

HTTP-DTNSec: An HTTP-Based Security Extension for Delay/Disruption Tolerant Networking

Lucas William Paz Pinto[1], Bruno L. Dalmazo[2(✉)], André Riker[3], and Jéferson Campos Nobre[4]

[1] University of Vale do Rio dos Sinos, São Leopoldo, Brazil
lwppinto@unisinos.br
[2] Federal University of Rio Grande, Rio Grande, Brazil
dalmazo@furg.br
[3] Federal University of Pará, Belém, Brazil
afr@ufpa.br
[4] Federal University of Rio Grande do Sul, Porto Alegre, Brazil
jcnobre@inf.ufrgs.br

Abstract. Communication networks operating in challenging environments can be grouped by the concept of Delay/Disruption-Tolerant Networking (DTN). Different protocols can be used in DTN, such as the Bundle Protocol (BP) and the HyperText Transfer Protocol - DTN (HTTP-DTN). In this context, security properties are of fundamental importance in DTN like in regular networks. However, the challenges in DTN hamper the use of traditional security mechanisms. Although BP has been extended to include such mechanisms, there is still no analogous extension for HTTP-DTN. In this paper, we propose the HTTP-DTNSec, a security extension for HTTP-DTN. This extension improves the confidentiality and integrity of HTTP-DTN as well as updates the base protocol for HTTP/2. The proposed extension was implemented as a proof of concept and it was used to perform experiments in a simulated environment. These experiments show that HTTP-DTNSec performed the transfer of packages (i.e., a group of related objects) in a safe manner and with an increase in performance concerning HTTP-DTN. Finally, we provide some concluding remarks and future directions.

Keywords: DTN · HTTP-DTN · Network security

1 Introduction

It is important for many applications to support communication in challenging environments such as InterPlanetary Networking, Battlefield Networking, Rural communications, and Underwater Communications. The Delay/Disruption-Tolerant Networking (DTN) emerged to covers all these environments [8,20], since they present difficulties for the operation of a traditional network, such

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12949, pp. 325–340, 2021.
https://doi.org/10.1007/978-3-030-86653-2_24

as the Internet. The conditions found in DTN environments cause some problems in computer networks, such as long periods of disconnections that hinder end-to-end communication.

In the context of the IETF, the Bundle Protocol (BP) [5,17] has been developed in order to enable the transmission of messages between DTN nodes, through a logic of storage and sending of packets. This allows the nodes to store all the packets they receive locally, and send them when a connection is available with another node. However, some authors point out problems in the BP, for example, problems of synchronization and complexity [22]. Thus, alternative approaches to BP were created in order to avoid such difficulties. One of these alternatives is Hyper Text Transfer Protocol - Delay-Tolerant Networking (HTTP-DTN) [23].

Security properties are critical in DTN as in traditional networks [7]. However, there are additional difficulties in implementing delay or disconnected-tolerant security mechanisms [13,16]. In this context, an extension of BP has been developed, named *Bundle Protocol Security Specification* (BPSec) [4], to support end-to-end integrity and confidentiality. However, this extension only applies to BP, which means that other DTN protocols, such as HTTP-DTN, do not support these important security aspects.

The present paper proposes *HTTP-DTNSec*, a security extension proposal for HTTP-DTN. Such a proposal incorporates the security properties found in BPSec, maintaining the desirable characteristics of HTTP-DTN. In addition, HTTP-DTNSec updates the HTTP version used in HTTP-DTN (i.e., from HTTP/1.1 to HTTP/2). Such an update allows a performance increase in the exchange of messages. The results show that the extension is feasible, incorporating new features as well as maintaining adequate performance.

This article is structured as follows. Section 2 presents the theoretical background on DTN and security challenges that exist in this type of network. Section 3 describes the HTTP-DTNSec specification. Section 4 contains the experiments carried out as well as an analysis of the obtained results. Section 5 presents the related work. Finally, Sect. 6 presents the conclusions and future work.

2 Background

The DTN architecture proposed by the IETF [20] describes the Bundle Layer, which interfaces with the lower layers using Convergence Layer Adapters (CLAs) to carry out the transport of information. The end-to-end transport semantics are redefined and are now confined within each DTN hop. Thus, CLA protocols for different transport protocols have been developed, (e.g., UDP [18] and TCP [19]). This makes it possible to use the most suitable transport protocol for each hop.

BP defines the concept of blocks, which are a set of information referring to data and metadata that make up a bundle. The IETF defines three blocks [17]: the Primary Block, the first block in a bundle which contains information regarding source and destination addressing; the*Payload* Block, which carries the

data that the application wants to send via the BP; and the Extension Block, which has a variety of data that can provide additional information needed to process a bundle along the network.

This section is organized as follows. Initially, aspects related to security in DTN will be presented. Next, HTTP-DTN will be described in more detail.

2.1 DTN Security

There are several challenges in implementing security mechanisms in DTN. Some of these challenges occur due to the computational processing of such mechanisms [12,21]. For example, the use of digital certificates is difficult since a trusted third party for verifying the authenticity of certificates may not be available at the time of processing such verification.

