DTN Research Group
Internet-Draft
Expires: October 27, 2013

W. Moreira
P. Mendes
SITILabs, Universidade Lusofona
R. Ferreira
D. Cirqueira
E. Cerqueira
ITEC, Universidade Federal do Para
April 25, 2013

# Opportunistic Routing based on Users Daily Life Routine
# draft-moreira-dlife-02

Abstract

This document is written in the context of the Delay Tolerant Networking Research Group and will be presented for reviewing by that group. This document defines dLife, an opportunistic routing protocol that takes advantage of time-evolving social structures. dLife belongs to the family of social-aware opportunistic routing protocols for intermittently connected networks. dLife operates based on a representation of the dynamics of social structures as a weighted contact graph, where the weights (i.e., social strengths) express how long a pair of nodes is in contact over different period of times. It considers two complementary utility functions: Time-Evolving Contact Duration (TECD) that captures the evolution of social interaction among pairs of users in the same daily period of time, over consecutive days; and TECD Importance (TECDi) that captures the evolution of user's importance, based on its node degree and the social strength towards its neighbors, in different periods of time. It is intended for use in wireless networks where there is no guarantee that a fully connected path between any source - destination pair exists at any time, a scenario where traditional routing protocols are unable to deliver bundles. Such networks can be sparse mesh, in which case intermittent connectivity is due to lack of physical connections, or dense mesh, in which case intermittent connectivity may be due to high interference or shadowing. In any case, intermittent connectivity can also be due to the availability of devices (e.g., unavailable due to power saving rules). The document presents an architectural overview followed by the protocol specification.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

## Copyright and License Notice



## Table of Contents

## 1. Introduction

The pervasive deployment of wireless personal devices is creating the opportunity for the development of novel applications. The exploitation of such applications with a good performance-cost tradeoff is possible by allowing devices to use free spectrum to exchange data whenever they are within wireless range, specially in scenarios where it is difficult to find an end-to-end path between any pair of nodes at any moment. In such scenarios every contact is an opportunity to forward data. Hence, there is the need to develop networking solutions able to buffer messages at intermediate nodes for a longer time than normally occurs in the queues of conventional routers (cf. Delay-Tolerant Networking [RFC4838]), and routing algorithms able to bring such messages close to a destination, with high probability, low delay and costs.

Most of the proposed routing solutions focus on inter-contact times alone [Chaintreau06], while there is still significant investigation to understand the nature of such statistics (e.g., power-law, behavior dependent on node context). Moreover, the major drawback of such approaches is the instability of the created proximity graphs [Hui11], which changes with users' mobility.

A trend is investigating the impact that more stable social structures (inferred from the social nature of human mobility) have on opportunistic routing [Hui11], [Daly07]. Such social structures are created based on social similarity metrics that allow the identification of the centrality that nodes have in a cluster/community. This allows forwarders to use the identified hub nodes to increase the probability of delivering messages inside (local centrality) or outside (global centrality) a community, based on the assumption that the probability of nodes to meet each other is proportional to the strength of their social connection.

A major limitation of approaches that identify social structures, such as communities, is the lack of consideration about the dynamics of networks, which refers to the evolving structure of the network itself, the making and braking of network ties: over a day a user meets different people at every moment. Thus, the user's personal network changes, and so does the global structure of the social network to which he/she belongs.

When considering dynamic social similarity, it is imperative to accurately represent the actual daily interaction among users: it has been shown [Hossmann10] that social interactions extracted from proximity graphs must be mapped into a cleaner social connectivity representation (i.e., comprising only stable social contacts) to improve forwarding. This motivates us to specify a routing protocol

aware of the network dynamics, represented by users' daily life routine. We focus on the representation of daily routines, since routines can be used to identify future interaction among users sharing similar movement patterns, interests, and communities [Eagle09].

Existing proposals [Costa08], [Hui11], [Daly07] succeed in identifying similarities (e.g., interests) among users, but their performance is affected as dynamism derived from users' daily routines is not considered. To address this challenge, we propose dLife that uses time-evolving social structures to reflect the different behavior that users have in different daily periods of time: dLife represents the dynamics of social structures as a weighted contact graph, where the weights (i.e., social strengths) express how long a pair of nodes is in contact over different period of times.

dLife considers two complementary utility functions: Time-Evolving Contact Duration (TECD) that captures the evolution of social interaction among pairs of users in the same daily period of time, over consecutive days; and TECD Importance (TECDi) that captures the evolution of user's importance, based on its node degree and the social strength towards its neighbors, in different periods of time.

