed to all nodes; for the others, between 4% and 46% of nodes were not reprogrammed.



(a) program file size (b) reprogramming time (c) reprogramming time for various MSP430 platforms

Fig. 5: The effect of the node manufacturer on reprogramming time for a single-node: with differently-sized programs, several platforms can be seen to have significant differences in programming time.

protocol state. These can render a node non-programmable and non-addressable after the error occurs.

Fig. 4 presents a histogram with the percentage of failed nodes of a subset of TUDμNet [6] testbed SUT jobs since its first deployment 10 months ago. This set only contains jobs addressing more than 10 nodes, with other types of failures omitted (spanning around 200 jobs). This shows that only in 11.35% of the jobs no nodes failed, while in all other jobs at least one node did. It is important to note that the referred USB topologies conformed to the USB standard, and that none of these faults were due to faulty nodes, nor occurred at one specific position within a topology. *These node failures occurred in patterns that are hard to track down or reproduce*.

Such nodes resemble the so called fail-stop behavior[2]: nodes stop being reprogrammable, identifiable through an error message, and remain in this state until being serviced. Two types of solutions exist: A purist approach including tracing and debugging the entire software (including USB drivers, OS kernel, and serial bootstrap loader) and hardware stack proved to be very tough in prototype environments. Alternatively, methods to reconnecting the node to the USB port can be explored, removing power from the node and its USB converter, and reenabling it. Without servicing failed nodes, the number of available (i.e., reprogrammable) nodes in the testbed drops *monotonically*. In a permanent, unattended testbed, non-programmable nodes that require manual intervention imply bad experiment repeatability.

Note that technically, the USB 2.0 bandwidth is sufficient for the maximum possible amount of nodes (126 nodes at 38,400bps represent $\sim$1% of 480Mbps). The following section will characterize the failures for different topologies.

IV. Evaluation

In order to evaluate the different backchannel scenarios, we proceeded with the same basic set of steps: a) physical preparation of the topology (connection of hubs, cables and/or nodes), b) verification of power on all nodes, c) verification of correct, stable node enumeration (i.e., registration) at the host gateway, d) execution of microbenchmark. The microbenchmark software has minimal impact on the measured results, since it simply consists of repeatedly reprogramming the node(s), which was done with the default bootstrap loaders provided by Contiki and TinyOS for the respective sensor node and host platform combination. The implementation was written in Perl and exploits parallel processes for reprogramming the nodes as necessary. This approach allowed us to identify topology-related (i.e., spatial) issues. Although tests lasted from several hours to a couple of days, we could only capture a fraction of the temporal issues that can emerge on a long-term, permanent testbed as presented before.

Across our tests we resorted to binary images of three sizes. A `hello world` program represented our smallest image. Scopes [9], a sensornet node grouping system, was used as mid-size image. Finally, we use the binary image from the ukuFlow [5] workflow engine as our largest test program. The program sizes for two popular platforms, the Telos and Z1 WSN nodes, are summarized in Fig. 5a.

A. Gateway to Single Node Tests

We evaluated more than 50 cases with single-node topologies in total. For this purpose we elaborated a simple microbenchmark which consisted of sequentially reprogramming the tested node. This sequence was repeated until the node failed, or reached 1,000 iterations. In case of a failure, the procedure was restarted, either by automatically rebooting the gateway when the device's root hub turns off attached devices, or otherwise by manually reconnecting the node. This was repeated 25 times to ensure statistical validity.

Fig. 5b presents the reprogramming time for both Telos and Z1 nodes with the three file sizes described earlier, when nodes were connected directly to a gateway. As expected, the larger the file size, the longer the average reprogramming cycle was (averages are connected by dotted lines). The error bars show an outlier, which is normally the first iteration, where the bootstrap loader and the program image files must be loaded into memory. Interestingly, no major differences were

[2]This is in contrast to fail-silent failures, where the device would provide no hints that it has become non-programmable.

