An Overview of μDTN: Unifying DTNs and WSNs

Georg von Zengen, Felix Büsching, Wolf-Bastian Pöttner, and Lars Wolf
Technische Universität Braunschweig
Institute of Operating Systems and Computer Networks (IBR)
Mühlenpfordtstr. 23, 38106 Braunschweig
Email: {vonzeng | buesching | poettner | wolf}@ibr.cs.tu-bs.de

***Abstract*—Wireless Sensor Networkss (WSNs) nowadays are deployed in miscellaneous fields of application. Some use cases – especially those which include mobility – would benefit from a communication protocol that handles connection losses by design instead of assuming a continuous end-to-end connection. In this work, we describe the implementation of a Delay Tolerant Network (DTN) protocol for WSNs, that is based on the Bundle Protocol (BP) specifications. The BP specifies a "store, carry and forward" network protocol that can be used in scenarios with changing network topologies and unstable links. With μDTN, we present a DTN implementation for Contiki OS which is compatible to Contiki's network stack and is able to handle disruptions without packet loss. Furthermore, it is interoperable with BP implementations running on Linux.**

I. Introduction

DTN has its origins in interplanetary communication, where usually a continuous end-to-end connection cannot be assumed [1]. In harsh deep-space environments, traditional communication protocols are inappropriate due to several reasons such as long distances resulting in high latency, which again makes connection oriented protocols like TCP unmanageable, and the absence of a continuous end-to-end connection requires a different approach than common communication protocols. The DTN architecture [2] bases on a "store, carry and forward" concept and is able to compensate these shortcomings. The BP [3] is a specification for an implementation of this concept.

Looking closer at WSNs, it turns out that some of the problems are the same for interstellar communication and communication in WSNs. The unreliable and unpredictable radio channel is just one problem that can be addressed by a protocol that is intentionally tolerant to delay or disruption. Adding mobility to a WSN implies connection losses, temporary separated nodes and split networks. This can either be handled by an application in combination with more or less intelligent routing, or, as in our approach, by the implementation of a generic DTN protocol for WSNs.

Implementing a well-known protocol such as the BP has the additional benefit of interoperability with existing BP implementations for other platforms. This enables seamless network integration and reduces hassles with new and proprietary network protocols.

To overcome the stated problems, this paper presents our BP implementation for 8- and 16-bit microcontrollers called μDTN. It is integrated into the Contiki network stack and sends bundles directly in IEEE 802.15.4 [4] radio frames. It is based on Contiki OS and is interoperable with IBR-DTN [5] for Linux.

The remainder of the paper is organized as follows: In Section II related work regarding DTN-approaches in WSNs is presented. The architecture and functionality of μDTN for Contiki is detailed in Section IV, after some design decisions in Section III. A preliminary evaluation is given in Section V. Finaly, Section VI concludes the paper and provides the resources for downloading μDTN.

II. Related Work

Several publications have used DTN techniques in WSN context, without using the standardized BP. One early example of such projects is ZebraNet [6], which aims at tracking wildlife in Kenya. Seal-2-Seal [7] tracks contacts between wild animals, while LUSTER [8] aims at monitoring environmental parameters to be used by ecologists. All those projects use a proprietary delay tolerant communication protocol, whereas also the BP would have been suitable. These examples show, that a need for delay tolerant communication in WSNs exist.

Two published DTN implementations for WSN nodes exist. DTNLite [9] implements DTN concepts on TinyOS, but does not use the BP. ContikiDTN [10] uses the BP over TCP Convergence Layer (TCPCL) in IEEE 802.15.4 networks. To the best of our knowledge, no existing solution for using the BP in IEEE 802.15.4 networks without the need for existing layers 3 and 4 is available.

We conclude, that previous DTN implementations for WSN nodes either did not use the BP or are based on IP for hop-by-hop communication.

III. Design Decisions

In order to make the (comparably) heavy Bundle Protocol suitable for the limited resources of sensor nodes, we had to take a number of design decisions.

In contrast to all other approaches, μDTN does not build upon traditional transport- and network-layer protocols like TCP/IP. It is located directly above the MAC-Layer and is designed around the IEEE 802.15.4 convergence layer. The support of multiple convergence layers has been neglected in favor of higher performance and code efficiency.