## 1.1. Applicability of the Protocol

This section describes the applicability of the dLife protocol in terms of the networking protocol stack and in terms of the usage scenarios that are representative of the daily life experience of most people. The latter aims to check which are the communication challenges that can be mitigated by deploying a delay-tolerant routing protocol. The focus goes to scenarios involving mission-critical environments that represent sporadic situations that require a spontaneous and efficient exchange of information, as well as communications in urban environments, which can also benefit from the existence of a DTN.

This document does not focus on scenarios such as space networks and rural area networks, since space networks rely on the usage of single links with extreme long delay, while most of the potential rural area scenarios will require a store-and-carry system (ferry type) and not so much a store-carry-forward system.

### 1.1.1. Protocol Stack

The dLife protocol is expected to interact with the Bundle Protocol agent for retrieving information about available bundles and for requesting bundles to be sent to another node (cf. Section 2.4.1). It

is expected that the associated bundle agents are then able to establish a link, over the TCP convergence layer [I-D.irtf-dtnrg-tcp-clayer]) or the UDP convergence layer [I-D.irtf-dtnrg-udp-clayer]) to perform this bundle transfer.

In what concerns information needed for the operation of dLife, dLife does not impose any requirements for data reliability transfer to avoid restricting its applicability. Hence, data exchange may take place over transport protocols that do not provide neither message segmentation or reliability, nor in order delivery. Hence, dLife provides itself the capability to segment protocol messages into submessages. Submessages are provided with sequence numbers, and this, together with the capability for positive acknowledgements allows dLife to operate over an unreliable protocol such as UDP or potentially directly over IP. As said for the bundle agent, the communication medium used to send dLife messages can include different technologies such as Bluetooth and Wi-Fi.

Moreover, dLife expects to be able to use bidirectional links for information exchange; this allows information exchange to take place in both directions over the same link, avoiding the need to establish a second link for information exchange in the reverse direction.

### 1.1.2. Applicability scenarios

The identified scenarios aim to illustrate the applicability of dLife in real scenarios. In technical terms, dLife aims to target networks where we may not find any end-to-end path between any pair of nodes at some moment in time. The lack of end-to-end path may be due to node mobility and availability (e.g., switching off radios), aspects that create connectivity patterns that are correlated with the daily habits of citizens. Human behavior patterns (often containing daily or weekly periodic activities) provide one example where dLife is expected to be applicable, independently of the type of personal device: it can be of explicit usage (e.g., smartphones) or of implicit usage (e.g., embedded devices).

Scientific results [Moreira12a] [Moreira12b] show that dLife is able to benefit from the predictability of human behavior in daily periods of time even in the presence of few contacts. However, the behavior predictability can be estimated more accurately with a higher number of events.

#### 1.1.2.1. Urban Areas Networks

This seems to be the most challenging scenario to analyze the applicability of DTN employing dLife as the store-carry-forward routing protocol. A study of DTN routing for urban scenarios may

bring a coherent understanding about the advantages and challenges of using a DTN system in the daily life of millions of people. A study of the applicability of DTN routing in urban scenarios may benefit from a good understanding of the per-person bit density (available capacity per second/hour/day) in a major metropolitan area.

In a urban area there are several examples of networking scenario that can gain from the applicability of dLife, such as: Urban "dark" places due to high mobility (e.g., fast trains), indoors (e.g., subway systems, in-building) and outdoor (e.g., areas with closed APs, areas with significant interference); Off-load of cellular networks, since cellular operators do not like to have data traffic unrelated to services provided in their networks; The cost of cellular wireless data, which decreases the relation quality/price of cellular data communications; Networks of embedded objects, which will require delay-tolerant communications over short-distance wireless interfaces and not over cellular ones.

#### 1.1.2.2. Mission-critical Networks

At any point, natural catastrophes can happen and such type of network can be deployed in order to facilitate rescue and medical operations. Another type of situation that a mission-critical network may be formed is in hostile environment such as war scenarios. Independently of the scenario for its application, this type of networks must be readily available through any sort of Wi-Fi enabled equipment (PDAs, cell phones, laptops, APs) which are expected to cooperate with the aim of helping the dissemination of information. Information must reach the interested parties as quickly as possible to achieve fast results for the actions being taken.