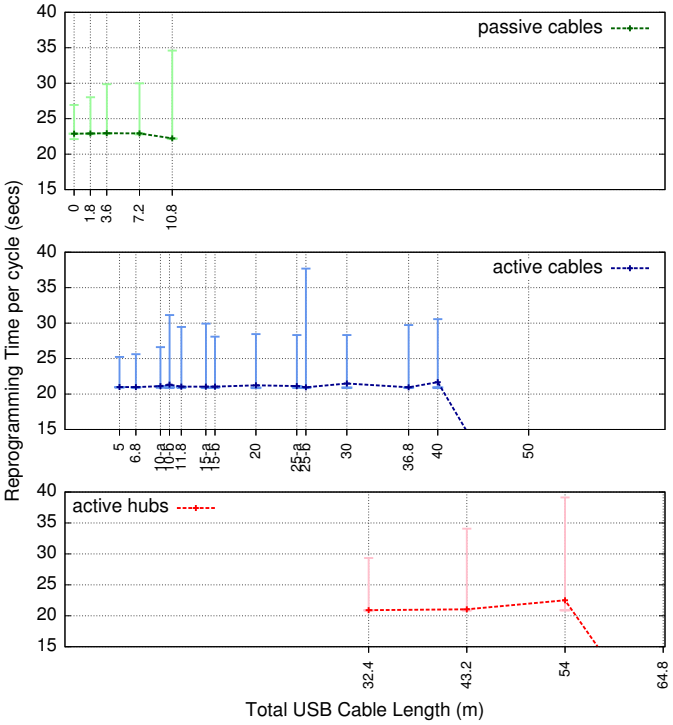


Fig. 6: Reprogramming time vs. total cable length.

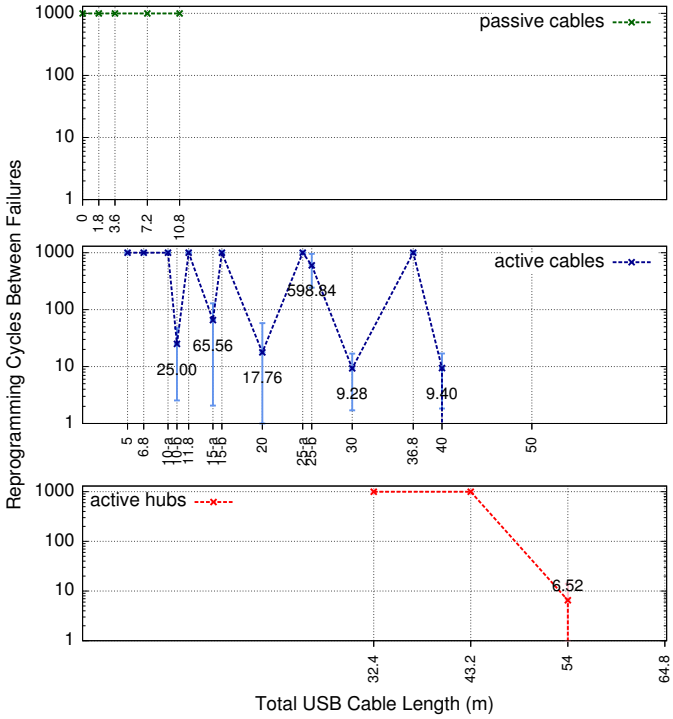


Fig. 7: RCBF vs. total cable length.

noticeable between the two node types, although they used a different USB converter chipset (FTDI versus SiLabs). A closer look at different manufacturers of these nodes revealed similar reprogramming times (cf. Fig. 5c), except for moteiv's Tmote Sky nodes, which took longer and had a higher variability. We believe this could be due to these nodes belonging to some of the very early manufactured revisions. In terms of reliability, no differences could be observed across these sensor nodes (not depicted).

How far can we put the gateway and node apart? For many environments, long connections between a gateway and a node are advantageous. Since the USB standard dictates a maximum cable length of 5 meters, we evaluated various topologies with different cables and hubs for reprogramming performance and reliability (Figs. 6 and 7). The components used in these topologies, their order, and the resulting number of USB layers, are listed in Table IV.

By chaining standard 1.8 meter *passive cables*, it was possible to power a node located up to 10.8 meters. Though this is surprisingly well beyond the USB specification, nodes were correctly enumerated and worked reliably. As expected, the time performance variance grew with the total length, with the average reprogramming time dropping slightly with length. All nodes used in these topologies were correctly recognized and reprogrammed for the full 1,000 iterations. From 12.6 meters onwards, nodes were not enumerated anymore.

Conclusion 1: standard passive cables will work to cross a distance from gateway to a node of up to 10 meters.

When resorting to *active cables*, the 10 meter limit was overcome using various 5 and 10-meter cables of this kind.

