A Mobile Code Bundle Extension for Application-defined Routing in Delay and Disruption Tolerant Networking

Carlos Borrego[a,*], Sergi Robles[a], Angela Fabregues[a], Adria Sánchez-Carmona[a]

[a]*Departament d'Enginyeria de la Informació i de les Comunicacions*
Universitat Autònoma de Barcelona
Barcelona, Spain

Abstract

In this paper, we introduce software code to improve Delay and Disruption Tolerant Networking (DTN) performance. DTN is extremely useful when source and destination nodes are intermittently connected. DTN implementations use application-specific routing algorithms to overcome those limitations. However, current implementations do not support the concurrent execution of several routing algorithms. In this paper, we contribute to this issue providing a solution that consists on extending the messages being communicated by incorporating software code for forwarding, lifetime control and prioritisation purposes. Our proposal stems from the idea of moving the routing algorithms from the host to the message. This solution is compatible with Bundle Protocol (BP) and facilitates the deployment of applications with new routing needs. A real case study based on an emergency scenario is presented to provide details of a real implementation. Several simulations are presented to prove the feasibility and usability of the system and to analyse its performance in comparison to state-of-the-art approaches.

Keywords: Active-DTN, aDTN, networking, DTN, routing, Bundle Protocol, Metadata Extension Block, mobile-code, disaster recovery

1. Introduction

Portable devices such as mobile phones or tablets are widely used in daily life. They are generally equipped with wireless-enabled communication, GPS receivers and/or touch screens. The existence of these devices has improved outdoor applications in a great variety of situations. Particularly, they can directly connect with each other. The most popular network configurations for this kind of connections are Ad hoc and Mobile Ad hoc (MANET) [4]. This kind of configurations does not require other infrastructure than the connected devices themselves. New communication paradigms are emerging to fill the void for some specific settings not covered by Ad hoc and MANET. This is the case of Delay and Disruption Tolerant Networking (DTN) RFC 4838 [15].

DTN implementations are extremely useful when no concomitant network links connect the source and the destination nodes at transmission time. This is typical in emergency and disaster recovery scenarios where the conventional network infrastructure collapses. It is also common in the space and in undeveloped areas where no conventional network is available. Ad hoc communications are defined when nodes keep connected solely during message transmission. This is also the case of MANET. The capability of DTN to work with intermittently connected nodes makes it suitable for scenarios like the previously described. The lack of infrastructure requirements makes DTN applicable to restoring network connectivity even co-existing with other networking solutions. DTN approaches provide a cheap, easy and ready-to-use deployment.

DTN has strong foundations such as the Bundle Protocol (BP), RFC 5050 [46]. Many groups have been working on their formalities for several years [22, 38]. Moreover, NASA is using DTN in the International Space Station [16]. However, there are still a number of issues to be solved being routing one of the most problematic. We consider routing as the process of selecting *which* messages are to be transmitted (prioritisation and lifetime control) and *where* to (forwarding). These issues need innovative solutions that have not been normally used on the Internet. The rationale for this is that applications running on such poorly connected networks require different routing algorithms for their specific problems. In contrast to what happens on the Internet, no general purpose routing algorithms exist which satisfy the requirements of all applications at once. One of the design principles of the Delay-Tolerant Networking Architecture, as defined in [15], is to:

"Provide coarse-grained classes of service, delivery options, and a way to express the useful life-

*Corresponding author
Email addresses: cborrego@deic.uab.cat (Carlos Borrego), sergi.robles@uab.cat (Sergi Robles), fabregues@deic.uab.cat (Angela Fabregues), adria.sanchez@deic.uab.cat (Adria Sánchez-Carmona)

time of data to allow the network to better deliver data in serving the needs of applications."

This principle can be summarised as "DTN routing algorithms must be application-defined". The problem lies in the fact that with current specifications, any DTN implementation must be devoted to a single application or at least to applications with similar routing needs. This often implies requiring to establish a network per application. With the aim of following this principle of allowing application diversity, we propose a solution without the need for any software deployment or maintenance. Our proposal incorporates the application-specific routing algorithms as software code to be carried by the messages. By this way, messages can be forwarded, lifetime controlled and prioritised using algorithms suitable for the applications to which they belong. The concept underneath is mobile code [42], that is, a well-known technology designed for this purpose.

In this paper, we define Active-DTN, that is, a DTN where messages have an active decision behaviour instead of being pure data wrappers. This behaviour makes possible the use of application-specific routing allowing, at the same time, the concurrent execution of applications with different routing needs. Active-DTN follows BP in order to be compatible with current implementations. It defines the Mobile code Metadata Extension Block (MMEB) to allow code to be carried, and three code types are proposed: the forwarding, the lifetime control and the priority of messages. This paper also provides the description of aDTN, the first Active-DTN implementation and a discussion about how can it be applied to a disaster recovery scenario with several rescue teams sharing the same network. The simulation experiments performed in the context of disaster recovery show the soundness of this proposal evaluating several components of Active-DTN.

The paper starts with all the relevant state of the art information, in Section 2, paying special attention to the BP. Next, we provide a full description of Active-DTN in Section 3 and its block definition as MMEB in Section 4. Section 5 defines three code types for forwarding, lifetime control and priority. In Section 6, some security considerations are discussed. Then, aDTN, the first Active-DTN implementation, is explained in Section 7. The paper follows with a discussion about its applicability to disaster recovery, in Section 8, followed by Section 9 with the simulation experiments. Finally, Section 10 contains the conclusions.

2. Related Work

There are two main developing paradigms related to networks characterised by intermittent connectivity, asymmetric bandwidths, long and variable latency and ambiguous mobility patterns. The most relevant to this study is the paradigm of the IRTF Delay Tolerant Network Research group.[1] They defined the DTN architecture, RFC 4838 [15], and BP, RFC 5050 [46], which are an abstract service description for the exchange of what they denoted by bundles in DTN. A bundle is a series of contiguous data blocks containing enough semantic information to allow the application to make progress where an individual block may not. These bundles carry the application information from a source to a destination following the store-carry-and-forward paradigm. That is: each node stores application data that can forward whenever the node contacts another node. The bundle architecture behaves as an overlay network.

The second paradigm belongs to the Haggle project, [45]. Haggle is a one-way communication architecture which its main purpose is to take advantage of brief connection opportunities. As in the BP, Haggle proposes solutions to scenarios with intermittent network availability suffering from long delays by switching messages and performing opportunity-oriented behaviours. Haggle allows the application messages on every DTN node to choose among a limited number of routing protocols. These routing protocols must be previously deployed. However, in intermittently connected scenarios, this deployment is not easy to conduct. In DTN, due to the idiosyncrasy of the network, performing such deployments does not guarantee that when a message arrives at an intermediate node their optimal routing protocol will be available. Additionally, these nodes are normally hardware-limited and consequently the local maintenance of the different routing protocols is not easy to perform, as pointed out in studies like [37].

The two previously introduced paradigms accept disruptions as the idiosyncrasy of the problem. Contrarily, studies like [1] and [29] propose different ways of linking the existing partitioned networks. These proposals may be useful in some situations. Unfortunately, they are essentially based on adding infrastructure elements to the network. This is not always feasible due to the complexity added to the system, the economic cost of the solution, or the difficulty of finding the best location for the links. Furthermore, most of these proposals fail to consider networks of mobile elements such as those required by a practical application such as disaster recovery [34].

Interesting proposals, such as [52], suggest the possible use of the software-defined network technology (SDN) [48] in DTN and in particular in emergency scenarios. SDN is an open standards-based and vendor-neutral network approach that allows the applications to centrally control the behaviour of the network in a very agile way. By means of a logically centralised software program, SDN separates the network control from the forwarding functions. However, we believe this could lead to many problems when applying it directly to DTN because the centrally managed approach is not appropriate for intermittently connected

[1]The main site of the IRTF Delay Tolerant Network Research group is `http://www.dtnrg.org`.

scenarios where the control information is not guaranteed to be properly distributed.

Some proposals have been published expressing concern about effective buffer management in DTN networks. Dimitriou et al. propose in [18] a way to accelerate transmissions by saving the bundle information in memory. In [23], Henriksson et al. propose as a routing strategy a ranking scheme using different classical caching models such as *most recently seen* and *most frequently seen*. Introducing queues in the bundle layer is neither a new concept. Lindgren et al. in [33] propose different strategies to drop bundles in the case the bundle cache buffer becomes full. In [27], the authors propose using information about encounters and locally collected statistics to derive an optimal policy based on global knowledge about the network. These strategies mainly use the bundle layer information. Other layers, like the application layer, should also be considered because they contain essential information. The criteria for bundle ordering or bundle dropping remain unchanged in each node, and they do not provide any deployment mechanisms to update these criteria dynamically taking into account the local context.

Transmissions in DTN networks can be much slower than in Internet Protocol (IP) networks. Information sent on DTN networks can travel from node to node for a very long period of time until it reaches its final destination. For some applications, the transmission speed is simply too slow. For the rest of applications, a priority mechanism among bundles is needed to guarantee an admissible transmission speed for critical application tasks. Nevertheless, Seligman et al. propose in [47] a solution to handle storage congestion that does not use priority. Instead, they propose to migrate storage data to neighbours. Other studies like [17] and [13] attempt to solve the same problem by proposing a hop-by-hop local-flow control mechanism. Unfortunately, these proposals take only into account local information and fail to consider data from other nodes.

Several proposals around the name of *active networks* were published during the early nineties. They are included in an interesting survey that can be found in [51]. These networks perform user-driven computation at network nodes in order to adapt the network to changing requirements. This is done by selecting the most suitable routing algorithm among a preset set of algorithms provided by the network. It is an approach towards application-specific routing although they have never been applied to DTN.

In [8], the authors propose using context-based information for routing in opportunistic networks. These studies are very interesting but, unfortunately, they do not provide the different applications that employ the network with ways of seeing this context information from the application perspective.

There are few studies using mobile code in DTN scenarios. [10] proposes a new paradigm called *store-carry-process-and-forward* that uses mobile code to improve the integration of wireless sensor networks and grid computing infrastructures. It proposes the implementation of a delay tolerant grid service, the computer element, to give computing access to an intermittently connected wireless sensor network. The result is an intelligent system which takes the routing problem, adapts itself dynamically to intermittent disconnections and improves the coexistence of multiple grid applications. Moreover, in [31] a similar proposal is introduced using message relay. Unfortunately, it is an algorithmic approach that does not use any application-specific routing algorithm. Furthermore, no architecture details nor implementation details were provided by the authors.

Additionally, the authors of the present study have recently published in [9] a study which presents a general purpose, multi-application robot sensor network based on mobile code. This intelligent system can work in DTN scenarios. Mobile nodes host software mobile code with task missions and act as DTN routers following the store-carry-and-forward paradigm. A proposal for a real-world application in the context of refugee camp management is presented as well.

Even though there has been considerable ongoing work on mobile code and DTN network infrastructure integration, there is not yet a standard solution flexible enough to fit the need for application-specific routing algorithm diversity in DTN. In this work, we outline a novel approach to overcoming this limitation and permit application-specific forwarding, lifetime control, and priority algorithms.

3. Active-DTN

Active-DTN is a DTN where bundles have an active routing behaviour. Instead of being data wrappers only, active bundles can carry software code for forwarding, lifetime control or prioritisation purposes. The movement of software code from a device to another is widely studied under the concept of mobile code. In Active-DTN, mobile code provides routing flexibility supporting the usage of many routing algorithms simultaneously. In fact, each bundle can carry its own routing algorithm to solve these three DTN issues. In other words, mobile code brings a solution for the need of having at the same time application-specific forwarding, lifetime control and prioritisation algorithms.