In mission-critical networks there are several examples of networking scenario that can gain from the applicability of dLife, such as: Disaster networks where no (maybe very few) infrastructure is available since it may have been destroyed; Military networks, where communications can be established using devices carried by soldiers as well as other military vehicles and easily deployed equipments.

### 1.2. Relation with the DTN Architecture and Networking Model

The DTN architecture introduces the bundle protocol [RFC5050], which provides a way for applications to "bundle" an entire session, including both data and metadata, into a single message, or bundle, that can be sent as a unit. The bundle protocol provides end-to-end addressing and acknowledgments. Hence, dLife is intended to provide routing services in a network environment that uses bundles as its data transfer mechanism, but could also be used in other intermittent environments.

From a networking model perspective, a DTN is a network of self-organizing wireless nodes connected by multiple time-varying links, and where end-to-end connectivity is intermittent. Even in urban scenarios, it is possible to face intermittent connectivity due to dark areas, such as inside buildings and metropolitan systems, as well as public areas with closed access points or even places overcrowded with wireless access points. Unavailability of wireless connectivity can be also a result of power-constrained nodes that frequently shut down their wireless cards to save energy.

From a conceptual point of view, a DTN consists of a node meeting schedule and workload. A node meeting schedule is a directed multigraph where each direct edge between two nodes represents a meeting opportunity between them, and it is annotated with a starting time of the meeting, the ending time of the meeting, if known, the size of the transfer opportunity (i.e., contact capacity), and the contact type. The workload is a set of messages. Each message can be represented by the source, destination, size, time of creation at the source, and priority. In dLife, a contact is defined by the tuple <starting time, end time, contact duration> and a message by the tuple <source, destination, size>.

A DTN model encompasses the notion of type of contact or connectivity. In current networks, the connectivity of a link or path is generally given as a binary state (i.e., connected or disconnected). In DTNs, a richer set of connectivity options is required to make efficient routing decisions. Most importantly, links (and paths, by extension) may provide a scheduled, predicted or opportunistic communication.

Scheduled contacts imply some a priori knowledge about adjacent nodes regarding future availability of links for message forwarding. Scheduled links are the most typical cases for today's Internet and satellite networks. Predicted contacts correspond to communication opportunities wherein the probability of knowing whether a link will be available at a future point in time is strictly above zero and below one. Such links are the result of observed behavior (e.g., a person may use its home Internet connection with significant probability for any given time period) being characterized using statistical estimation. Predicted links are only becoming of serious concern recently, namely in ad-hoc wireless networks where node mobility may be significant.

In more challenging environments, such as mission-critical networks for instance, the future location of communicating entities may be neither known nor predictable. These types of contacts are known as opportunistic. Such opportunistic contacts are defined as a chance to forward messages towards a specific destination or a group of

destinations. In such unpredictable scenario, it is important to take into account the time that a node must wait until it meets another node again (i.e., inter-contact time), the duration of these contacts (i.e., contact duration), and the quality of the contact in terms of the set of information that can be transferred (i.e., contact volume).

Independently of the type of connectivity, a contact in a DTN is direction-specific. For example, a dial-up connection originating at a customer's home to an Internet Service Provider (ISP) may be scheduled from the point of view of the customer but unscheduled from the point of view of the ISP. In what concerns contacts, dLife assumes direction-specific opportunistic contacts (starting time, end time, contact duration) which occur with some probability in pre-defined daily time periods.

Another concept that must be introduced is that of network behavior. As networks can be formed on-the-fly, their behavior can either be deterministic or stochastic, depending on the type of used links. In this draft, we focus on dynamic scenarios where the behavior of the network is described in stochastic terms, based on users' mobility and social behavior. In a dynamic scenario, users move around carrying their personal devices, which opportunistically come into contact with each other, resulting in topology changes.

### 1.3. Differentiation to other Opportunistic Routing Proposal

Due to intermittent connectivity, routing protocols based on the knowledge of end-to-end paths perform poorly, and numerous opportunistic routing algorithms have been proposed instead. Some opportunistic routing protocols use replicas of the same message to combat the inherent uncertainty of future communication opportunities between nodes. In order to carefully use the available resources and reach short delays, many protocols perform forwarding decisions using locally collected knowledge about node behavior to predict which nodes are likely to deliver a content or bring it closer to the destination.