TABLE IV: Some of the single-node topologies tested to reach a certain length between gateway and node. (p.c. = passive cable; a.c. = active cable; a.h. = active hub)

length (m)	components	USB layers
0.0	direct	2
1.8	1 x 1.8m p.c.	2
3.6	2 x 1.8m p.c.	2
7.2	4 x 1.8m p.c.	2
10.8	6 x 1.8m p.c.	2
5.0	1 x 5m a.c.	3
6.8	1 x 5m a.c. + 1 x 1.8m p.c.	3
10-a	2 x 5m a.c.	4
10-b	1 x 10m a.c.	3
11.8	1 x 10m a.c. + 1 x 1.8m p.c.	3
15-a	1 x 5m a.c. + 1 x 10m a.c.	4
15-b	1 x 10m a.c. + 1 x 5m a.c.	4
20	2 x 10m a.c.	4
25-a	2 x 10m a.c. + 1 x 5m a.c.	5
25-b	5 x 5m a.c.	7
30	3 x 10m a.c.	5
36.8	3 x 10m a.c. + 1 x 5m a.c. + + 1 x 1.8m p.c.	6
40	4 x 10m a.c.	6
50	5 x 10m a.c.	7
32.4	3 x 6 x 1.8m p.c., 5 x a.h.	7
43.2	4 x 6 x 1.8m p.c., 5 x a.h.	7
54.0	5 x 6 x 1.8m p.c., 5 x a.h.	7
64.8	6 x 6 x 1.8m p.c., 5 x a.h.	7

Since these work internally as a hub, technically up to 5 of these can be chained, thus potentially reaching 50 meters with active cables plus a last passive segment of 10.8 meters. With these components, however, correct enumeration was

found to be limited to a maximum of 50 meters. Average reprogramming time and variance grew with cable length, with the reliability decreasing considerably. With three 10-meter cables, for instance, we observed an average of 9.28 reprogramming cycles between failures (RCBF). Remarkable was also that having a 10-meter cable as the last segment always led to poor reliability.

Conclusion 2: active cables extend the distance to the gateway, at the cost of decreasing reliability, to 40 meters.

By employing *active hubs* and passive cables, the length was stretched further. Inter-hub lengths of 5.4, 7.2, 9.0 and 10.8 meters were tried, for a total of 32.4, 43.2, 54 and 64.8 meters, respectively. The longest length achieved with a reliable behavior was 43.2 meters. At 64.8 meters, nodes could be correctly powered and enumerated, but not reprogrammed. All other topologies were either extremely unreliable, or nodes were enumerated but could not be reprogrammed.

Conclusion 3: active hubs allow extending the distance to the gateway, at the cost of routing power to the hubs, to 43 meters.

B. Gateway to Multiple Node Tests

When connecting multiple nodes to a gateway, reliability can be expected to drop as the USB backchannel's topology becomes more complex. This was already noticed in steps b (verifying power) and c (enumeration) of the evaluation methodology. When having more than 64 nodes, and thus more than 3 layers of 4-port USB hubs, enumeration became very unreliable. This was due to sections of the tree not being powered in a stable manner. We believe that this is an issue in the USB handshake protocols. We managed to power 64 nodes and have them registered with the OS, though this required some effort since at this scale, the topology became very sensitive to cable quality. Fig. 8 presents the multi-node topologies that worked reliably (summarized in Table V).

The microbenchmark for multiple-node topologies was parameterized to support several concurrent processes. Fig. 9 exemplifies two instances of its execution. In Fig. 9a, the amount of nodes equals the amount of processes (n=p=4). At the third inner iteration, nodes 2 and 4 fail to reprogram; the others continue. Once all nodes fail, the system is reinitialized and the whole procedure is repeated (25 times). In Fig. 9b there are more nodes than processes (n=8, p=3), therefore, at least three rounds are necessary in each inner iteration. We observed that the fewer nodes a round had, the shorter the reprogramming time was.

