DTN7: An Open-Source Disruption-tolerant Networking Implementation of Bundle Protocol 7

Alvar Penning[1], Lars Baumgärtner[3], Jonas Höchst[1,2], Artur Sterz[1,2], Mira Mezini[3], and Bernd Freisleben[1,2]

[1] *Dept. of Math. & Computer Science, Philipps-Universität Marburg, Germany*
{penning, hoechst, sterz, freisleb}@informatik.uni-marburg.de
[2] *Dept. of Electr. Engineering & Information Technology, TU Darmstadt, Germany*
{jonas.hoechst, artur.sterz}@maki.tu-darmstadt.de
[3] *Dept. of Computer Science, TU Darmstadt, Germany*
{baumgaertner, mezini}@cs.tu-darmstadt.de

Abstract. In disruption-tolerant networking (DTN), data is transmitted in a store-carry-forward fashion from network node to network node. In this paper, we present an open source DTN implementation, called DTN7, of the recently released Bundle Protocol Version 7 (draft version 13). DTN7 is written in Go and provides features like memory safety and concurrent execution. With its modular design and interchangeable components, DTN7 facilitates DTN research and application development. Furthermore, we present results of a comparative experimental evaluation of DTN7 and other DTN systems including Serval, IBR-DTN, and Forban. Our results indicate that DTN7 is a flexible and efficient open-source multi-platform implementation of the most recent Bundle Protocol Version 7.

Keywords: delay-tolerant networking · disruption-tolerant networking

1 Introduction

Delay- or disruption-tolerant networking (DTN) is useful in situations where a reliable connection to a communication infrastructure cannot be established, e.g., during environmental monitoring in remote areas, if telecommunication networks are destroyed as a result of natural or man-made disasters, or if access is blocked due to political censorship. In DTN, messages are transmitted hop-to-hop from network node to network node in a store-carry-forward manner. There might be larger time windows between two transmissions, and the next node to carry a message might be reached opportunistically or through scheduled contacts.

There are several mobile DTN appications, such as FireChat [13] and Serval [12], that rely on peer-to-peer networks of smartphones, where the preinstalled Wi-Fi or Bluetooth hardware of the mobile devices is used to create a large mesh network. µPCN [10] is a special purpose DTN application for planetary communication, and IBR-DTN [8] is a popular DTN platform, but does not implement the recently released Bundle Protocol (BP) Version 7 [5].

In this paper, we present DTN7, which (to the best of our knowledge) is the first and only freely available, open source implementation of the most recent draft of Bundle Protocol Version 7 (BP7) (draft version 13). DTN7 is designed to offer extensibility by allowing developers to easily replace or add individual components. DTN7 is a general purpose DTN software with support for several use cases, such as enabling communication in disaster scenarios or providing connectivity in rural areas. Our contributions are:

- We provide a memory-safe and concurrent open-source implementation of BP7 (draft version 13), written in the Go programming language.
- With its highly modular design and its focus on extensibility by providing interfaces to all important components, DTN7 is a flexible basis for DTN research and application development for a wide range of scenarios.
- We compare DTN7 with other well-known DTN systems including Serval, IBR-DTN, and Forban, using the CORE network emulation framework.
- Several experiments to mimic different DTN test cases, i.e., a chain of up to 64 nodes with different payload sizes, are conducted.
- The presented DTN7 software[4], the evaluation framework and its configurations[5], and the experimental fragments[6] are freely available.

The paper is structured as follows. Section 2 discusses related work. In Section 3, we briefly explain BP7. Section 4 discusses DTN7's design and implementation. Section 5 describes experimental results. Section 6 concludes the paper and outlines areas of future work.

2 Related Work

This section briefly reviews relevant publications in the area of DTN software.

2.1 DTN Software Implementations

IBR-DTN [8] is a lightweight, modular DTN software for terrestrial use. The Interplanetary Overlay Network (ION) focuses on the aspects of extreme distances in space [3]. DTN2 is the reference implementation of the BP, developed by the IETF DTN working group [7]. These three implementations are based on RFC 5050, i.e., BP Version 6 [19].

Designed for small satellites in low earth orbit, µPCN can be used to connect different regions of the world. It also implements BP Version 6, as well as an older draft of version 7 [10]. Furthermore, an older version of BP7 is implemented in Terra [15].

Serval focuses on node mobility by providing implementations that run on smartphones, as well as by incorporating different radio link technologies [12].