We previously identified [Moreira11] that most of the opportunistic routing prior-art considered the replication-based forwarding scheme, while only 15% were based on single-copy and flooding-based forwarding schemes. Among the replication based solutions, approximately 69% consider a contact-based approach (e.g., frequency of encounters) and 31% (the latest ones) investigate the trend based on social similarity metrics (e.g., community detection). Contact-based proposals consider every contact among nodes to update the proximity graph and implement metrics such as the number of times nodes meet, contact frequency and the last time a contact occurs.

Besides PROPHET [Lindgren04], the most cited replication-based proposal, other examples based on contact metric are Prediction [Song07], and Encounter -Based Routing [Nelson09].

Most of the existing opportunistic routing solutions are based on some level of replication. Among these proposals, emerge solutions based on different representations of social similarity: i) labeling users according to their social groups (e.g., Label [Hui07]); ii) looking at the importance (i.e., popularity) of nodes (e.g., PeopleRank [Mtibaa10]); iii) combining the notion of community and centrality (e.g., SimBet [Daly07] and Bubble Rap [Hui11]); iv) considering interests that users have in common (e.g., SocialCast [Costa08]). Such prior-art shows that social-based solutions are more stable than those which only consider node mobility. However, they do not consider the dynamism of users' behavior (i.e., social daily routines) and use centrality metrics, which may create bottlenecks in the network. Moreover, such approaches assume that communities remain static after creation, which is not a realistic assumption. On the other hand, prior-art also shows that users have routines that can be used to derive future behavior. It has been proven that mapping real social interactions to a clean (i.e., more stable) connectivity representation is rather useful to improve delivery. With dLife, users' daily routines are considered to quantify the time-evolving strength of social interactions and so to foresee more accurately future social contacts than with proximity graphs inferred directly from inter-contact times.

### 1.4. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. Node Architecture

In this section we describe the architecture of a dLife node, which performs its routing decisions based on two utility functions: TECD to forward messages to nodes that have a stronger social relationship with the destination than the carrier; TECDi to forward messages to nodes that have a higher importance than the carrier.

With TECD each node computes the average of its contact duration with other nodes during the same set of daily time periods over consecutive days. Our assumptions is that contact duration can provide more reliable information than contact history, or frequency when it comes to identifying the strength of social relationships. The reason for considering different daily time periods relates to

the fact that users present different behavior during their daily routines. If the carrier and encountered node have no social information towards the destination, forwarding takes place based on a second utility function, TECDi.

We start this section by describing the different components of the node architecture, followed by an explanation of how to route information based on the dynamics of the network that dLife is able to capture by computing TECD and TECDi. Finally, we describe the implemented forwarding strategy and the needed interfaces.

## 2.1. dLife Components

In order to perform forwarding based on the social daily behavior of users, dLife comprises the following main computational elements:

o Social Information Gatherer (SIG) - responsible for: i) keeping track of the contact duration of each encounter between nodes; and, ii) obtaining the social weights and importance of encountered nodes (i.e., potential next forwarders). As dLife considers different daily samples, corresponding to different periods of time, it is imperative to keep track of nodes' contacts in each sample. Additionally, upon the need to replicate a bundle, SIG will obtain the social weight between the encountered node and nodes it has met as well as the importance of such node.

o Social Information Repository (SIR) - responsible for storing a list with encountered nodes and contact duration to such nodes. At the end of every daily sample, SIR will also store social weights and importance of the encountered nodes computed by SW and IA (see below). Additionally, SIR will temporarily store the social weight and importance of an encountered node when the need for replicating a bundle arises.

o Social Weighter (SW) - responsible for determining the social weight between nodes according to their social interaction throughout their daily routines. At the end of every daily sample, SW will interact with SIR to determine the total contact time nodes spent together and the average duration of contacts in order to compute the social weight between nodes.

o Importance Assigner (IA) - responsible for computing the importance of a node taking into account the importance of encountered nodes and its social weight to such nodes. At the end of every daily sample, IA will interact with SIR in order to compute the importance of a node in the system.

o Decision Maker (DM) - responsible for deciding whether replication

should occur. DM will interact with SIR to obtain relevant information in order to take decisions.

## 2.2. Routing Algorithm

The dLife protocol applies the social opportunistic contact paradigm to decide whether bundle replication is feasible. Its decision is based on social weight (w_(x,y)) towards the bundle's destination or on the importance (I(x)) of the encountered node (i.e., potential next forwarder) in the system.

If the encountered node has better relationship with the bundle destination than the carrier in a given daily sample, it receives a bundle copy, since there is a much greater chance for the encountered node to meet the destination in the future. If relationship to the bundle destination is unknown, replication happens only if the encountered node has higher importance than the carrier.