TABLE V: Multi-node topologies in detail.

grid	nodes	area (m^2)	density (n/m^2)	USB layers	USB hubs
a) 3x3	9	12.96	0.69	5	3
b) 5x3	15	25.92	0.57	7	5
c) 4x6	24	48.60	0.49	6	8
d) 6x6	36	103.68	0.34	5	21
e) 7x7	49	147.91	0.33	5	21
f) 8x8	64	147.91	0.43	5	21

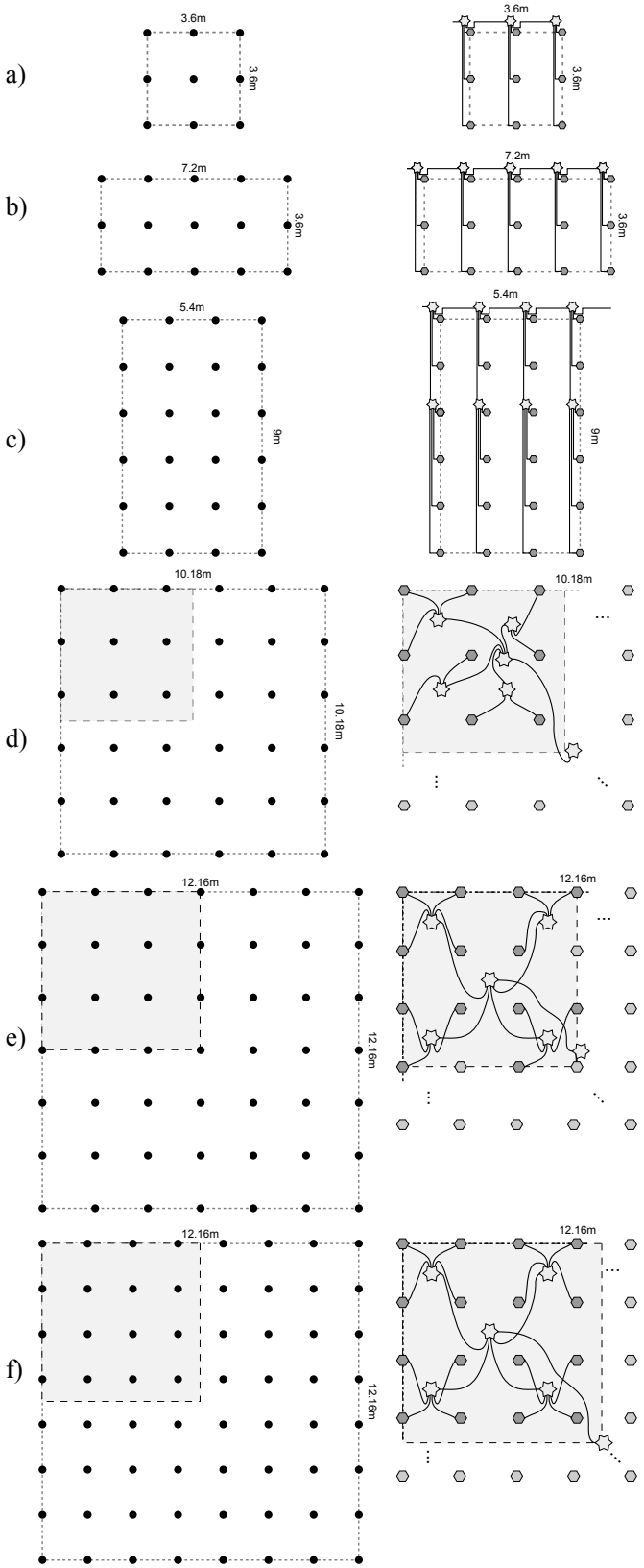


Fig. 8: Sensor node grid deployments and underlying USB topologies (details in Table V).

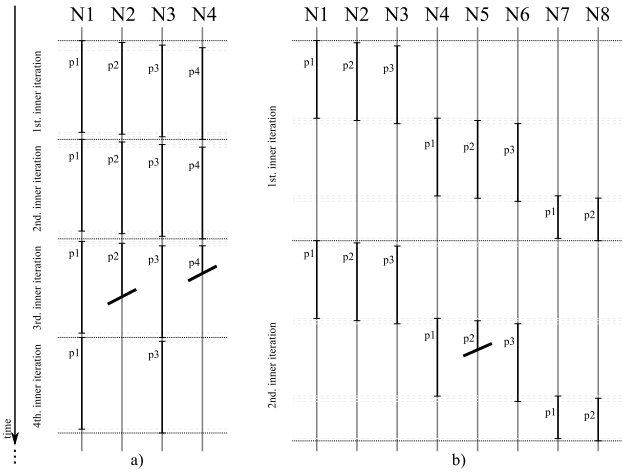


Fig. 9: Microbenchmark sample instances.