[4] `https://github.com/dtn7/dtn7-go`
[5] `https://github.com/dtn7/adhocnow2019-evaluation`
[6] `https://ds.mathematik.uni-marburg.de/dtn7/adhoc-now_2019.tar.gz`

Forban is a peer-to-peer file sharing application that uses common Internet protocols like IP and HTTP to transmit files in a delay-tolerant manner [9]. With FireChat [13], it is possible to send messages via DTN without relying on Internet access or direct peer contacts.

Many of the mentioned DTN systems implement the BP as specified in RFC 5050 [19]. While some implement a draft of BP7, none of them implements the most recent draft. Serval, Forban, and FireChat have their own protocol definitions, which are not compatible with the BP. Furthermore, the mentioned implementations cannot be extended in a modular manner, are not written in developer-friendly high-level programming languages and are not intended as general purpose DTN platforms, but are designed for specific use cases. FireChat is not freely available, and thus cannot be extended.

2.2 DTN Software Evaluations

IBR-DTN, DTN2, and ION were evaluated by Pottner et. al [14]. For a payload of 1 MB, DTN2 and IBR-DTN produced almost identical results. ION was slower in the conducted measurements. Furthermore, the interaction of the three DTN implementations was evaluated by transferring bundles between them, and the times measured varied significantly.

IBR-DTN was used to evaluate the connection between a stationary DTN node and a moving vehicle [8]. This vehicle passed the stationary node at an average speed of 20 km/h, and the transmission rate was measured in relation to the distance. Data could be transmitted within a range of about 200 meters.

Serval was experimentally evaluated in our previous work [2], for scenarios with 48 nodes in a hub topology, 64 nodes in a chain topology, and 100 nodes in disjoint islands connected over time. The results indicate that Serval can achieve high network loads, while CPU usage remains relatively low.

3 Bundle Protocol Version 7

This section gives an overview of bundle protocols, referring to RFC 4838 [6] and the current version of the Bundle Protocol (BP) [5]. The latter has version number 7 and is currently still in active development. We discuss the status of the 13th draft from April 2019 below.

3.1 Basic Concepts

Endpoints. In DTN, there are nodes and endpoints. Nodes exchange bundles according to the store-carry-forward principle. Bundles are addressed at endpoints, or more precisely, their characterizing *Endpoint Identifier* (EID), which might not be a currently existing part of the network. Fig. 1 shows an example of a scenario, where sensor nodes produce readings to be consumed by data sinks. The temperature bundle is addressed directly to `dtn:s3`, where the lux bundle is headed to `dtn:sink/lux`, an EID that is handled by two nodes, and thus

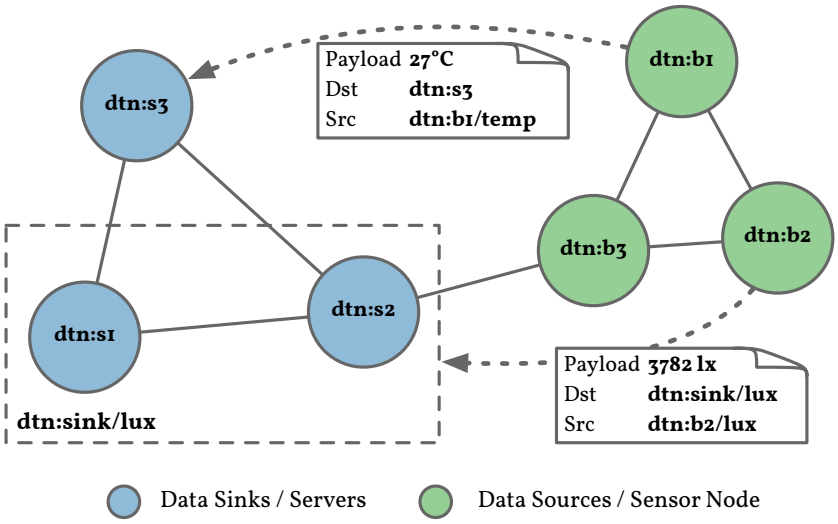

Fig. 1: Example sensor node scenario with multiple endpoints.

a multicast. BP7 is endpoint scheme agnostic and supports the null endpoint for anonymous bundles. In BP version 6, only endpoints are defined, so it is not possible to address dedicated nodes.

Bundles and Blocks. Packets in a DTN consist of multiple *Blocks* to form logical units called *Bundles*. In Fig. 2, an example bundle containing the mandatory Primary Block, and two Canonical Blocks, namely a Hop Count Block and the actual Payload Block, is shown, following the example of Fig. 1.

