Electronic Communications of the EASST
Volume 37 (2011)

Workshops der wissenschaftlichen Konferenz
Kommunikation in Verteilten Systemen 2011
(WowKiVS 2011)

IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation

Sebastian Schildt, Johannes Morgenroth, Wolf-Bastian Pöttner and Lars Wolf

10 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation

Sebastian Schildt, Johannes Morgenroth, Wolf-Bastian Pöttner and Lars Wolf

Institute of Operating Systems and Computer Networks
Technische Universität Braunschweig,
Braunschweig, Germany
Email: `[schildt|morgenroth|poettner|wolf]@ibr.cs.tu-bs.de`

Abstract: In this paper we present IBR-DTN, a lightweight, modular and portable Bundle Protocol implementation and DTN daemon. IBR-DTN is especially suited for embedded platforms, which allows to leverage the benefits of a fully compliant Bundle Protocol daemon in cost sensitive distributed sensing applications. We line out IBR-DTN's extensible software architecture and introduce the modules included in the standard IBR-DTN distribution. To give an impression of the performance that can be expected when using IBR-DTN, we perform a series of benchmarks and compare the outcome with DTN2 performance, the reference implementation of the Bundle Protocol.

Keywords: DTN, Bundle Protocol, Embedded Linux

1 Introduction

Today it is accepted within the networking community that continuous end-to-end connectivity will not be available all the time, not even in wired networks. Especially in MANETs with low-powered devices the need for a store-and-forward architecture can not be ignored. In recent years the approach of Delay Tolerant Networking gained interest in the community for coping with the challenges of intermittently connected networks.

DTNs are a good choice for distributed data sensing and aggregation as presented in [LDP+07]. The standard protocol used in DTNs is the Bundle Protocol [SB07]. Unfortunately the Bundle Protocol reference implementation DTN2 is not suited for Embedded Systems, which precludes its deployment in many applications that are constrained by costs or energy. In this paper we present IBR-DTN which aims to be a fully compliant Bundle Protocol daemon and is designed to run on OpenWRT[1], which is a Linux distribution specifically built to run on embedded hardware (see section 3). An early precursor of IBR-DTN has been introduced in [DLMW08].

This paper will detail the overall software architecture of IBR-DTN. We also give some basic benchmarks comparing the performance with the DTN2 reference implementation.

[1] http://openwrt.org/

2 Related Work

In [LDP+07] Lahde et. al. presented a system that relies on DTN for the distributed sensing of air pollution in urban environments. A DTN based sensor network for lake water quality monitoring or urban noise monitoring was presented in [MGH+07]. While using embedded Linux nodes the authors did not use a Bundle Protocol compliant DTN implementation.

The most well known implementation of the Bundle Protocol is the DTN2 reference implementation[2]. It provides a flexible framework for DTN related experiments and can be configured and managed by a TCL console and configuration files. Extensions for routing, storage and convergence layers are easily attachable through XML interfaces. However, the extensive use of external libraries makes it difficult to port the DTN2 software to embedded systems. ION (Interplanetary Overlay Network) [Bur07], is another implementation of the Bundle Protocol that has been written by the JPL and is intended to be used in spacecrafts. It has been developed for VxWorks but can run on Unix-like systems such as Linux as well. There has also been some research about using DTN in low-power sensor networks, where nodes are only equipped with a few kByte of RAM and a small microcontroller [PN03]. However, due to resource constraints those approaches only aim at supporting basic DTN and Bundle Protocol concepts in WSNs and usually these implementations are not interoperable with the Bundle Protocol.

Several other implementations are currently in development. ByteWalla[3] is a Java implementation designed for Android devices. Other implementations for Android and Symbian devices and for Python exist, but those only implement limited functionality and do not provide the features of a full DTN daemon.

3 Supported Platforms

IBR-DTN is specifically designed for and tested against uClibc.uClibc is a streamlined C standard library for resource constrained systems. While the concrete amount of memory needed for operation depends on the configuration and usage scenario, we will give two exemplary numbers: On an x86 machine running IBR-DTN under Ubuntu for several days the resident set size (RSS) of the IBR-DTN daemon process is 3244 KiB, for a MIPS node with OpenWRT under similar conditions the RSS is 2428 KiB.