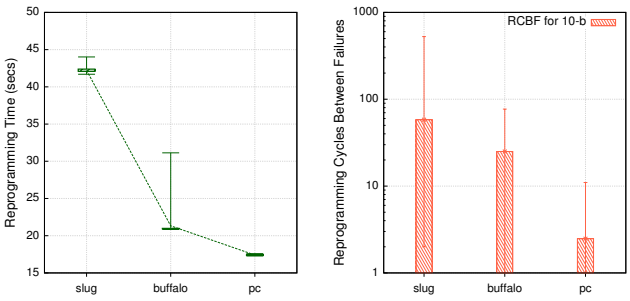


Fig. 10: Comparison of gateway platforms of Table II.

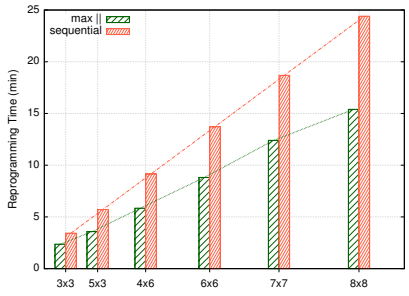
What difference does the host gateway make? The selection of the gateway platform plays a major role in the overall testbed costs. We compared the reprogramming performance of the three host platforms of Table II. The left plot in Fig. 10 shows that faster gateways also exhibited faster average reprogramming cycles. This suggests that the more nodes a topology has, the better suited a more powerful gateway is. This assumes that the topology is reasonably designed: the scenario with only the 10-meter cable shows that the reliability decreases with more powerful gateways (Fig. 10, right barchart).

Conclusion 4: The choice of gateway platform should link to the speed at which all its nodes need to be programmed.

Sequential or parallel reprogramming? A topic that arises when reprogramming multiple nodes from a single gateway is that this should ideally be done in parallel since this might save time, compared to a sequential reprogramming. The limited resources of single-board computer gateways, however, constrain the degree of parallelism. The table in Fig. 11a presents our findings on the maximum degree of parallelism (row called *max* $\|^\circ$) of each gateway; reprogramming more nodes caused the host platform to hang. (Note that a PC could probably reprogram more than 59 nodes, but this was a limit in our test topologies due to power and enumeration.) The bottom part of the table indicates in how many rounds a

		Slug	Buffalo	PC
max $\|^\circ$		5	8	59
rounds for # nodes	**4**	1	1	1
	12	3	2	1
	24	5	3	1
	48	10	6	1
	64	13	8	1|2

(a) parallel capacity



(b) total time

Fig. 11: Possibilities for parallelism.

topology of a given amount of nodes can be divided in order to exploit parallelism. Evidently, the slug will require many rounds to reprogram large topologies, while a PC could do it in one or two rounds. Fig. 11b presents the observed average time it took to the Buffalo gateway to reprogram once all of the nodes in each of the topologies of Fig. 8, both with the maximum degree of parallelism (bottom) and sequentially (top). The diverging curves show that parallelism should be preferred. From the reliability perspective, it was not relevant how nodes were reprogrammed.

Conclusion 5: Topologies with many nodes should exploit parallelism to reduce the reprogramming overhead time.

What difference do USB hubs make? We have in these experiments inspected a total of 10 4- and 7-port USB hubs, bus- and self-powered. In our experiments, no noticeable effects were obtained, neither in terms of reprogramming time, nor in reliability. The next section indicates, however, which hubs are to be preferred for a testbed.

V. USB Power Control to Enhance Reliability

Manually reconnecting nodes to the USB cabling, in order to remove power temporarily and cause a hard reboot effect, is a costly solution to the backchannel problems. By using a feature of USB 2.0 hubs, namely hub port power control (HPPC) [14], it is possible to achieve the same effect, but without requiring manual intervention. Power control was first used in TWIST [7] to emulate node deaths. Here we resort to it in order to increase the reliability of the testbed. Next we describe the procedure to exploit this functionality.

A. Exploiting Power Control

The power control procedure begins by constructing a tree reflecting the attached USB topology (as for example the one shown in Fig. 3). This is done by exploring operating system's data structures. For each element in the tree, the OS provides metadata such as whether the element is a sensor node or a hub (and whether it is active or passive), its manufacturer, the product ID and other descriptors. Details matter, since many hubs for instance share the vendor and product IDs, but are very different internally.