Bundle

Primary Block	Hop Count Block	Payload Block
Version: `7` Control Flags: *Status requested for reception* CRC Type: *None* Destination EID: `dtn:sink/lux` Source node EID: `dtn:b2` Report-to EID: `dtn:b2` Creation Timestamp: (`0`, `23`) Lifetime: `3600000`	Type Code: `9` Number: `2` Control Flags: *None* CRC Type: *None* Data: (`64`, `42`)	Type Code: `1` Number: `1` Control Flags: *None* CRC Type: *None* Data: `0E C6`

Fig. 2: A bundle transmitting a lux value from `dtn:b2` to `dtn:sink/lux`.

Primary Block. Each bundle begins with a (since BP7 immutable) *Primary Block* (see Fig. 2), containing meta-information about the bundle with the following fields: Version; Bundle Processing Control Flags to provide information on the bundle, including fragmenting and reporting information; an optional CRC Checksum (added in BP7 and not available in BP version 6); Destination EID, Source Node ID and Report-To EID, as endpoints for administrative records regarding this bundle; Creation Timestamp, consisting of the actual timestamp and an incrementing sequence number; Maximum Lifetime of a bundle, expressed in microseconds after creation time; Fragment Offset and Total Data Length, if fragmented and indicated by the bundle process control flags.

Canonical Block. Payload and Extension Blocks in Fig. 2 are summarized as *Canonical Blocks*. These contain a payload in addition to a few block-specific characteristics. A Canonical Block consists of a Type Code to identify the kind of block, Number to address the specific block, Control Flags and Data.

The actual payload of the bundle is located in the Payload Block at the end of each bundle. In addition to sending user data from application programs, status information is also sent within bundles, called Administrative Records, automatically created and sent by DTN software as a response to a previous bundle. Extension Blocks are Canonical Blocks containing further information relevant for a DTN router depending on its configuration. In contrast to BP version 6, the BP7 specification defines the Previous Node Block, Bundle Age Block, and Hop Count Block, and allows user-defined blocks to be added.

3.2 Node Components

Bundle Protocol Agent. The *Bundle Protocol Agent* (BPA) offers BP and DTN specific services. It executes procedures of the BP. For example, communication between Application Agent and Convergence Layer Adapter (see below) is managed. The BPA also constructs bundles for the Application Agent.

Application Agent. The interface between the BPA and an application is defined as an *Application Agent* (AA). A generic AA needs the ability to receive incoming bundles and compose outbound bundles for user applications and services. Furthermore, an EID must be assigned for local bundle delivery.

Convergence Layers. Bundles are exchanged over connections between nodes of different types and characteristics, and connections are unidirectional or bidirectional, or vary in transmission speed and bandwidth. Depending on the connection technology used, more or less complex protocols are required for delivery, called *Convergence Layer (CL) Protocols* (CLP). A *Convergence Layer Adapter* (CLA) is an implementation of a CLP. There are two CLPs defined by the IETF DTN group to exchange bundles over a TCP connection, the bidirectional TCP Convergence Layer Protocol (TCPCL) [20] and the unidirectional Minimal TCP Convergence Layer Protocol (MTCP) [4]. In addition to transport layer CLs,

there are approaches based on other technologies, e.g., DTN2 defining a Bluetooth and a serial CL, or IRB-DTN featuring an e-mail CL.

4 DTN7

In this section, we present the design and implementation of DTN7.

4.1 Requirements Analysis

There are several requirements that should be satisfied by DTN software. First, DTN software operating on a variety of laptops, smartphones, and routers should run on several hardware architectures (e.g., x86, ARM, and MIPS), based on the most popular operating systems (e.g., Linux, macOS, and Windows). Second, the individual components of the DTN software should be exchangeable. For example, there is the need to support different storage backends, CLAs, and DTN routing protocols. A suitable programming interface enabling concurrent execution is required for the interaction of components. Furthermore, a CLA implementation is required as well as a peer discovery mechanism to enable automatic establishment of connections between nodes. Finally, applications should to be independent of the DTN software, to allow easy creation of further applications and tools. Thus, a convenient interface between the DTN software and applications is required.

4.2 Implementation Decisions

As a result of these requirements, we selected the Go programming language[7] to develop DTN7. Go offers a large standard library and is rather developer-friendly. Its strengths are the simple creation and integration of programming libraries. Moreover, Go enforces good style guides and clean code plus provides memory-safety guarantees to increase security and stability of written programs. Thus, Go makes maintaining code and bringing in new developers very easy. The source code including all required dependencies are compiled into a single, static executable, removing the need for interpreters or further libraries. Furthermore, the Go compiler allows simple (cross-)compilation for many operating systems and processor architectures. The concept of concurrency is implemented in Go through the interaction of Goroutines and Channels; concurrency was one of the design priorities of the language designers.