In Active-DTN, when a new application intends to use the network, no network reconfiguration or deployment is required. Moreover, in order to be compatible with existing DTN implementations, Active-DTN satisfies BP and defines a Metadata Extension Block (MEB) to specify how software code must be carried by bundles. See Figure 1 for a graphical representation of Active-DTN as a DTN extension including related work. Whenever an active bundle arrives at a non Active-DTN node, the routing code included in the active bundle will be ignored. However, the bundle will be processed in any case using the node's default algorithm.

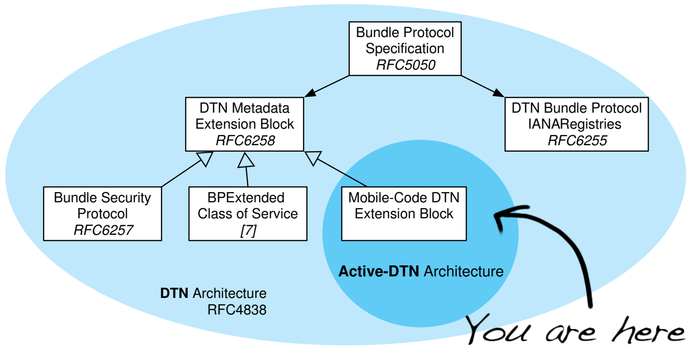

Bundle Protocol Specification RFC5050
DTN Metadata Extension Block RFC6258
DTN Bundle Protocol IANARegistries RFC6255
Bundle Security Protocol RFC6257
BPExtended Class of Service [7]
Mobile-Code DTN Extension Block
Active-DTN Architecture
DTN Architecture RFC4838
You are here

Figure 1: Active-DTN extends DTN architecture adding more flexibility being fully compatible with current specification of the Bundle Protocol.

Making a parallelism with the IP and its Routing Table, we define a data structure to be stored in Active-DTN nodes for routing purposes. The existence of a unique and known routing algorithm shared among all nodes makes the use of the routing table suitable for IP. Nevertheless, the existence in Active-DTN of several routing algorithms given by the applications being launched makes it necessary the use of a more flexible data structure. We propose the use of a tree structure as in the specification of the Simple Network Management Protocol (SNMP) [14] to store routing information. This tree is a way of representing the hierarchical structure of the routing information. We denote this tree by *Routing Information Tree* (RIT). Every Active-DTN node has one with different information. This information is used by the routing software codes to make decisions. Routing data are accessible through paths from the root of the RIT that we define as productions in ASN.1 [5]:

$< relPath >::=< word > \mid < relPath >$ "/" $< relPath > \mid$ ""
$< path >::=$ "/" $< relPath >$

Figure 2 represents an example of a RIT instance for a node with a local branch for position sensor data and the list of neighbour nodes. In addition, three branches are available for three applications with different routing needs. Their difference is clear due to the information stored in each branch. RIT branches are created by the routing algorithms themselves. The correspondence between applications and branches shown in Figure 2 is not necessary. Several applications can share the same routing algorithm, and thus, share the same RIT branch too. In fact, the name of the branch does not need to be the name of the application. It is the case in this figure for simplicity. Similarly, an application can send bundles using different routing algorithms.

The content of the RIT can be accessed and modified using primitives like `put` and `get`:

- `put(path,value)` stores a `value` at the given `path` of the RIT. Existing paths are overwritten. Non-existing paths are created in order to allocate the value.
- `value = get(path)` returns the `value` stored at the `path`. Non-existing paths imply an empty return.
- `set(path,label)` sets a `label` to a given `path` of the RIT.

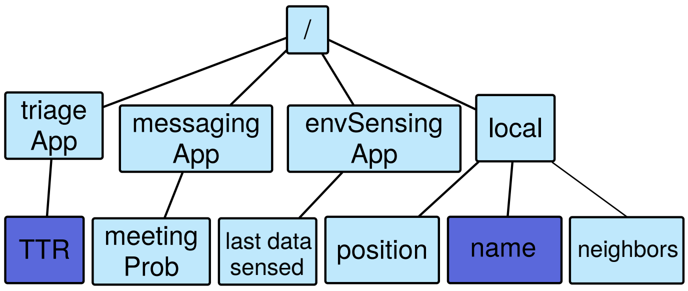


Figure 2: Example of a RIT instance depicted as a tree diagram with square boxes for path terms. In this RIT, three different applications and the local platform store data. Dark squared boxes represent RIT paths set as announceable.

Routing software codes may tag elements from the RIT as announceable using the *set* primitive. By this way, the applications can choose which routing information may be shared and spread over the Active-DTN network using beacon messages (broadcast messages sent to announce node's presence).

For an Active-DTN node to be able to execute a forwarding, lifetime control or prioritisation algorithm, the node must be able to understand its software code and know when it should be executed. Multiple software code languages are currently available in several formats: source code, bytecode, binary code, etc. Besides, the RIT structure may be accessed in very different ways. In order to specify all this information, a *software architecture* must be defined identifying all this.

4. Mobile code Metadata Extension Block

The specification of the bundle data block is included in RFC 5050. RFC 6258 defines how a bundle can be extended by means of adding a set of Metadata Extension Blocks (MEB). These extensions have been used in the literature for security [50] and priority [12] purposes as represented in Figure 1. Nevertheless, for Active-DTN, what we need is to be able to include software code in the bundle to include forwarding, lifetime control, and prioritisation algorithms. To that end, we define Mobile code Metadata Extension Block (MMEB) as a type of MEB with the necessary fields for the inclusion of software code. In this section, we provide a general description of the bundle and MEB data block structure and specify MMEB data structure.

The purpose of MEBs is to carry additional information that nodes can use to make processing decisions regarding bundles. The original bundle structure consists of a primary block followed by several payload blocks. The primary block contains the source and destination of the bundle using a service based on Endpoint Identifiers (EIDs). The payload is the data blocks to be transferred. MEBs extend the original structure of a bundle being included between the primary and the payload blocks. Figure 3 shows a graphical representation of a MEB illustrating its inclusion in a bundle.

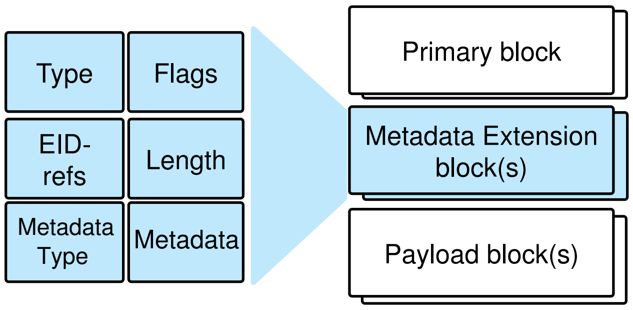


Figure 3: MEBs fit between the primary and the payload blocks of a bundle. Short field names are used to simplify its graphical representation.

The list bellow provides details about MEB fields using parentheses to remark the short field names included in Figure 3:

- Block-type (Type) is a single byte. According to RFC 6255 [7], the different Bundle block types have different block-type codes starting from the first available. The block-type code for the metadata block is 0x08.
- Block processing control flags (Flags) is a non-fixed length field encoded in Self-Delimiting Numeric Values (SDNV) format as defined in RFC 5050. Thus, this field follows the bundle definition described in the BP.
- EID-references (EID-refs) is an optional composite field that contains references to EIDs. When EIDs are available, the 6th bit flag denoted by "Block contains an EID-reference field" must be set. EID-references allow blocked MEBs to be ignored or employed only in some specific bundle nodes.
- Block length (Length) is the total MEB length expressed in SDNV format as defined by the BP for every block except the primary bundle block.
- Metadata type field (Metadata Type). Encoded in SDNV, this field indicates which metadata type is to be used to interpret the metadata field and the EID-references in the optional Block EID-reference field.
- The metadata (Metadata) is the bundle metadata. Its format depends on the software architecture that has been specified in this MEB.

Nevertheless, enabling bundles to carry software code requires representing this code in a bundle. To that end, we define MMEB as a new type of MEB and we propose to allocate the Block-type 0x10 from the *Bundle Block Types Registry* defined in RFC 6255.

MMEB splits the metadata field content of MEB defining several new fields most of them encoded in SDNV format. We define a CodeType field similar to the Block-type in MEB to allow the inclusion of three types of routing algorithms in a bundle: forwarding, lifetime control, and prioritisation.

Figure 4 illustrates the MMEB and its fields using short names as previously defined. A complete description of the MMEB fields is included bellow. Notice that short field names are represented between parentheses. The MMEB fields are:

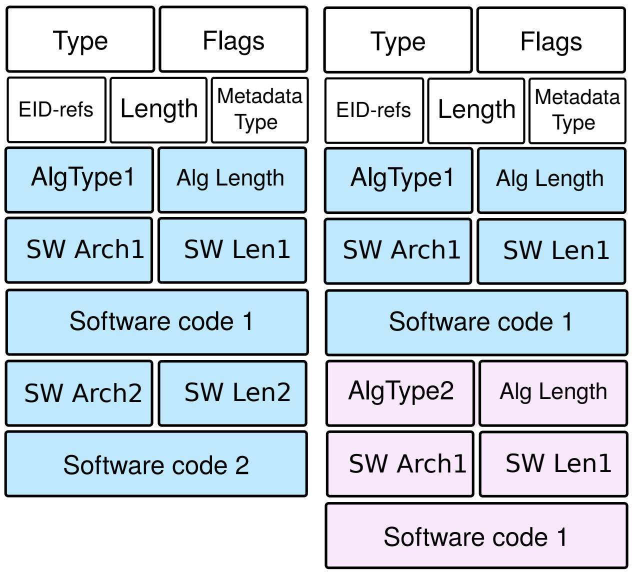


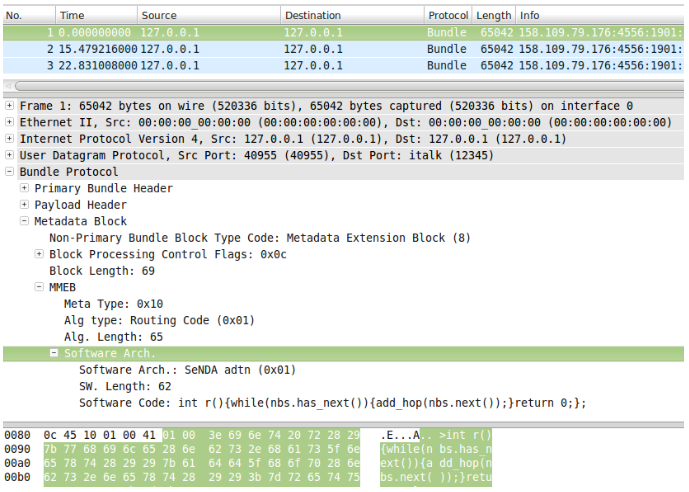
Figure 4: The MMEB splits the metadata field of a MEB. As a result, several new fields are added that are represented in dark. Two examples are depicted in this figure. For simplicity, short field names are used. At the left side of the figure, we represent a MMEB with a single-algorithm type, a forwarding code, for example. This algorithm type is represented using two different software architectures. At the right side of the figure, we represent a MMEB with two algorithm types, a forwarding code, and a prioritisation algorithm, for example. Both algorithm types are represented using the same software architecture.

- Algorithm type (AlgType). It is a two-byte field representing the type of algorithm. Three types of algorithms are described in this article: forwarding, lifetime, and priority.
- Algorithm length (AlgLength) is the total length for this algorithm expressed in SDNV format. Several algorithms can be included in one MMEB.
- Software Architecture (SW Arch). It is a two-byte field which identifies the software architecture as explained in Section 3. An example value for the *Software Architecture* field could be aDTN 1.0 platform, a software architecture that provides specific functions to access information such as the RIT, the format of

the software code or how to access the list of neighbours in order to implement the different algorithm types.

- *Software code length* (SW Len) is the length of the Software code field expressed in SDNV format.
- *Software code* (Software code) It is a field with variable length. This field contains the software code itself.

