Technische Universität Braunschweig

Institute of Operating Systems and Computer Networks

Amphisbaena: A Two-Platform DTN Node

S. Rottmann and R. Hartung and J. Käberich and L. Wolf

Authors post-print published on 2017-08-28
Originally published in *2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*
Publisher version available at
DOI: 10.1109/MASS.2016.039

Abstract:

In most Wireless Sensor Network deployments, the energy supply is a major challenge. Especially for nodes with high computational power or high bandwidth communication interfaces, the required size of batteries might increase to infeasible levels, even if the option of energy harvesting exists. For many use cases, some nodes are idling most of the time and transmitting only a few bytes from time to time. In this paper, we present a two-platform node consisting of a high-power and a low-power platform. Both platforms are using the same Delay-Tolerant Networking (DTN) architecture and the same protocols. A novel concept offers the opportunity for both platforms to appear as a single node to communication partners. The high-power part is running a full-featured Linux operating system, the low power platform is built around an energy-efficient 32-bit microcontroller and is able to fulfill tasks, which would have required to wake up the high power node in a conventional setup. Our system can increase the energy efficiency in WSN scenarios where the demand of bandwidth and computational performance is strongly fluctuating.

2016 IEEE 13th International Conference on Mobile Ad Hoc and Sensor Systems

Amphisbaena: A Two-Platform DTN Node

Stephan Rottmann, Robert Hartung, Jan Käberich and Lars Wolf
Institute of Operating Systems and Computer Networks
Technische Universität Braunschweig
Braunschweig, Germany
Email: [rottmann | hartung | kaeberic | wolf]@ibr.cs.tu-bs.de

***Abstract*—In most Wireless Sensor Network deployments, the energy supply is a major challenge. Especially for nodes with high computational power or high bandwidth communication interfaces, the required size of batteries might increase to infeasible levels, even if the option of energy harvesting exists. For many use cases, some nodes are idling most of the time and transmitting only a few bytes from time to time. In this paper, we present a two-platform node consisting of a high-power and a low-power platform. Both platforms are using the same Delay-Tolerant Networking (DTN) architecture and the same protocols. A novel concept offers the opportunity for both platforms to appear as a single node to communication partners. The high-power part is running a full-featured Linux operating system, the low power platform is built around an energy-efficient 32-bit microcontroller and is able to fulfill tasks, which would have required to wake up the high power node in a conventional setup.**

Our system can increase the energy efficiency in WSN scenarios where the demand of bandwidth and computational performance is strongly fluctuating.

I. Introduction

A major challenge when installing wireless networks outdoors is the energy supply if no access to the grid is available. In those cases, the nodes have to be powered by batteries. Energy harvesting devices like Photo Voltaic (PV) panels may increase the lifetime of such an installation. But still, especially for devices with a high computational power, the possible run time and thus availability is limited.

Depending on the actual use case of the networks, some nodes do not need to be powered up all the time and may be shut down when not in use. Here, an additional device to wake up the main platform is required. In deterministic communication scenarios, a simple time based schedule may be sufficient, but dynamic wake-up events may not be possible.

In many dynamic scenarios, it is crucial that a node can be reached for communication at any time. Thus, a device is needed which is able to power up the network node in demand.

Thinking of a Wireless Sensor Network (WSN), one might imagine a network with different kinds of nodes. Some of them might have the task to measure and transmit the temperature, which can easily be done with a microcontroller. Others require a full-featured operating system like Linux, if their task is to take and process a picture at regular time intervals using computer vision algorithms. Still, most of the time, such a node might just forward the occasional data packets for other participants in the network while mostly idling.

Thus, it would not be efficient to keep the system powered up waiting for a forwarding request. As a conclusion the full-featured node should only be active on demand to save energy.

For such situations, we propose a two-platform DTN [1], [2] node which appears to other participants in the network as a single node offering different communication channels like IEEE 802.15.4 and WLAN. Our system consists of a low-power, low-energy platform based on a 32-bit microcontroller and a high-power platform which is a Single Board Computer (SBC) running Linux such as the *Raspberry Pi*[1]. Both platforms are connected to each other via USB and run their own implementation of a DTN software stack. Both are able to operate on their own, but as a novel feature can cooperate and appear as a single node within the network.

On the Raspberry Pi running Linux, *IBR-DTN* [3] is used in combination with a USB-WiFi dongle. The microcontroller-based platform presented in this paper uses our implementation *miniDTN* which is also presented in this paper. For communications, different types of low-power radio may be plugged onto our newly designed board. So far a *LoRa*[2] and an *IEEE 802.15.4* variant are available.

Since both platforms appear as a single node to neighbors, the low-power board is able to detect if a message received via the low-power radio link is destined for the high-power SBC or if the low-power link's bandwidth is not sufficient to forward a bundle. After booting the full featured Linux board, already received data and additional information can be transferred via USB.

The remainder of this paper is structured as follows. In Section II, we show related work which has been done in this field, followed by a detailed description of the design of our implementation in Section III. Besides the hard- and software, our concept of combining both platforms to a single node are discussed. Section IV explains the concepts of communication on which the evaluation in

[1] http://www.raspberrypi.org
[2] http://www.lora-alliance.org

2155-6814/16 $31.00 © 2016 IEEE
DOI 10.1109/MASS.2016.23

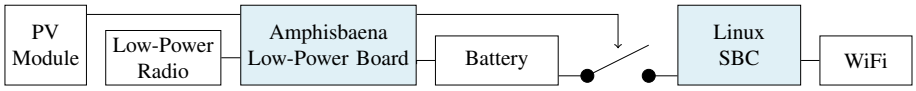


Fig. 1. Schematic diagram of the hardware

Section V is based. The paper is concluded in Section VI.