Because of implementation efficiency for wireless sensor nodes, which are usually limited in processing power and

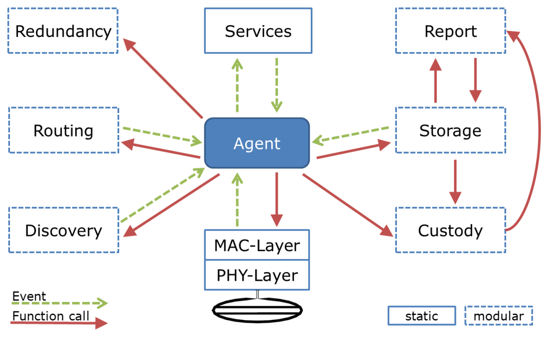

Fig. 1. Block diagram of μDTN's architecture.

memory, some compromises in compatibility to the Bundle Protocol specification were necessary: Compressed Bundle Header Encoding (CBHE) [11] is the only supported variant of μDTN's primary block. CBHE leads to smaller bundles that are easier to process and have less communication overhead, because here addresses are coded as numbers instead of strings. In order to further reduce communication and processing overhead, μDTN uses a slightly modified format for status and custody reports. Instead of using the (specified) string-based EIDs, μDTN also uses CBHE-style addresses in these bundles.

The current version of μDTN does not support fragmentation of bundles. Splitting bundles into fragments the size of the MTU of IEEE 802.15.4 leads to significant overhead and has therefore be neglected. We are currently working on mechanisms for segmentation of bundles between two nodes to overcome the overhead. Consequently, the current version can only send bundles that fit into one IEEE 802.15.4 radio frame.

IV. Architecture

μDTN is designed and implemented for the Contiki OS which offers a significant number of supported hardware platforms. μDTN uses a modular architecture which allows configuring different implementations of the various components during compile time. This makes the design flexible and efficient at the same time to cope with the tightly limited resources of wireless sensor nodes. During the design and implementation process, we have prioritized the memory efficiency higher than the actual speed because memory is a limited resource on most of the Contiki target platforms.

μDTN is split up into different modules as shown in Figure 1. The "Agent" is the central entity and cannot be exchanged while other modules may have different implementations. Interfaces are defined by structs that contain function pointers so that every module can use its own naming convention. Modules have to be selected at compile-time, dynamic loading is not supported at the moment.

Currently, we have interfaces and implementations for the following modules:

- *Discovery*: Discovers other nodes to transfer bundles
- *Routing*: Decides which bundles are send to which nodes
- *Storage*: Stores bundles on a node
- *Redundancy*: Checks if a given bundle has been received before
- *Custody*: Decides about custody questions, handles custody communication
- *Status Reports*: Handles bundle status reports

In the remainder of this section, we will detail information about the interfaces of the different modules as well as the currently available implementations.

A. Agent

μDTN is implemented as a single daemon protothread [12] process that we call "Agent". This process acts as a central entity and enables event-based communication between the different modules of μDTN as well as event-based communication with applications that want to send and/or receive bundles. While modules can send events to the agent, the agent interprets them and calls the appropriate functions of the interested modules. Also, the agent is responsible to communicate with the underlying layers of the stack to eventually send and receive radio frames. Due to efficiency considerations, our implementation supports only one convergence layer building on the Contiki Netstack interface to communicate with other nodes.

B. Data Structures

To temporarily keep bundles in RAM a C structure is used. This structure contains a Contiki MMEM pointer to the encoded representation of the bundle but in addition also keeps a copy of the attributes that are used frequently during processing, routing or storing a bundle. Also, the structure keeps a table of offsets and lengths of the different attributes in the primary bundle block to allow for fast random access to all attributes. This also enables to quickly check whether an SDNV value has changed its length whenever the value is changed.

C. Discovery Module

The discovery module is responsible to discover nodes than can be reached via the IEEE 802.15.4 radio link. Newly discovered nodes are reported to other modules, so that the routing module can send bundles to this neighbour. μDTN currently has two alternative discovery mechanisms.

1) Reactive Node Discovery: Different to most common BP implementations, μDTN can use a reactive discovery approach. A node that has bundles to send periodically broadcasts a beacon to all neighbors. Nodes receiving the beacon will respond and bundles can be exchanged. This adaptive way of discovering nodes is expected to be more energy efficient than traditional periodic broadcasting approaches.

2) IP-ND based Node Discovery: DTN IP Neighbor Discovery (IPND) [13] is a well-known mechanism to discover nodes in IP-based networks. While the specification is IP-specific, the employed data format is generic and can be easily adapted to IEEE 802.15.4-based DTNs. We have implemented a discovery module that periodically broadcasts IPND-style packets in IEEE 802.15.4 radio frames to discover other nodes. When receiving the beacons, the receiver can infer the originators IEEE 802.15.4 address, the PAN and the EID of the node.

D. Routing Module

The routing module is responsible to decide which bundles should be forwarded to which neighbor in range. It is informed whenever a new neighbor is discovered. Also, it is notified whenever a new bundle is stored or a stored bundle is deleted. The routing module usually waits for the discovery of a new neighbor. Then, the module will decide if one or multiple of the stored bundles should be forwarded to that new neighbor. All bundles that should be transmitted are stored in a list and passed on to the agent as an event. The daemon will then interpret the event and forward all bundles to the respective neighbor. For each successfully transmitted bundle, the routing module receives a callback from the agent to keep track of the current status of the bundles in the list.