To support exchangeability of DTN7's components, we structured our implementation into *Bundles* and its corresponding *Store*, *Convergence Layer Adapters*, *Peer Discovery*, the *Application Agent*, *Routing*, and the *Core* package needed to connect the individual packages. The modules in the these packages are designed as generic interfaces and example implementations, e.g., there exists an interface for routing in general and an epidemic routing implementation. We

[7] `https://golang.org`

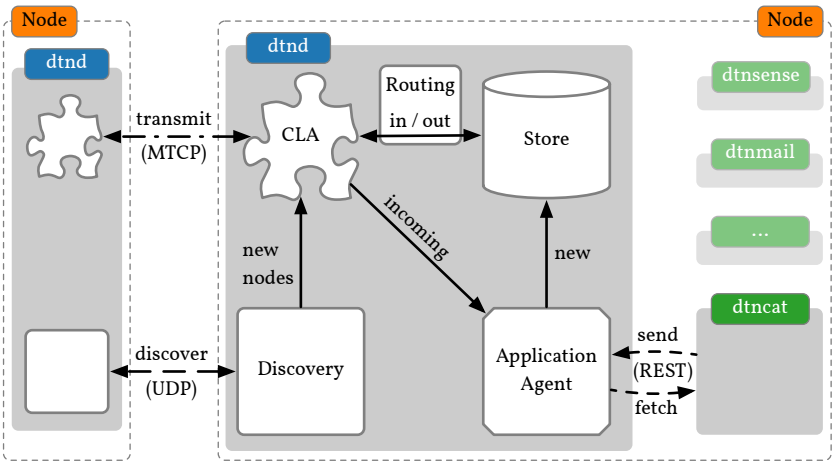

Fig. 3: Architecture and data flow in DTN7

decided to use MTCP for exchanging messages between two DTN7 nodes due to its simplicity. A third party application can also use parts of DTN7 as a library to, e.g., create and serialize bundles via the corresponding package. To make programming of applications against these interfaces simple and programming language independent, we decided to use a RESTful API.

4.3 DTN7 Architecture

Fig. 3 shows the modules of DTN7 and their interaction. The arrows indicate the way a bundle is internally processed in DTN7. The links between two distinct DTN7 nodes are shown by both an active CLA and the Discovery on the figure's left hand side. Multiple client connections to the AA from within the node are delineated on the node's right hand side.

To store bundles locally, a serialized version as defined in BP7 is written to the file system. A central index of all known bundles manages their meta-data and links point to information of the specific file. This index supports a fast lookup of bundles. The module providing this functionality is called *Store*.

In DTN7, an AA is implemented as a RESTful Web API to support both dispatching and fetching of bundles. The API does not interact with entire bundles, but only with a subset of its fields. This allows a client to send a new bundle by only supplying the destination EID and a payload. Such a request can easily be created from the command line or possible third-party software. When fetched over the API, selected fields of those bundles are returned and the bundles will be removed from the store afterwards.

The concept of different CLs and their CLAs is also present in DTN7's architecture with an implementation of MTCP. Based on a specific CL's characteristics, bundles might be transferred in a uni- or bidirectional way. Thus, a CLA in DTN7 must supply one or multiple modules for inbound and outbound bundle

processing. The unidirectional MTCP is designed using modules for sending and receiving bundles.

To support connections in dynamic networks, a *Peer Discovery* mechanism is provided. It announces a node's existence and listens for potential neighbors. This discovery mechanism broadcasts all of the node's CLAs continuously and notifies about received CLAs.

The previously defined components are linked together within DTN7's *Core* package. A central processing pipeline consumes both newly created and inbound bundles. Within this pipeline, a bundle will be marked to be delivered to a subset of known CLAs, to a local AA or to be discarded for later processing or even removed. The Core's internal links, visualized in Fig. 3, are related to the concept of a BPA, and serve as an interface between CLAs and the AA.

Every bundle that is not addressed to a particular node will be forwarded over one or multiple CLAs to neighboring nodes. The decision about which CLAs to select is made by a routing algorithm. To support the use of different routing algorithms, a generic interface needs to be informed about inbound bundles and, furthermore, a tight cohesiveness to the core is required. DTN7 implements an epidemic routing module, which is notified about received bundles, to memorize both sender and receiver. Before dispatching, the epidemic routing algorithm compiles a subset of known connections which have not received this bundle yet.

Finally, DTN7 is also intended to be used as a library and allows fast development of DTN applications. In particular, bundle package creation, serialization, and deserialization might be useful in other software.

