Ghost: Software and Configuration Distribution for Wireless Sensor/Actor Networks

Marcel Baunach
Department of Computer Engineering
Am Hubland
University of Würzburg
baunach@informatik.uni-wuerzburg.de

ABSTRACT

Wireless sensor and actor networks commonly consist of a rather large number of more or less widely distributed nodes. The resulting spatial complexity arises severe software related maintenance problems like application code and configuration updates. This paper presents the Ghost subsystem for efficient remote software maintenance in such networks. Besides safety, security and operation in heterogeneous environments we also address practical aspects like performance and resource requirements. Some results from a real-world installation will close this paper.

1. INTRODUCTION

In order to achieve a long lifetime for sensor/actor networks (henceforth simply called WSN), these distributed systems are subject to regular maintenance cycles. Besides hardware related issues like the renewal of power supplies or the replacement and attachment of modules, software related modifications are also very common. The latter allow application updates for integration of new functionality or for fixing bugs. Sometimes, there is just the need to reset a node or to modify its configuration for changing its behaviour or individual role in the overall system.

The problem's scope and intensity differs according to the evolution stage of the system. During software development, frequent updates (test versions) for few nodes can be expected. The frequency diminishes rapidly with the final release but then affects significantly more nodes within the original environment. But still, several updates might be required suddenly during the system runtime. It is similar with hardware. During development there are commonly few nodes but possibly they are already of different architectures. Often, these nodes are densely placed and easily accessible. After system deployment their number and heterogeneity increases within a quite ample environment. Then, some of them are hardly or not accessible at all. The conventional method to update sensor nodes on demand by means of mobile computers and debug-interfaces is safe and secure indeed but in any case extremely complex, annoying or even impossible. To counter this problem, the Ghost subsystem offers an efficient method for accomplishing software and configuration modifications via image distribution over the communication infrastructure which is available anyway within such networks. Since data dissemination protocols have received wide attention within the WSN community, several similar approaches like Typhoon [7], FiGaRo [8] and Deluge [6] already exist. Yet, such systems commonly provide their own customized communication/routing protocol or even require a special runtime environment on the nodes. In our opinion, both premises are adverse, since remote maintenance is important indeed but should neither define the actual application's communication nor affect the overall system performance. This is why Ghost was designed to use any available protocol and needs no modifications or extension of the used operating system. Instead, our focus aims on resource-aware integration into any software besides providing efficient and reliable operation. In this paper we present the central concepts of the Ghost remote maintenance subsystem but refer to radio communication for our concrete real world test bed.

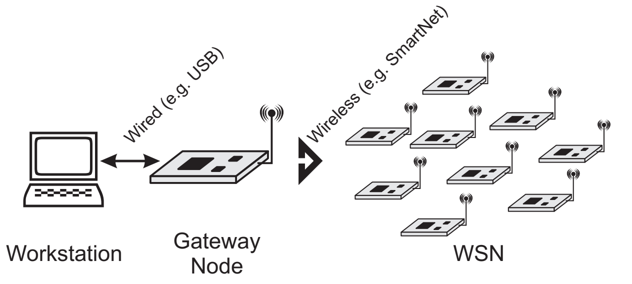

Figure 1: Ghost infrastructure for wireless remote maintenance

2. CONCEPTS AND OPERATION

Figure 1 shows wireless remote maintenance as one possible Ghost infrastructure. In this case, data is transferred from a workstation computer to a dedicated gateway node. The gateway creates data packets and transmits them to the destination nodes by any network protocol available in the WSN. Depending on this protocol, unicasts and broadcasts including routing are possible. If supported, groupcasts offer the special possibility to modify software in a role specific manner simultaneously for certain subsets of nodes.
Ghost is implemented as add-on for integration into any WSN application. To be always available for remote commands, it runs in parallel to each node's individual software and performs three basic operations:

1. reception and integrity check of Ghost command and data packets (image fragments) from any of the node's communication interfaces (radio, infrared, ethernet, CAN, etc),
2. direct execution of Ghost commands and buffering of images in an external flash memory, and finally
3. reprogramming of affected memory blocks (haunting).