Managing cryptographic keys presents challenges in DTN [3]. This is due to the fact that the acceptance of a session key, or even the exchange of public keys between the origin and the end of an end-to-end communication may not be possible, due, for example, to the lack of connection when sending the message [6]. Thus, a mechanism for the distribution of keys must be defined considering the communication problems in DTN. However, some insecure strategies can be adopted due to these problems. An example of such strategies is the use of keys shared with the nodes, which would reduce the complexity of managing them. This sharing is not adequate, since if the compromise occurs on any node, the entire DTN would be compromised and the key assignment process would need to be carried out again [14].

In the context of the IETF, security policies are applied through additional blocks to the bundle, which add end-to-end integrity and confidentiality services to BP [4]. Additional BPSec Blocks are: Block Integrity Block (BIB), used to guarantee the authenticity and integrity of the package block that the BIB sender wishes to ensure, and the Block Confidentiality Block (BCB), used to indicate that a part of the packet has been encrypted at the node originating from the BCB. BPSec is applied by definition only to nodes that accept its implementation, these nodes are known as Security Aware Nodes. There may be nodes in a DTN that do not implement BSP and that can communicate with all DTN nodes.

The authentication of the information in the BIB can be verified by any node that is in the path of the BIB sender and the recipient of the bundle. When a node other than the one originating from the bundle adds a safety block, it will be called the Security Origin Node of that block. The Security Origin Node can define a destination for the block you just added, which will not necessarily be the final destination of the bundle. This feature allows security policies to be applied on specific parts of the network, preventing unnecessary security processing at nodes that are not at risk.

2.2 HTTP-DTN

HTTP-DTN is a protocol developed as an alternative to BP in DTN. After several failures were pointed out in BP [22,23] proposed the use of HTTP as a protocol for exchanging messages in DTN. Such proposal is accomplished through some small changes of headings and addition of a store and forward logic. HTTP in this case is not used as on the Internet where it works end-to-end, instead the HTTP-DTN architecture is hop-by-hop (i.e., HTTP transfers between neighbouring nodes). HTTP requires only a reliable form of delivery, regardless of how the transport protocol does it, thus allowing it to operate over a wide variety of transport protocols [24].

HTTP-DTN is used as a Peer-to-Peer (P2P) protocol in the sense that multiple files can be transferred in two directions simultaneously between two communicating nodes. Thus, the DTN nodes do not have a client and server relationship as there is in the use of HTTP on the conventional Internet. For example, if a package sent from one node to another needs to pass through 4 other nodes on the way, there will be 5 separate HTTP transactions, one between each pair of nodes. Thus, the control of the delivery of the message passes from the source node to the destination node, for each hop between nodes. When two nodes contact each other, a request for files to be copied, saved, or sent can be made between nodes using the HTTP PUT and HTTP GET commands. A node also always has the option to reject a package by checking for error codes.

The addition of headers to HTTP is necessary for the operation of HTTP-DTN, they are:

- *Content-Destination*: specifies the final destination of the package being sent. This header can be used for routing applications that use HTTP-DTN, so that those applications decide which network paths to use.
- *Content-Source*: contains information about the origin identifier of the package. The existence of this header is also mandatory on packets, as well as the ability to process it on each node.
- *Content-Length*: loads the size in bytes of the information being carried in the package.
- *Content-ID*: allows files to be identified only within the context of the Source Information Header.
- (*Content-MD5*: adds the ability to verify end-to-end integrity. This header must be generated at the origin node that first sent the packet, and can be checked at any node before the destination to identify errors or corrupted data in the packet. If there is a problem with the package, the node can then either discard it or request the shipment of the damaged parts in the package.
- *Content-Host*: enables the identification of the node that sent the information and how to find it in the DTN. This header can be left blank in the package if you only need to request files available from a neighboring node. When a node sends a packet on the network, the information contained in the Source Information Header must be the same as that existing in the Host Header field.

Many other HTTP headers are directly useful for HTTP-DTN. For example, *ETag* which provides an ID for unique copies of files on the network, and can be used to provide global identifiers (*GUID*) for each version of a file.

2.3 HTTP/2

The specifications described by [2], define that HTTP/2 requests are made in frame format. There are several types of framework defined in the protocol, each with a different function. However, they all share a series of headers that are: size (*Length*) of the *payload* frame (24 bits) represented in bytes; type (*Type*) (8 bits) that determines the format and semantics of the frame; *flags* (8 bits) is reserved for Boolean *flags* specific to the frame type; R (1 bit) is not changed at any time and must exist when sending a request and be ignored when handling the frame; Stream identifier (*Stream Identifier*) (31 bits) represents an identification for the *stream* to which the frame in question belongs; the structure and content of the *payload* of the *frame* is entirely dependent on the value defined in the Type field. In Fig. 1 you can see the structure of the headers.