Once a gateway has constructed its USB tree, HPPC can be applied to all nodes or a selected one. Switching power of all nodes can be used, e.g., to do a testbed *soft reboot*

of the lowest, sensor node tier. For this case, the procedure starts traversing the tree from the root hub and, in a post-order fashion, switching power (on or off, as requested) of all of a hub's ports, assuming it supports HPPC. Note that it does not suffice to stop at the first USB hub that supports HPPC, since downstream hubs could be self-powered (thus connected nodes would remain unaffected).

Switching power of a particular node (without affecting the others) is used if for instance a test job is running on some nodes within the tree, and some other need to be restarted or reprogrammed. After searching for the target node in the tree, the procedure checks whether its direct parent hub supports HPPC. If it does, power switching is requested for the specific port of that hub to which the node is attached (this is the case for node N1 in Fig. 3). If it does not, the procedure backtracks through parents until it finds one that does. Since this could imply switching power to nodes in the common branch, care must be taken to consider this undesired side effect (e.g., N7 is safe through hub 3, but N4 is not safe through hub 1). We have additionally implemented a *force* option, which aggressively ignores switching other node's power. This is useful for troubleshooting tasks.

B. Reliability Improvements

Applying HPPC has shown to overcome many (though not all) of the issues in our testbed. Fig. 12 presents a quantification of the effects of HPPC on three of the problematic single node topologies (20, 30 and 54 meters), extended with an HPPC-able USB hub connected to the root. The plot shows the relative improvement in RCBF in these three cases, and suggests that the more complex the USB topology, the higher the improvement achieved by applying HPPC before reprogramming a node.

Enabling HPPC in a USB topology for any given node works best when having USB hubs that support this function as its direct parent. Although the USB standard specifies that bus-powered hubs are required to implement this function (self-powered hubs *might*) [14], in practice, few of all the USB hubs we tried were manufactured with the necessary circuitry to support it, to the point that it is very hard to find any on the market. Mass production, however, does not imply that this circuitry adds a significant price to the USB hub[3], and the reliability gains greatly outweigh its cost. *Conclusion: using HPPC is an inexpensive mechanism to improve reliability for an unattended testbed operation.*

C. Alternatives

As indicated earlier, some gateways' root hub USB circuitry effectively powers down attached USB devices when doing a software reboot. Besides implying shutting down critical services running on the gateway, boot cycle time (especially of single board computers) can take minutes, reducing the testbed availability. Although we have not searched extensively, among the hardware we inspected only the Buffalo WZR-HP-G300NH (rev. 2.0) routers exhibited this behavior.

[3]One such hub is the DeLock 4-port 87445, which retails for around $7.

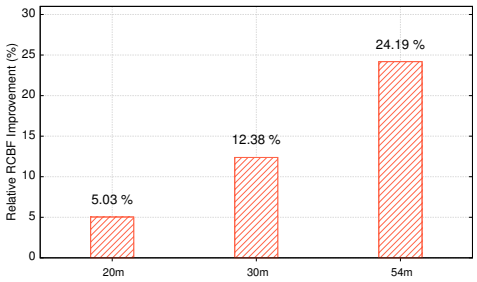


Fig. 12: Effects of Hub Port Power Control (HPPC) on the reliability within the testbed.

VI. Conclusions

Using the USB backchannel in a WSN testbed is a double-edged sword: On one hand, presently available hardware and software for WSN deployments do not guarantee a reliable USB communication, resulting in severe reliability issues that are exacerbated when the USB topology is enlarged. On the other hand, extending such USB backchannels leads to significantly fewer gateways – and therefore costs. Investigating this trade-off has been this paper's main goal.

From a case study with a deployed WSN testbed, we show the urgent need for such investigation: during a 10-month period, we observed that for a 46-node testbed in an office environment, only about 11% of the jobs were successfully deployed to all targeted nodes. All other jobs had one or more nodes becoming unresponsive during reprogramming due to a USB backchannel failure, and remained so until they were manually serviced.

Our analysis of possible designs of such USB backchannels has led to the following recommendations:

- passive cables will work to cross distances till 10m
- active cables can extend that distance, though at a cost of reliability
- active hubs further extend this to 43m, but they need power
- faster gateways reduce reprogramming time, but reliability might decrease
- parallel reprogramming can and should be practiced

with no remarkable differences noticed between different types of hubs or cables, apart from one particular 10-meter cable.

As a particularly interesting method, the cost-effectiveness of using hub port power control (HPPC) is highlighted, which allows failed nodes to be rebooted without servicing them manually. Tests in unreliable topologies showed that rebooting nodes by applying HPPC before each reprogramming cycle can increase the reliability by up to 24%.