In order to compute the social weight between nodes and their importance, dLife uses parameters that are determined as nodes interact in the system. A brief explanation of the parameters is given below:

CD_(x,y)
: Refers to the contact duration between nodes, i.e., time nodes spent in the range of one another, which would allow them to exchange information. Within a given daily sample, there could happen different contacts with varied lengths.

TCT_(x,y)
: Refers to the total contact time between nodes within a given daily sample. It is given by the sum of all CD_(x,y) in that specific daily sample.

AD_(x,y)
: Refers to the average duration of contacts for the same daily sample over different days. It is a Cumulative Moving Average (CMA) of the average duration, considering the TCT_(x,y) of the current daily sample and average duration in the same daily sample of the previous day, AD_(x,y)_old.

w_(x,y)
: Refers to the social weight between nodes at a given daily sample. It reflects the level of social interaction among such nodes throughout their daily routines.

I_(x)
: Refers to the importance of a node in the system. The importance

is influenced by how well a node is socially related to other important nodes.

N_(x)
Refers to the neighbor set of a node x, which it encountered in the current daily sample.

dumping factor (d)
Refers the level of randomness considered by the forwarding algorithm.

daily sample (Ti)
Refers to the time period in which the contact duration will be measured to determine social weight and node importance.

As nodes interact, their CD_(x,y) is collected and used to determine TCT_(x,y), AD_(x,y), w_(x,y), and I_(x) at the end of every daily sample. If dLife is configured with a high number of daily samples, the social weight and node importance will be more refined. Thus, it is recommended the usage of twenty-four (24) daily samples representing each hour of the day: the first daily sample refers always to the zero hour of the day when the node is started.

Being able to identify the current daily sample allows a proper computation of social weights and importance. Hence, in the case of node failure (e.g., node crash) or node shutdown (e.g., lack of battery), nodes need to know exactly in which daily sample they stopped interaction, and more importantly how many daily samples have elapsed since then (elapsed_ds). To guarantee that, the equation below is used:

elapsed_ds = cnds * (ed - 1) + (cds - 1) + (cnds - lds) (1)

where:

cnds
is the configured number of daily samples.

ed
refers to the number of elapsed days.

cds
refers to the current daily sample (the one in which the node came back on).

lds
refers to last daily sample (in which the node failed or shut down).

With this, the node knows how many daily samples have elapsed and can proceed with the update of social weights and importance to reflect the lack of interaction that happen in reality.

### 2.2.1. Time-Evolving Contact Duration (TECD)

The TECD utility function considers the duration of contacts (representing the intensity of social ties among users) and time-evolving interactions (reflecting users' habits over different daily samples).

Regarding the notations used in the equations presented in this sub-section: sumk(...) denotes summation for k from 1 to n; sumj(...) denotes summation for j from i to i+t-1; sumy denotes summation from all y belonging to N(x).

Two nodes may have a social weight, w_(x,y), that depends on the average total contact duration they have in that same period of time over different days. Within a specific daily sample Ti, node x has n contacts with node y, having each contact k a certain contact duration, CD_(x,y). At the end of each daily sample, the total contact time, TCT_(x,y), between nodes x and y is given by the equation below where n is the total number of contacts between the two nodes.

TCT_(x,y) = sumk(CD_(x,y)) (2)

The Total Contact Time between users in the same daily sample over consecutive days can be used to estimate the average duration of their contacts for that specific daily sample: the average duration of contacts between users x and y during a daily sample Ti in a day j, denoted by AD_(x,y) is given by a cumulative moving average of their TCT in that same daily sample, TCT_(x,y), and the average duration of their contacts during the same daily sample Ti on the previous day, denoted by AD_(x,y)_old, as shown in the equation below.

AD_(x,y) = (TCT_(x,y)+(j-1)*AD(x,y)_old)/j (3)

The social strength between users in a specific daily sample Ti may also provide some insight about their social strength in consecutive k samples in the same day, i+k. This is what we call Time Transitive Property. This property increases the probability of nodes being capable of transmitting large data chunks, since transmission can be resumed in the next daily sample with high probability.

TECD is able to capture the social strength w_(x,y) between any pair of users x and y in a daily sample Ti based on the average duration

AD_(x,y) of contacts between them in such daily sample and in consecutive t-1 samples, where t represents the total number of daily samples. When k>t, the corresponding AD_(x,y) value refers to the daily sample k-t. In the equation below the time transitive property is given by the weight t/(t+k-i), where the highest weight is associated to the average contact duration in the current daily sample, being it reduced in consecutive samples.