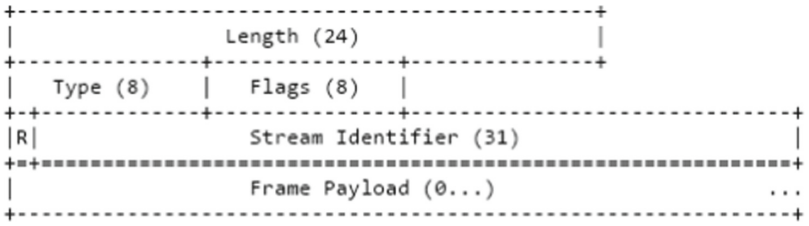


Fig. 1. Headers shared between all frames. Adapted from [2]

2.4 Headers-Type Frames in Requests

During an encounter with another security-aware node, there are a number of predefined requests that will occur. Such requests are intended to ensure that packages in transit on the network reach their final destinations.

A GET-type request is a HEADERS-type frame. It has the objective of obtaining from the paired node, any package whose recipient is the node that made the request. The headers sent in the *Header Block Fragment* field of the frame are:

- *method*: GET string.
- *Content-Destination*: identification of the destination node.

A PUT request is used to inform the paired node that a packet is being sent. Unlike a GET, this request consists of two frames, one of the HEADERS type and the other of the DATA type. In the HEADERS table, a series of headers is informed, containing values referring to the packet being sent. The DATA table contains the content to be sent. The headers sent in the *Header Block Fragment* field of the frame are:

- *method*: PUT string.
- *Date*: date of the package generation.
- *Host*: identification of the custody node of the package.
- *Content-Source*: identification of the source node.
- *Content-Destination*: identification the destination node.
- *Content-Integrity*: hash of the package content.
- *Content-Length*: size of the package content.

For each request made in a meeting, there are 3 types of possible responses. All are HEADERS type frames, followed by 0 or more DATA frames. Only one header of status type is sent in each HEADERS frame. The answers are: confirmation of the existence of packages (status header with a value of 200), used to answer a GET request, indicating the presence of packages for the node that made the request; denial response (status header with a value of 404), used as a response to a GET request, responding to the requesting node that there are no packages for it; and receipt confirmation (status header with value 202), used to respond to PUT requests and signalling that the package was successfully received and processed.

All frames are transported through a stream. A stream is by definition a sequence of frames, independent and bidirectional, exchanged between a client and a server during an HTTP/2 connection. The characteristics of a stream are: a single HTTP/2 connection can contain multiple open streams; streams can be established unilaterally by one of the points, or shared by the client and server; streams can be terminated by both the client and the server; streams are processed in the order they are received; streams are identified by an integer. This value is assigned to the stream by the point that started the request. Client-initiated streams will have odd values, and even streams when they are initiated by the server.

3 HTTP-DTNSec: Security and Performance Enhancements for the HTTP-DTN

This section aims to present the proposed solution in the context of the present work. Such solution is the extension of HTTP-DTN to provide support for security properties. HTTP-DTNSec is a protocol that allows packets to be exchanged securely on HTTP-based DTN, both hop-by-hop and end-to-end.

To ensure integrity and security in the transport of packets, HTTP-DTNSec implements an encryption solution in its solution when sending messages. Such a

solution enables authentication, confidentiality and integrity of messages through its structure. Each package sent from one node to another goes through the encryption process, ensuring a level of security hop-by-hop. Besides, with the addition of extra headers in HTTP-DTNSec, it is also possible to guarantee encryption in end-to-end messages.

HTTP-DTNSec is based on HTTP/2 because it allows some performance improvements, since it support multiple requests and responses on the same connection. With HTTP/1.1, it is possible to have a client send more than one request, it is necessary that for each of them there is a different connection [2]. With HTTP/1, there are the occurrence of repeated and verbose requests, which cause unnecessary network traffic and also excessive use of the TCP congestion window.

Another important aspect for HTTP-DTNSec is that it can use an efficient coding of the HTTP header fields. So, it is possible for HTTP-DTNSec to add priority to some requests, leaving more important packages to be processed first.

The rest of this section is organized as follows. Firstly, it presents the hop-by-hop and end-to-end encryption aspects. Next, it describes how two nodes negotiate to establish a communication.

3.1 Hop-by-Hop and End-to-End Encryption

HTTP-DTNSec was developed to ensure hop-by-hop and end-to-end security in DTN. In order to achieve this goal, all packets sent from one node to another are encrypted and processed. In addition to ensuring confidentiality, HTTP-DTNSec also ensures the integrity and authenticity of a message. The message in this way also has its integrity checked at the destination, and in the event of an inconsistency a new request to obtain the package again is made.

The encryption process is performed when the message is first sent and the decryption occurs when it reaches its destination. This process is addressed in the HTTP-DTNSec protocol using the Content-Destination header from the HEADERS table. The session key is then also encrypted, but using the public key of the destination. The destination uses its private key to decrypt the session key. With the session key, the destination can decrypt the message and the signature. The destination generates a hash of the message, and with the public key of sender verifies in the signature that the hash matches the sent one.