Acknowledgments

This work is partly funded by the LOEWE Priority Program Cocoon, http://www.cocoon.tu-darmstadt.de, as well as the Research Training Group GRK1362, *Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments*.

REFERENCES

[1] Riccardo Crepaldi, Simone Friso, Albert Harris III, Michele Mastrogiovanni, Chiara Petrioli, Michele Rossi, Andrea Zanella, and Michele Zorzi. The Design, Deployment, and Analysis of SignetLab: A Sensor Network Testbed and Interactive Management Tool. In *3rd Int. Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, TridentCom, pages 1–10, may 2007.

[2] Manjunath Doddavenkatappa, Mun Choon Chan, and A.L Ananda. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *7th Int. Conference on Testbeds and Research Infrastructure for the Development of Networks and Communities*, TridentCom, pages 302–316, Shanghai, China, apr. 2011.

[3] Matthias Dyer, Jan Beutel, Thomas Kalt, Patrice Oehen, Lothar Thiele, Kevin Martin, and Philipp Blum. Deployment Support Network, A Toolkit for the Development of WSNs. In *4th European Conference on Wireless Sensor Networks*, EWSN'07, pages 195–211. Springer-Verlag, feb. 2007.

[4] Emre Ertin, Anish Arora, Rajiv Ramnath, Mikhail Nesterenko, Vinayak Naik, Sandip Bapat, Vinod Kulathumanit, Mukundan Sridharant, Hongwei Zhangt, and Hui Cao. Kansei: a Testbed for Sensing at Scale. In *5th Int. Conference on Information Processing in Sensor Networks*, IPSN'06, pages 399–406, New York, NY, USA, apr. 2006. ACM.

[5] Pablo E. Guerrero. The ukuFlow Macroprogramming System. http://www.dvs.tu-darmstadt.de/research/ukuflow/, 2012.

[6] Pablo E. Guerrero, Alejandro Buchmann, Abdelmajid Khelil, and Kristof Van Laerhoven. TUDμNet, a Metropolitan-Scale Federation of Wireless Sensor Network Testbeds. In *9th European Conference on Wireless Sensor Networks*, feb. 2012.

[7] Vlado Handziski, Andreas Köpke, Andreas Willig, and Adam Wolisz. TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Networks. In *2nd Int. Workshop on Multihop Ad Hoc Networks: from Theory to Reality*, REALMAN'06, pages 63–70, New York, NY, USA, may 2006. ACM.

[8] Jonathan W. Hui and David Culler. The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale. In *2nd Int. Conference on Embedded Networked Sensor Systems*, SenSys'04, pages 81–94, New York, NY, USA, nov. 2004. ACM.

[9] Daniel Jacobi, Pablo E. Guerrero, Ilia Petrov, and Alejandro Buchmann. Structuring Sensor Networks with Scopes. In *3rd European Conference on Smart Sensing and Context*, EuroSSC'08, Zurich, Switzerland, oct. 2008. IEEE Communications Society.

[10] Aslak Johansen, Thomas Sorensen, and Philippe Bonnet. Service and Experiment: Towards a Perpetual Sensor Network Testbed without Backchannel. In *8th Int. Conference on Mobile Adhoc and Sensor Systems*, MASS'11, pages 626–633. IEEE, oct. 2011.

[11] MEMSIC. Interface Boards Datasheets. http://www.memsic.com.

[12] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *4th. Int. Conference on Information Processing in Sensor Networks*, IPSN'05, pages 364–369, Piscataway, NJ, USA, apr. 2005. IEEE Press.

[13] Philipp M. Scholl, Kristof Van Laerhoven, Dawud Gordon, Markus Scholz, and Matthias Berning. jNode: a Sensor Network Platform that Supports Distributed Inertial Kinematic Monitoring. In *9th Int. Conference on Networked Sensing Systems*, INSS'12, pages 1–4, Antwerp, Belgium, jun. 2012.

[14] USB Implementers Forum. *Universal Serial Bus Specification Revision 2.0*, apr. 2000.

[15] Geoffrey Werner-Allen, Patrick Swieskowski, and Matt Welsh. MoteLab: a Wireless Sensor Network Testbed. In *4th Int. Conference on Information Processing in Sensor Networks*, IPSN'05, pages 483–488, Piscataway, NJ, USA, apr. 2005.