Currently IBR-DTN is targeting Linux plattforms. It is 64 bit safe and porting to other POSIX compliant OS's should be feasible with minimal effort. Besides a standard Linux desktop/server distribution, the main platform for IBR-DTN deployment is a *Ubiquiti RouterStation Pro* board running OpenWRT. The RSPro is based on an Atheros AR7161 SoC (MIPS 24K) running at 680MHz with 128MiB RAM and 16MiB Flash. The (physically) smallest node known to run IBR-DTN is a *Memsic (formerly CrossBow) IMote 2* sensor node. This node is based on a Marvell PXA271 (StrongARM) CPU running at 416 MHz with 32 MiB RAM and 32 MiB Flash. The node runs OpenEmbeddedand includes a TI ChipCon 2420 IEEE 802.15.4 compliant transceiver. The public DTNBone node of the IBR is a *Buffalo TeraStation Pro* NAS running a proprietary Buffalo firmware based on kernel 2.4. It is based on a Freescale MPC8241 SoC (MPC603e PPC) running at 266 MHz with 128 MiB RAM. IBR-DTN runs inside a OpenWRT chroot providing the needed

[2] https://sourceforge.net/projects/dtn/

[3] http://sourceforge.net/projects/bytewalla/

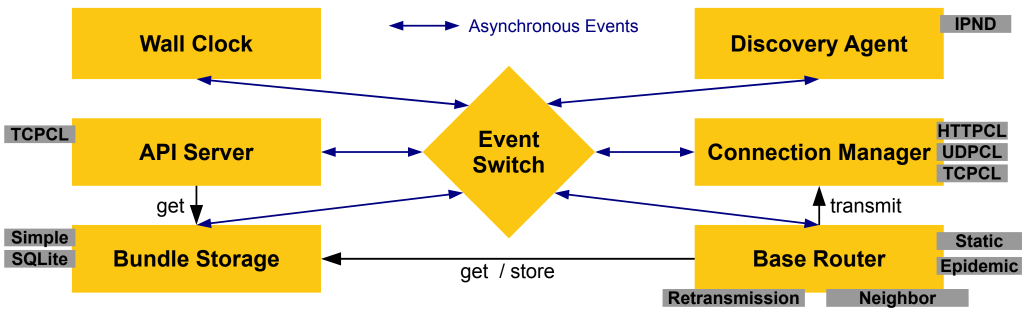

Figure 1: IBR-DTN architecture overview

libraries. Another NAS IBR-DTN has been sucessfully tested on is a *QNAP TS-219P NAS* using a QNAP proprietary Linux distribution running kernel 2.6. It is based on the Marvell Kirkwood 88F628 SoC (ARM11) and includes 512 MiB RAM. Additionally the *Technologic systems TS-7500 SBC* running OpenWRT is supported. This board uses a Cavium networks CNS2132 SoC (ARM 9) running at 250 MHz and includes 64 MiB RAM.

4 Architecture

Figure 1 shows an architectural overview of IBR DTN. As IBR-DTN is targeted for embedded platforms, it is fully compatible with uClibc, a system library specifically tailored for embedded systems. Two general guidelines have been followed developing for IBR-DTN: One goal is to keep external requirements (e.g. libraries), to a minimum. Often these are not available on certain platforms because there are not ported, or because the space needed for all dependencies would be prohibitive. A small set of external dependencies also makes sure that porting IBR-DTN to new platforms is relatively easy. Another goal is to keep the software as modularized as possible. The functionality in IBR-DTN is implemented in loosely coupled modules that communicate using an event mechanism. Thus at compile time IBR-DTN can be specifically tailored to the capabilities of the used target platform. Also if some functionality is dependent on external libraries (i.e. IBR-DTN's upcoming Bundle Protocol Security Extensions are based on OpenSSL) , the dependency applies to the optional module only.

In the following sections we will discuss IBR-DTN's standard modules.

4.1 Event Switch

At the core of IBR-DTN is the *Event Switch*. We have defined a set of standard events which the *Event Switch* dispatches to all relevant sub-modules. Events that trigger extensive processing in the module can be queued in a private work queue of the module's thread. This allows IBR-DTN to achieve a high degree of concurrency between the daemons modules. Existing and new IBR-DTN modules can receive and raise events to communicate with other parts of the daemon. Of course custom IBR-DTN modules can introduce new application-specific events. There exist standard events related to routing of bundles, operation of the storage and the availability of nodes.