Figure 2 demonstrates a process of sending a package with end-to-end security. First, Node A encrypts the message using public key C shared between all nodes. Then, it sends the message to Node B and it verifies that its identifier does not correspond to the final destination of the package. Node B stores the package with the encrypted message and waits for an opportunity to send it to Node C. When a connection is established with Node C, Node B sends the package. Finally, Node C verifies the final destination of the package and with the private key C obtains the decrypted message.

HTTP-DTNSec does not perform cryptographic key management. However, for the use of an asymmetric encryption process, it is essential that the source and destination nodes have their respective public keys. The way in which this

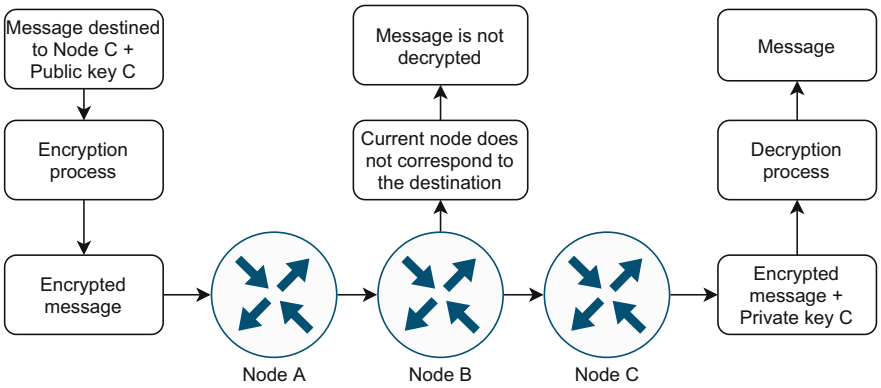


Fig. 2. End-to-end secure flow in HTTP-DTN

exchange will take place is outside the scope of HTTP-DTNSec (some alternatives are presented in [14]). Using end-to-end encryption, it is not possible to verify the integrity of a hop-by-hop package, as the hash of the message because it is encrypted can only be verified at the destination. In this way, if there was any corruption of the package during the end-to-end shipment, it would only be detected at the final destination.

The procedure to obtain the value to guarantee the integrity, occurs after the encryption of the package. In order to be able to verify the hop-by-hop integrity, an extra field is used in the PUT request HEADER frame, called Content-Integrity. First, a hash of the encrypted content is generated and the result is inserted in the Content-Integrity field. The node that receives the PUT request, must then generate a hash of the encrypted content of the received DATA frame. Finally, compare the value you obtained with the value sent in the Content-Integrity field. In case of inconsistent values, the node will respond with a Negative Response. In Fig. 3, it is possible to observe the sequence diagram of the message exchange.

HTTP-DTNSec, by definition, must be executed by all nodes that are part of the DTN. If there is a node that does not support it, it cannot be considered as a source or destination of packets. Besides, HTTP-DTNSec does not define a particular solution for encryption, so different solutions can be implemented according to the limitations found in a specific DTN.

3.2 Negotiation for Communication

A meeting between two nodes using HTTP-DTNSec can be divided into two distinct moments, the negotiation and the exchange of messages. The negotiation moment uses the HTTP/2 connection opening to initiate communication between nodes. HTTP-DTNSec does not change the way messages are exchanged at this time, it only defines default configuration values for the connection.

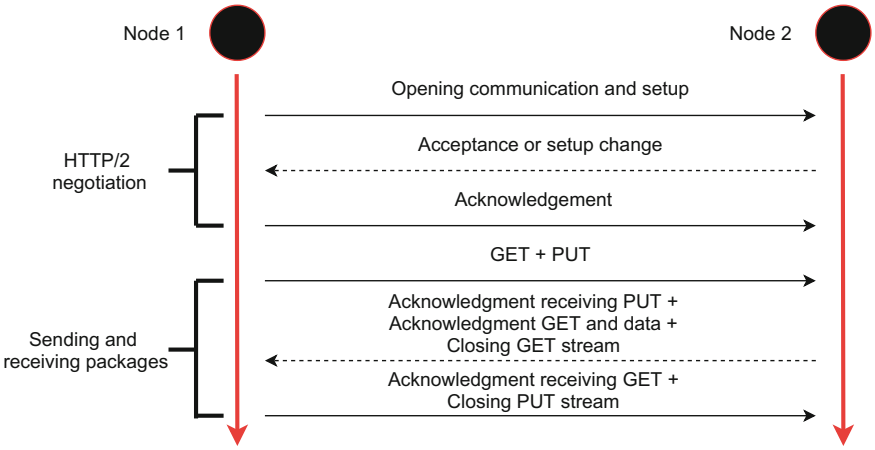
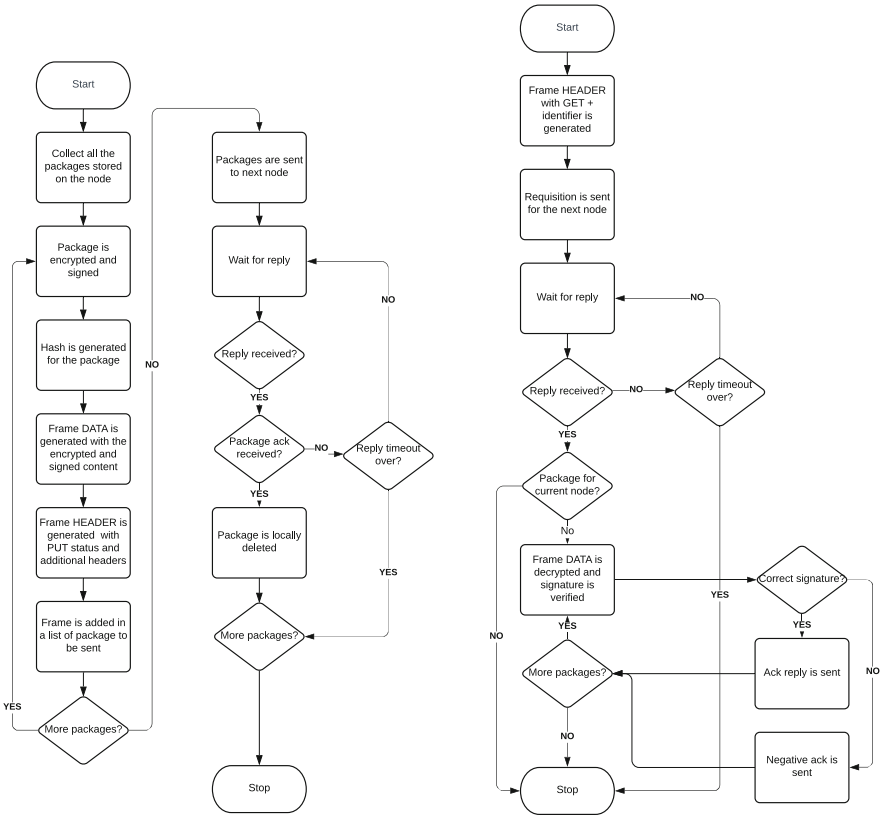


