TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Chair of Computer Networks

Comparison of Bundle Protocol version 6 and 7

Marcel Beyer

August 1, 2019

The Bundle Protocol is an end-to-end protocol for the exchange of messages in Delay Tolerant Networks (DTN). Version 6 of the Bundle Protocol was published 2007 as an experimental specification in RFC 5050 [1]. After gaining experience from different implementations of this experimental specification, version 7 of the Bundle Protocol (BPbis) was published as an Internet Draft in 2016 [2]. This document is intended to provide an overview of the changes between those two versions of the Bundle Protocol.

This comparison regards version 12 of BPbis.

1 Unchanged fundamentals

Since the new Internet Draft is not a new protocol, but an update for the Bundle Protocol, the basic principles of both versions were kept.

So the nodes in both versions have the same components, which are one or more Convergence Layer Adapters (CLA), the Bundle Protocol Agent (BPA) and the Application Agent (AA). The CLAs are responsible for the communication with underlying network layers. The services and procedures of the Bundle Protocol are offered and executed by the BPA. The AA has an Administrative element and an Application-specific element. The former is responsible for creating administrative records, which are needed for the communication between different nodes of the Bundle Protocol, e.g. for status reports, while the latter is the link to upper network layers.

All bundles, the data units of the Bundle Protocol, are formed in both Bundle Protocol versions of two or more blocks. Each bundle has to contain one primary block and one payload block. All other blocks are optional and called “Extension Blocks”.

The procedures of sending and receiving bundles also have not changed between version 6 and 7 of the protocol.

The idea of bundle fragmentation, where a bundle could be split into several smaller ones somewhere on the way of transmission, is also implemented in both versions the same way.

2 Changes

Although the fundamentals of the Bundle Protocol were kept between version 6 and 7, the changes are so significant that implementations for one of the two versions are not compatible with the other one.

2.1 Convergence Layer Adapter

For the communication between two nodes the Bundle Protocol uses other protocols of lower network layers. The components using the services of those lower protocols to send and receive bundles are called Convergence Layer Adapters.

For those lower protocols are no concrete ones specified. There are only some requirements for those in the Bundle Protocol, which may be supplemented by other DTN protocols like the Bundle Security Protocol.
In Version 6 the requirement is, that a bundle could be sent to *all* reachable nodes, which are in the destination Endpoint ID of the bundle.
Version 7 claims the CLA to send the bundle to only *one* reachable node.
Both versions expect the CLA to deliver received bundles to the Bundle Protocol Agent, which will then process the bundles on behalf of the Bundle Protocol.

Version 7 also lists some examples like the retransmission of corrupt bundles or reporting on the transmission status of bundles as requirements which may be added by other DTN protocols.

2.2 Encoding of bundles

Version 6 of the Bundle Protocol was a "classical" type length value (TLV) formatted Protocol. To get a variable length of integer values, version 6 used SDNV [3] to encode them.

In the discussions for version 7 of the Bundle Protocol SDNV was perceived as an unhandsome way to encode integers. Instead of SDNV the JSON derived CBOR [4] came up. But CBOR is not only able to encode integers. Because there are also more objects like strings, arrays and many other, it is now used as a replacement for the TLV formatting.

Using CBOR has also some other benefits. Achieving a higher transmission and processing rate is one of the goals of CBOR, which are also goals for delay tolerant networks, and so for the Bundle Protocol. Also, CBOR was designed to be used on systems with limited resources, which may also apply to many systems used in DTN.
Furthermore CBOR decoders are more widespread and tested than decoders for the Bundle Protocol in version 6, which may cause them to be more robust.

The drawback of using CBOR is the loss of clear text readability.