II. Related Work

To the best of our knowledge, we are the first to develop a two-platform node that appears to be a single node within the network. Several systems have been proposed which can power up a node to allow a communication contact. Some rely on the fact that different communication standards are sharing the same radio frequencies. For instance, an IEEE 802.15.4 radio can be used to detect the presence of other communication systems like Wireless Local Area Network (WLAN) [4]. The low-power radio cannot decode the actual data, but if it recognizes any transmission, the high-power platform is powered up. While this is reasonable for rural and remote areas where usually no data transfers take place, this would cause problems in cities where a lot of devices using WLAN might be detected. In order to prevent unnecessary wakeup events, the authors have extended the approach and added the possibility to transmit a signature addressing a specific node using the WLAN interface [5].

Another option, which does not require an additional full-featured radio, has been proposed by Spenza et al. [6]. In this work, the authors built an envelope detector from scratch with discrete electronic components. An ultra-low-power microcontroller is able to decode a signal received by the HF part of the circuit similar to the aforementioned project.

A node which consists of two platforms using diverse wireless links and protocols is presented in [7]. Each system consists of a low-power node with an IEEE 802.15.4 radio and a Linux board with a WLAN IEEE 802.11n interface. The low-power platform is used only for control traffic (power up the next hop). The use case described in their work focuses on a line-shaped multi hop topology without considering meshed networks with many paths. Nodes other than source or sink are solely used for data forwarding. The use case described is real time end-to-end data transmission for surveillance cameras, etc.

In the *DieselNet* project [8], so-called *Throwboxes* have been designed. Those devices consist of a battery and two computing devices, a *Soekris* SBC running Linux with a WLAN interface and a TelosB mote with an XTend radio module which has a communication range much higher than the WLAN interface. In the testbed, buses of the public transportation system driving in the city would send beacon messages containing their position via the long-range radio. From these beacons, a movement profile is calculated on the TelosB mote. Using Markov chains, the probability of a contact via WLAN can be calculated. If a connection can be established, the Linux board will be powered up [8].

A multi-platform approach is implemented in *mPlatform* [9]. Here, the authors present a system which can consist of multiple processing units connected to each other by a custom bus system. Thus, for each application the optimal configuration can be found.

We have presented a similar system consisting of two boards (Linux and low-power) in [10]. There, each board has its communication links (WLAN and IEEE 802.15.4) and protocols. Different to this paper and similar to the *DieselNet* approach, only the Linux board was able to exchange data in a DTN since the low-power link was used for control and signaling traffic only.

Based on the experiences made with this system, we extended the functionality significantly. Besides hardware improvements like a more flexible and efficient charge controller and a much more powerful Microcontroller Unit (MCU), the main aspect of our new platform is the collaboration of both parts. This allows to keep the Linux board powered off in most cases. The new MCU offers an integrated USB link as well as an Ethernet port. These interfaces offer many new possibilities for operation, such as a gateway node. When not in use, most components can be turned off to save energy.

III. System Architecture

In this section, we will describe our contributions, both in soft- and hardware. The "two-headed" node *Amphisbaena*[3] consists of two computing platforms, as shown in Figure 1. The colored blocks are running a DTN implementation.

A. Hardware Architecture

All components are powered by a battery which may be charged from a PV panel. The system implements a Maximum Power Point Tracking (MPPT) charge controller maximizing the energy harvested from the PV panel. The battery's state of charge is monitored to avoid a deep discharge cycle. Currently, we have two different kinds of radio modules which can be plugged into the board: an IEEE 802.15.4 radio (*Atmel AT86RF233*) or a *LoRa* module based on a *Semtech Sx1272*. These radios offer a low data rate at a small current consumption. Especially the latter can be used for very long range

[3] A serpent with two heads from Greek mythology.

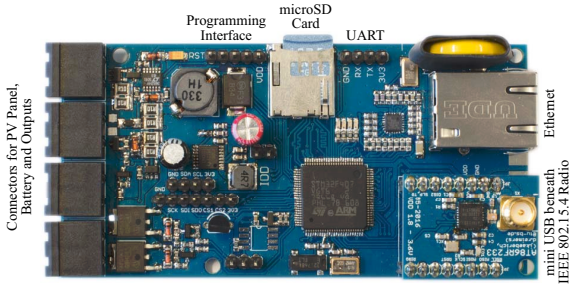


Fig. 2. The *Amphisbaena* board

communications: A few kilometers are possible, even in urban areas and with a piece of wire as antenna [11].

We chose a *Raspberry Pi* as the Linux SBC shown in Figure 1. It can be turned on or off by the low-power platform. Both boards are connected via USB to each other, allowing data transfers and communication. This link is used for:

- **Control Messages:** Coordinating the cooperation, synchronizing date and time, notification of upcoming shutdown (which may be interrupted).
- **Bundle Exchange:** Transfer DTN bundles received by the low-power platform.
- **Use Radio Link:** When awake, the SBC is able to transmit and receive data via the radio attached to the microcontroller.