As introduced in Section 3, the software code has access to all the resources accessible by the node using the RIT functions provided by the software architecture. For security reasons, constrained access may be applied so that, for instance, a software code would have access only to particular branches in the RIT. In [43], the research group of the authors of this paper has already presented an identity-based access control system for code-contained bundles. This system uses bundle software code hashes to give access to keys that allow partial access to the RIT. Security services provided are confidentiality, integrity, and indirectly authenticity.



```
No.   Time          Source      Destination  Protocol  Length  Info
    1 0.000000000   127.0.0.1   127.0.0.1    Bundle    65042   158.109.79.176:4556:1901:
    2 15.479216000  127.0.0.1   127.0.0.1    Bundle    65042   158.109.79.176:4556:1901:
    3 22.831008000  127.0.0.1   127.0.0.1    Bundle    65042   158.109.79.176:4556:1901:

+ Frame 1: 65042 bytes on wire (520336 bits), 65042 bytes captured (520336 bits) on interface 0
+ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
+ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
+ User Datagram Protocol, Src Port: 40955 (40955), Dst Port: italk (12345)
- Bundle Protocol
  + Primary Bundle Header
  + Payload Header
  - Metadata Block
      Non-Primary Bundle Block Type Code: Metadata Extension Block (8)
    + Block Processing Control Flags: 0x0c
      Block Length: 69
    - MMEB
        Meta Type: 0x10
        Alg type: Routing Code (0x01)
        Alg. Length: 65
      - Software Arch.
          Software Arch.: SeNDA adtn (0x01)
          SW. Length: 62
          Software Code: int r(){while(nbs.has_next()){add_hop(nbs.next());}return 0;};

0080  0c 45 10 01 00 41 01 00  3e 69 6e 74 20 72 28 29   .E...A.. >int r()
0090  7b 77 68 69 6c 65 28 6e  62 73 2e 68 61 73 5f 6e   {while(n bs.has_n
00a0  65 78 74 28 29 29 7b 61  64 64 5f 68 6f 70 28 6e   ext()){a dd_hop(n
00b0  62 73 2e 6e 65 78 74 28  29 29 3b 7d 72 65 74 75   bs.next( ));}retu
```

Figure 5: Content of a bundle with a MMEB routing extension as shown by the Wireshark application. A simple epidemic forwarding software code can be read at the bottom part of the figure.

In Figure 5, a bundle is captured by the Wireshark network protocol analyser application [39][2] and its MMEB is shown containing a classic epidemic forwarding algorithm.

5. MMEB Code Types

This work is about allowing forwarding, lifetime control and prioritisation diversity by including the routing software code into bundles. As introduced in the previous section, MMEB block definition incorporates a CodeType field to specify the algorithm type of the software code. We propose the use of a code type registry similar to RFC 6255 [7]. In this section, we propose three CodeTypes. Those algorithms are denoted by forwarding (code 0x01), lifetime (code 0x02), and priority (code 0x03).

The **forwarding algorithm** refers to the path decision task. It takes the bundle destination as a parameter and computes the list of nodes where the bundle must be forwarded to. This list is named *nextHopList* following the terminology presented in [15].

$$nextHopList \subseteq neighbourNodes \cup \{node\}$$

The list of nodes where the bundle must be forwarded to (*nextHopList*) must be a subset of the set formed by the list of neighbours (*neighbourNodes*) and the local node (*node*). This latter is added to this set to allow the routing algorithms to keep a copy of the bundle in the local node. Contrarily, an empty list being returned means that the bundle can just be discarded.

Lifetime control is crucial to avoid bundles endlessly circulate through the network. The Internet Protocol (IP), for example, uses the Time to live (TTL) field included in each datagram. This lifetime control mechanism consists of a number that gets decremented by every router it goes through. When the TTL is 0, the datagram is discarded. A similar approach has been proposed in [21] for BP. However, the time to deliver a message is much more difficult to foresee in BP than in IP. We claim that, in DTN, lifetime control algorithms must be more complex than simple counters or timestamps attached to the message because of the heterogeneity and unpredictability of the network. We propose using application-specific software codes and to include them in bundles as we already do with forwarding algorithms. Thus, every bundle can carry its own lifetime control algorithm that may be evaluated on every bundle hop to decide whether a bundle should be discarded or not. This decision is made from the point of view of the application in terms of the local context found in the RIT. This mechanism allows applications to decide how to discard their bundles in terms of their own criteria beyond the classical time-based approaches. An example for contextual application-based lifetime control, which will be extended in Section 8.2, could be an algorithm that uses the node's GPS position to discard bundles depending on their location.

Lifetime control has no parameters. However, it can query and modify the RIT. The purpose of this procedure is to compute whether the bundle must be discarded or not. Therefore, it returns a Boolean value where `0x1` means true, and `0x0` means false. Next line represents a call to this procedure:

```
mustBeDiscarded = lifetimeControl()
```

Another crucial issue is bundle prioritisation. By setting bundle priorities, we establish a processing order among

[2]Source code for the Wireshark aDTN extension can be found at: *https://senda.uab.cat/wiki/aDTN* in Section *Wireshark aDTN Bundle Plug-in*.

bundles in custody. The BP contains two bits to define three types of priorities in its header. Using the MEB, the extended class of service described in [12], enhances the BP allowing the sender to specify additional priorities among bundles. These priorities, defined in RFC 5050 and [12], are equivalent to those defined in the *Differentiated Services* field in the Internet Protocol Suite. They define an order for IP datagrams based on precedence classes. This field could force routers to act on datagrams. As described in [30], it is seldom used, and it does not guarantee the priority set. However, we claim that in DTN prioritisation algorithms must be application-specific and included into bundles as software code. The idea is to allow priorities to be dynamic in terms of the context. By this way, the application may control bundle prioritisation on every hop in a very flexible and context-aware way.

Bundle prioritisation has no parameters. However, it can query and modify the RIT. This procedure computes the priority level of a bundle and returns a positive integer number representing this level being `0x00` the highest priority, and `0xFF` the lowest. Next line represents a call to this procedure:

```
level = bundlePriority()
```

The use of MMEB provides much flexibility for the definition of application-specific algorithms that can make use of contextual information stored in the node's RIT. Nevertheless, there is no guaranty on information to be present and updated in the RIT. Therefore, it is recommended to use a defensive programming strategy to code the procedures to include in MMEB in order to prevent errors to occur. In any way, in case of error, the node executes the default procedure with the same type of algorithm.

6. Security considerations

In the context of DTN networks, security is a very complex issue to achieve. There is a wide range of security services that should be analysed in this type of networks. Services that are usually secured in recent publications, like, for example [19], are authentication and privacy. However, in this section we want to focus on the security problems derived from the fact of having software code in the messages. We discuss in this section several solutions to be considered to improve the security of our proposal. On one hand, we propose to use the standard Bundle Security Protocol to obtain MMEB authentication and integrity services. Secondly, we discuss how to define security services to the access the RIT. Finally, we propose the sandboxing technique as a way to limit the execution environment to routing software codes.

Routing software codes proposed in this study may be signed by software providers, the routing software code programmers. These providers may be the source application itself or an external entity. Software providers may sign routing software codes so DTN nodes may decide on every bundle hop whether to execute the routing software codes or not. This decision is done in terms of the trust relationship with the software provider.

Routing software code signatures and its corresponding signer certificates can be added to the MMEB as an additional field to the ones presented in Section 4. However, we believe that using the standard Bundle Security Protocol (BSP) [20] is an elegant and standard solution to provide routing software code authentication and integrity. This protocol provides data integrity and confidentiality services for the Bundle Protocol. Different services are provided to protect the bundle payload and the rest of the blocks from the bundle. These services are the Bundle Authentication Block (BAB), the Payload Integrity Block (PIB), the Payload Confidentiality Block (PCB) and the Extension Security Block (ESB).

We consider that the ESB may be very useful to provide to our proposal with software code authenticity and integrity. This security solution places the ESB in the bundle in the same position as the MMEB we want to secure. The original MMEB block that contains the routing software code is signed by the source application and encapsulated in an ESB block. From the point of view of the bundle, this block is placed at the same sequential position as the original MMEB block.

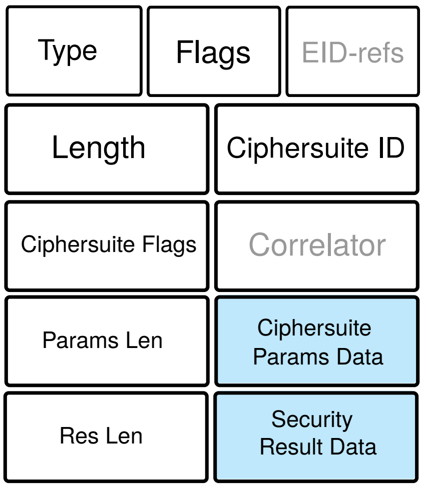


Figure 6: Extension security block fields. Dark background fields contain MMEB information. The MMEB is placed in the *Ciphersuite Params Data*. Its signature with the key identifier or certificate used to sign in the *Security Result Data* field. Depicted in light gray font, fields that are not needed.

Following the BSP, we discuss now how the ESB should be defined as presented in [19]. In Figure 6, all the fields of the ESB are depicted.

- The *Type* field should be set to *0x09*, indicating that this block is an ESB.

- The *Flags* should be defined as in all Bundle Protocol blocks except the primary bundle block, as described in [46].

- The *EID-refs* field is not needed because no additional Endpoint Identifiers must be referenced.
- The *Block data length (Length)* field indicates the length of the block, as in all Bundle Protocol blocks except the primary bundle block.
- The *Ciphersuite ID* indicates the cryptographic algorithms and implementation rules to provide the security services. In our case, a new ciphersuite must be defined[3] to provide the hop-by-hop routing software code authentication service.
- *Ciphersuite Flags* indicate which fields are present in this block.
- The *Correlator* field is used when more than one related block is inserted. Since the MMEB is unique, this field is not needed.
- The *Ciphersuite Flags* field informs about the presence of security-source EID, security-destination EID or ciphersuite-parameters. In our case, only the ciphersuite-parameters flag should be set.
- The original MMEB routing extension is placed in the *Ciphersuite Params Data* field as an item *10: encapsulated block* with length *Params Len*.
- The key or certificate used to sign the MMEB and the signature is placed in the *security-result data* as item *3: key-information* with length *Res Len*.

Secondly, the access to the RIT is also an issue that should be discussed from the security perspective. It is desirable to preserve confidentiality and integrity in the routing data stored in the RIT. For this purpose, there are many studies that propose different solutions for access control in similar structures like the RIT, like, for example [3]. However, these solutions are not applicable when it comes to scenarios where the network connectivity is intermittent.

We propose to employ security strategies like [2] where the security access control is based on the identity of the requester. Services covered by this proposal guarantee the confidentiality and integrity of the information used for routing published on every DTN node in the RIT. The idea behind this publication can be applied to our proposal: access control can be performed in terms of the very same routing algorithm. When a routing software code requests access to some branch from the RIT, access control is evaluated by the DTN node using a hash function of the very same routing software code.

Finally, in order to allow unverified applications to run untrusted programs, the sandboxing technique can be extremely useful. Sandboxing [41] is a form of software virtualization that limits the execution environment to software codes. This technology is far from being a research-only product: there are many examples of sandboxing, such as Google Native Client [53], that are being used in browsers to allow web-based applications to run at near-native speeds.

The objective of using sandboxing in our proposal is to control the local resources of the DTN node that is executing the routing software codes. These resources include file descriptors, memory, file system space, RIT branches or access to other bundles in custody. Sandboxing may be used without the ESB, or it can be applied in combination with it to improve the DTN node security.

The main challenge for network security in aDTN networks is preventing malicious code actions. The scheme presented in this section is a possible way to solve it, but each particular scenario could require different solutions, taking advantage, where possible, of other available security tools such as public-key infrastructures or cryptographic token devices.