2.3 Custody

An important concept of the Bundle Protocol is the custody of nodes for bundles. Custody ensures the transmission of bundles while they are not transmitted directly from one sender to a receiver, but forwarded hop-by-hop from one node to another through the network, until they reach the receiver. Thus, there is no need for a direct and stable connection between sender and receiver. Also, bundles have not to be retransmitted the whole way if there are errors between two nodes on the route, because nodes have to store bundles they have accepted custody for until another node accepts custody for this bundle.

In version 6 custody was specified directly in the Bundle Protocol. For version 7 custody was excluded from the Bundle Protocol to a separate Bundle-in-Bundle-Encapsulation Protocol (BIBE) [5]. If custody is needed, BIBE will now be used as a convergence layer. BIBE itself implements custody like it was implemented in version 6 of the Bundle Protocol.

Reasons for the exclusion were that a reliable transmission should be realized in a convergence layer and that custody also causes some difficulties. One of those difficulties is, that custody is not able to detect partial data loss. If there is only a single part of the bundle not transmitted correctly, the whole bundle will be retransmitted, which may be several hundreds of megabytes. Also, custody has no negative acknowledges to notify the sender of data loss. Since nodes are not engaged to accept custody for bundles, the sender of a bundle has no way to determine how long it would take until the next node accepts custody. Hence, he has no clue if the bundle got already lost and needs a retransmission or not.

But there are situations where all these disadvantages are better than e.g. TCP. An example situation could be a transmission between nodes on two different planets which do not have a direct connection. If there are satellites between those planets, which pairwise have time slots where they can communicate, custody is able to ensure the transmission.

So in version 7 custody can be used if needed.

2.4 Identifying single nodes

In some cases it is necessary to address a single node. An example therefor might be a status report to the node which sent a bundle. Version 7 of the Bundle Protocol introduced Node IDs for this purpose.

Bundle endpoints are sets of one or more nodes that identify themselves for a purpose. Each endpoint has an endpoint ID to identify it. An endpoint which contains a single node is called "singleton endpoint".

This concept was already specified in version 6. New in version 7 is, that every node has to be in a singleton endpoint until there is no need to address this node anymore. The ID of this singleton endpoint is now called "node ID" and is used to identify the node. Thus, node IDs solve the problem that nodes had not to stay in their singleton endpoint in version 6, which may have lead to undeliverable status reports and other messages to this node.

2.5 Integrity

In version 6 the Bundle Protocol had no possibilities to check the integrity of the transmitted data. Therefor version 7 specifies optional checksums for every block. So it is also possible to add checksums to only some blocks in a bundle.

To realize the checkusms, all blocks have two new fields: CRC type and CRC value, where CRC value is only present, when the type is not zero, which means that this block does not have a checksum. All other numbers in the type field specify which algorithm was used to calculate the checksum in the CRC value field.

There are only two CRC types specified for version 7: standard X-25 CRC-16 and standard CRC32C (Castagnoli).

2.6 Bundle format

Regardless of the version, every bundle has to consist of at least two blocks. The first block of every bundle is the so called "primary block" which contains all basic information about the bundle, including routing information. The "payload block" is the second block every bundle has to have. As the name says, it carries the payload data of the bundle.

In version 6 the payload block could be on an arbitrary position in the bundle. To enable the decoder to detect the end of the bundle, the block processing control flags had a "last block"-flag which was set to 1 in the last block. Version 7 defines that the last block has to be the payload block. Thusly the payload block indicates the end of the bundle and makes the last block-flag superfluous.

There were also changes to the separate components of a bundle, which are described in the following sections.

2.6.1 Primary Block

Many fields were retained from version 6 to 7 with minor changes, which arise from other changes in the bundle protocol. Naturally the value of the version number-field was incremented to 7. The bundle processing control flags, creation timestamp, lifetime, fragment offset as well as the destination-, source and report-to-addresses can be found in both versions. The bundle processing control flags were restructured, which can be found later in this paper. While in version 6 all addresses (source, destination, report-to) were split up in two links to the dictionary — one for the scheme offset and one for the scheme-specific part — in version 7 each address is a CBOR array with two elements (scheme offset and ssp). The dictionary from version 6, where all blocks could reference addresses from, was removed in version 7 since it made creating bundles a nightmare. Accordingly, the two fields “dictionary length” and “dictionary” are not longer present.