4.2 Discovery Agent

The *Discovery Agent* supports node discovery through pluggable discovery modules. Currently we implement DTN IP Neighbor Discovery (*IPND*) version one and two as specified in [EB10] and IP-Discovery frames compatible to DTN2. The *Discovery Agent* generates Node Events related to the discovery or disappearance of neighbors. When a new neighbour is detected by the Discovery Agent, the Routing modules will check whether there are any Bundles that have to be transferred to the new found contact.

4.3 Connection Manager

Lower level protocols which provide connectivity between DTN daemons are called convergence layers (see [SB07]). In IBR-DTN, convergence layers are implemented as modules. The *Connection Manager* manages the instances of these modules based on the daemon configuration. Each convergence layer provides an interface to transfer bundles to neighboring nodes. If such a transfer succeeds or fails, an event is generated. Incoming bundles are also announced via a global event and stored in the *Bundle Storage*. IBR-DTN comes with four built-in convergence layers listed below:

- TCPCL: The TCP convergence layer compatible with IETF draft [DO08]. TCPCL uses a handshake mechanism between DTN daemons and includes the ability to split bundles into segments which are acknowledged by the receiving daemon.
- UDPCL: UDP convergence layer compatible with IETF draft [KO08]. UDPCL provides a very simple way of transfer bundles to other daemons. Since it requires that a bundle fits into a single UDP datagram, the maximum bundle size is limited.
- HTTPCL: A convergence layer that can use an HTTP server to send and receive bundles. This convergence layer is based on libcurl[4].
- LowPANCL: This convergence layer supports the 802.15.4 MAC protocol [IEE07] commonly used in sensor networks. Currently LowPANCL works with transceivers supported by the Linux ZigBee stack[5]. This convergence layer has been successfully used with iMote2 sensor nodes (see section 3).

4.4 Bundle Storage

As DTN's use a store-and-forward paradigm, a DTN node needs the capability to buffer bundles for an extended period of time. A *Bundle Storage* module provides an interface to store and retrieve bundles by different parameters, e.g. the bundles unique id or the destination. IBR-DTN supports three different types of bundle storage listed below:

- Memory: This is a non-persistent bundle storage and is used by default if no storage path is set. All bundles are kept in RAM in this case. The maximum amount of RAM used for bundles can be limited. The OpenWRT distribution of IBR-DTN includes measures to fallback to

[4] http://curl.haxx.se/
[5] http://sourceforge.net/apps/trac/linux-zigbee/

memory storage if there is a system failure that prevents the usage of on-disk storage. This enables a node to operate with reduced capabilities in case of a problem so it is still reachable for diagnostic and maintenance purposes.

- File based storage: If a storage path is set in the configuration, IBR-DTN will use a persistent storage based on simple files. The functionality is equal to the memory-based storage, but all bundles are serialized to disk. Thus bundles will survive scheduled daemon restarts as well as power failures. Since the bundles are no longer hold in memory, the amount of memory required by the daemon is reduced.
- SQLite: This storage uses an SQLite database as backend. The SQLite storage is currently in beta. It can store more meta information for bundles and is useful for more complex routing modules.

The base interface for all these modules ensures that all storage engines implement a basic set of functions and can be transparently interchanged without modifying other parts of the daemon. The Memory and File storages are examples of basic storages. The SQLite storage also can be used as a drop-in replacement for the File or Memory storage, but if the SQLite storage's advanced features are used by some module, this module will not be able to work with the basic storages.