4.4 Resulting Programs

DTN7 contains a DTN daemon, referred to as `dtnd` in Fig. 3, for storing and exchanging bundles and interfacing with applications. Currently, an example DTN application (`dtncat` in Fig. 3) for sending and receiving bundles, implemented as a command line tool, is included. `dtnd` initializes the previously defined modules according to the configuration provided by the user. `dtncat` processes user input, which is handed over to `dtnd`'s AA RESTful interface. The input is then encapsulated inside the Payload Block of a newly created bundle by `dtnd`. This bundle's Primary Block will be populated with basic defaults, like disabled CRC, and a delivery report request. As shown in Listing 1.1, `dtncat` is called by passing parameters on the command line. The first option selects between receiving or sending bundles. The local `dtnd`, running the RESTful API, is addressed by the second parameter. When sending new bundles, the content is read from the standard input.

```
# Sending a bundle
$ dtncat send http://localhost:8080 dtn:s2 <<< "3782 lx"

# Retrieving a received bundle
$ dtncat fetch http://localhost:8080
```

Listing 1.1: `dtncat` example

5 Experimental Evaluation

In this section, we experimentally evaluate DTN7 and compare it with other DTN software.

5.1 Emulation Environment

To evaluate DTN7 in a realistic manner, we emulated up to 64 nodes in the network emulation framework *Common Open Research Emulator* (CORE) [1]. CORE can emulate nodes using Linux namespaces to allow the execution of native binary programs, which is not possible with purely simulation-based approaches like NS-3 [16, 18]. All experiments were performed on Intel Xeon E5-2698 CPUs with 80 cores at 2.20 GHz and 256 GB RAM. To execute the total number of 1,440 experiment runs, we used MACI, a framework for extensive and reproducible experiments [11].

DTN Software. We compared DTN7 with three popular DTN software solutions. *Serval*[8] is a software suite centered around protocols designed for infrastructure independent communication [12]. To be able to transfer files in intermittently connected networks, Serval relies on Rhizome, a custom DTN bundle protocol with epidemic routing. In our evaluation, we used the latest stable Serval release, which is from April 2016, since the recent development version has stability issues. *IBR-DTN*[9] is an implementation of BP Version 6, aimed to be lightweight and fast [8]. For comparability, we use the epidemic routing extension instead of the default PRoPHET protocol used by IBR-DTN. We use the current HEAD of the git repository to include the latest bug fixes. *Forban*[10] is mainly used as a local peer-to-peer file sharing application using an epidemic routing protocol based on HTTP. We used the latest HEAD of the git repository, but had to introduce our own patches to make Forban usable.

Payload Sizes. DTN software is used in multiple applications and scenarios. Serval, e.g., offers the SMS-like application MeshMS for short text messages. IBR-DTN can be used in environmental monitoring, where transmission of short audio recordings or images might be required. Therefore, we selected four different file sizes, representing a wide range of possible applications. All files were generated randomly with the same seed for reproducibility in six sizes:

- *64 KiB* for compressed images or map data;
- *1 MiB* representing small images or short audio recordings;
- *5 MiB*, e.g., smartphone images and audio recordings;
- *25 MiB* representing longer audio recordings or short videos;
- *50 MiB* for HD videos typically recorded by smartphones;
- *100 MiB*, e.g., 4k smartphone videos [2, 17, 21].

[8] `https://github.com/servalproject/serval-dna/tree/batphone-release-0.93`
[9] `https://github.com/ibrdtn/ibrdtn`
[10] `https://github.com/adulau/Forban`

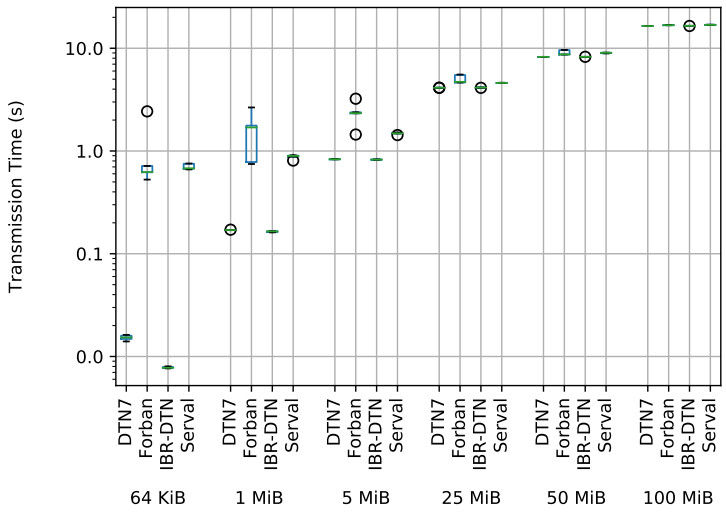

Fig. 4: Bundle transmission time for the 1-hop topology and different payload sizes