7. aDTN: an Active-DTN implementation

The first implementation of Active-DTN is the Active-DTN platform named aDTN[4]. This aDTN is a Bundle Protocol agent, as defined in RFC 5050 [46], which includes a software platform that supports Active-DTN's Mobile code Metadata Extension Blocks (MMEB). In Figure 7, its different modules are depicted. This platform has been implemented by our research group SeNDA[5] taking the BP Application Server as reference and supporting the basic three MMEB code types: forwarding, priority and lifetime.

The main component of aDTN is the Bundle Agent that has a modular structure. Bundles arrive at the Bundle Agent through the Bundle I/O Manager that is a module listening for bundles either coming from the network or applications running on the DTN node. The Bundle I/O Manager checks bundle destination in order to delegate bundles whenever is possible. Otherwise, the bundle is enqueued. To keep the bundle agent delay and disruption tolerant, the communication between platform applications and the Bundle I/O Manager is asynchronous.

The module in charge of forwarding bundles is the Custody Manager Module. This module dequeues bundles in order to execute a forwarding algorithm for them. Depending on the output of the execution, the bundle is forwarded or enqueued again. See Algorithm 1 for more details. This algorithm loops over the bundles to be processed (m) and then over the available neighbours (n). The computation complexity of this procedure is quadratic ($O(mn)$). This complexity is similar to other Custody

[3]We suggest defining ESB-RSA-SHA256-EXT with ciphersuite ID value 5 (first available in the IANA registry).

[4]Source code can be found at: *https://github.com/SeNDA-UAB/aDTN-platform*.

[5]Security of Networks and Distributed Applications (SeNDA) is a research group within the Department of Information and Communications Engineering of the Universitat Autònoma de Barcelona.

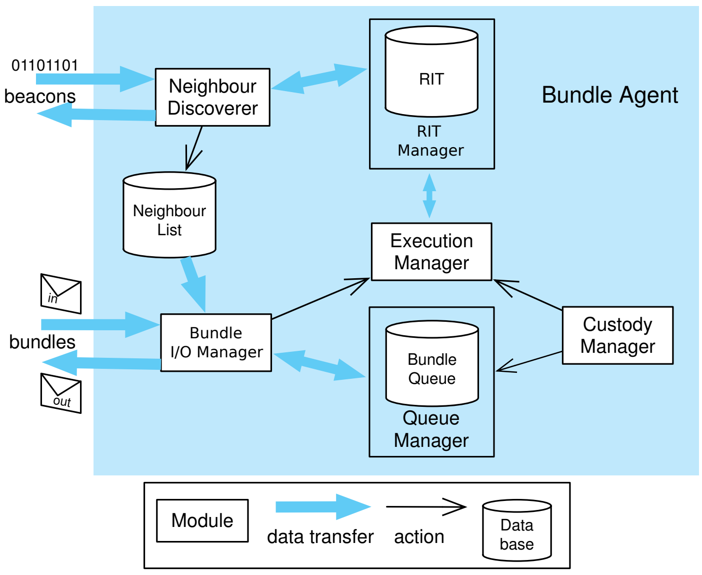


Figure 7: Structure of the aDTN platform.

Managers from other implementations of the Bundle Protocol such as DTN2 [40] since they behave in a very similar way. Concerning its space complexity, the amount of space the Custody Manager employs is equal to every Bundle Protocol implementation: linear to the number of bundles in custody. Bundles are written to disk to provide persistence and to prevent message volatility if the node is powered off.

Algorithm 1 Bundle Agent using the Custody Manager to process bundles.

```
Ensure: executor is a reference to the Executor Module with procedure execForwarding(·) for executing the forwarding algorithm.
Ensure: queue is a reference to the bundle queue with procedures enqueue(·) and dequeue(·).
Ensure: forward(·) is a procedure for forwarding bundles to other nodes.
Ensure: FORWARDING is a constant representing the MMEB codeType value for the forwarding algorithm.
1:  procedure PROCESS
2:      loop
3:          bundle ← queue.dequeue()
4:          nextHopList ← executor.exec(FORWARDING, bundle)
5:          for node ∈ nextHopList do
6:              if thisNode = node then
7:                  queue.enqueue(bundle)
8:              else
9:                  forward(node, bundle)
10:             end if
11:         end for
12:     end loop
13: end procedure
```

The different routing algorithms to execute for a given bundle are the ones the very same bundle contains. Therefore, when available, the software code of the bundle's MMEB for routing must be used. Nevertheless, when this extension is not present, or there is no software code in it that is compatible with the platform,[6] then a default routing algorithm must be executed.

For security reasons, the execution of software codes provided by the bundles must be supervised. For instance, access to system functionality must be denied by default. Only access to particular structures like the RIT is available. Moreover, the execution time is limited. When the execution time limit is reached, the execution must be cancelled, and a default software code must be executed instead. Similarly, a default software code must be executed whenever the bundle's software code execution fails.

The module in charge of the software code execution is denoted by the Execution Manager. This module is aware of the format that is supported by the platform; it supervises the software code execution and knows what are the default software codes to use in case they fail. The Execution Manager is available for running algorithms software codes. The aDTN platform is prepared to receive forwarding, lifetime and priority software codes.

Current aDTN implementation supports ISOC99-compliant code only. TinyCC [6] is used to compile the C software code because it can quickly [7] compile to native x86, x86-64 and ARM code in devices with slow processors and few disk space. Support to other software code language formats can be provided by extending the Execution Manager.

The access to the bundle queue is regulated by the Bundle Queue Manager that provides internal services for enqueuing and dequeuing bundles. Moreover, this module is in charge of controlling bundle lifetime and re-scheduling the queue by priority. In order to perform those tasks, a procedure similar to Algorithm 1 line 4 is provided.

The RIT is implemented in aDTN as a JSON file [8] where elements are tree nodes. Elements can have several tags but only a single value. The current version of the aDTN RIT supports content announcement with *announceable* tags set to true. Tags can also be used to extend the implementation to support access restriction to particular branches only. The interaction with the RIT is only possible through the RIT Manager that is a module that provides a public API and several internal services. All primitives introduced in Section 4 have been implemented. Sensors connected to the platform can write information into the RIT using the public API. For example, it is pos-

[6]As described in Section 4, several software codes can be included per MMEB. Active-DTN nodes are not required to support any specific format.

[7]For example, a TTR-based forwarding algorithm in a2.6 GHz Intel Core i5 takes on average to compile 0.048s.

[8]JSON (JavaScript Object Notation) is a lightweight easy to read and write data-interchange format to transmit data objects consisting of attribute/value pairs (*http://www.json.org/*).

sible to set a GPS sensor to keep the local position of the platform up to date by updating a special RIT path.

aDTN platform nodes advertise their presence sending simple beacon messages. A node is considered a neighbour if a beacon message has recently been received from it. Beacon messages contain the node end-point identification. In addition, as introduced in Section 3, they can also include a small amount of announceable information from the RIT. Notice that, by this way, routing information can be spread over the Active-DTN network. The Information Exchange Module is in charge of periodically sending beacon messages and keeping neighbour information up to date. To that end, this module makes use of a RIT Manager internal service that provides announceable information. Additionally, this module decides the amount of information to be shared. An example of content access restriction in the RIT is the writing access to the neighbour list that must be only possible from the Information Exchange Module.

The current implementation has been coded in C language and provides a communication software architecture for applications to interact with the platform. As in TCP/IP, an aDTN socket is provided that needs to be bound before being used. The Software Code 2 provides a programming example on how to send bundles with MMEB extensions. In this example, a bundle is sent containing the forwarding software code provided in Software Code 1 as MMEB extension. Notice that the current implementation supports a C-based software architecture. In Software Code 1, an example of a routing code is presented.

```
// Forwarding algorithm
int forwarding(char* dest){
        nbs=RIT.getNeighbors();
        //Starts decision
        int minttr = 100; // maxTTR=100
        int ttr = 0;
        int len;
        while(nbs.has_next()){
                char triageBranch[160] = {0};
                char* endPointId = nbs.next();
                snprintf(triageBranch, 159,
"/%s/TTR", endPointId);
                ttr = atoi(RIT.get(triageBranch));
                if(ttr < minttr){
                        minttr = ttr;
                        cln_hops();
                        add_hop(endPointId);
                }else if(ttr == minttr){
                        add_hop(endPointId);
                }
                add_hop(nbs.next());
        }
        return 0;
};
```

Software Code 1: Example of forwarding algorithm software code. The list of neighbours in this software architecture is obtained by the function *RIT.getNeighbors()*. DTN nodes are added to the list of nodes to be forwarded using the *add_hop(endPointId)* function. Access to the RIT is done by the *RIT.get(branch)* function.

```
  // Creating a bundle socket
  int bundle_sock = adtn_socket();
  // Setting data for origin and destination
  sock_addr_t origin, destination;
  ...
  // Binding the socket
  adtn_bind(bundle_sock,&origin);
  // Adding a routing code type MMEB
  adtn_setcodopt(bundle_sock, 0x01,
"nbs=RIT.getNeighbors(); int minttr = 100; int ttr
= 0; int len; while(nbs.has_next()){ char
triageBranch[160] = {0}; char* endPointId =
nbs.next();          snprintf(triageBranch, 159,
"/%s/TTR", endPointId); ttr =
atoi(RIT.get(triageBranch)); if(ttr < minttr){
minttr = ttr; cln_hops(); add_hop(endPointId);
}else if(ttr == minttr){ add_hop(endPointId);}
add_hop(nbs.next()); } return 0; ");
  // Sending data using the bundle socket
  adtn_sendto(bundle_sock, destination, "This is
the payload.");
```

Software Code 2: Example of C code used to send a bundle with the forwarding MMEB software code developed in Software Code 1.

8. Discussion of a Practical Application

Disaster recovery actions after emergencies such as terrorist attacks or meteorological calamities are difficult to conduct. A priori connected areas become precipitously disconnected and isolated. The high deploying speed of DTN networks makes them an excellent solution for communication until the original network is restored[9]. In this section, we define an emergency scenario based on [34] and use it as case base scenario to show the advantages of using an Active-DTN network.

Medical Emergency Team (MET) members arriving at the disaster scenario follow an emergency protocol that consists in localising victims and classifying them given their health condition. This classification is denoted by *triage* and defines four different victim statuses: status 0 for deceased, status 1 for seriously injured, status 2 for injured and status 3 for mildly injured. MET members move freely around the disaster area collecting information and providing it back to the Emergency Coordination Centre (ECC). It is crucial to perform all this process as quick as possible as the soonest the information arrives at the ECC, the earliest the victims will be attended and the whole disaster scenario will be recovered. Notice that no network available means that each MET member must collect the data and carry it at least until he/she returns to the ECC. That is, MET members must constantly be deciding whether to continue collecting data or returning to the ECC for these data to be used. Moreover, it also means that much time is lost while people are going and returning to the ECC that, for logistic reasons, may be relatively far away from the critical area.

A DTN network can be deployed in disaster scenarios using mobile devices[10] carried by the members of the rescue

[9]Some kinds of disaster require several days or even weeks to restore the original network.

[10]By mobile devices we refer to mobile phones, tablets and similar global purpose devices that people daily use.

team and sensor devices deployed at the ECC and along the disaster area as illustrated in Figure 8. Mobile and sensor devices are the nodes that form the intermittently-connected network that use opportunistic contacts among nodes to allow bundles to jump node to node from the source towards the destination.