4.5 Base Router

The *Base Router* manages different routing modules, which can operate concurrently. The routing modules receive events about arriving or departing nodes from the *Discovery Agent* and they are informed whenever new bundles arrive in the storage. If a routing module considers itself responsible for a given bundle, it can contact the *Connection Manager* and request the appropriate convergence layer to transfer the bundle to the next hop. IBR-DTN includes the following routing modules:

- Static: Routes and available links are configured statically, i.e. the convergence layer information for a given EID is supplied a priori through the configuration file. This module assumes that configured links are available permanently.
- Neighbor: The Neighbor routing module routes packets to nodes discovered by a *Discovery Agent*. When using *IPND*, nodes within the same subnet are reachable.
- Epidemic: IBR-DTN contains an epidemic routing [VB00] implementation. Epidemic routing is a variant of flooding. It aims to transfer all bundles to all nodes participating in epidemic routing and thus given unlimited resources it can guarantee the eventual delivery of a given bundle. Instead of using summary vectors, IBR-DTN uses a more efficient Bloom-Filter[BM04] mechanism which has been proposed as an optimization in [VB00]. As an extension to epidemic routing IBR-DTN manages a purge vector which will be distributed to all neighbors ensuring that delivered bundles are deleted throughout the network.
- Retransmission: When a convergence layer signals an error transmitting a bundle to the next hop, the bundle will be requeued and stays in the storage. Depending on the type of error (permanent, temporary) and the convergence layer the *Retransmission* module will try to repeat the transmission.

Routing modules receive events from the Discovery Agent and interact with the storage. Currently all shipped routing modules can be combined with any of IBR-DTNs bundle storages.

4.6 Wall Clock

The *Wall Clock* module determines the current global time in the DTN by evaluating the local host's clock. Unlike in other network protocols the timestamps used in Bundle Protocol count the seconds since 1/1/2000. In addition to basic time querying functionality this module provides a global time tick event, which is dispatched every second through the *Event Switch* to all other modules. This event is used by simple modules to initiate recurring tasks.

4.7 IBR-DTN API

Currently IBR-DTN reuses the bundle streaming protocol (part of the TCP convergence layer) to provide a socket based API interface. This can either be a TCP socket which allows to run the daemon and DTN applications on different machines, or a Unix Domain Socket, which is more efficient when operating locally.

To avoid reimplementing the complex bundle streaming protocol in each application, an IBR-DTN library can be linked to applications simplifying the creation of bundles. This approach has the advantage that all supported DTN features of the daemon are immediately available as they can be enabled by setting the appropriate bundle headers. However, since this interface does not support out-of-band messages, runtime configuration of e.g. routing modules is not possible. Also it is difficult to use this API from other environments than C++.

4.8 Extensions & Customization

One aim of IBR-DTN is to provide an easy and non-invasive method for the extension of core functionalities like storage, routing and connectivity. To achieve this, we introduced the *Event Switch* and used an threaded environment which allows a loose coupling between the different modules IBR-DTN consists of.

Only two steps are necessary to introduce a new module into IBR-DTN: Firstly, the new module needs to implement the base class of the correct family of modules (depending on whether the new module is for example a Routing Module or a Discovery Agent). Each module can choose to run as a separate thread or not by implementing the corresponding *Component* class. Secondly, the configuration in the *main()* method has to be extended to reflect the existence of the new module. The basic services a module has to provide are defined by the interface of the base module class. The access to other modules (e.g. wall clock, storage or convergence layer) is supported through global singleton objects. Asynchronous and one-to-many interaction is supported by events using the *Event Switch*.

4.9 Tools

For basic usage several standard shell tools are provided with the IBR-DTN distribution. `dtnsend` and `dtnrecv` can send and receive files and standard in-/output. The ping application `dtnping` sends out bundles to a DTN Endpoint Identifier and waits for a bundle with the same payload as reply. The response time is measured and printed out. With `dtntracepath` the path of a bundle can be discovered using "bundle forwarded" reports from each hop as defined in [SB07]. `dtntunnel` is an experimental tool that tunnels IP traffic through a DTN connection. Additionally, there are `dtninbox` and `dtnoutbox` which can transfer files between two directories on different computers. For simple applications the `dtntrigger` application provides a lightweight alternative to the API.

5 Evaluation

In this section we compare the performance of IBR-DTN to the DTN2 reference implementation. Both implementations have been measured using their respective default configuration. Additional configuration was only performed insofar it was necessary for the experiment. For this evaluation we used computers equipped with a Pentium IV 3.2 GHz including 2 GiB RAM running Ubuntu Linux. For the network experiments the nodes were connected using a full-duplex 1 GBit Ethernet link. The peak bandwidth of this configuration transferring raw TCP data is 480 MBit. We used the standard tools `dtnsend` and `dtnrecv` available for both IBR-DTN as well as DTN2. The IBR-DTN version used was 0.4.3, the DTN2 version was 2.70.