With our node, we target outdoor network installations where a wired mains power supply is not available. Since the whole system is powered by a battery, all components have to be energy efficient and should be turned off when not in use. Basic communication in the DTN can be done by the low-power board: Sensor data can be read and routed to a sink without the need to boot the SBC. Also, small to medium bundles of other nodes can the forwarded via IEEE 802.15.4 or LoRa. The Linux board will only be booted if a task cannot be handled by the low-power platform or upon request.

A picture of the board is shown in Figure 2. Depending on the configuration, a single board (PCB and components) costs about USD 70. Populating the Ethernet parts for example, adds about USD 10.

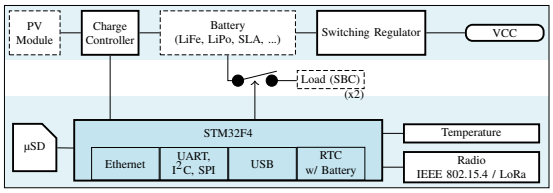


Fig. 3. Architecture of the *Amphisbaena* board. Dashed boxes are connected externally using wires

A block diagram of the *Amphisbaena* hardware presented in this paper is shown in Figure 3. The components on the board can roughly be classified into two groups, a) the power supply including the charge controller and b) the processing unit. Both are explained in more detail now.

a) Power Supply: For energy management purposes, the current and voltage delivered by the PV module and battery can be monitored by the microcontroller. A high-side current monitor *INA138* is used to meter the currents. The result is fed into the integrated Analog to Digital Converter (ADC) of the MCU. The voltage dividers for scaling the voltages to appropriate values can be turned off using MOSFETs, which is used to save energy.

The blocks shown in Figure 3 are:

- **PV Module:** A PV panel is used to harvest electrical energy from sunlight. The open circuit voltage should not exceed 34 V, which is the maximum input voltage of the charge controller.
- **Charge Controller:** The *LT3652HV* charge controller IC is able to charge different types of batteries (Li-Ion/LiPo/$LiFePO_4$/Lead-Acid). MPPT is implemented using this controller.
- **Battery:** Electrical energy is stored in the battery. Its type depends on the use case, available mechanical space and other factors.
- **Switching Regulator:** An efficient switching regulator *LM43603* is used to supply the board with 3.3 V.

b) Processing Unit/MCU: The major part of the board is the microcontroller and its peripherals. Besides the communication task, the MCU also controls all peripherals on the board. Voltages and currents are being monitored, a MPPT controller is implemented in the software which maximizes the energy harvested from the PV panel. All user applications for the low-power board run on the MCU. An example for a very simple task could be monitoring a temperature and transmitting the data using the DTN communication stack.

- **STM32F4:** A 32-bit microcontroller (*STM32F407VGT* [12]) is used as the MCU on the low-power board. It runs with *FreeRTOS*[4] as an operating system and provides several communication interfaces such as Ethernet, UART and USB. An RTC with an external backup battery is used to keep the time and wake the CPU from deep sleep modes. The Linux board is able to read the time from this RTC after booting, so no time synchronization via NTP, GPS or radio controlled clocks is required.
  The MCU has a maximum clock rate of 168 MHz, but can be scaled down on demand.

[4] `http://www.freertos.org`

- **Temperature:** 1-wire sensors (*DS18B20*) are used to monitor the temperature on the board, at the battery and at the PV panel. This makes it possible to adjust end-of-charge-voltages. One sensor is mounted on the PCB, additional ones can be connected via a 3-pin header.
- **µSD:** The micro SD card is used to store configuration options (DTN endpoint identifier, Ethernet configuration, ...) as well as DTN bundles.
- **Load Outputs:** Two outputs can be switched on and off by the MCU. When switched on, external loads are powered directly from the battery. The Linux board (*Raspberry Pi* in our case) is connected with its own voltage regulator to one of these outputs. Each output is able to provide several Amperes for the connected load.
- **Radio:** A daughter board with a radio can be plugged into the base board. At the moment, two radios with a compatible pin configuration are available. Both use the Serial Peripheral Interface (SPI) bus for communication. The LoRa module itself does not have an external connector for an antenna, so we integrated a SMA connector on the base board. The same type of plug is used on the breakout board for the *Atmel* IEEE 802.15.4 radio, which makes it easy to use in other applications. With an external antenna, the board can also be put into a robust metal housing with the antenna outside.
- **Additional Interfaces:** More peripheral components can be attached to the board using several buses, such as SPI or I^2C.

The PCB we built has the dimension of 5x10 cm. The hardware is designed in a way that most components can be completely shut off with MOSFETs if they are not needed. This means that no current is being consumed when devices are in sleep mode. All connectors for communication (mini-USB, SMA for antennas, Ethernet) are located on the short edge of the board, screw terminals for battery, PV panel and the SBC are available on the opposite edge. Four holes (∅ 3 mm) can be used to mount the PCB.

B. Software Architecture

1) Embedded: The embedded Delay-Tolerant Network (DTN) implementation called *miniDTN* runs on the *FreeRTOS* operating system and is based on *µDTN* [13]. We chose to port the software from the Contiki OS[5] due to the greater availability and thus support of *FreeRTOS* on 32-bit MCUs. The Bundle Protocol Agent is running as a *Task* in FreeRTOS. This makes it easy to extend existing applications with our Bundle Protocol implementation.