In consequence of removing custody from version 7 the field for the current custodian was removed from the primary block.

Changing the data format to CBOR allowed to remove the “block length” (total length of all blocks) and “total application data unit length” (total length of payload before fragmentation) fields.

To allow checksums the two fields “crc type” and “crc” were added.

2.6.2 Bundle Processing Control Flags

Most of the flags from version 6 were kept in version 7 although they are differently ordered. Furthermore, it was shortened from a 20 bit integer to a 16 bit integer.

The flags which indicate that the bundle is a fragment, must not be fragmented, an acknowledgment by the application is requested or status reports are requested were kept.

Since custody is no longer a part of the bundle protocol, all custody related flags were removed. A flag which indicated that the endpoint ID of the bundle is a singleton destination was also removed since node IDs can be detected by their separate namespace.

New in version 7 is a flag which requests the current time in status reports to determine how old such a report is. A flag indicating that the bundle contains a manifest block is also new. The manifest block itself is not specified in the Bundle Protocol and is thought to identify all blocks in a bundle which were present at the end of the creation of the bundle.

2.6.3 Canonical Blocks

Canonical blocks are all blocks except the primary block. As in version 6 they have still a block type code, block processing control flags and the block specific data. The type code is an integer which specifies the purpose of the block. The block-type-specific data contains the payload of the block as a CBOR string. In version 6 this had to be the last part of a canonical block since there were (depending on the type code) defined more TLV fields. Since the block data now is a single field containing a CBOR byte string, the block data length-field is no longer needed and was removed.

Because the dictionary in the primary block was abolished the EID reference list and the EID reference count in the canonical blocks were removed too.

Beside the two new fields for CRC (type and value) all blocks now have a block number. This number is an integer which uniquely identifies all blocks of a bundle. With this number blocks are able to reference among themselves, which is used for example by the BPSEC (bundle security) protocol.

2.6.4 Block Processing Control Flags

The main purpose of the block processing control flags is to define the handling of blocks which could not be processed by a node. Therefore, both versions of the bundle protocol have flags for requesting status reports, deleting the block from the bundle or dropping the whole bundle if a block could not be processed. Furthermore, there is a flag indicating that a block has to be repeated in every fragment if a bundle gets fragmented by a node.

Version 6 had the possibility to set a flag of the block processing control flags to 1 if the block was forwarded without processing (e.g. if the block type is unknown to the node). This flag was removed in version 7. Thus, nodes can only remove the affected block or drop the whole bundle, if allowed by the block processing control flags. Otherwise, the fact that a node was not able to process the block, will be ignored.

The "last block"-flag was removed since the payload block is the last block by definition. As the dictionary for endpoint IDs was removed, the flag which indicates that a block has references to this dictionary ("Block contains an EID-reference field") was also removed.

Those three freed bits and an additional one were marked as reserved for future versions. In version 6 were no reserved bits in the block processing control flags.

2.7 Extension Blocks

Extension Blocks are all blocks except the primary and the payload block and are defined in version 6 and 7 of the bundle protocol, where in version 7 the first concrete extension blocks were defined. But also the in v7 defined blocks are not all possible blocks, since other protocols can define more extension blocks. That is the reason why nodes have to get along with all (known and unknown) types of extension blocks. As described in the section above, version 6 had the ability to mark blocks as not processed while version 7 has only the ability to delete the block or the whole bundle.

The following extension blocks are defined in version 7 of the bundle protocol.

2.7.1 Previous Node

The block-type-specific data of this block is the node ID of the node which forwarded the bundle to the current node. Each bundle should contain one occurrence of this block type except the bundle is on the node which created it.