TECD = w_(x,y) = sumj(t/(t+k-i)*AD_(x,y)) (4)

### 2.2.2. TECD Importance (TECDi)

As social interaction may also be modeled to consider the node importance, TECDi computes the importance, I_(x), of a node x (cf. equation below), considering the weights of the edges between x and all the nodes y in its neighbor set, N_(x), at a specific daily sample Ti along with their importance.

TECDi = I_(x) = (1-d)+d*sumy(w_(x,y)*I_(y)/N_(x)) (5)

TECDi is based on the PeopleRank function [Mtibaa10]. However, TECDi considers not only node importance, but also the strength of social ties between bundle holder and potential next hops. Another difference is that with TECDi the neighbor set of a node x only includes the nodes which have been in contact with node x within a specific daily sample Ti, whereas in PeopleRank the neighbor set of a node includes all the nodes that ever had a link to node x. Note that the level of randomness may vary with the application scenario. Unless previously experimented, it is suggested that dumping factor is set to 0.8.

## 2.3. Forwarding strategy

Independently of the application scenario, each node MUST employ a forwarding strategy. The first rule is that if the encountered node is the final destination of a bundle, the carrier SHOULD prioritize such bundles by employing the prioritized forwarding strategy, described below.

We use the following notation for the description provided in this section. Nodes A and B are the nodes that encounter each other, and the strategies are described as they would be applied by node A.

### 2.3.1. Basic Strategy

Forward the bundle only if w_(B,D) > w_(A,D) or I_(B) > I_(A)

When two nodes A and B meet in any daily sample Ti, node A gets from

node B: a) the updated list of all neighbors of B, including the social weights that B has towards each of its neighbors, as well as the importance of B; b) the list of the bundles that B is carrying (bundle identifier, plus Endpoint Identifier (EID) of the destination); c) the list of the latest set of bundles acknowledged to B (the size of the list of acknowledged bundles returned by B depends on the local cache size and policy). The information about the social weight, importance, bundle list, and acknowledged bundles received from node B are referenced in node A as w_(B,x)_recv, I_(B)_recv, bundleList(IDn, destinationEIDx)_recv, and ackedBundleList(IDn, destinationEIDx)_recv, respectively.

For every bundle that A carries in its buffer, and i) is not carried by B, ii) has not been previously acknowledged to B, and iii) B has enough buffer space to store it, node A sends a copy to B if B has already encountered the bundle's destination D and its weight in w_(B,D)_recv is greater than A's weight towards this same destination D. Otherwise, bundles are replicated if I_(B)_recv is greater than A's importance in the current daily sample Ti.

Finally, node A will update its own ackedBundleList and discard bundles that have already been acknowledged to node B as described in Section 3.3.3.

### 2.3.2. Prioritized Strategy

Similar to the basic forwarding strategy, being the only difference the fact that prior to sending bundles, node A will first send those bundles that have node B as destination.

## 2.4. Interfaces

This section provides a specification of the two major interfaces required for dLife operation: a) the interface between the dLife routing agent and the bundle agent; b) the interface between the dLife routing agent and the lower layers.

### 2.4.1. Bundle Agent

The bundle protocol [RFC5050] introduces the concept of a "bundle agent" that manages the interface between applications and the "convergence layers" that provide the transport of bundles between nodes during communication opportunities. This draft extends the bundle agent with a routing agent that controls the actions of the bundle agent during communication opportunities.

This section defines the interfaces to be implemented between the bundle agent and the dLife routing agent. The defined interfaces

follow the general definition that was defined for the PRoPHET proposal.

In this document, we assume that functions in a complete bundle agent supporting dLife are distributed in such a way that reception and delivery of bundles are not carried out directly by the dLife agent, being the bundles placed in a queue available and managed by the dLife agent. In this case, this interface allows the dLife routing agent to be aware of the bundles placed at the node, and allows it to inform the bundle agent about the bundles to be sent to a neighbor node. Therefore, the bundle agent needs to provide the following interface/functionality to the routing agent:

Get Bundle List
: Returns a list of the stored bundles and their attributes to the routing agent.

Send Bundle
: Notifies the bundle agent to send a specified bundle.

Drop Bundle Advice
: Advises the bundle agent that a specified bundle may be dropped by the bundle agent if appropriate.