Therefore, the Ghost subsystem is organized in two modules (→ Fig. 2): Steps 1 and 2 are executed during regular node operation (high level task). During step 3 the operating system along with the entire application is stopped (low level functions). After haunting, the device is reset.

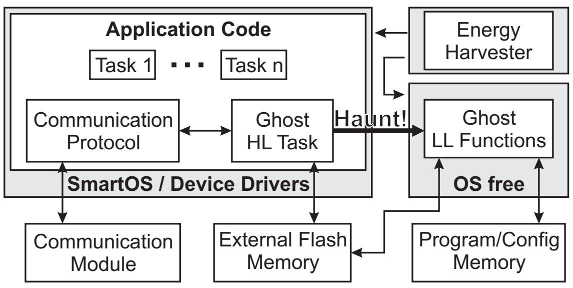

Figure 2: Ghost integration with WSN applications

2.1 Ghost Functionality

Always keep in mind, that software based remote maintenance subsystems stay passive for most of the entire system uptime whereas the fraction of their active operation time is relatively small. Thus, the complexity and permanent resource requirements (CPU, RAM, ROM) must be adjusted carefully to the available performance of the nodes and a well balanced cost-benefit-ratio is required.

This is why the high level module runs as regular task and requires no further adaptation of the remaining software. This task accepts five basic commands from virtually any communication channel:

1. **New**: initiates a new image transmission by sending the preconditions required to run or use the image (e.g. CPU type). This is required to avoid accidental installation of incompatible software or configurations which would render the node inoperable.
2. **Data**: successively transmits data fragments until the entire image is transferred. The fragments are limited by the payload size of the underlying network protocol.
3. **Reset**: is used for explicit restarting of nodes.
4. **Query**: retrieves application, hardware and runtime related information about the node. This always includes application type, version, build number, CPU type, remaining energy and uptime. Additional data is optional and node-specific but always limited by the payload of a single network data packet.
5. **Haunt**: initiates updating of program and/or configuration by any previously received image. This is done in Ghost low level mode and finally invokes an implicit node reset.

Figure 3a shows an example for a possible remote software update procedure via a gateway node. Depending on the requirements and remaining memory resources, an additional clone-function can be statically linked into the Ghost subsystem:

6. **Clone**: requests the currently running application code from a node.

By first sending a query, a node can check its neighbourhood for available software types and versions. Thereby newly deployed nodes can integrate themselves almost autonomously into a running WSN by requesting the appropriate software from neighbour nodes. Figure 3b shows an example. Yet, this autonomy requires that each new node is equipped with a certain initial application containing the Ghost subsystem and the knowledge about its future role in the WSN. This role is defined by the application type, which is unique for each kind of software and hardware combination (e.g. router for TelosB nodes, ultrasound sensor for SNOW5 nodes, etc.). Likewise, nodes can stay up-to-date by observing their environment for newer versions of their own software. In contrast to the common method, where a source node (gateway) pushes the software to each affected node, cloning allows updates to be pulled over the whole network without explicit routing or flooding. This desired virus like spreading is extremely useful for very large networks where the individual handling of each single node would become too complex. Of course, the network protocol must remain compatible between versions to support any kind of long term self-maintenance.

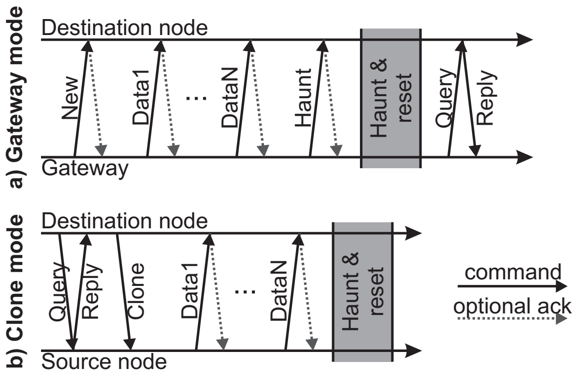

Figure 3: Ghost communication process

2.2 Security

Secure data transmission is very critical in wireless sensor networks. This is especially true for the deployment and maintenance of software and configuration images since a tapping attacker might draw conclusions about operation and vulnerabilities of the overall network. Of course, prevention of hostile code injections or node takeovers is also mandatory.