Fig. 3. Sequence diagram of message exchange

The node that initiates the negotiation will be considered as an HTTP/2 client, while the other paired node will be the server. The client node opens the connection by sending a PREAMBLE frame together with 2 SETTINGS frames. The first SETTINGS frame contains the configuration values for the connection, while the second is an empty SETTINGS frame that signals the end of the message. At this moment the communication is made through a single stream, this value is 0. The server node when receiving the message, using the same stream confirms the settings defined by the client and responds with 2 SETTINGS frames. The first frame can be used to modify some value of the connection configuration, while the second is an empty frame SETTINGS to signal the end of the message. Finally, the client node responds with an empty SETTINGS frame, signalling that the connection is now defined and is ready to send HTTP-DTNSec requests.

After the negotiation, when the client node has a package to be sent ahead, it will open a new stream with a value equals 1 and execute the logic of Fig. 4 (a). First, the node obtains all packages saved locally and for each one, it performs the encryption process and creation of the HTTP/2 frames used by HTTP-DTNSec. When all frames are created, the node then sends all packages to another paired node. The node waits for a receipt response for each sent package, deleting those that were successfully received locally, and keeping those that received problems.

The server node, upon receiving the PUT request, will check for integrity of each package received, in the case of a corrupted package, the server will return an error message. Then, the node will then check if it is the final destination of the message, if not, it will just store the package, in the other case it will decrypt the contents of the packages. Finally, it returns a receipt message to the paired node. At the end of the request, the client node will close the stream of value 1.



a) Execution flow diagram for PUT on HTTP-DTN for sending packages

b) Execution flow diagram for GET on HTTP for sending packages

Fig. 4. Execution flow diagrams

At the same time the PUT request is made, the client node will also carry out the GET request. So, the node opens a new stream of value 3 and executes the logic of Fig. 4. First the node sends a request to the paired node, asking if there are packages that it is the destination. When there is no response or there are no packages for the node, the stream is closed. In case of packages, the node decrypts the package's content and sends a response confirming receipt to the paired node.

The server node when receiving the GET request will later check for the existence of packages whose destination is from the requesting node, otherwise, it sends a denial message. When packages exist, the node encrypts the content of the package and creates the HTTP/2 frames to perform the upload. Finally, the server waits for a receipt response and in case of confirmation it will delete the packages received correctly. At the end of the request, the server node will close the stream of value 3.

4 Evaluation

This section presents the implementation and gives further details on the evaluation settings used to evaluate HTTP-DTNSec. In addition, we present the preformed experiments and the obtained results. Finally, we discuss the obtained results.

4.1 Implementation and Environment Settings

HTTP-DTNSec was based on an existing implementation of HTTP-DTN [15]. Our codes were developed in Python due to the main libraries and the framework used to produce itself were elaborated on in this language. The following resources were used for its development:

- Twisted[1]: development framework for networking applications.
- Hyper-H2[2]: library for the development of HTTP/2 applications.
- GnuPG[3]: library that implements the OpenPGP format (encryption and integrity).
- SQLite[4]: simple database employed for package storage.