5.1 API and bundle storage performance

This experiment focuses on API and bundle storage performance and did not involve any network transfers. On each node we measured the time to store and receive 100 bundles of varying payload size. The 100 bundles were send one-by-one using the `dtnsend` application and subsequently retrieved en-bloc using the `dtnrecv` application. For IBR-DTN we used the *Memory* and *File* storage subsystem. For DTN2 we used the *BerkelyDB* storage which resides in the filesystem and the *Memory* storage which resides in RAM. For each test and storage we performed ten runs. The IBR-DTN daemon was running continuously and has not been restarted between subsequent runs. The DTN2 daemon had to be restarted between different bundle sizes, because we observed continuously deteriorating performance for each run. Test runs were not running in parallel: We made sure to receive all bundles of a previous run before starting to send new bundles. The results of this test can be seen in figures 2 and 3 with logarithmically scaled axes. We plot the arithmetic mean and standard deviation for each payload size.

For the store performance (figure 2) it can be seen, that IBR-DTN significantly outperforms DTN2 when using disk based storage and small payload sizes. IBR-DTNs store performance using disk-based storage is almost similar to the memory-based performance. As DTN2's performance improves when operating in memory it outperforms IBR-DTN in this testcase for very small bundle sizes. For bigger bundle sizes performance between DTN2's and IBR-DTN's RAM based storages is virtually identical. Also the non-increasing time for small bundle sizes shows that both IBR-DTN's and DTN2's performance is limited by the attainable bundle frequency and not by a bandwidth limit of the storage, i.e. the processing overhead per bundle is the limiting factor.

For the receive case (figure 3) IBR-DTN again shows similar performance in the file and memory storage cases. For Bundle sizes $\geq 5kByte$ IBR-DTN significantly outperforms DTN2 in both cases. For bundle sizes up to 500 bytes (DTN2) or up to 5000 bytes (IBR-DTN) performance is limited by the bundle frequency.

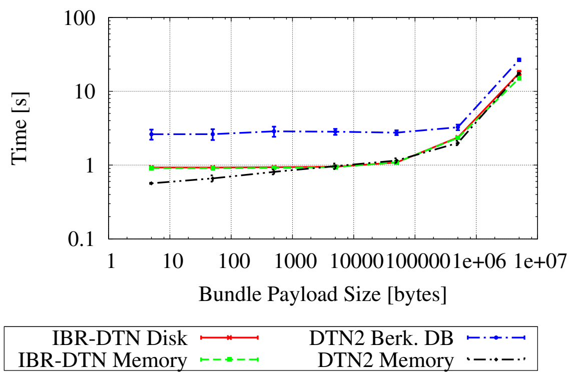

Figure 2: API store performance

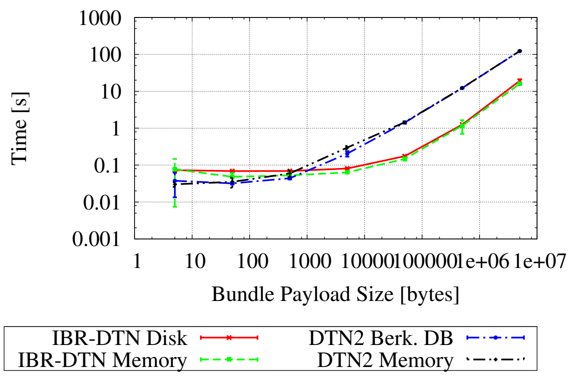

Figure 3: API receive performance

5.2 Networking performance

In this experiment we measure the network performance of IBR-DTN and DTN2. Two nodes are connected via Ethernet, but connectivity is blocked. On the sender node we use `dtnsend` to inject 1000 bundles destined for the receiver. After the bundles have been received by the sending daemon we open the connection and measure the time until all bundles have arrived at the receiver. This is a good indicate of the network performance of a daemon as this approach eliminates the influence of the API and the storage write performance on the sender. However this test still includes the time spent in the the sender's storage retrieval functions and the receivers storage writing functions. For IBR-DTN we used iptables for connection handling for DTN2 we used the builtin TCP API to add or remove the corresponding route on the fly.