Network Topologies. We used a chain topology of three different lengths, where nodes are connected pairwise, to benchmark the different DTN software systems. The first node is sending a bundle destinated to the last node in the chain. To get the baseline performance of the interacting components, a chain of two nodes was used. We measured the time it takes to read the data, serialize the bundle, send it over the network, deserialize it at the receiver and deliver it to the application. With 32 nodes, the forwarding capabilities were investigated. For an even larger scenario, we used 64 nodes, to evaluate how the DTN software systems behave when node numbers increase. We used a bandwidth of 54 MBit/s to match the speed of an IEEE 802.11g network.

Measurements. To measure CPU utilization for each process on every node, we used *pidstat*, which is part of the *sysstat* package[11]. Additionally, *bwm-ng*[12] was used for network statistics per node and network interface. Finally, every used DTN software logged both the timestamp of sending and receiving bundles, such that a detailed analysis of transmission time and network distribution can be performed.

5.2 Results

Transmission Times. Figs. 4 and 5 show the bundle transmission times on the y-axes and payload sizes on the x-axes for the 1-hop and 64-hops topologies,

[11] http://sebastien.godard.pagesperso-orange.fr/man_pidstat.html
[12] https://github.com/vgropp/bwm-ng

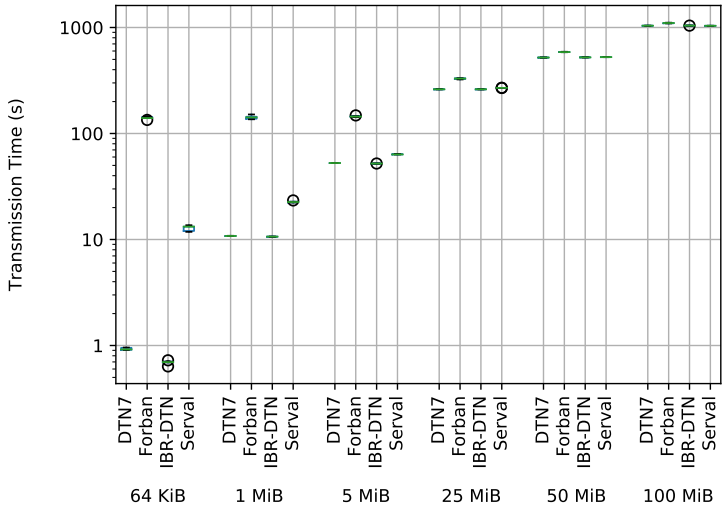

Fig. 5: Bundle transmission time for the 64-hops topology and different payload sizes.

respectively. Regardless of chain length and file size, DTN7 and IBR-DTN are always the fastest DTN software systems. The larger the files become, the transfer times of all DTN systems converge. This is due to the network configuration. All DTN systems manage to completely fill the 54 Mbit/s available, which is easier to achieve with larger files. As a result, the transfer times for large files hardly vary at all.

For a single hop, Forban and Serval take about the same time for transmitting files (e.g., about 0.6 seconds for 64 KiB files), but Forban shows a higher variance. For longer chains and files below 50 MiB, the differences between Forban and Serval are more noticable. DTN7, however, is still up to 140 times (64 KiB over 1 hop) faster than Serval. Particularly in chat or text based applications, the speed advantage of DTN7 can be crucial if a message arrives below 0.01 seconds rather than after one second.

These results indicate that both BP6 and BP7 have a relatively small protocol overhead compared to the protocols used by Serval and Forban, which is especially noticeable for small files. The larger the files or the longer the chain, the less weight the low protocol overhead carries. Furthermore, it is also remarkable that DTN7, which is written in Go, does not take longer to transmit larger files from end to end in the chain, although IBR-DTN is implemented in C++ and optimized for speed. In terms of transmission speeds, Forban takes longer than the other DTN software systems, although differences get smaller the bigger the files are. One explanation is that Forban has a pull-based approach where it actively downloads new bundles after an announcement was received. There-