The obvious technique is to encrypt all communication. Yet, the used algorithm must provide an adequate security level at appropriate resource requirements. Furthermore, the key selection needs careful consideration. Where an individual key for each node provides highest security (especially, if nodes can be stolen along with the secret), a simultaneous updating of several nodes by a single encrypted image is not possible any more. The same holds for cloning, since key exchange methods among today's low-power nodes are not practicable. Thus, common keys increase the performance while significantly reducing the network load altogether with energy consumption.

Our current implementation focuses on usage during WSN development in research and education. Thus, we prefer performance to security and use either no encryption at all or the TEA algorithm [12] for a minimum of security. Then, the password is not necessarily node specific, but can depend on the application type, version and build number. This way and despite of a regular password change with each update, it is still possible to supply several nodes simultaneously with new software. At receiver side, the high level task validates the packets regarding encryption and CRC checksum before executing control commands or buffering new images.

2.3 Safety and Reliability

Safety and reliability are other important factors in remote maintenance systems. This includes the guaranteed and complete reception of command and data packets at the desired destination nodes. Concerning the communication, Ghost completely relies on the used network and routing protocol. This accounts for compactness and flexibility of the Ghost subsystem. Just the data packets are successively numbered with each new transmission.

While communication is shifted to a WSN specific protocol, Ghost integrates some mechanisms to guarantee smooth completion of the actual update process and to prevent the node's total breakdown caused by unpredictable errors while haunting. This is why the Ghost subsystem is divided into two parts.

As the high level task runs as part of the application and in parallel to the user tasks it is permanently ready to process Ghost packets (→ Fig. 2). If a haunt command arrives, the Ghost task checks for sufficient remaining energy to perform the expensive update operation. If energy is low, haunting is deferred to allow a possibly available energy harvester to recharge the power supply. As soon as enough energy is available or if the charge level can not be determined at all, the Ghost task stops the operating system and passes control to the low level functions for haunting the system.

Checking the remaining energy before each update is very important since many sensor nodes are based on microcontrollers with program flash ROM. Besides, many applications are statically linked and hard to modify at runtime. In both cases, the node's program memory must be erased before installing the new software. A power breakdown between erasing and complete reflashing would indispensably corrupt the node and require manual repair – if this is possible at all. Indeed, the energy requirements for updating a node may not be underestimated. They depend on the electrical characteristics of program and flash memory and on the image size. The latter defines the number of read/write cycles and in consequence the total duration of this critical process. Section 4 shows the energy requirements of a concrete Ghost implementation.

Yet, underestimated energy resources as well as other unpredictable problems can still lead to node reset or hanging during the update. Thus, the Ghost low level functions configure an available watchdog timer to force an reset in case of a hanging node. A reset always leads to a well defined system state, since the Ghost low level part will never be erased from the ROM. In particular, it contains an emergency function (→ Fig. 4), which is automatically executed at node start-up to recover from awkward situations. First, it tests a flag indicating the state of the last update. On success, the regular application is started and the operating system takes over control. On failure, the power supply is again checked repeatedly until there is sufficient energy to restart the update.

3. IMPLEMENTATION

As useful and indispensable remote maintenance is for large scale or frequently updated sensor networks, as complex are the resulting requirements and challenges for a concrete implementation. Thereby, some problems are rooted in the underlying network (hardware and protocols), in the node's architecture and in the applied operating system. Our implementation of Ghost uses the SNoW5 [2, 5] sensor node and the *Smart*OS [3] operating system. Within the test application presented below we used the *Smart*Net wireless communication protocol.