We repeated this experiment with varying bundle payload sizes ranging from 5 bytes to 500 kBytes. Test runs were not running in parallel: We made sure that all bundles of a previous run have been received before starting to send new bundles. In this experiment we use the TCP convergence layer. For each bundle payload size and storage type we performed 10 runs.

The IBR-DTN daemon was running continuously and has not been restarted between subsequent runs. The DTN2 daemon had to be restarted between different bundle sizes, because we observed continuously deteriorating performance for each run when using the Memory storage. This effect manifests itself as high standard deviations in the DTN2 Memory test, as the test for each bundle size is repeated 10 times.

The results of this experiment can be seen in figure 4 with a logarithmically scaled x-axis. We plot the arithmetic mean and standard deviation for each payload size. First it can be seen that even with 500kByte bundle size neither DTN2 nor IBR-DTN were able to saturate the used link (480 MBit). IBR-DTN achieves an average speed of $\sim$ 310 MBit while DTN2 tops at only $\sim$ 245 MBit for the memory based storage and significantly slower ($\sim$ 98 MBit) for the more praxis relevant disk based storage. This is especially an issue for DTNs, where it is a goal to transfer as much data as possible during a possibly short contact. For IBR-DTN the reason is, that default TCP and Bundle Protocol parameters are adapted for wireless links with smaller bandwidth. We assume the same is true for DTN2. In all testcases IBR-DTN outperforms DTN2 in both the memory and disk-based storage cases. Even more IBR-DTNs disk based storage outperforms DTN2's memory based storage for all tested bundle sizes.

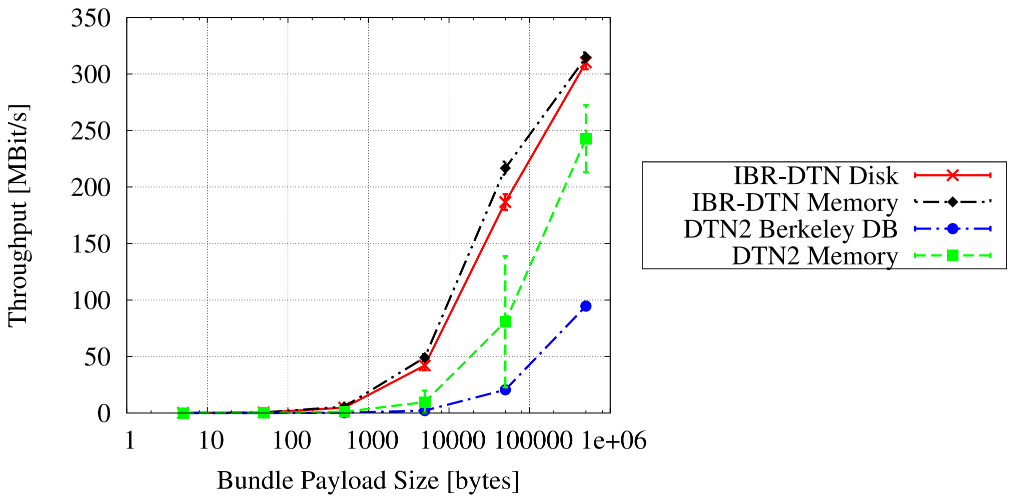

Figure 4: Throughput performance

5.3 IBR-DTN embedded performance

In this section we want to give a small insight into what performance can be expected running IBR-DTN on embedded hardware. For this experiment we used the Routerstation Pro platform (see sec. 3) running OpenWRT. As only IBR-DTN runs on this platforms there is no comparision with DTN2 or ION. For each test we did 10 runs with 100 Bundles of 1 MByte. We used disk storage on an SD Card. The RSPro are connected using a GBit LAN Port. Using this setup the peak iPerf performance between two RSpro boards using TCP is 251 MBit/s.