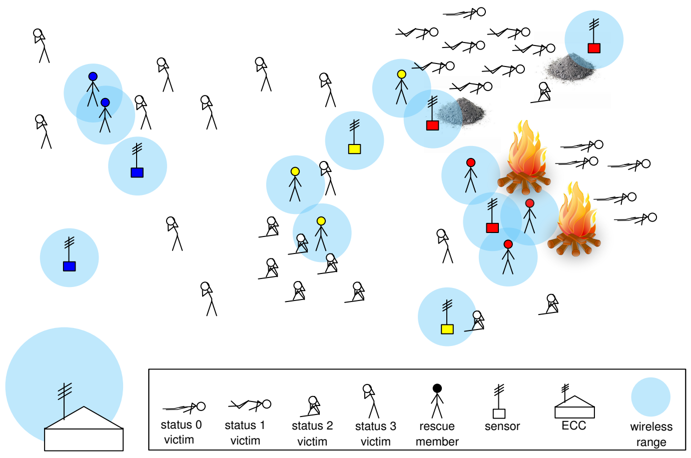


Figure 8: Disaster scenario with victims spread around and the rescue team performing the *triage*. The ECC is already deployed as well as some portable devices. The rescue team members are carrying mobile devices. Outer circles around each of them represent wireless ranges. There are three groups of team members: medical (yellow), firemen (red) and policemen (blue).

Using a regular DTN benefits the MET efficiency letting them send *triage* information about victims to the ECC. This application creates a bundle per victim containing the victim location and status. The destination of this bundle is the ECC. Thus, the bundle can be forwarded to the mobile devices of other MET member contacted in the field to take advantage of their trips to the ECC. As a direct consequence, the number of trips can be reduced. DTN provides a good solution to MET efficiency problems. However, this solution could be not enough for environments where MET members do not regularly contact. In addition, there are many other applications for supporting the whole rescue team that would not be compatible with the *triage* application because, as explained in previous sections, DTN performs very well for applications that share the same forwarding algorithm. To get the benefits of DTNs and provide forwarding diversity support among several applications, we propose to use Active-DTN.

Active-DTN provides a better solution to emergency scenarios than traditional DTN. It engages all kind of rescue team members to participate in the network with application diversity. We envisage many useful applications ranging from internal messaging to environmental sensing applications. Among the latest, we would like to mention the following three: pollution and nuclear radiation measurement,[11] smoke and fire detection,[12] sound and movement detection,[13]. The coexistence of all these applications using the same network decreases the global deploying cost and time. Furthermore, it increases the frequency of node contacts as more users would be willing to contribute to establishing the network using their devices.

In the following sections, we discuss some issues that emerge from applying Active-DTN to disaster recovery and how we deal with them.

8.1. *Application-specific forwarding diversity*

In emergency scenarios, there is a need for several applications to coexist, for example, triage applications, internal messaging, and environmental sensing. However, there is no single-forwarding algorithm suitable for all kind of applications. The most suitable forwarding algorithm for triage application is based on the fact that, for security reasons, MED members must notify to the emergency coordinator the expected time to return (TTR) to the ECC. The neighbour nodes with soonest TTR are then selected by the forwarding algorithm as the next hop. Algorithm 2 describes an example of forwarding algorithm based on TTR for the triage application. It assumes that the RIT is similar to the one represented in Figure 2 with several nodes at the */triageApp/TTR/* path representing the TTR value of those nodes.

As it happens with other routing algorithms, such as [32], Algorithm 2 needs to compare local values to others from every single neighbour to make a routing decision. As a consequence, the computation complexity of this algorithm is linear to the number of neighbours a node has. Other routing protocols, like Spray and Wait [49], do not need to iterate over the available neighbours if, for example, the number of copies of a message to be forwarded is less than the number of neighbours. Instead, the space computational cost is constant: the local TTR variable is the only variable that should be stored in the node so it can be compared to the ones from its neighbours.

In contrast to the triage application, a probabilistic algorithm is more suitable for internal messaging in order to take advantage of the fact that some rescue team members meet more often than others. Similarly, this happens with environmental sensing applications. These applications use epidemic algorithms in order to spread alarms rapidly when danger is detected. Following our approach, many forwarding algorithms can coexist being very rapid to deploy as a set of MMEB. In Algorithm 3, an example of epidemic forwarding algorithm is represented.

[11]The Fukushima incident just after the tsunami required nuclear radiation measurement.

[12]Smoke and fire detection applications can help firemen to be aware of apparently extinct fires to come alive from their ashes in the forest.

[13]Sound and movement detection applications can help rescue teams finding victims under collapsed buildings after an earthquake.

Algorithm 2 A forwarding algorithm based on TTR.

```
Require: maxTTR is the highest possible TTR
Require: rit is a reference to the RIT
Require: "/local/neighbours" is the standard path to the neighbours stored in the RIT
Ensure: Returns the set of nodes where the bundle must be forwarded to.
 1: procedure FORWARDING(Destination)
 2:     NeighbourNodes $\leftarrow$ rit.get("/local/neighbours")
 3:     if Destination $\subseteq$ NeighbourNodes then
 4:         return Destination
 5:     end if
 6:     Dest $\leftarrow$ Destination $\cap$ NeighbourNodes
 7:     min $\leftarrow$ maxTTR
 8:     NextHop $\leftarrow \emptyset$
 9:     for node $\in$ NeighbourNodes $\setminus$ Dest do
10:         ttr $\leftarrow$ rit.get("/triageApp/TTR/"node)    ▷ Getting the TTR info for this node from the RIT.
11:         if ttr $<$ min then
12:             min $\leftarrow$ ttr
13:             NextHop $\leftarrow \{node\}$
14:         else if ttr $=$ min then
15:             NextHop $\leftarrow$ NextHop $\cup\{$node$\}$
16:         end if
17:     end for
18:     return NextHop $\cup$ Dest
19: end procedure
```

Algorithm 3 The traditional epidemic forwarding algorithm.

```
Require: "/local/neighbours" is the standard path to the neighbours stored in the RIT
Ensure: Returns the set of nodes where the bundle must be forwarded to.
 1: procedure FORWARDING(Destination)
 2:     NeighbourNodes $\leftarrow$ rit.get("/local/neighbours")
 3:     if Destination $\subseteq$ NeighbourNodes then
 4:         return Destination
 5:     end if
 6:     return NeighbourNodes $\cup$ {thisNode}
 7: end procedure
```

In fact, apart from allowing application-specific forwarding, our approach supports bundle-specific forwarding letting different forwarding algorithms to be carried by bundles of the same application. This is especially suitable in the case of environmental sensing applications. In these applications, epidemic forwarding can be limited to relevant sensed data while regular data can be communicated using, for instance, Spray&Wait's algorithm [49]. Additionally, different initial number of copies can be set depending on how critical the sensed data is.

8.2. Bundle congestion alleviation

Congestion alleviation in Active-DTN is application-specific being mainly performed by the lifetime control and the prioritisation algorithms. The amount of bundles in custody that a node can support is finite. Using aDTN nodes, the maximum number of supported bundles in custody is given by the capacity of the Custody Manager queue. When the bundle queue is full, and a new bundle needs to be in custody, any of these bundles in custody must be discarded. For some applications, losing some bundles could have almost no effect. This is the case, for instance, of bundles belonging to an environmental sensing application that contain information about non-critical sensed values. However, for other applications, the damage is unacceptable. This is the case of the triage application where discarding a bundle means leaving a victim unattended.

Lifetime control is crucial to alleviate bundle congestion. Bundles may be perishable, the information that they contain may decay on time. We claim that only the application can know whether a bundle is not useful anymore, and this will depend on the context of the node. The common approach is similar to using the TTL field from IP and discard all bundles after a particular number of seconds or hops. Instead, what we propose is to let the bundles carry its own lifetime control algorithm. Note that this empowers bundle lifetime control because it allows to take into account the context of the node from the point of view of the application. As an example of the flexibility that this kind of lifetime control provides, we define Algorithm 4, a lifetime control algorithm based on the node's GPS position. Disaster scenarios are geographically restricted to a given damaged area. Bundles being carried by nodes that have left this area can be safely discarded. This is not possible using traditional TTL-like lifetime control approaches.

Algorithm 4 When using this GPS position based lifetime control algorithm, bundles are discarded if they are located outside a given area.

```
Require: rit is a reference to the RIT.
Require: isDamagedArea($\cdot$) is a predicate that returns whether a GPS position is included in a damaged area.
Ensure: Returns whether the bundle must be discarded.
 1: procedure LIFETIMECONTROL
 2:     position $\leftarrow$ rit.get("/local/position")
 3:     return $\neg$ isDamagedArea(position)
 4: end procedure
```

When the previously described congestion alleviation method is not enough, the node can still discard the last bundles of the queues. In this case, bundle prioritisation is crucial to prevent high-priority bundles to be discarded. According to the aDTN model, the Custody Manager scheduling policy states that less prioritised bundles will be at the end of the queue. Therefore, no major computation is necessary to decide which bundle to discard.

8.3. Contextual bundle prioritisation

As introduced before, bundle prioritisation is used for scheduling bundles in custody. The capability of Active-DTNs to use application-specific bundle prioritisation algorithms brings the possibility to define context-aware algorithms that can take advantage of application-specific routing information stored in the RIT to deal with priority criteria changes. For instance, in Triage the prioritisation algorithm adjusts the bundle priority according to the victim status and the danger level of the damaged area.

According to the [34], status 1 victims are seriously injured victims. Given the fact that status 0 victims are already deceased, it seems reasonable to give maximum priority to any information related to status 1 victims. However, this depends on the danger level in the damaged area. When the danger level is very high, rescue team members can be ordered to leave seriously injured victims die in order to save their own life. Assuming a danger protocol that defines 5 danger levels being level 4 the highest, a bundle priority algorithm like Algorithm 5 would be appropriate for status 1 victim bundles. As represented in the algorithm, when the danger level is lower than 3, the bundle has a high priority in order to reduce the rescue time, see line 4. Contrarily, when danger level is higher, the bundle priority decreases a great deal as it will only be used for counting deceased victims, see line 6. This algorithm is a good example of the degree or flexibility provided by Active-DTN in comparison to traditional DTN approaches where all this application-based context-aware decisions cannot be implemented.

Algorithm 5 Triage bundle prioritisation for status 1 victims.

```
Require: rit is a reference to the RIT.
Require: "/triageApp/dangerLevel" is the path used by
    the triage app to store the current danger level.
Require: HIGH is the highest priority level.
Require: LOW is the lowest priority level.
Ensure: Returns the priority level.
1: procedure BUNDLEPRIORITY
2:     level ← rit.get("/triageApp/dangerLevel")
3:     if level < 3 then
4:         return HIGH          ▷ Victim must be rescued.
5:     else
6:         return LOW           ▷ Victim must be counted.
7:     end if
8: end procedure
```

9. Simulation results

Active-DTN is our approach for a DTN that supports concurrent applications with different routing needs. We have conducted several experiments using simulations in an emergency scenario that corresponds to the practical application described and discussed in Section 8. In this kind of scenarios, we prove that Active-DTN improves network latency and delivery ratio by the use of several forwarding algorithms at the same time. We confirm that RIT spreading improves the performance, and contextual prioritisation works better than static prioritisation. We checked the convenience of using lifetime control as MMEB, and it largely improves the network average latency. Finally, we wanted to confirm that bundle size increase due to MMEB has no major negative consequences. For all these reasons, we claim that Active-DTN is highly beneficial for environments with routing diversity like emergency scenarios. In this section, we describe the simulation experiment in more detail.

9.1. Environment

The experiment has been conducted over the Opportunistic Network Environment (TheONE) simulator [26]. This simulator was created to run simulation experiments on opportunistic networks like DTN and to provide reports and graphical representations of the results. TheONE represents the communication among nodes as well as the nodes movement. It simulates intermittent disconnections by also taking into account the node transmission range. One of the advantages of this simulation environment is that it is quite simple to configure and intuitive to extend and modify. In fact, we have modified the simulator to represent an Active-DTN network and study the peculiarities of our networking approach.