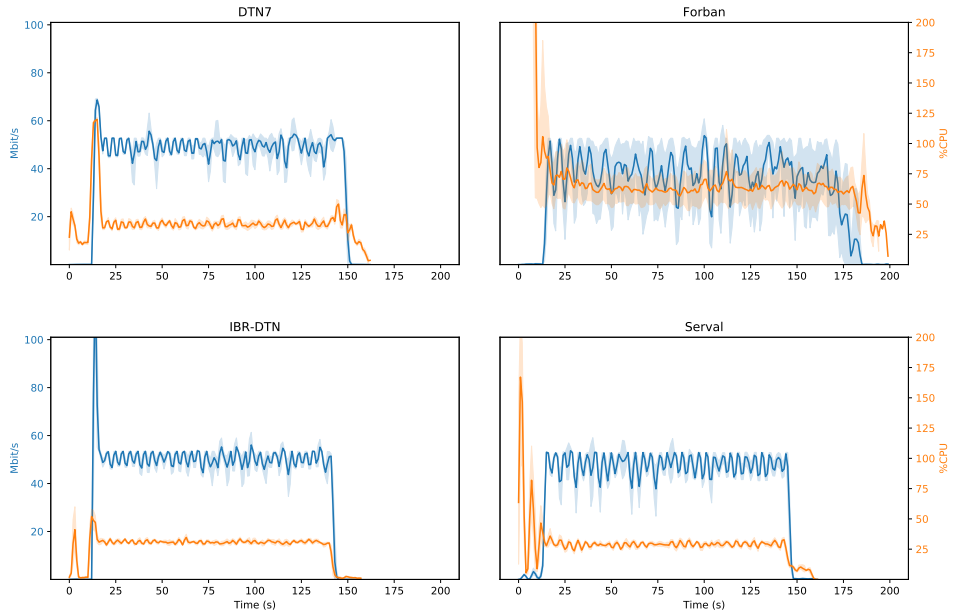

Fig. 6: CPU and network usage for transmitting 25 MiB over 32 hops.

fore, the announcement interval is a natural barrier. If quick data exchange is necessary, the other solutions provide better performance.

CPU Usage and Network Utilization. Fig. 6 shows CPU usage and network utilization for transmitting 25 MiB over 32 hops. On the x-axes, the time for the entire experiment in seconds is shown, the left y-axes denote the network usage in Mbit/s and the right y-axes show the CPU usage in %, both of the entire network. The bold graphs denote the sum over all nodes, averaged over all experiment repetitions. The shaded areas denote the error band.

DTN7 requires about 34.3% of the available CPU (standard deviation of 16.7%). At the beginning of an experiment, DTN7 shows a short peak in CPU usage resulting from the first node, where the file is converted to base64, sent to the DTN7 AA, which decodes the file again, packs it into a bundle, and starts the transmission. Further nodes only have to retransmit the bundle and do not require the steps mentioned above. Forban uses about 163.1% CPU (646.3%). Forban shows a small peak at the start of the experiment, indicating the overhead when starting its daemons, where four Python interpreters have to be started. Additionally, the file has to be hashed at the beginning of the experiment. Serval

consumes 29.3% (24.6%) CPU. Serval has an additional hashing step, which results in higher CPU load at the start of the experiment. With only 26.9% (13.1%), IBR-DTN is the most efficient tested DTN software in terms of CPU usage.

In terms of network usage, DTN7 reaches about 42.0 Mbit/s (19.7 Mbit/s) for transmitting bundles from node to node, while Forban achieves about 32.8 Mbit/s (22.8 Mbit/s). IBR-DTN and Serval achieve 42.3 Mbit/s (23.7 Mbit/s) and 39.5 Mbit/s (20.0 Mbit/s), respectively. Although the theoretical total network load for the entire network can be up to 1.674 Gbit/s, the tested DTN software systems used only the maximum bandwidth per link, which is 54 Mbit/s, in peak situations. This indicates that every DTN software needs to receive the entire bundle before transmitting it to the next node.

To summarize, DTN7 requires slightly more CPU utilization than IBR-DTN and Serval, but has the advantage of transmitting files faster than all other DTN systems in most cases, as shown in Section 5.2.

6 Conclusion

We presented an open source DTN implementation, called DTN7, of the recently released Bundle Protocol BP7 (draft version 13), written in the Go programming language. DTN7 is designed to offer extensibility and supports multiple use cases, such as enabling communication in emergency and disaster scenarios or providing connectivity for rural areas. Furthermore, we presented results of a comparative experimental evaluation of DTN7 and other DTN systems including Serval, IBR-DTN, and Forban. Our results indicated that DTN7 is a flexible and efficient open-source multi-platform implementation of the most recent version of BP7.

There are several areas for future work. For example, the BP does not define any kind of security or privacy mechanisms, although optional extension exist. This opens the field of DTN-related security and privacy research based on DTN7. Furthermore, for sensor networks or deployments in rural areas, DTN7's energy consumption should be evaluated. Due to DTN7's modular routing interface, new DTN routing algorithms for vehicular ad-hoc networks or UAV-based information dissemination should be investigated. Finally, new Convergence Layers based on emerging radio technologies, such as LoRa or mmWave communication, could be developed.