2.7.2 Bundle Age

This block type contains the number of microseconds between the creation time of the bundle and the time at which it was most recently forwarded. Therewith nodes without an accurate clock should be able to determine if the lifetime of a bundle expired. To calculate this time all signal propagation times over all episodes of transmission between nodes and all residence times on nodes are added up.

2.7.3 Hop Count

The intention of this block type is to delete all bundles which will never be delivered due to forwarding errors. Therefor it contains the two integers "hop limit" and "hop count". The hop limit is only set at creation time of the bundle to the maximum number of hops which are allowed for this bundle. On every hop, which means every time the bundle gets forwarded, the hop count is incremented by one. As soon as the hop count value exceeds the hop limit value, nodes are intended to drop the bundle.

3 Changed Demands

I see three main goals which led to the changes between version 6 and 7 of the bundle protocol: simplification, robustness and future flexibility.

Simplification is achieved by excluding custody which is not needed for all bundles. Furthermore, complex structures like the dictionaries were removed.

The robustness was increased by using CBOR instead of TLV/SDNV as a modern data structure which is more widespread and thus better tested than the decoder for bundles specified in RFC 5050 (version 6 of the bundle protocol).

CBOR also increased the future flexibility of the protocol since there are no longer e.g. fields with a specified length. Adding new elements to arrays has no impact on the backwards compatibility which is not true for TLV fields. Ignoring unknown extension blocks is also important for backwards compatibility if new blocks will be introduced in the future. Additionally, all flag-fields in version 7 have reserved bits while all bits of the block processing control flags were used in version 6.

By specifying Extension Blocks some known problems of version 6 could be resolved. So all bundles in routing loops will be dropped as soon as the hop count increases over the hop limit. Nodes without an accurate clock are now able to determine if the lifetime of a bundle exceeded using the bundle age.

4 Implementations

There are four implementations for the bundle protocol in version 6 (RFC 5050): the reference implementation of the DTN Research Group *DTN2*, the *Interplanetary Overlay Network (ION)* which is developed by the JPL and focused on spacecraft flights, *IBR-DTN* which is developed at the TU Braunschweig for embedded systems and *microPCN* developed at TU Dresden for POSIX-Systems.

The only one of those implementations which already supports version 7 of the bundle protocol is microPCN. For ION the implementation of version 7 is anticipated in the milestone for version 4.0.0 of ION. The support of version 7 is proposed for IBR-DTN since its makers published a job advertisement for the implementation.

Terra, an implementation with focus on terrestrial DTN, and PyDTN, a python implementation by the Slovakian company X-works, are two new implementations which only support version 7 of the bundle protocol. PyDTN has shown to be interoperable with microDTN.

	V6	**V7**
Convergence Layer Adapter	send to all reachable nodes	send to a (single) node
Data format	bit pattern, SDNV	CBOR
Custody	Specified in BP	BIBE, Conv. Layer
Identifying single nodes	Singleton endpoints	constant Node IDs
Checksums	no checksums	every block
Bundle Format	"last block"-flag	Payload Block is last block
Primary Block	Dictionary for EIDs	Immutable
Canonical Blocks	References to EIDs	Block number
Extension Blocks	no concrete blocks specified	Previous Node, Bundle Age, Hop Count

Table 1: Summary of changes

References

[1] K. Scott and S. Burleigh. Bundle Protocol Specification. *RFC 5050*, November 2007

[2] S. Burleigh, K. Fall and E. Birrane. Bundle Protocol Version 7. *Internet Draft*, November 2018

[3] W. Eddy and E. Davies. Using Self-Delimiting Numeric Values in Protocols. *RFC 6256*, May 2011

[4] C. Bormann and P. Hoffman. Concise Binary Object Representation (CBOR). *RFC 7049*, October 2013

[5] S. Burleigh. Bundle-in-Bundle Encapsulation. *Internet Draft*, January 2019