TheONE allows the user to define different types of messages with configurable size. In our experiment, a TheONE message represents a bundle. The official version of the simulator assumes that, as it is the case in most opportunistic networks, there is a single-forwarding algorithm at a time to forward all messages. Therefore, there is a single-forwarding algorithm procedure that is called in TheONE every time an event needs to be forwarded.

Active-DTN networks support the concurrent use of many different forwarding algorithms employing the forwarding algorithm provided by the bundle to forward it. In a real scenario, the software code of the forwarding algorithm is carried by the bundle using a MMEB. To simulate this behaviour, we have extended TheONE messages to represent bundles with configurable payload and MEB size. Moreover, we implemented a forwarding algorithm procedure to select the forwarding algorithm to execute. By this implementation, each bundle can be forwarded employing its corresponding forwarding algorithm.

TheONE simulator provides reports about the overall communication of the network. To analyse the performance of Active-DTN in more detail, we have extended the reporting functionality of the simulator to discriminate results also per message type. By this way, we can report the communication results obtained by each forwarding algorithm. Then, to facilitate the visualisation of results and simplify its analysis, we have implemented the functionality to create reports following the Google's Motion Chart[14]. The chart represents the evolution of the network latency and delivery ratio (x- and y-axis) as well as the number of forwarded bundles (size and colour of the bullet representing the application in the chart) per forwarding algorithm.

Finally, the official TheONE release does not take into account energy consumption of nodes. We extended it to measure the existing energy measured by the remaining time of power autonomy. We also provide the needed functionality to configure the energy consumption speed taking into account that nodes consume more when they are transferring data than when they are just waiting for new data to be transferred.

9.2. Settings

As described in Section 8, a rescue team involves many different people, e.g.: doctors, fireman and coordination staff. Each of them carries a mobile device. In addition, a portable device is placed at the ECC and several sensor devices are spread over the area. They all together form the communication network. For this experiment, victims are also given a mobile device when the medical team performs the Triage procedure to them. Transmit speed for wireless devices is fixed to 15Mbps[15]. The buffer size for small devices such as the wireless sensors is a random value between 1MB to 20MB. The buffer size is fixed to 500MB for mobile devices and the ECC. The transmit range for the small devices is 3 meters, and it is 500 meters for the mobile nodes. The emergency area contains 100 nodes: 65% of them represent victims, 10% represent doctors, 10% are sensors, 7% are firemen and 7% are coordination personal and 1% the ECC node.

The emergency area occupies $20Km^2$. Mobile nodes present in the affected area produce different mobility patterns. On one hand, the paramedic personnel follow a variation of the random-walk movement model in which several points of interest have been defined. These nodes come back to the Emergency Coordination Centre (ECC) every less than 3 hours and return to the emergency scenario after a maximum of 1 hour. Firemen behave slightly differently from the paramedic personnel: they enter the emergency scenario directly to where a fire has occurred and come back after a random time, less than 6 hours. Coordination personnel follow a task-driven movement model in which nodes perform a node searching movement model combined with a list of points of interest to be visited. Mobile device nodes can move up to 3m/s. Other nodes such as the ECC and the wireless sensors are static.[16]

[14]Google's Visualisation Motion Chart *http://goo.gl/otSnq1*

[15]15Mbps is a fairly realistic speed because although Wi-Fi dongles like Wi-Pi for single-board computers like Raspberry Pi allow transmission speeds up to 150Mbps, the USB controller slows the transmission speed down to values around 15Mbps.

[16]The implementation of the movement models is available at *https://senda.uab.cat/wiki/aDTN*.

Four applications coexist in this scenario. An alert application is used to broadcast alerts among all people in the rescue area. Another application permits notifying an event to a particular person. There is an application for sensor data retrieval, and finally, the Triage application previously described in this paper.

Messages created during the simulations represent alert and communication messages, images and sensed data. These messages have a variable size from 10KB to 50KB. This size is representative of messages used in emergency scenarios as learnt by our research group from projects like the ones presented in [36] and [34] and some others from other external studies like [25] and [11]. By default, the network configuration sets nodes to create all bundles with 10KB for payload and to forward them using PRoPHET [32]. Moreover, bundles are created periodically using a random value between 1 and 10 as the amount of seconds to wait for next bundle to be created. We defined other network configurations based on the default configuration by changing some of its features. For instance, we set other bundle creation time thresholds, change the forwarding algorithm and require bundle extensions. When extensions are present, the size of these extensions is 200B. All simulations represent 300 hours of activity. Finally, battery consumption is simulated by decreasing a maximum value of 10h per computation action, storage access or radio usage. The batteries are recharged and set to fully charged every time the node goes back to the ECC.

9.3. Tests & results

In this work, we conduct the experimentation performing tests to validate several hypotheses about our networking approach. In the following, we enumerate those hypotheses, describe the network configurations being used and illustrate the results of their simulation execution. With the aim of avoiding noise, we repeated every execution 20 times and illustrated and analysed their average results.

Test 1 *Forwarding diversity improves network performance.*

The main advantage of Active-DTNs is that many forwarding algorithms can run concurrently with no need for network re-configuration. In this study, we claim that forwarding diversity improves network performance. Therefore, we set a network configuration with several forwarding algorithms and compare its performance to other network configurations with a single-forwarding algorithm. Concretely, we employ PRoPHET and Larod as previously done in [35] and [25]. Moreover, we also employ the TTR algorithm described in previous sections. Then we employ them all together in a single configuration where we define the forwarding algorithm to use by each type of bundles being sent. We denote this configuration by *multi-forwarding.*[17]

The probabilistic forwarding algorithm PRoPHET takes into account the frequency of opportunistic contacts among nodes to forward the bundle to the neighbour node with higher contact frequency. The forwarding algorithm Larod [28] forwards bundles to the node geographically closer to the destination node. Finally, TTR (see Algorithm 2) takes advantage of the existing time-to-return information and forwards bundles to the neighbour node that will return sooner to the ECC.

The multi-forwarding configuration is set to forward all bundles generated by the Triage application using the TTR forwarding algorithm because this algorithm is specifically designed to accelerate the latency of bundles with ECC as destination. Bundles from the notification application use PRoPHET, the alerts application uses epidemic forwarding [24] unless the alert is addressed to a specific physical area, for example, an area on fire. In that case, Larod is the forwarding algorithm to use.

To check whether the bundle size affects the network performance, we set network configurations to create all their bundles with a fix payload size. The set of payload size values used in Kilobytes is {10, 15, 20, 25, 30, 35, 40, 45, 50}. Note that, even though the payload size is fixed, the bundle size is larger in multi-forwarding configurations because bundles in those configurations use a MMEB.

The most popular metrics to use to evaluate network performance are latency and delivery ratio. The latency is the time for bundles to communicate from the source to the destination. In Figure 9, we can see a summary of the simulation results indicating the average latency per network configuration over the payload size of its bundles. Therefore, we identify a network configuration by its forwarding algorithm and the bundles' payload size.

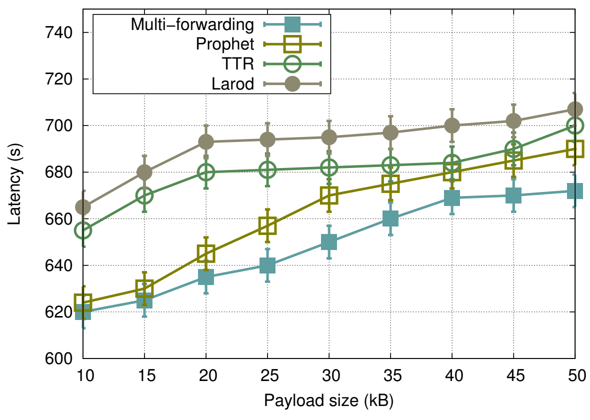


Figure 9: Latency over payload size for executions with different forwarding: forwarding diversity, Larod, PRoPHET, and RTT. The Forwarding diversity option performs better independently of the size of the payload.

[17]The source code for routing algorithms can be found at https://senda.uab.cat/wiki/aDTN in Section "TheOne resources".

As it can be seen in Figure 9, latency time increases as the size of the message increases. In the different routing protocols, the gradient of the graphic decreases as the payload size increases due to how the buffers get full. This evolution of the performance in terms of the size of the message is a common issue in this kind of networks, and it can be seen in studies like [44]. As illustrated, PRoPHET performs well. However, the multi-forwarding configuration has the lowest latency time regardless of the payload size.

The delivery ratio is the percentage of bundles arrived at their destination among those that were sent. During the simulation execution, bundles are periodically created by nodes using random values between 1 and 100 as the time to wait in seconds between bundles being created. Therefore, the number of bundles created is quite constant. If we set the configurations to create bundles forever, during the whole execution, the last bundles sent will not have the chance to arrive at the destination because the execution will stop immediately after their creation. After running some simulations, we decided to set all nodes to stop creating bundles some time before stopping the execution. We set that amount of time to be double the observed latency. We illustrate the impact of this change in Figure 11 with the number of bundles in the network at the end of the simulations.

The results using this new configuration where bundle creation is stopped before finishing the simulation execution are represented in Figure 10. In our scenario, the connection window is sometimes very small. As a consequence, when two nodes meet, the bigger the messages are, the easier the messages will not be forwarded: messages may be started to be forwarded but then aborted. This is the reason why, in Figure 10, when the payload size increases, the delivery ratio decreases.

Observing Figure 10, we can see that although TTR shows a good delivery ratio, the highest delivery ratio corresponds to the multi-forwarding configuration.

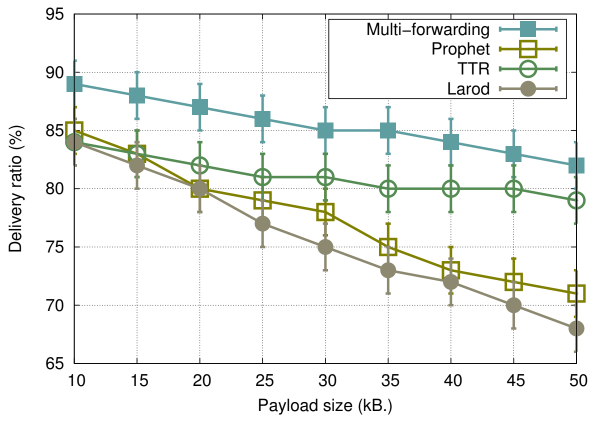


Figure 10: Delivery ratio over payload size for executions with different forwarding: forwarding diversity, Larod, PRoPHET, and RTT. The Forwarding diversity option performs better independently of the size of the payload.

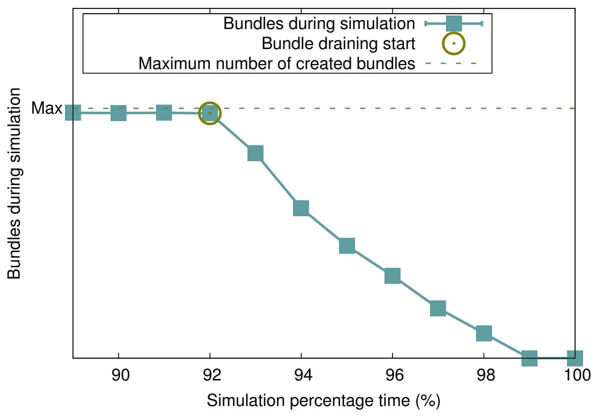


Figure 11: Number of messages present in the scenario as a function of the percentage of the simulation time. Message creation is stopped after double the observed latency on previous simulations.

According to the latency and the delivery ratio measures, the multi-forwarding configuration is considerably better than the single configurations that lead to confirm that forwarding diversity improves the network performance. However, we wanted to study the overhead generated by our proposal. To that end, we measured the most important overheads introduced by our proposal. These overheads include the bundle size overhead due to the bundle extension, the buffer occupancy overhead on average during the simulation, the CPU overhead, the energy consumption and the node's message drop overhead, all of them as a function of the size of the payload message. Then, we compare them to the default configuration results and illustrate the result of the perceptual overhead in Figure 12.