Acknowledgement

This work is funded by the HMWK (LOEWE Natur 4.0 and LOEWE emergenCITY) and the DFG (SFB 1053 - MAKI).

References

1. Ahrenholz, J.: Comparison of CORE Network Emulation Platforms. In: 2010 Military Communications Conference (Milcom). pp. 166–171. IEEE (2010)

2. Baumgärtner, L., Gardner-Stephen, P., Graubner, P., Lakeman, J., Höchst, J., Lampe, P., Schmidt, N., Schulz, S., Sterz, A., Freisleben, B.: An Experimental Evaluation of Delay-Tolerantf Networking with Serval. In: 2016 IEEE Global Humanitarian Technology Conference (GHTC). pp. 70–79. IEEE (2016)
3. Burleigh, S.: Interplanetary Overlay Network An Implementation of the DTN Bundle Protocol. Tech. rep., JPL (2007)
4. Burleigh, S.: Minimal TCP Convergence-Layer Protocol. Tech. rep., IETF (2019)
5. Burleigh, S., Fall, K., Birrane, E.J.: Bundle Protocol Version 7 (draft version 13). Tech. rep., IETF (2019)
6. Cerf, V.G., Burleigh, S.C., Durst, R.C., Fall, K., Hooke, A.J., Scott, K.L., Torgerson, L., Weiss, H.S.: Delay-Tolerant Networking Architecture. Tech. Rep. RFC 4838, IETF (2007)
7. Demmer, M., Brewer, E., Fall, K., Jain, S., Ho, M., Patra, R.: Implementing Delay Tolerant Networking. Tech. rep., Intel Research Berkeley and University of California, Berkeley (2003)
8. Doering, M., Lahde, S., Morgenroth, J., Wolf, L.: IBR-DTN: An Efficient Implementation for Embedded Systems. In: Third ACM Workshop on Challenged Networks. pp. 117–120. ACM (2008)
9. Dulaunoy, A.: Forban: A P2P Application for Link-local and Local Area Networks (2016), `https://github.com/adulau/Forban`
10. Feldmann, M., Walter, F.: µPCN - A Bundle Protocol Implementation for Microcontrollers. In: 2015 Int. Conf. on Wireless Communications & Signal Processing (WCSP). IEEE (2015)
11. Froemmgen, A., Stohr, D., Koldehofe, B., Rizk, A.: Don't Repeat Yourself: Seamless Execution and Analysis of Extensive Network Experiments. In: 14th Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT'18) (2018)
12. Gardner-Stephen, P.: The Serval Project: Practical Wireless Ad-Hoc Mobile Telecommunications. Tech. rep., Flinders University, Adelaide, Australia (2011)
13. Open Garden: Firechat (2019), `https://www.opengarden.com/firechat/`
14. Pöttner, W.B., Morgenroth, J., Schildt, S., Wolf, L.: Performance Comparison of DTN Bundle Protocol Implementations. In: 6th ACM Workshop on Challenged Networks. pp. 61–64. ACM (2011)
15. RightMesh: Terra: Lightweight and Extensible DTN Library (2018), `https://github.com/RightMesh/Terra`
16. Riley, G.F., Henderson, T.R.: The NS-3 Network Simulator. In: Modeling and Tools for Network Simulation, pp. 15–34. Springer (2010)
17. Schildt, S., Morgenroth, J., Pöttner, W.B., Wolf, L.: ibr-dtn: A lightweight, modular and highly portable bundle protocol implementation. Electronic Communications of the EASST **37** (2011)
18. Schwerdel, D., Hock, D., Günther, D., Reuther, B., Müller, P., Tran-Gia, P.: ToMaTo - A Network Experimentation Tool. In: International Conference on Testbeds and Research Infrastructures. pp. 1–10. Springer (2011)
19. Scott, K.L., Burleigh, S.: Bundle Protocol Specification. Tech. Rep. RFC 5050, IETF (2007)
20. Sipos, B., Demmer, M., Ott, J., Perreault, S.: Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4. Tech. rep., IETF (2019)
21. Trono, E.M., Arakawa, Y., Tamai, M., Yasumoto, K.: Dtn mapex: Disaster Area Mapping through Distributed Computing over a Delay-tolerant Network. In: 2015 Eighth International Conference on Mobile Computing and Ubiquitous Networking (ICMU). pp. 179–184. IEEE (2015)