[5] `http://www.contiki-os.org`

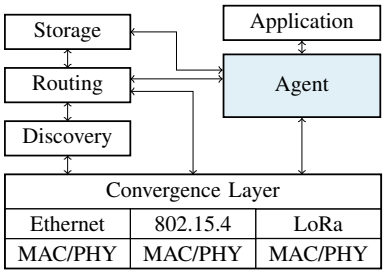


Fig. 4. Schematic diagram of the software architecture

Currently two Convergence Layers (CLs) are implemented in *miniDTN*. Both are datagram based and compatible with their *IBR-DTN* [14] counterparts. One of them is based directly on top of the IEEE 802.15.4 MAC layer ("`dgram:lowpan`"), while the other one uses UDP. LwIP[6] is used as the IP stack for communication over Ethernet. New CLs for *miniDTN* can easily be developed by implementing a simple abstracted interface.

Nodes may be connected via multiple links to each other at the same time. A central routing module decides which link should be used to transfer the data. New rules can be implemented to support application specific routing.

A simplified schematic diagram of *miniDTN*'s software architecture is shown in Figure 4. The *Agent* is the central component of the implementation. Each *Application* has to pass its bundles to the *Agent*. Bundles are stored in the *Storage*, which may either be on the microSD card or in the RAM of the MCU. The *Routing module* submits bundles to the *Convergence Layer* to be sent over the air or wire. Neighbors are discovered via beacon messages on the *Convergence Layer* and handled by the *Discovery module*. Information is passed to the *Routing module*, which decides which *Convergence Layer* to use.

2) On High-power Node: The operating system running on the high-power node is a Debian-based Linux. For communications, *IBR-DTN* is used, which is a lightweight implementation of RFC5050 [2]. More information on this software can be found in [3], [14] and online[7]. The SBC is connected to the embedded board via USB. Modifications to the *IBR-DTN* daemon enable the cooperation of both systems. After starting, control messages are exchanged which allow to adopt the current state and data exchange.

C. Cooperation of DTN Implementations

The concept of two a network node consisting of two individual platforms which appear as a single one

[6] `http://savannah.nongnu.org/projects/lwip`
[7] `http://www.ibr.cs.tu-bs.de/projects/ibr-dtn`

to the outside offers many opportunities in wireless networks. Communication partners do not need to know which service is running on which platform. No explicit messages have to be exchanged which are used to request the booting of the Linux board.

Both DTN implementations can exchange control data and bundles with each other. Both Bundle Protocol agents know of the services running on the other platform and may forward received data to the other one. Sensors attached to the low power platform can be accessed using the USB link from the Linux board which allows the SBC to fulfill all the tasks which would be done by the *Amphisbaena* board itself when the Linux part is powered off. This means that no data will be lost and no delays occur by letting the high power platform to take control.

At the moment, the SBC will be powered up if data is being received which has to be handled by the Linux part or if the IEEE 802.15.4/LoRa-link is not powerful enough for the traffic demand. This can be detected by a congestion of the radio link or by the requirements of the data transmissions. This will be discussed in Section IV. An application may decide if the data should be transferred saving energy and thus may need longer to reach its destination, or if it is urgent and the reception is time critical.

After booting the SBC, a neighbor in the network would only see that the node with the same ID has activated a new communication link like WLAN. Later, this functionality will be extended by actively announcing the possibility to activate the high power board. This can be used for routing decisions, especially in a network of more nodes.

IV. Network Performance

When switching between the low-power platform and the Linux-based SBC in an application, it is important what performance at which cost of energy can be expected in the different modes. This is not only depending on the communication medium, but also on the processing speed, storage performance and other tasks running on the platforms.

In order to be able to make a decision which communication link to use or to switch to the SBC with a higher performance, both the time needed for data transfers and the energy demand need to be known.

We tried to cover most communication scenarios which might appear in a real world DTN or WSN. In many situations, a host is sending a small-sized request to another communication partner, which either sends a small acknowledgment or a bigger portion of data. To model these situations, a `ping`-like application has been implemented which sends a data packet and specifies (in this data) the requested size of the answer. As bundle sizes modeling a request/ACK or payload data, 64 Byte and 1024 Byte have been chosen.

The applications used for the data transfers are running in the user space of each DTN implementation. Using the existing `dtnping` application shipped with *IBR-DTN* gives better results in means of lower Round Trip Times (RTTs), but does not depict a real world scenario since an application developed by a user will not be compiled into the daemon. The communication between the daemon and the application itself also takes some time and processing overhead.

The expected outcome of our measurements is to create a function which returns the best option for transmitting data with given characteristics. One can decide, if the result should be better in terms of energy demand or time. The clock frequency of the MCU has an influence, on both the energy and time. Such results are discussed in Section V.

The cost C in means of time and energy of a single data transfer is defined as

$$C(s_{tx}, s_{rx}, M_i, f_{CPUj}) = (E, T)(s_{tx}, s_{rx}, M_i, f_{CPUj}) \tag{1}$$

s_{tx} and s_{rx} are defining the (expected) amount of data to be transferred. M_i is the communication medium, which might be IEEE 802.15.4, Ethernet or WLAN (on the SBC), f_{CPUj} is the clock frequency of the MCU. E and T are the energy and time needed for the transfer.