We have implemented a modified version of flooding, in which bundles are sent to a maximum of n neighbors before they are deleted. The implementation avoids to send bundles to nodes from which this bundle was received previously and avoids sending bundles back to the originator. Bundles that have successfully been delivered to the destination are deleted from storage.

E. Storage Module

The storage module is responsible to store a bundle on a local medium and to read and return the bundle upon request by the agent. The BP specifies, that the creation timestamp, the sequence number, the fragment offset and the source EID of the bundle uniquely identify a bundle. Since comparing a tuple of this information is time consuming, the storage module assigns an internal number for each bundle that allows a more efficient access to this specific bundle. The length of this number if currently limited to 16 bit to maintain acceptable speed on most platforms but could be extended to 32 bit (or even more) depending on the platform.

1) Storage based on CFS: Contiki comes with the Coffee File System (CFS) [14] which allows to store files on the flash that is present on most popular sensor nodes. We have implemented a storage module that stores each bundle in a separate file of the CFS. In addition, the storage module keeps a list of currently stored bundles in memory. The list includes metadata such as the time at which the bundle has been received, the size of the bundle, the lifetime of the bundle and the internal number of the bundle that also resembles the CFS filename. The storage module regularly checks the lifetime of all stored bundles and eventually deletes a bundle if the lifetime has elapsed. Bundles that are deleted are signaled to the agent including a reason for deletion to be used in the status and custody reports. Upon startup of a node, the storage module repopulates the list by reading the necessary information from flash memory.

A storage module based on the FAT filesystem is in preparation.

2) Storage based on volatile RAM: We have implemented an additional storage module that stores all bundles in the RAM of the microcontroller. As for the CFS-based storage a list with meta information for each bundle is kept in memory. Each entry of this list now contains a pointer to the MMEM block in which the full bundle is stored in memory. While bundles stored in memory will not survive node reboots, write times to RAM are orders of magnitude faster than in flash. Storing a bundle to flash with CFS may take around 512 ms, while storing such bundle to RAM takes less than 4 ms.

F. Redundancy Module

Since bundles may be duplicated and transported on different paths to the destination, the destination node has to avoid sending multiple copies of the same bundle to applications. The redundancy module has the responsibility to decide whether a bundle should not be delivered to a local service because it is a duplicate. We have designed this functionality as a separate module since comparing the tuple of information that is used to uniquely identify a bundle is time consuming and can be optimized to specific application requirements.

G. Status Reports Module

Status reports are bundles of a special type that inform nodes about the status of a bundle. The format of the implemented status reports deviates slightly from the specification as outlined in Section III. This module is responsible to create and interpret status report bundles and is invoked either by the agent or by the custody module.

H. Custody Module

The custody module is primarily responsible to create and interpret the custody reports. Further, the module has to decide whether a node can become the custodian of a given bundle. Each node maintains a list with all bundles for which the nodes is the custodian. Whenever the node accepts custody for a new bundle, the meta information of this bundle is inserted into the list. Further, the nodes sets a timer to allow retransmission in case that no other node wants to take custody for this bundle.

I. Application Programming Interface

The Application Programming Interface (API) can be used by applications to send and receive bundles. The communication between applications (services in BP terminology) in Contiki Protothreads and the agent is realized completely asynchronously using events. First of all, an application has to register an endpoint by filling a struct with parameters and appending the structure to the appropriate event that is sent to the daemon. After the registration, an application is ready to receive bundles. Incoming bundles are signaled to the application by an event originating at the agent. The application receives the complete bundle structure and has to extract the relevant information. To send a bundle the application has to allocate a bundle structure and then use various API functions to fill in the necessary fields and attributes of the bundle. The API also allows adding custom blocks to the bundle while intermediate daemons that do not understand the block typically forward the block. Finally, the application sends an event to the daemon with the bundle structure appended and the daemon will then take care of storing and transmitting the bundle.

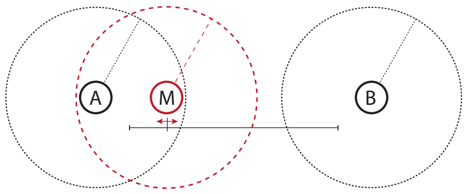

Fig. 2. Node M is moving between the communication radii of nodes A and B. In a DTN, node M could store, carry and forward bundles between A and B.