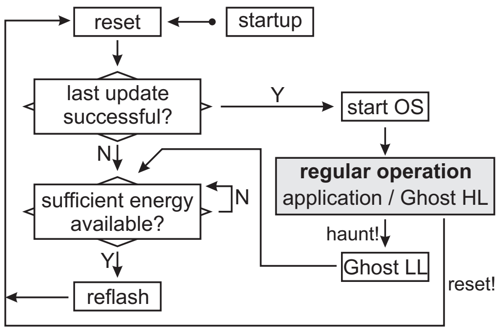

Figure 4: Ghost low level operation during system runtime

The SNoW5 [2, 5] nodes integrate a MSP430F1611 [10] controller with 10 kB RAM and 48 kB ROM running at 8 MHz. An external 2 MB flash memory allows simultaneous buffering of various Ghost program and configuration images. Communication is done via a CC1100 [9] radio transmitter supporting data rates up to 500 kbps.

*Smart*OS [3] was developed for real-time operation of low-power and low-performance devices like sensor nodes. The preemptive scheduling of prioritized tasks allows prompt processing of internal and external events. Altogether with the sophisticated resource management policy, it allows the easy composition of arbitrary tasks to still efficient applications. Thus, *Smart*OS based software can easily be extended by the Ghost subsystem.

Finally, *Smart*Net was developed for both, CSMA/CA or slotted wireless communication. It allows broadcasts, groupcasts, the integration of routing protocols and self-organizing communication via e.g. HashSlot [1]. The *Smart*Net high priority task processes incoming data packets and redirects them to the appropriate receiver tasks. Similar to TCP ports in IP networks, this is done by each packet's port ID and destination address. In turn, the receiver task is signalled and reactivated by *Smart*OS according to its priority.

Fig. 2 shows the integration of the Ghost subsystem into a *Smart*OS based software. Since *Smart*OS is fully preemptive, it is sufficient to link the Ghost modules into the the WSN application. Besides adding the high level task, the linker also places the low level module adequately to start the execution of the emergency recovery function at system start prior to the operating system. The required memory (RAM/ROM) adds up of the Ghost modules plus the communication protocol and the flash memory driver. However, the last two components are often part of the application anyway and thus mean no extra system load. Yet, a slim and OS independent version of the flash memory driver is required within the low level module to gain read access to this device. In general, performance losses caused by Ghost are negligible since only the relatively rare Ghost commands trigger the execution of the high level task. As already mentioned in section 2.3, an energy harvester is optional but can account significantly to the system stability during memory update processes.

4. TESTBED AND RESULTS

For evaluating the just described techniques under real world conditions, we used a SNoW Bat [4] indoor localization system comprising 45 nodes. In this section we will present some results concerning update speed, resource requirements and energy efficiency.

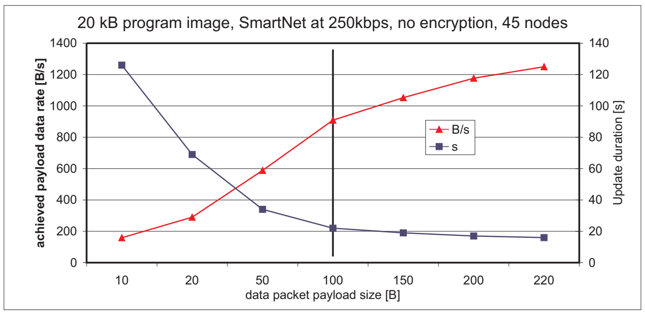

Figure 5: Ghost performance for program image update

In fact, these metrics are not only Ghost dependent but also reflect some characteristics of the applied communication protocol, the node's software architecture and hardware components.

Regarding the ROM size, our current Ghost implementation takes <880 B without and <1 kB with clone functionality (depending on the compiler's optimization level). The RAM requirements of the high level module are about 126 B fixed plus the buffer size for the data packet payload. As figure 5 shows, the latter has significant impact on the data dissemination time. Thus, we commonly chose 100 B for trade-off between speed and memory consumption.

When updating the software within our test bed, we found that *Smart*Net managed to deliver the data very reliable, though the nodes' main application also made intensive use of wireless communication during the remote maintenance process. Finally, reprogramming all 45 nodes by an image groupcast commonly took the same time as reprogramming a single one (approx. 22 sec). In comparison, manual flashing of the same WSN system via JTAG successively node by node took approximately 1 hour and was simply too much time-consuming. In fact, this expected speed-up was the initial main motivation for developing the Ghost system at all.