Equation 2 and Equation 3 give the minimum energy or time needed for the transmission defined by s_{tx} and s_{rx} given the communication mediums $M_0 \dots M_M$ and available clock frequencies $F_{CPU0} \dots F_{CPUN}$ of the *Amphisbaena* board.

$$E_{min}(s_{tx}, s_{rx}) = min(E(s_{tx}, s_{rx}, M_0, f_{CPU0}), \dots \dots E(stx, s_{rx}, M_M, F_{CPUN})) \tag{2}$$

$$T_{min}(s_{tx}, s_{rx}) = min(T(s_{tx}, s_{rx}, M_0, f_{CPU0}), \dots \dots T(s_{tx}, s_{rx}, M_M, F_{CPUN})) \tag{3}$$

When switching on the SBC in order to transfer larger amounts of data or using a more powerful communication link such as WLAN, the boot time T_{Boot} and the energy needed for the boot process E_{boot} have to be considered. During the boot process, no data exchange with the SBC is possible.

For all combinations of bundle sizes, communication media and clock frequencies, measurements have to be conducted. In our measurements, the set of communication links of the *Amphisbaena* board considered, consists of Ethernet and IEEE 802.15.4, the set of clock frequencies is 24 MHz, 30 MHz, 84 MHz and 144 MHz.

The sequence of data exchange between nodes during the evaluation is shown in Figure 5. At time T_0, a bundle is created and passed from the application to the

miniDTN agent. The bundle is parsed by the agent and sent on wire do the destination at the time T_1.

Between T_1 and T_2, the bundle is received by *IBR-DTN* and a simple application parses the data and generates the answer bundle. After *miniDTN* received this bundle, an acknowledgment is sent out on the Bundle Protocol (BP)'s CL which is not seen in the user space.

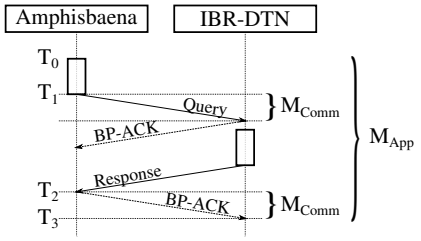


Fig. 5. Communication Protocol. For larger packets, *Query* or *Response* may consist of several packets on the wire, each of them will get ACK'ed.

V. Evaluation

A. Description of Measurements

The general architecture for the measurements is shown in Figure 6. We want to measure energy consumption of the *Amphisbaena* board including the MCU and all peripherals, such as the Radio or Ethernet PHY. A *Control PC* is used to configure the bundle sizes and other parameters like destination and count on the MCU.

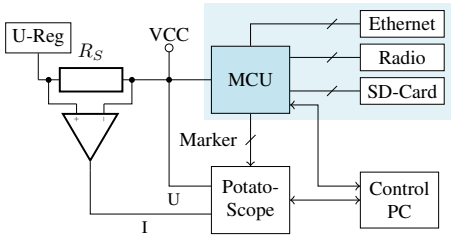


Fig. 6. Measurement setup

For measuring the energy consumption – that is supply voltage and current – we use the PotatoScope [15], a microcontroller-based oscilloscope.

With this integrated measurement solution that offers to set markers in the code, we are able to directly measure energy consumption of parts of the software. Markers are input pins of the PotatoScope that can be pulled high or low. The state of the markers are saved along with each sample, allowing for a fine-grained annotation of the measurement. Measurements were taken at a sample rate of 100 kHz.

Two markers are implemented in the software called M_{Comm} to signal activity on the wire and M_{App} covering the whole communication. These are also shown in Figure 5. The usage of markers for the scenario described in Section IV is illustrated in Figure 7. At bundle creating time T_0 a rising edge of the Application-marker can be seen. A falling edge occurs right after receiving the answer from the other station at T_3. During the actual data transfer over the wire starting at T_1 and T_2, the marker M_{Comm} is held high. The total energy demand of a data transfer can be calculated using Equation 4.

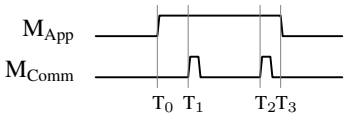


Fig. 7. Waveforms of markers (not to scale)

$$\int_{T_0}^{T_3} u(t) \cdot i(t) dt \tag{4}$$

The knowledge of the timepoints $T_0 \ldots T_3$ allow debugging and improving the software since they can be used to figure out when delays during the transmission occur. In the lab, it is even possible to connect the markers of the communication partner, which allows an even better insight in the communication process.

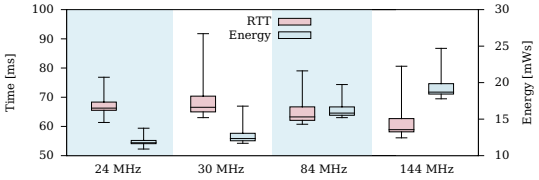
B. Network Setup

Both, data throughput and latency have been measured between *IBR-DTN* and *miniDTN* for the data bundles of the sizes and clock frequencies of the MCU given in Section IV. On both systems the datagram-based CL [14] `dgram:udp` or `dgram:lowpan` have been used.

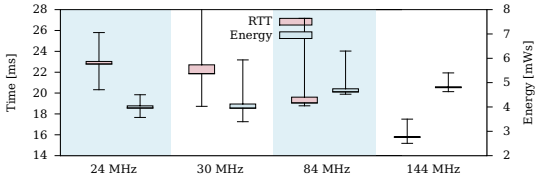
For evaluating the transmissions on Fast-Ethernet, we built a dedicated network consisting of the three nodes (*Raspberry Pi* and two *miniDTN* boards) and a switch.