The proposed scenario simulates an environment where two nodes meet and exchange messages by sending and receiving packages. These received the identification of Node A and Node B, in both the Windows operating system was used and the HTTP-DTNSec protocol was installed. Node B is a virtual machine, whose host is Node A. Both were configured with fixed IPs in an isolated network, where only the two machines exist. Two text files of different sizes were created in node B, so that they could be sent to node A. The file sizes are respectively File1 with 24 bytes, File2 with 1216 bytes. Finally, to verify the packets exchanged between nodes, Wireshark packet capture software was used. Figure 5 represents the environment defined for the execution of experiments.

[1] Twisted - https://twistedmatrix.com.
[2] Hyper-H2 - https://python-hyper.org/projects/hyper-h2/.
[3] GnuPG - https://www.gnupg.org/.
[4] SQLite https://www.sqlite.org.

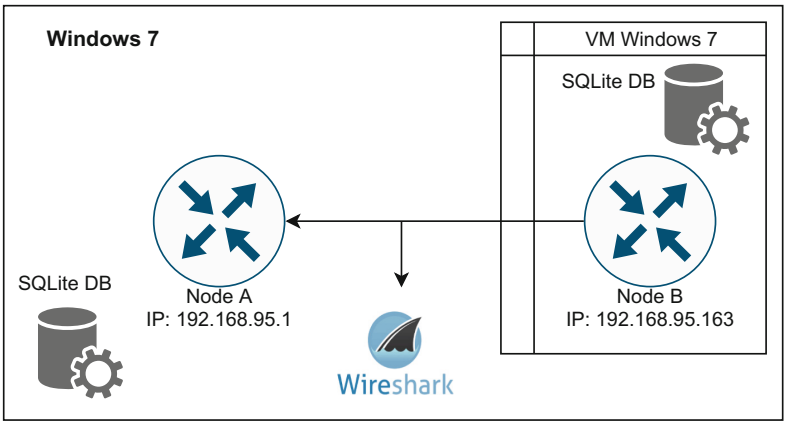


Fig. 5. Environment Settings for the HTTP-DTNSec evaluation

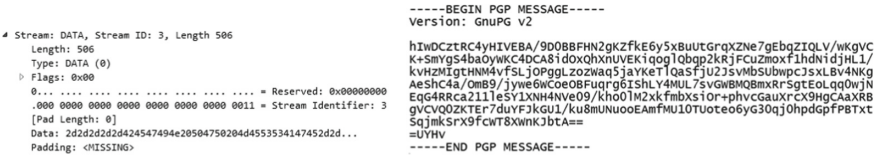


Fig. 6. HTTP-DTNSec send process

4.2 Experiments

Three meetings between the nodes were simulated in the proposed environment. At each meeting, one of the existing files in Node B was sent to Node A. In order to verify the changes to the HTTP-DTN protocol suggested in the 3 section, each meeting was also executed using it. The following captures made by Wireshark, correspond to the sending of the two files of different sizes. In the captures, only the DATA frames are shown using HTTP-DTNSec and beside the content of the Data field. Figure 6 represents the sending of File 2.

Finally, the following captures represent the number of messages exchanged by the nodes during a meeting, using HTTP-DTNSec and HTTP-DTN. Figure 7 shows the sending of a package using HTTP-DTN, while Fig. 8 shows the exchange of HTTP-DTNSec messages.

4.3 Discussion

When observing the packet captures of the meetings that used HTTP-DTNSec, it was possible to verify that the protocol is applying the encryption proposed in the 3 section when sending the frames that contains the packages. In addition, when analyzing the size of the packages, it was noticed that the OpenPGP encryption is actually using the compression described by [11]. This compression is causing

small files, when going through the encryption process, to have the size of their package increased, but large files are reduced in size. In Fig. 9, a comparison on the content sent size considering HTTP-DTNSec and HTTP-DTN.

Through the captures that demonstrate the amount of messages exchanged between nodes using HTTP-DTNSec and HTTP-DTN, it can be seen that the HTTP-DTN protocol during a meeting, only sends one message while HTTP-DTNSec sends four. However, it is important to understand that HTTP-DTNSec performs PUT and GET requests during a single connection, while HTTP-DTN would need to establish different connections for each of the requests.

5 Related Work

Several feasibility studies about using security properties in DTN point out the network communication overheads as a significant barrier. In light of this, this section presents the most prominent research initiatives related to DTN security.

Duarte *et al.* [9] present a solution for the autonomous management of computer networks. The authors also propose applying the solution in DTN environments using a proof of concept of HTTP-DTN to carry out the storage and sending messages between the network nodes. The results achieved by the authors show the operation of the HTTP-DTN protocol and that its use on the network did not increase significantly the traffic data flow. In addition, the open-source implementation of HTTP-DTN made by the authors served as a basis for the practical development of the present work.