The total data throughput of IBR-DTN between two RouterStaion Pro's is 33.85 MBit/s (± 5.44 MBit/s). The throughput from and to the PC is with 45.36 MBit/s (± 12.10 MBit/s) slightly faster. The achieved performance is significantly lower than raw TCP throughput, as the Bundle Protcol is used on top of TCP, i.e. in addition to the TCP overhead the Bundle Protocol overhead comes into play. Also the speed of the used storage can limit the throughput, as all bundles have to be written to the SD Card. Still the result show, that it would be possible to saturate a 54 MBit/s 802.11g WiFi link (net data rate 22.5 MBit/s).

6 Conclusions and Future Work

We presented IBR-DTN, a lightweight, modular and highly portable DTN stack. IBR-DTN has a small memory footprint and can be used on virtually every platform supporting Linux, especially embedded platforms based on uClibc. This enables the deployment of a full featured DTN solution in wireless sensor networks. IBR-DTN is interoperable with DTN2, the Bundle Protocol reference implementation. In most cases its speed supersedes that of DTN2, although both implementations were not able to saturate the provided link using their respective default configurations.

Currently we are working on support for the Bundle Protocol security extensions [SFWL10]. In the future we will include optimizations for high bandwidth links into IBR-DTN. Also we intend to augment the IBR-DTN TCPCL API with a lightweight and ubiquitous alternative such as an

XML-RPC based API. The most recent IBR-DTN version can be found on the project page at http://www.ibr.cs.tu-bs.de/projects/ibr-dtn/.

Acknowledgements

This work has been partially supported by the NTH School for IT Ecosystems and by EFRE (European fund for regional development) project OPTraCom (W2-800-28895).

References

[BM04] A. Broder, M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics* 1(4):485–509, 2004.

[Bur07] S. Burleigh. Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol. In *4th IEEE Consumer Communications and Networking Conference*. Pp. 222–226. Jan. 2007.

[DLMW08] M. Doering, S. Lahde, J. Morgenroth, L. Wolf. IBR-DTN: an efficient implementation for embedded systems. In *CHANTS '08: Proceedings of the third ACM workshop on Challenged networks*. Pp. 117–120. ACM, New York, NY, USA, 2008.

[DO08] M. Demmer, J. Ott. Delay Tolerant Networking TCP Convergence Layer Protocol. *IETF Draft*, 2008.
http://tools.ietf.org/pdf/draft-irtf-dtnrg-tcp-clayer-02.pdf

[EB10] D. Ellard, D. Brown. DTN IP Neighbor Discovery (IPND). *IETF Draft*, 2010.
http://tools.ietf.org/pdf/draft-irtf-dtnrg-ipnd-01.pdf

[IEE07] IEEE. 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs) . *IEEE Standard for Information technology*, 2007.

[KO08] H. Kruse, S. Ostermann. UDP Convergence Layers for the DTN Bundle and LTP Protocols. *IETF Draft*, 2008.
http://tools.ietf.org/pdf/draft-irtf-dtnrg-udp-clayer-00.pdf

[LDP$^+$07] S. Lahde, M. Doering, W.-B. Pöttner, G. Lammert, L. Wolf. A practical analysis of communication characteristics for mobile and distributed pollution measurements on the road. *Wireless Communications and Mobile Computing* 7(10):1209–1218, Jan 2007.

[MGH$^+$07] P. McDonald, D. Geraghty, I. Humphreys, S. Farrell, V. Cahill. Sensor Network with Delay Tolerance (SeNDT). *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on DOI - 10.1109/ICCCN.2007.4318006*, pp. 1333–1338, 2007.

[PN03] R. Patra, S. Nedevschi. DTNLite: A Reliable Data Transfer Architecture for Sensor Networks. Technical report CS294–1, Berkeley, 2003.

[SB07] K. Scott, S. Burleigh. Bundle Protocol Specification. RFC 5050 (Experimental), Nov. 2007.
http://www.ietf.org/rfc/rfc5050.txt

[SFWL10] S. Symington, S. Farrell, H. Weiss, P. Lovell. Bundle Security Protocol Specification. *IETF Draft*, 2010.
http://tools.ietf.org/pdf/draft-irtf-dtnrg-bundle-security-17.pdf

[VB00] A. Vahdat, D. Becker. Epidemic Routing for Partially-Connected Ad Hoc Networks. Technical report CS-200006, Duke University, May 2000.