IBR-DTN was running on a *Raspberry Pi Model B* and a *Core i7-3770*-based PC. The latter is for comparison only, since in our use case, we aim at embedded low-power SBCs in the network. We did not attach an IEEE 802.15.4 radio to the PC. DTN implementations other than *IBR-DTN* do not support the datagram-based CL at the moment. Thus, all tests could only be performed with *IBR-DTN*.

During the wireless measurements, the nodes have been placed on a desk. Tests have been conducted in an office environment of our lab where many participants are occupying the 2.4 GHz band using a variety of different technologies. This may lead to collisions during data transfers, but packet loss may also happen in real world deployments due to many factors like interference or path loss.



(a) 64x64 Byte over IEEE 802.15.4 to Raspberry Pi



(b) 64x64 Byte over IEEE 802.15.4 to Amphisbaena

Fig. 8. Data Transfer of 64 Byte to Raspberry Pi and Amphisbaena using IEEE 802.15.4

C. Measurement Results

The communication performance is heavily dependent on the communication link and even the communication partner. For example, Figure 8 shows the time and energy consumption of a data transfer of 64 Byte from *Amphisbaena* to a Raspberry Pi respectively another *Amphisbaena* board. As expected, the time of the data transfer decreases with an increasing clock frequency of the MCU while the energy demand is also rising. The higher processing time (60 ms to 70 ms instead of 15 ms to 25 ms) when communicating with a Raspberry Pi is a result of the Linux operating system and the architecture of *IBR-DTN* compared to the lightweight *miniDTN* implementation running on a real time operating system.

In some cases, the clock frequency of the MCU does not have a great effect on the time for a data transmission. This is especially true when using Ethernet for communicating with a Raspberry Pi. An example is shown in Figure 9. For all clock frequencies, the RTT is nearly constant around 63 ms while the energy demand is increasing from 22 mWs per data exchange to 33 mWs. The amount of data transferred (64/64 Byte in Figure 9(a) and 1024/1024 Byte in Figure 9(b)) has hardly any influence.

Most of the delays observed in Figure 9 are the results of processing overhead on the Raspberry Pi. This is proven by measurements of the same data sizes from an *Amphisbaena* board to *IBR-DTN* running on a PC with a faster CPU (Intel Core i7-3770). The RTT in this scenario is much lower between 2 ms and 5 ms for all clock speeds.

Figure 10 depicts a scenario in which a large amount of data has been transferred between two *Amphisbaena* boards using IEEE 802.15.4. Here we can see that the RTT reaches its minimum not at the highest clock frequency but at 84 MHz. In this configuration, the fastest clock speed should not be chosen since it does not decrease the time any further. Only the energy consumption increases when switching to 144 MHz.

1) Comparison of Energy Demand: Figure 11 shows the energy consumption of a Raspberry Pi using the `dgram:udp` CL when communicating with an Intel Core i7-3770 based PC. This measurement has been done to show the baseline for what performance is possible when using a Raspberry Pi. Comparing only these figures, it would not make much sense to boot the SBC which is quite expensive as shown in TABLE I.

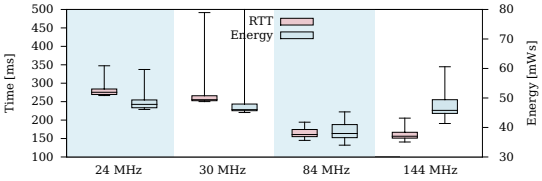


Fig. 10. Large data transfers using IEEE 802.15.4 (1024x1024 Byte)

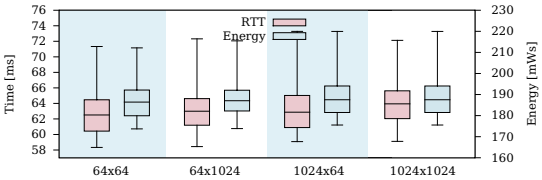


Fig. 11. Energy demand of Raspberry Pi

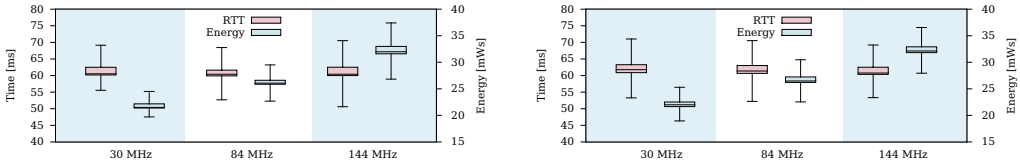
TABLE I
ENERGY CONSUMPTION FOR BOOTING SBC UNTIL FIRST COMMUNICATION IN DTN.

SBC Type	Medium	Boot-Time	Energy
Raspberry Pi Mod B[1]	Ethernet	49.3 s	69.230 Ws
Raspberry Pi Mod B[1]	WLAN	†103.23 s	203.963 Ws
Raspberry Pi2 Mod B[2]	Ethernet	23.0 s	21.927 Ws
Raspberry Pi2 Mod B[2]	WLAN	24.1 s	30.157 Ws

[1] Running Raspbian *wheezy*
[2] Running Raspbian *jessie*
† Association with WLAN AP may take very long

For determining the energy demand of the boot process, we configured the Linux to start the DTN daemon as early as possible after booting. The SBC connects automatically to a network (WLAN, which may take a very long time for establishing the association with the access point, or via Ethernet). Directly after starting the DTN daemon, a bundle is to be sent to another host which measures the time and energy demand of the boot process. This gives realistic values for the time it takes to be able to communicate after booting.