Acked Bundle Notification
: Bundle agent informs routing agent whether a bundle has been delivered to its final destination and time of delivery.

### 2.4.2. Lower Layers and Interface

To accommodate dLife operation on different types of wireless technology, the lower layers SHOULD provide the following functionality and interfaces.

Neighbor discovery and maintenance
: A dLife node needs to: i) know the identity of its neighbors; ii) when new neighbors appear; iii) when old neighbors disappear. Some wireless networking technologies might already contain mechanisms for detecting neighbors and maintaining state about them. Hence, neighbor discovery is not mandatory as a part of dLife. However, if the underlying networking technology does not support neighbor discovery and maintenance services, a simple neighbor discovery scheme using local broadcasts of beacon messages COULD be used, assuming that the underlying layer supports broadcast messages. The operation of the protocol is as follows:

    1. Periodically a dLife node does a local broadcast of a beacon that contains its identity and address.

2. Upon reception of a beacon, the following can happen:

o The sending node is already in the list of active neighbors. Update its entry in the list with the current time. At this point dLife should start the Neighbor Sensing procedure as mentioned in Section 3.1.

o The sending node is not in the list of active neighbors. Add the node to the list of active neighbors and record the current time.

3. If a beacon has not been received from a node in the list of active neighbors within a predefined time period, it should be assumed that this node is no longer a neighbor. The entry for this node should be removed from the list of active neighbors.

The lower layers MUST provide the two functions listed below:

New Neighbor
: Signals the dLife agent that a new node has become a neighbor (a node that is currently within communication range of the current node, based on the used wireless networking technology). At this point dLife should start the Neighbor Sensing procedure as mentioned in Section 3.1.

Neighbor Gone
: Signals the dLife agent that one of its neighbors has left.

Sender/Receiver Local Address
: An address used by the underlying communication layer (e.g., an IP or MAC address) that identifies the address of the sender and receiver nodes. This address and its format is dependent on the communication layer that is being used by the dLife layer. This address must be unique among the nodes that can currently communicate.

## 3. Protocol Overview

This section provides a description of the three operational phases of dLife, namely: neighbor sensing, information exchange, and bundle reception policies

### 3.1. Neighbor Sensing Phase

The operation of dLife depends on how nodes interact, i.e., considering all the potential contact opportunities to exchange

information. Thus, nodes running dLife MUST employ a mechanism for neighbor discovery (cf. Section 2.4.2) and neighbor sensing.

If the underlying networking technology does not support neighbor discovery and maintenance services, a mechanism as described in Section 2.4.2 can be provided.

When a node (new or already met) is discovered, dLife performs the following operation:

Start Contact Duration Counting
: dLife starts counting the contact duration for the purpose of later computing social weights and importance. In the case the peer node has been encountered before within the same daily sample, dLife checks if there were any changes in the metadata (e.g., Social Weights and Node Importance list, ackedBundleList) of the current node since the last encounter. If so, the current node will start the Hello Procedure.

Hello Procedure
: dLife sets up a link with the neighbor node through the Hello message exchange as described in Section 5.3. The Hello message exchange allows nodes to exchange information about their EID, storage capacity, current time, and timer value. Once the link has been set up the protocol may continue to the Information Exchange Phase (cf. Section 3.2) during which the lower layer is responsible for detecting broken links.

Stop Contact Duration Counting
: dLife stops counting the contact duration after detecting that the neighbor is gone, through the notification received from the lower layer (cf. Section 2.4.2).

In order to make use of this time dependence, dLife maintains a list of recently encountered nodes identified by the EID, as described in Section 5.2. Each entry of such list includes information that the node uses to update the status of the current communication session and to gather information about previous contacts. The size of this list is controlled, because due to low storage capacity of nodes, the information related to neighbors that are not in contact and towards which the current node has a social weight lower than a predefined threshold SOCIAL_DROP can be dropped from the list.

In order to make use of this time dependence, dLife maintains a list of recently encountered nodes identified by the Endpoint Identifier (EID), as described in section 5.2. Each entry of such list includes information that the node uses to update the status of the current communication session and to gather information about previous

contacts. The size of this list is controlled, because due to low storage capacity of nodes, the information related to neighbors that are not in contact and towards which the current node has a social weight lower than a predefined threshold SOCIAL_DROP can be dropped from the list.