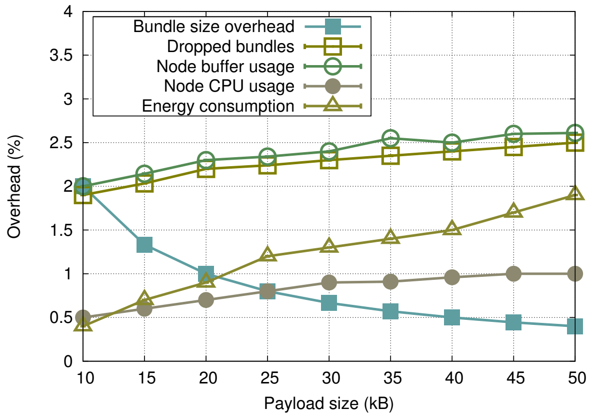


Figure 12: Bundle size overhead created by the extension blocks, buffer occupancy on average during the simulation overhead and message drop overhead, all of them as a function of the size of the payload message.

As illustrated in this figure, the overhead can be considered low as it is always under 3%. The increase of overhead over the payload size is also low. In fact, the message size overhead reduces over the payload size because the MMEB is fixed while the measure is relative. As it can be seen, the energy consumption increases stronger than CPU when payload size is increased. This is because a great part of the energy consumption of a DTN node is spent on the radio wireless radio. The bigger the message is, the more energy is used to transmit it.

Analysing the latency, the delivery ratio and the added overhead of the multi-forwarding configuration over the rest of the single-forwarding configurations, we can confirm that the multi-forwarding configuration performs better enough to pay off the overhead added by our proposal. Therefore, for all these reasons, we conclude confirming that forwarding diversity considerably improves the network performance.

Test 2 *RIT spreading increases network delivery ratio.*

Another contribution of Active-DTNs is the use of the RIT, see Section 7, to store routing information in nodes. aDTN nodes spread part of this information using beacon messages. The aim motivating the use of RITs is to assist the routing algorithm with information generated from other nodes assuming that this information improves the communication increasing the delivery ratio. By this test, we want to prove that this claim is true. Therefore, we set two network configurations using TTR as forwarding algorithm. This forwarding algorithm has been selected because the TTR information of nodes is stored in the RIT. The difference between the two configurations is that only one of them spreads RIT information. We execute 20 simulations of these two configurations with each of the following maximum bundle creation times {1, 35, 70, 105, 140, 175, 205, 240} in seconds.

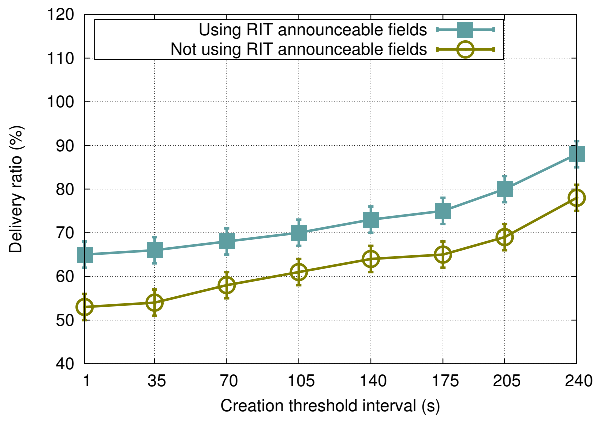


Figure 13: Bundle arrival efficiency as a function of the message creation interval. Spreading RIT fields tagged as announceable performs better than another scenario that does not.

As seen in Figure 13, the delivery ratio of the spreading configuration is substantially higher for each maximum bundle creation time used. Therefore, we conclude that the convenient use of the RIT with spreading values is beneficial for the network as it increases its delivery ratio.

***Test* 3** *Contextual prioritisation is effective.*

The Bundle Protocol header reserves seven bits to specify the bundle class of service that is a static prioritisation mechanism to establish a processing order among bundles in custody. In Active-DTN, the use of MMEB with bundle prioritisation code allows the definition of context-aware prioritisation algorithms. These algorithms can be sensitive to changes in the neighbour node list and the RIT. By this test, we check whether context-aware prioritisation is more effective than static prioritisation measuring the effectiveness as the amount of latency reduction of high-priority bundles.

When no priority is set, all bundles are delivered with a similar latency. Nevertheless, when priority is set, high-priority bundles are expected to have a lower latency. Several executions have been performed using three different configurations. The static prioritisation configurations use the class of service field in the bundle header to set the bundle relevance before sending the bundle. This priority is static, does not change during communication. The contextual prioritisation configuration uses the bundle prioritisation MMEB to set its relevance as a software code computes the priority of the bundle in terms of information stored in the RIT. This priority is dynamic and can change during communication. The non-prioritised configuration does not prioritise relevant bundles.

The configurations are set to change the bundle priority at intervals of every one fifth of the simulation time. The priority is set to either maximum or minimum. All bundles created during the same period of time are set to have the same priority. When the priority changes, only those bundles with contextual priority will be aware and affected by the change. After the executions, we classify bundles arriving at the destination by their priority and compute the latency mean of high-priority bundles. This value is compared to the overall bundle latency to compute the latency difference of high-priority bundles over the overall mean. Figure 14 illustrates the results.

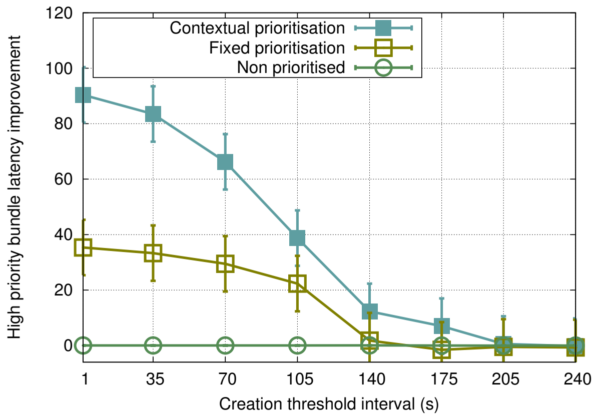


Figure 14: Prioritised bundles over total mean as a function of the number of messages. Contextual prioritisation performs better than the schemes with no prioritisation and source-prioritisation.

As can be seen in Figure 14, in priority changing scenarios, the relative latency of high-priority bundles using contextual prioritisation criteria is much smaller than the latency of high-priority bundles using static prioritisation criteria. This is due to the capability of bundles in contextual prioritisation criteria to be aware of priority changes. Therefore, according to our experiments, contextual prioritisation with MMEB bundle prioritisation is more effective than static prioritisation using class of service field or non-prioritisation.

***Test* 4** *Dynamic lifetime control reduces network congestion.*

Lifetime control aim is network congestion reduction. The more efficiently messages are lifetime controlled, the fewer useless messages will be present in the network. As a consequence, node's buffers are more efficiently used, and congestion is alleviated.

In the Bundle Protocol, bundles can be lifetime controlled using the bundle time-to-live field. In this case, the time-to-live must be a priori set. This means that it is necessary to estimate the latency of messages in order to set a time-to-live at least as long as its future delivery time. To guarantee a certain probability of the bundle being delivered, the time-to-live is often a much longer than necessary. Instead, in Active-DTN we can use a dynamic lifetime control where time to live is determined by a software code that can take as input information stored in the RIT. Notice that this alternative does not require latency estimations.

To prove that dynamic lifetime control is better reducing network congestion, we run several simulations with two network configurations with different lifetime control mechanisms. One of them is dynamic, and the other is static. The dynamic one allows the application to control the bundle's lifetime from an application perspective. In these simulations, victims' information messages are lifetime controlled using technical medical algorithms. Instead, sensors and fire information messages use geographical and meteorological algorithms. Finally, coordination information messages are discarded using data from task completion information, when available.

As can be seen in Figure 15, latency in dynamic lifetime control is always lower than latency in static lifetime control. As latency directly depends on congestion, we can claim that dynamic lifetime control reduces network congestion.

***Test* 5** *Contrary to deployment-based solutions, carrying routing algorithms ensures optimal routing usage.*

As introduced in Section 2, there are proposals like Haggle [45] that allow application messages to choose among a limited number of routing protocols which have been previously deployed. However, as already explained in Section 2, in DTN networks, this deployment is not a trivial issue. We have performed several simulations to under-

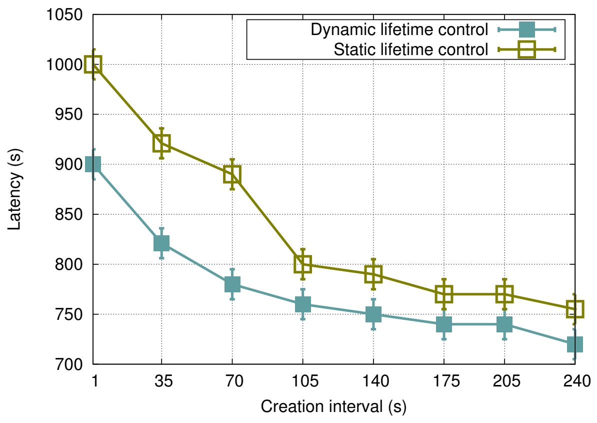


Figure 15: Latency time as a function of the number of the message creation interval for two different scenarios: dynamic lifetime control using MMEB and static one.

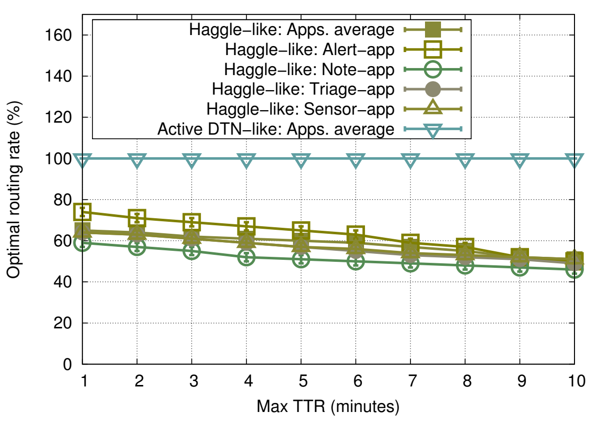


Figure 16: Percentage of the messages routed with their optimal routing algorithm as a function of the time to return to the ECC. For the Haggle-like approach, the different applications simulated are shown on average and separately. For the Active-DTN approach, the result is shown on average for every application.

stand how difficult is to perform these deployments and to study which are their limitations. In Figure 16, we analyse the percentage of messages routed using their optimal routing protocol as a function of the mobile nodes maximum time to return to the ECC. In this scenario, the four applications previously simulated in this section coexist. The simulation starts with a network formed by the medical personnel and the victims. In *t=0*, a single application runs in the network. The additional three applications and the rest of the DTN nodes are incorporated gradually to the simulation after three hours. The fact that every node is incorporated in different moments prevents the possibility of an initial routing software deployment. Mobile nodes are updated with new routing algorithms every time the come back to the ECC. Static nodes remain with a single routing algorithm during the whole simulation period.

As it can be seen, using our proposal, 100% of the messages are routed using their optimal routing algorithm. Instead, using a Haggle-like solution the different applications are routed using their optimal routing algorithm in 51% to 65% of the cases, on average, depending on the maximum TTR. We claim, consequently, that using our proposal helps the bundles to be routed using their optimal routing algorithm.

10. Conclusions

In this article, we have presented a network solution based on mobile code to improve DTN networks. Our proposal enables the nodes to execute routing software codes carried by the messages themselves. This proposal follows the Bundle Protocol specification by extending the bundles with forwarding, lifetime control and prioritisation algorithms. This allows the coexistence of different applications needing to use different routing algorithms. As a consequence, and unlike classical DTN approaches, when this type of applications intends to use the network, no network reconfiguration or deployment is required.