Asokan *et al.* [1] discuss the application of integrity assurance in DTN bundles. This paper presents the problem of verifying integrity in fragmented bundles (e.g., a large bundle is divided into several smaller bundles). However, the hash function that is applied to guarantee integrity is used over the entire bundle, thus, in the case of a fragment loss, the receiving node is unable to obtain the same hash value. The authors suggest that the hash function be applied to each bundle fragment individually. Therefore, the integrity check can be performed even in the case of lost bundles and the receiving node can request selective retransmission. The authors state that their proposal has the disadvantage of increasing the network traffic and the cpu load of nodes.

```
15:33:30 192.168.95.163    192.168.95.1    HTTP    1514 Continuation
```

Fig. 7. Message exchange using HTTP-DTN

```
15:30:16 192.168.95.163    192.168.95.1      HTTP2    117 Magic, SETTINGS
15:30:16 192.168.95.1      192.168.95.163    HTTP2    102 SETTINGS, SETTINGS
15:30:16 192.168.95.163    192.168.95.1      HTTP2    721 SETTINGS, HEADERS, HEADERS, DATA, DATA
15:30:16 192.168.95.1      192.168.95.163    HTTP2     90 HEADERS, HEADERS
```

Fig. 8. Message exchange using HTTP-DTNSec

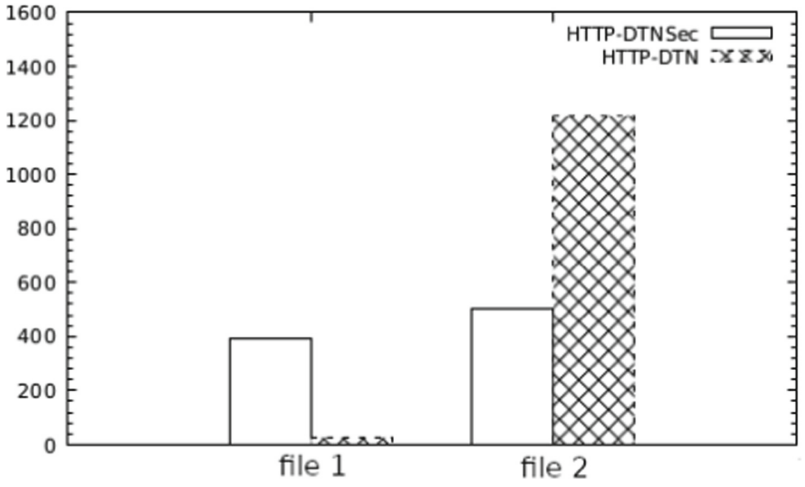


Fig. 9. Comparison on the content sent size considering HTTP-DTNSec and HTTP-DTN

Farrell and Cahill [10] analyze several issues related to DTN security. More specifically, they discuss some security decisions regarding the BP and BPSec. Also, the authors point out the reason for not using widespread security mechanisms, such as TLS, is often due to network limitations. Such limitations also hamper managing DTN keys, in which several messages are necessary to carry out the exchange of keys among nodes. The use of digital certificates may also not be feasible in many places, as there may be unavailability of connections when checking the validity with a trusted third party.

6 Final Remarks

Several security challenges can be highlighted in DTNs. Although security mechanisms are important in all communication networks, vulnerabilities are even more critical in some DTN environments, such as InterPlanetary Networking and Battlefield Networking. In the context of the IETF, the Bundle Protocol (BP) has been extended to enable safe messages exchange in challenging environments through the BPSec. In order to deal with the complexity of BP, alternatives with HTTP-DTN were developed. However, the security properties found at BPSec have not been extended to such alternatives.

In this paper, we propose the HTTP-DTNSec, a security extension for HTTP-DTN. Our extension aims to improve the confidentiality and integrity of HTTP-DTN, allowing it to have a security level comparable to BP. Besides, HTTP-DTNSec updates the version of HTTP used in HTTP-DTN. The performed evaluation shows that HTTP-DTNSec performed the transfer of packages in a safe manner and with an increase in performance concerning HTTP-DTN. The

main reason behind such increase is due to HTTP-DTNSec being able to send more than one request per connection through the use of HTTP/2.

Besides the encouraging results found in the experiments carried out, some future works can also be proposed. First, it is necessary to deploy and analyze HTTP-DTNSec on an operational DTN scenario. Secondly, key management mechanisms should be considered in HTTP-DTNSec, similarly to what occurs in BPSec. Finally, it would be interesting to investigate the use of HTTP/3 as a transport for HTTP semantics since the use of QUIC as a transport protocol can lead to different impacts on HTTP-DTNSec.