V. Evaluation

We have performed extensive evaluation with μDTN, but have to omit detailed results due to space limitations. The application-layer throughput of μDTN on INGA with a bundle payload size of 80 bytes is 47.58 kbit/s. Furthermore, we have measured the round-trip round-trip to be 23.13 ms in a single-hop scenario in the university laboratory.

The evaluation of the disruption tolerant use case was performed in the Cooja Simulator. We designed a scenario as shown in Figure 2: Nodes A and B are placed in a setting, where they could not directly communicate with each other. Using BonnMotion [15], a third node (node M) moved between these nodes and so was able to collect and deliver packets from A to B and vice versa. Movement time was 60 seconds for one direction, connection time (M in A's or B's communication radius) was 15 seconds in each case. We used the "CSMA" MAC layer and the "nullrdc" RDC implementation. Further, a fixed bundle creation frequency of 0.07 Hz has been configured to avoid overloading the memory of the nodes. The full μDTN stack including discovery and routing as outlined in Section IV is used. The result of this simulation is, that in all 3 simulation runs μDTN was able to deliver all 1000 packets generated by the sender. We conclude, that μDTN can handle network disruptions without loosing application data. We have also evaluated interoperability with IBR-DTN and found, that bundles can be transmitted in both directions. More evaluation results can be found in [16].

VI. Conclusion

In this paper we presented μDTN, a BP implementation for Contiki. We have outlined the design decisions as well as the architecture of μDTN. While certain aspects are not completely compliant with the protocol specification, basic interoperability is possible.

The evaluation has shown μDTN's ability to cope with disruptions in the network. Furthermore we have seen, that the throughput is acceptable and improvements are planned. Further evaluation (in a testbed using INGA [17]) – which was omitted due to space reasons – showed, that μDTN is able to sustain 0 % packet loss even in networks with very high link losses.

The μDTN implementation is under open-source license and can be downloaded at:
http://www.ibr.cs.tu-bs.de/projects/mudtn/.

Acknowledgments

The Lower Saxony research network "Design of Environments for Ageing" acknowledges the support of the Lower Saxony Ministry of Science and Culture through the "Niedersächsisches Vorab" grant programme (grant ZN 2701).

References

[1] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 27–34, New York, NY, USA, 2003. ACM.

[2] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay-Tolerant Networking Architecture. RFC 4838 (Informational), April 2007.

[3] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), November 2007.

[4] The Institute of Electrical and Electronics Engineers, Inc. Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). IEEE 802.15.4-2006, September 2006.

[5] Sebastian Schildt, Johannes Morgenroth, Wolf-Bastian Pöttner, and Lars Wolf. IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation. *Electronic Communications of the EASST*, 37:1–11, Jan 2011.

[6] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li Shiuan Peh, and Daniel Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *SIGOPS Oper. Syst. Rev.*, 36(5):96–107, October 2002.

[7] A. Lindgren, C. Mascolo, M. Lonergan, and B. McConnell. Seal-2-Seal: A delay-tolerant protocol for contact logging in wildlife monitoring sensor networks. In *Proc. of MASS 2008*, 29 2008-oct. 2 2008.

[8] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, and J. Porter. LUSTER: wireless sensor network for environmental research. In *Proc. of SenSys '07*, pages 103–116. ACM, 2007.

[9] Rabin Patra and Sergiu Nedevschi. DTNLite: A Reliable Data Transfer Architecture for Sensor Networks. Technical Report CS294–1, Berkeley, 2003.

[10] Max Loubser. Delay Tolerant Networking for Sensor Networks. Master's thesis, Swedish Institute of Computer Science, 2005.

[11] S. Burleigh. Compressed Bundle Header Encoding (CBHE). RFC 6260 (Experimental), May 2011.

[12] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, pages 29–42, New York, NY, USA, 2006. ACM.

[13] D. Ellard and D. Brown. DTN IP Neighbor Discovery (IPND). *IETF Draft*, 2010.

[14] Nicolas Tsiftes, Adam Dunkels, Zhitao He, and Thiemo Voigt. Enabling Large-Scale Storage in Sensor Networks with the Coffee File System. In *Proceedings of the 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2009)*, San Francisco, USA, April 2009.

[15] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla, and Matthias Schwamborn. Bonnmotion: a mobility scenario generation and analysis tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 51:1–51:10, ICST, Brussels, Belgium, Belgium, 2010.

[16] Wolf-Bastian Pöttner, Felix Büsching, Georg von Zengen, and Lars Wolf. Data Elevators: Applying the Bundle Protocol in Delay Tolerant Wireless Sensor Networks. In *Proceedings of the 9th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (IEEE MASS 2012)*, 2012.

[17] Felix Büsching, Ulf Kulau, and Lars Wolf. Demo: INGA - An Inexpensive Node for General Applications. In *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*, SenSys '11, Seattle, WA, USA, 2011. ACM.