Finally, table 1 shows the energy requirements of a single node during various operation modes. One can see clearly, that updating the program memory imposes significant load on the power supply compared to regular operation. Yet, intentionally provoked node breakdowns due to current peaks or empty batteries were safely handled by the low level recovery function.

5. CONCLUSION AND OUTLOOK

In our paper we presented the Ghost remote maintenance subsystem for wireless sensor and actor networks. We showed, that efficient and fast software and configuration updates are possible without the need for a predefined communication protocol or especially modified system software. Instead, Ghost integrates easily into any WSN application and accepts data from arbitrary sources (wired or wireless). Thereby it requires little memory and computational power. Besides, it supports role specific software deployment by either direct transmission via a gateway or by autonomous self-maintenance of the nodes via cloning. This way, simultaneous updates are even possible in heterogeneous networks. Finally, the sophisticated low level module along with the optional encryption allows extremely safe and secure operation for reliable remote maintenance. Our current work in this field addresses viral data dissemination techniques, power aware operation in general and energy harvesting in particular.

Table 1: Energy requirements during sensor node operation

low system load (no radio)	17mW
heavy system load (no radio)	26mW
data reception (radio) and buffering (flash)	69mW
haunting process (copy flash to ROM)	81mW

6. REFERENCES

[1] M. Baunach. Speed, Reliability and Energy Efficiency of HashSlot Communication in WSN Based Localization Systems. In Verdone [11], pages 74–89.

[2] M. Baunach, R. Kolla, and C. Mühlberger. SNoW^5: A versatile ultra low power modular node for wireless ad hoc sensor networking. In P. J. Marrón, editor, *5. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, pages 55–59, Stuttgart, 17.–18. July 2006. Institut für Parallele und Verteilte Systeme.

[3] M. Baunach, R. Kolla, and C. Mühlberger. Introduction to a Small Modular Adept Real-Time Operating System. In Distributed Systems Group, editor, *6. Fachgespräch Sensornetzwerke*, pages 1–4, Aachen, 16.–17. July 2007. RWTH Aachen University.

[4] M. Baunach, R. Kolla, and C. Mühlberger. SNoW Bat: A high precise WSN based location system. Technical Report 424, Institut für Informatik, Universität Würzburg, May 2007.

[5] M. Baunach, R. Kolla, and C. Mühlberger. SNoW^5: a modular platform for sophisticated real-time wireless sensor networking. Technical Report 399, Institut für Informatik, Universität Würzburg, Jan. 2007.

[6] J. W. Hui and D. E. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In J. A. Stankovic, A. Arora, and R. Govindan, editors, *SenSys*, pages 81–94. ACM, 2004.

[7] C.-J. M. Liang, R. Musaloiu-Elefteri, and A. Terzis. Typhoon: A Reliable Data Dissemination Protocol for Wireless Sensor Networks. In Verdone [11], pages 268–285.

[8] L. Mottola, G. P. Picco, and A. A. Sheikh. FiGaRo: Fine-Grained Software Reconfiguration for Wireless Sensor Networks. In Verdone [11], pages 286–304.

[9] Texas Instruments Inc., Dallas (USA). *CC1100 Single Chip Low Cost Low Power RF Transceiver*, 2006.

[10] Texas Instruments Inc., Dallas (USA). *MSP430x161x Mixed Signal Microcontroller*, Aug. 2006.

[11] R. Verdone, editor. *Wireless Sensor Networks, 5th European Conference, EWSN 2008, Bologna, Italy, January 30-February 1, 2008, Proceedings*, volume 4913 of *Lecture Notes in Computer Science*. Springer, 2008.

[12] D. J. Wheeler and R. M. Needham. TEA, a Tiny Encryption Algorithm. In B. Preneel, editor, *FSE*, volume 1008 of *Lecture Notes in Computer Science*, pages 363–366. Springer, 1994.