Comparing these values and considering only the single data transfers, it seems it would never be efficient



(a) 64x64 Byte (b) 1024x1024 Byte

Fig. 9. Data Transfer to Raspberry Pi using Ethernet

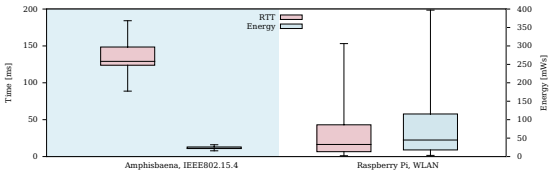


Fig. 12. Transmitting 1000*1024 Byte to a Raspberry Pi; data shown are for single bundles

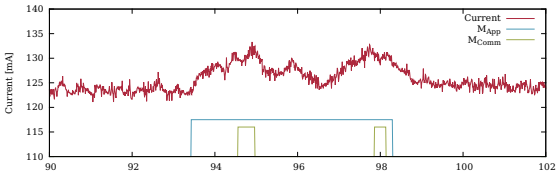


Fig. 13. Data Transfer with markers

to boot the SBC. On the other hand, we also transferred larger amounts of data in a row. For this evaluation, we did not send application-based ACKs other than the automatically generated ACKs by the Bundle Protocol.

We transmitted 1000 bundles of 1024 Bytes in two scenarios being 1) from the *Amphisbaena* board to a Raspberry Pi using IEEE 802.15.4 and 2) between two Raspberry Pis using WLAN. For the transmission between the Linux boards, we used the TCP-CL, which has not been implemented for *miniDTN* since it would not perform well over IEEE 802.15.4. The results are shown in Figure 12. Here we can see, that the transfer of *each single bundle* (1000 in total) with the size of 1024 Byte each takes much longer using IEEE 802.15.4, while the energy demand is in the same magnitude for both cases.

Using a full featured operating system like Linux on a SBC would allow to transmit larger bundles instead of rather small ones like 1024 Byte each. This can reduce the overhead significantly. Transmitting data amounts in the magnitude of Megabytes is infeasible for low power links like IEEE 802.15.4 which has a gross data rate of 250 kBit/s [16] and relatively small Packet Data Unit (PDU) sizes.

2) Markers: As an example, a real world measurement of a data transfer is shown in Figure 13. At about 93.5 ms within the measurement the user process starts to send data. The blue marker M_{App} shows the time in which the user process is active, the green marker M_{Comm} indicates that the Ethernet PHY is transmitting data. In this case, the process takes around 5 ms. The current consumption is shown by the red graph. One can see that the current rises during the processing and communication and falls afterwards to the idle level. Capacitors on the board lead to a delay of the process. Due to this fact, we transmitted the data without any delay in our measurements.

TABLE II
CURRENT CONSUMPTION ACCORDING TO DATASHEETS.

Component	Current @3.3 V	Power
MCU_{active} (168 MHz)	43.7 mA	144.2 mW
MCU_{idle} (25 MHz)	5 mA	16.5 mW
$MCU_{PowerDown}$ †	4 µA	13.2 µW
IEEE 802.15.4 $Radio_{RX}$	12 mA	39.6 mW
$Ethernet_{100\,Base-T}$	45 mA	148.5 mW
micro SD $Card_{R/W}$	100 mA	330 mW
Raspberry Pi (@5 V)	800 mA	4000 mW

†Can only be woken up by RTC.

3) Energy consumption of Amphisbaena: TABLE II shows the current and energy consumption of the individual components on the PCB. According to the datasheets, running at the highest possible clock frequency of 168 MHz, the MCU's current consumption is moderate with 43.7 mA. The actual value depends on the peripherals being used. Depending on the application and use case, the MCU may be clocked down. For example, if there are no tasks, the MCU can enter idle mode with a significantly lower clock rate and be woken up by an interrupt from the radio.

Accessing an SD card is very expensive in terms of energy [17], [18]. In order to keep the whole system available for a long time, it must be ensured that it should only be accessed when needed.

As seen in TABLE II, the energy consumption for the worst case (Radio TX, 100 Base-T Ethernet, SD transfer) is around 670 mW at 3.3 V. Fortunately, in most *wireless* scenarios, the Ethernet link would probably be not in use. Generally, the schedule for all components can be very different for each use case. This makes it impossible to calculate an average energy consumption. During our measurements, we have shown that the actual energy consumption is in many cases higher than given in the

data sheets.

The static measured energy consumption is shown in TABLE III at different clock speeds. Those values are measured when the DTN implementation is running, but not active data transfers (except the usual broadcasting of announcement messages) are active. We did not measure the current consumption of each component, but only the consumption of the complete *Amphisbaena* board.

TABLE III
MEASURED ENERGY CONSUMPTION OF *Amphisbaena*.