References

1. Asokan, N., Kostiainen, K., Ginzboorg, P., Ott, J., Luo, C.: Towards securing disruption-tolerant networking. Nokia Research Center, Tech. rep. NRC-TR-2007-007 (2007)
2. Belshe, M., Peon, R., Thomson, M.: Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540 (May 2015). https://doi.org/10.17487/RFC7540, https://rfc-editor.org/rfc/rfc7540.txt
3. Bhutta, M.N.M., Cruickshank, H., Nadeem, A.: A framework for key management architecture for DTN (KMAD): requirements and design. In: 2019 International Conference on Advances in the Emerging Computing Technologies (AECT), pp. 1–4. IEEE (2020)
4. Birrane, E.J., McKeever, K.: Bundle Protocol Security Specification. Internet-Draft draft-ietf-dtn-bpsec-27, Internet Engineering Task Force (February 2021). https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpsec-27, work in Progress
5. Burleigh, S., Fall, K., Birrane, E.J.: Bundle Protocol Version 7. Internet-Draft draft-ietf-dtn-bpbis-31, Internet Engineering Task Force (January 2021). https://datatracker.ietf.org/doc/html/draft-ietf-dtn-bpbis-31, work in Progress
6. Dalmazo, B.L., Vilela, J.P., Curado, M.: Online traffic prediction in the cloud. Int. J. Netw. Manag. **26**(4), 269–285 (2016)
7. Dalmazo, B.L., Vilela, J.P., Curado, M.: Triple-similarity mechanism for alarm management in the cloud. Comput. Secur. **78**, 33–42 (2018)
8. Ranjan Das, S., Sinha, K., Mukherjee, N., Sinha, B.P.: Delay and disruption tolerant networks: a brief survey. In: Mishra, D., Buyya, R., Mohapatra, P., Patnaik, S. (eds.) Intelligent and Cloud Computing. SIST, vol. 194, pp. 297–305. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-5971-6_32
9. Duarte, P.A.P.R., Nobre, J.C., Granville, L.Z., Tarouco, L.M.R.: A p2p-based self-healing service for network maintenance. In: 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, pp. 313–320. IEEE (2011)
10. Farrell, S., Cahill, V.: Security considerations in space and delay tolerant networks. In: 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT 2006), pp. 8-pp. IEEE, Pasadena (2006)
11. Finney, H., Donnerhacke, L., Callas, J., Thayer, R.L., Shaw, D.: OpenPGP Message Format. RFC 4880 (November 2007). https://doi.org/10.17487/RFC4880, https://rfc-editor.org/rfc/rfc4880.txt
12. Ivancic, W.D.: Security analysis of DTN architecture and bundle protocol specification for space-based networks. In: 2010 IEEE Aerospace Conference, pp. 1–12. IEEE (2010)

13. Kulkarni, L., Mukhopadhyay, D., Bakal, J.: Analyzing security schemes in delay tolerant networks. In: Satapathy, S., Bhateja, V., Joshi, A. (eds.) Proceedings of the International Conference on Data Engineering and Communication Technology. Advances in Intelligent Systems and Computing, vol. 468. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-1675-2_60
14. Menesidou, S.A., Katos, V., Kambourakis, G.: Cryptographic key management in delay tolerant networks: a survey. Future Internet **9**(3), 26 (2017)
15. Nobre, J.C., Duarte, P.A.P.R., Granville, L.Z., Tarouco, L.M.R.: Self-* properties and p2p technology on disruption-tolerant management. In: 2013 IEEE Symposium on Computers and Communications (ISCC), pp. 000676–000681 (July 2013). https://doi.org/10.1109/ISCC.2013.6755026
16. Paul, A.B., Biswas, S., Nandi, S., Chakraborty, S.: MATEM: a unified framework based on trust and MCDM for assuring security, reliability and QoS in DTN routing. J. Netw. Comput. Appl. **104**, 1–20 (2018)
17. Scott, K., Burleigh, S.C.: Bundle Protocol Specification. RFC 5050 (November 2007). https://doi.org/10.17487/RFC5050, https://rfc-editor.org/rfc/rfc5050.txt
18. Sipos, B.: Delay-Tolerant Networking UDP Convergence Layer Protocol. Internet-Draft draft-sipos-dtn-udpcl-01, Internet Engineering Task Force (March 2021). https://datatracker.ietf.org/doc/html/draft-sipos-dtn-udpcl-01, work in Progress
19. Sipos, B., Demmer, M., Ott, J., Perreault, S.: Delay-Tolerant Networking TCP Convergence Layer Protocol Version 4. Internet-Draft draft-sipos-dtn-tcpclv4-02, Internet Engineering Task Force (July 2016). https://datatracker.ietf.org/doc/html/draft-sipos-dtn-tcpclv4-02, work in Progress
20. Torgerson, L., et al.: Delay-Tolerant Networking Architecture. RFC 4838 (April 2007). https://doi.org/10.17487/RFC4838, https://rfc-editor.org/rfc/rfc4838.txt
21. Urunov, K., Vaqqasov, S., Namgung, J.I., Park, S.H.: Security issues for DTN mechanism of UIoT. In: The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014), pp. 1–3. IEEE (2014)
22. Wood, L., Eddy, W.M., Holliday, P.: A bundle of problems. In: 2009 IEEE Aerospace conference, pp. 1–17. IEEE (2009)
23. Wood, L., Holliday, P.: Using HTTP for delivery in Delay/Disruption-Tolerant Networks. Internet-Draft draft-wood-dtnrg-http-dtn-delivery-09, Internet Engineering Task Force (June 2014). https://datatracker.ietf.org/doc/html/draft-wood-dtnrg-http-dtn-delivery-09, work in Progress
24. Wood, L., Holliday, P., Floreani, D., Psaras, I., England, G.: Moving data in DTNS with http and mime making use of http for delay-and disruption-tolerant networks with convergence layers (2009)