We have described the feasibility, utility and usefulness of our proposal by applying it to a recovery scenario, as described in Section 8. We have also included in Section 9 experiments for several DTN applications in a realistic scenario. These experiments conclude that Active-DTN reduces network latency and increases delivery ratio by the use of several routing algorithms at the same time. It is confirmed that application-based information spreading improves the performance of the network. Furthermore, we have seen that contextual application-based prioritisation works better than static prioritisation. Additionally, we conclude that using application-based lifetime control improves the network average latency. Finally, we confirm that bundle size increase due to MMEB has no major negative consequences.

It is a fact that the complexity of the network as a whole has increased by moving the routing software code from the host to the message. Furthermore, there is an overhead of information transmitted since the bundles are carrying their routing algorithms in addition to the application data itself. As seen in Section 9, these overheads are low and do not affect network performance. We have learnt that the advantages of employing such a paradigm in scenarios, such as disaster recovery scenarios, absolutely outweigh the impediments related to a network solution with the added complexity.

The network solution presented in this paper opens new possibilities in opportunistic networks. The results presented give a flourishing indication that new scenarios besides the ones presented in this paper may take advantage of the combination of DTN and mobile code.

Acknowledgements

This work was partly supported by the Catalan AGAUR 2014SGR-691 project and the Spanish Ministry of Economy and Competitiveness TIN2014-55243-P project. The authors thank Daniel Ezquerra, Gerard Garcia and Rafael Páez for their excellent contribution to the aDTN implementation, Luca Bernabini for his advice on emergency scenarios and Peter Lovell and Stephen Farrell for their advice on the Bundle Security Protocol.

References

[1] Hisham M Almasaeid. *Data delivery in fragmented wireless sensor networks using mobile agents*. ProQuest, 2007.

[2] Joan Ametller, Sergi Robles, and Jose A Ortega-Ruiz. Self-protected mobile agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 362–367. IEEE Computer Society, 2004.

[3] Asmidar Abu Bakar, R Ismail, and J Jais. A review on extended role based access control (e-rbac) model in pervasive computing environment. In *Networked Digital Technologies, 2009. NDT'09. First International Conference on*, pages 533–535. IEEE, 2009.

[4] Meenakshi Bansal, Rachna Rajput, and Gaurav Gupta. Mobile ad hoc networking (manet): Routing protocol performance issues and evaluation considerations. Technical report, RFC 2501 (Informational), Jan, 1999.

[5] PT Barry. Abstract syntax notation-one (asn. 1). In *Formal Methods and Notations Applicable to Telecommunications, IEE Tutorial Colloquium on*. IET, 1992.

[6] Fabrice Bellard. Tcc: Tiny c compiler. *URL: http://fabrice. bellard. free. fr/tcc*, 2003.

[7] Marc Blanchet. Delay-Tolerant Networking Bundle Protocol IANA Registries, 2011.

[8] Chiara Boldrini, Marco Conti, Jacopo Jacopini, and Andrea Passarella. Hibop: a history based routing protocol for opportunistic networks. In *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, pages 1–12. IEEE, 2007.

[9] Carlos Borrego, Sergio Castillo, and Sergi Robles. Striving for sensing: Taming your mobile code to share a robot sensor network. *Information Sciences*, 2014.

[10] Carlos Borrego and Sergi Robles. A store-carry-process-and-forward paradigm for intelligent sensor grids. *Information Sciences*, 222(0):113 – 125, 2013.

[11] Raffaele Bruno, Marco Conti, and Andrea Passarella. Opportunistic networking overlays for ict services in crisis management. In *Proceedings of International Conference on Information Systems for Crisis Response and Management ISCRAM*, 2008.

[12] Scott Burleigh. Bundle protocol extended class of service (ecos), 2010.

[13] Scott Burleigh, Esther Jennings, and Joshua Schoolcraft. *Autonomous congestion control in delay-tolerant networks*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2006.

[14] Jeffery Case, Mark Fedor, Martin Schoffstall, and C Davin. *A simple network management protocol (SNMP)*. Network Information Center, SRI International, 1989.

[15] Vinton Cerf, Scott Burleigh, Adrian Hooke, Leigh Torgerson, Robert Durst, Keith Scott, Kevin Fall, and Howard Weiss. RFC 4838, delay-tolerant networking architecture. RFC 4838 (Informational), 2007.

[16] Michael Cooney. Nasa exploring groundbreaking space network to sustain large data dumps and trips to the moon, mars. *Available online: http://www.networkworld.com/community/blog/nasa-exploring-groundbreaking-space-network-sustain-large-data-dumps-and-trips-moon-mars*, 2013.

[17] Floriano De Rango, Mauro Tropea, Giovanni Battista Laratta, and Salvatore Marano. Hop-by-hop local flow control over interplanetary networks based on DTN architecture. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 1920–1924. IEEE, 2008.

[18] Stylianos Dimitriou and Vassilis Tsaoussidis. Effective buffer and storage management in DTN nodes. In *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*, pages 1–3. IEEE, 2009.

[19] Stephen Farrell. Security in the wild. *Internet Computing, IEEE*, 15(3):86–91, 2011.

[20] Stephen Farrell, Howard Weiss, Susan Symington, and Peter Lovell. Bundle security protocol specification. 2011.

[21] Zhenxin Feng. Data dissemination in delay tolerant networks. doctor of philosophy thesis, 2012.

[22] Delay Tolerant Networking Research Group. Delay tolerant networking research group website, December 2013. http://www.dtnrg.org.

[23] Dan Henriksson, Tarek F Abdelzaher, and Raghu K Ganti. A caching-based approach to routing in delay-tolerant networks. In *Computer Communications and Networks, 2007. ICCCN 2007. Proceedings of 16th International Conference on*, pages 69–74. IEEE, 2007.

[24] Che-Jung Hsu, Huey-Ing Liu, and Winston KG Seah. Opportunistic routing–a review and the challenges ahead. *Computer Networks*, 55(15):3592–3603, 2011.

[25] Peng Jiang, John Bigham, and Eliane Bodanese. Adaptive service provisioning for emergency communications with DTN. In *Wireless Communications and Networking Conference (WCNC), 2011 IEEE*, pages 2125–2130. IEEE, 2011.

[26] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Proceedings of the 2nd international conference on simulation tools and techniques*, page 55. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[27] Amir Krifa, Chadi Barakat, and Thrasyvoulos Spyropoulos. Message drop and scheduling in DTNs: Theory and practice. *Mobile Computing, IEEE Transactions on*, 11(9):1470–1483, 2012.

[28] Erik Kuiper. Node density, connectivity and the percolation threshold. doctor of philosophy thesis, 2010.

[29] Sookyoung Lee and Mohamed Younis. Optimized relay placement to federate segments in wireless sensor networks. *Selected Areas in Communications, IEEE Journal on*, 28(5):742–752, 2010.

[30] Fulu Li, Nabil Seddigh, Biswajit Nandy, and Diego Matute. An empirical study of today's internet traffic for differentiated services ip qos. In *Computers and Communications, 2000. Proceedings. ISCC 2000. Fifth IEEE Symposium on*, pages 207–213. IEEE, 2000.

[31] Qun Li and Daniela Rus. Communication in disconnected ad hoc networks using message relay. *Journal of Parallel and Distributed Computing*, 63(1):75–86, 2003.

[32] Anders Lindgren, Avri Doria, and Olov Schelén. Probabilistic routing in intermittently connected networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 7(3):19–20, 2003.

[33] Anders Lindgren and Kaustubh S Phanse. Evaluation of queueing policies and forwarding strategies for routing in intermittently connected networks. In *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on*, pages 1–10. Ieee, 2006.

[34] Ramon Martí, Sergi Robles, Abraham Martín-Campillo, and J Cucurull. Providing early resource allocation during emergencies: The mobile triage tag. *Journal of Network and Computer Applications*, 32(6):1167–1182, 2009.

[35] Abraham Martín-Campillo, Jon Crowcroft, Eiko Yoneki, and Ramon Martí. Evaluating opportunistic networks in disaster scenarios. *Journal of Network and Computer Applications*, 36(2):870–880, 2013.

[36] Abraham Martín-Campillo, Carles Martínez-García, Jordi Cucurull, Ramon Martí, Sergi Robles, and Joan Borrell. Mobile agents in healthcare, a distributed intelligence approach. In *Computational Intelligence in Healthcare 4*, pages 49–80.

Springer, 2010.

[37] Paolo Meroni, Elena Pagani, Gian Paolo Rossi, and Lorenzo Valerio. An opportunistic platform for android-based mobile devices. In *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*, pages 191–193. ACM, 2010.

[38] Helsinki University of Technology. Networking laboratory website, 2013. http://www.netlab.tkk.fi/engl.shtml.

[39] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit.* Syngress, 2006.

[40] Wolf-Bastian Pöttner, Johannes Morgenroth, Sebastian Schildt, and Lars Wolf. Performance comparison of DTN bundle protocol implementations. In *Proceedings of the 6th ACM workshop on Challenged networks*, pages 61–64. ACM, 2011.

[41] Vassilis Prevelakis and Diomidis Spinellis. Sandboxing applications. In *USENIX Annual Technical Conference, FREENIX Track*, pages 119–126. Citeseer, 2001.

[42] Antonio Puliafito and Orazio Tomarchio. Using mobile agents to implement flexible network management strategies. *Computer Communications*, 23(8):708–719, 2000.

[43] Adrián Sánchez-Carmona, Carlos Borrego, Sergi Robles, and Jordi Andújar. Control de acceso para mensajes pro-activos en redes dtn. In *Proceedings of XII Reunión Española sobre Criptología y Seguridad de la Información*, 2012.

[44] Gabriel Sandulescu and Simin Nadjm-Tehrani. Opportunistic DTN routing with window-aware adaptive replication. In *Proceedings of the 4th Asian Conference on Internet Engineering*, pages 103–112. ACM, 2008.

[45] James Scott, Jon Crowcroft, Pan Hui, and Christophe Diot. Haggle: A networking architecture designed around mobile users. In *WONS 2006: Third Annual Conference on Wireless On-demand Network Systems and Services*, pages 78–86, 2006.

[46] Keith L Scott and Scott Burleigh. Bundle protocol specification. RFC 5050 (Experimental), November 2007.

[47] Matthew Seligman, Kevin Fall, and Padma Mundur. Storage routing for DTN congestion control. *Wireless communications and mobile computing*, 7(10):1183–1196, 2007.

[48] Sakir Sezer, Sandra Scott-Hayward, Pushpinder-Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn? implementation challenges for software-defined networks. *Communications Magazine, IEEE*, 51(7):36–43, 2013.

[49] Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Cauligi S Raghavendra. Spray and wait: an efficient routing scheme for intermittently connected mobile networks. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 252–259. ACM, 2005.

[50] Susan Symington, Stephen Farrell, Howard Weiss, and Peter Lovell. Bundle security protocol specification, 2007.

[51] David L Tennenhouse, Jonathan M Smith, W David Sincoskie, David J Wetherall, and Gary J Minden. A survey of active network research. *Communications Magazine, IEEE*, 35(1):80–86, 1997.

[52] Noriki Uchida, Norihiro Kawamura, Nicholas Williams, Kazuo Takahata, and Yoshitaka Shibata. Proposal of delay tolerant network with cognitive wireless network for disaster information network system. In *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 249–254. IEEE, 2013.

[53] Bennet Yee, David Sehr, Gregory Dardyk, J Bradley Chen, Robert Muth, Tavis Ormandy, Shiki Okasaka, Neha Narula, and Nicholas Fullagar. Native client: A sandbox for portable, untrusted x86 native code. In *Security and Privacy, 2009 30th IEEE Symposium on*, pages 79–93. IEEE, 2009.