CPU Frequency	With Ethernet	Without Ethernet
30 MHz	354.6 mW	141.9 mW
84 MHz	432.3 mW	207.8 mW
168 MHz	529.4 mW	309.4 mW

D. Discussion of the results

We have shown that the costs of data transmissions are different for each scenario. In some cases, the clock speed of the MCU has hardly any influence on the time a data transfer takes which means that in most cases the lowest clock frequency should be chosen for this scenario (Figure 9). In other cases, the highest clock rate results in faster transmissions while increasing the energy demand (Figure 8). Conducting measurements for all combinations of clock speeds and data sizes are important since the "best" (either time or energy consumption) values are not necessarily be found at the highest or lowest clock frequency (Figure 10).

Our measurements have shown that the energy demand of the Ethernet PHY is much higher than given in the specification and shown in TABLE II, it is around 60 mW higher than expected. The current consumption of the MCU is also higher.

VI. CONCLUSION

In this paper, we have presented the versatile two-platform node "Amphisbaena" for a Delay-Tolerant Network (DTN) which is deployed outdoors. The option to charge a battery by a PV panel efficiently using MPPT allows the node to run self-sustaining for a long time due to its low power consumption while still being able to communicate. Our novel concept of a seamless switching between two platforms offers new possibilities for outdoor deployments. Measurements offer the opportunity to decide for which kind of communication which link or even system should be used.

The lightweight DTN implementation running in a real time operating system on a 32 Bit MCU outperforms Linux based implementations due to its reduced overhead, especially for small data amounts to be exchanged. If Megabytes of data have to be transmitted, the software on the MCU can decide to boot the full featured Linux on a SBC. A communication partner in the network would only realize that a new communication link is available. Further investigations on the announcement of the possibility to boot the SBC will be done in the future to benefit from this for routing strategies.

REFERENCES

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture," RFC 4838 (Informational), Internet Engineering Task Force, Apr. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4838.txt

[2] K. Scott and S. Burleigh, "Bundle Protocol Specification," RFC 5050 (Experimental), Internet Engineering Task Force, Nov. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5050.txt

[3] S. Schildt, J. Morgenroth, W.-B. Pöttner, and L. Wolf, "Ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation," *Electronic Communications of the EASST*, vol. 37, pp. 1–11, Jan 2011.

[4] N. Mishra, K. Chebrolu, B. Raman, and A. Pathak, "Wake-on-WLAN," in *Proceedings of the 15th international conference on World Wide Web - WWW '06*. New York, New York, USA: ACM Press, may 2006, p. 761.

[5] N. Mishra, D. Golcha, A. Bhadauria, B. Raman, and K. Chebrolu, "S-WOW: Signature based Wake-on-WLAN," in *2007 2nd International Conference on Communication Systems Software and Middleware*. IEEE, jan 2007, pp. 1–8.

[6] D. Spenza, M. Magno, S. Basagni, L. Benini, M. Paoli, and C. Petrioli, "Beyond duty cycling: Wake-up radio with selective awakenings for long-lived wireless sensing systems," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, apr 2015, pp. 522–530.

[7] I. Foche-Pérez, J. Simó-Reigadas, I. Prieto-Egido, E. Morgado, and A. Martínez-Fernández, "A dual IEEE 802.11 and IEEE 802.15–4 network architecture for energy-efficient communications with low-demanding applications," *Ad Hoc Networks*, Sep. 2015.

[8] N. Banerjee, M. D. Corner, and B. N. Levine, "An Energy-Efficient Architecture for DTN Throwboxes," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*. IEEE, 2007, pp. 776–784.

[9] D. Lymberopoulos, N. B. Priyantha, and F. Zhao, "mplatform: A reconfigurable architecture and efficient data sharing mechanism for modular sensor nodes," in *Information Processing in Sensor Networks, 2007. IPSN 2007. 6th International Symposium on*, April 2007, pp. 128–137.

[10] M. Doering, S. Rottmann, and L. Wolf, "Design and implementation of a low-power energy management module with emergency reserve for solar powered dtn-nodes," in *Proceedings of the 3rd Extreme Conference of Communication (ExtremeCom 2011), Manaus, Brazil*, 9 2011.

[11] *LongRange Transceiver*, RF Solutions Ltd. [Online]. Available: http://www.rfsolutions.co.uk/acatalog/DS-RFLoRa.pdf

[12] *STM32F405xx/STM32F407xx: ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera*, ST microelectronics, Oct 2015, docID022152 Rev 6.

[13] G. von Zengen, F. Büsching, W.-B. Pöttner, and L. Wolf, "An Overview of μDTN: Unifying DTNs and WSNs," in *Proceedings of the 11th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze (FGSN)*, Darmstadt, Germany, 9 2012.

[14] J. Morgenroth, "Event-driven Software-Architecture for Delay- and Disruption-Tolerant Networking," Ph.D. dissertation, Technische Universität Braunschweig, Jul. 2015. [Online]. Available: http://www.digibib.tu-bs.de/?docid=00061364

[15] R. Hartung, U. Kulau, and L. Wolf, "Distributed Energy Measurement in WSNs for Outdoor Applications," in *IEEE SECON 2016 Conference Proceedings*, London, UK, Jun. 2016, accepted for publishing.

[16] "Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans)," *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pp. 1–314, Sept 2011.

[17] *TS256M-2GUSD*, Transcend.

[18] *Samsung SD & MicroSD Card product family*, Samsung Electronics, Nov 2013, rev. 1.2.1.