### 3.2. Information Exchange Phase

The Information Exchange phase comprises the transfer of two types of metadata between connected nodes, through different messages described in Section 4:

o EID Dictionary
o Social Weights and Node Importance (SWNI)

Upon a communication opportunity, different sets of each type of metadata must be sent in each direction as explained further in this section. Each set may be transferred in one or more messages. In case a set of metadata needs more than one message to be completely transferred, it may be partitioned by the dLife protocol engine. The specification of dLife provides a submessage mechanism and retransmission that allows large messages to be transmitted in smaller chunks.

Each node running dLife is responsible for computing and updating their social weights towards previously encountered nodes as well as their own importance. Thus, in this operational phase, Social TLVs (Type-Length-Value messages), as defined in Section 4.3.4, are expected to reflect the latest updates regarding SWNI metadata, as well as to include the list of bundles carried by the peering node (bundleList) and the list of the latest bundles with acknowledged delivery (ackedBundleList). Social TLVs are generated throughout the information exchange phase upon updates of the SWNI information at the end of a daily sample.

As first step in the Information Exchange Phase one or more messages containing EID Dictionary metadata, EID Dictionary TLVs as defined in Section 4.3.3, MUST be sent to the peering node, if the list of encountered nodes is not empty. Such metadata contains a dictionary of the EIDs of the nodes that will be listed in the Social TLVs (cf. Section 3.2.1 for more information about this dictionary).

As a second step, one or more messages containing social metadata, Social TLVs, MUST be sent to the peering node, if the list of encountered nodes is not empty. This set of messages contains: i) a list with the EIDs of the nodes that the peering node has encountered so far and its social weights towards these nodes; ii) the importance of the peering node; iii) a list with the identifiers of the bundles

that the node currently carries and respective destinations EIDs; and iv) a list with the latest acknowledged bundles identifiers and respective destinations EIDs.

As a third step, upon reception and acknowledgment of the complete set of these messages, nodes MUST use one of the defined forwarding strategies (see Section 2.3) to decide which of the stored bundles (cf. Get Bundle List on Section 2.4.1) will be transferred to the peer, assuming that there are stored bundles.

The bundles to be sent to the peering node MUST be selected based upon the exchanged SWNI, bundleList and ackedBundleList information, as well as the available storage capacity on the receiving peering node. The bundles to be sent by the Bundle Agent (cf. Send Bundle on Section 2.4.1) SHOULD NOT exceed the peering node's capacity that MUST be indicated by the peer during the Hello procedure. The information to be passed to the Bundle Agent includes the number of bundles to be sent, where each bundle has an ID to be used for acknowledging their receipt.

### 3.2.1. EID Dictionary

The EID Dictionary, as used in PRoPHET, is a mapping between variable length EIDs [RFC4838] and String IDs coded as Self-Delimiting Numeric Values (SDNVs - see Section 4.1. of RFC 5050 [RFC5050]).

This dictionary is used by peering nodes to synchronize the EIDs of the nodes that they have encountered before. Each peer MAY add to the dictionary by sending a EID Dictionary TLV to its peer. To allow either peer to add to the dictionary at any time, the identifiers used by each peer are taken from disjoint sets: identifiers originated by the node that started the Hello procedure have the least significant bit set to 0 (i.e., are even numbers) whereas those originated by the other peer have the least significant bit set to 1 (i.e., are odd numbers). This means that the dictionary can be expanded by either node at any point of the information exchange phase and the new identifiers can then be used in subsequent TLVs until the dictionary is reinitialized.

The dictionary that is established only persists through a single encounter with a node (i.e., while the same link set up by the Hello procedure, with the same instance numbers, remains open).

Having more than one identifier for the same EID does not cause any problems. This means that it is possible for the peers to create their dictionary entries independently if required by an implementation, but this may be inefficient as a dictionary entry for an EID might be sent in both directions between the peers. It may be

required to inspect entries sent by the node that started the Hello procedure and thereby eliminate any duplicates before sending the dictionary entries from the other peer. Whether postponing sending the other peer's entries is more efficient depends on the nature of the physical link technology and the transport protocol used. With a genuinely full duplex link it may be faster to accept possible duplication and send dictionary entries concurrently in both directions. If the link is effectively half-duplex (e.g., Wi-Fi), then it will generally be more efficient to wait and eliminate duplicates.

If a node receives EID Dictionary metadata containing an identifier that is already in use, the node MUST confirm that the corresponding EID is identical to the EID in the existing entry. Otherwise, the node MUST send an ACK TLV (i.e., EID ACK - identifier/EID discrepancy) and ignore the EID Dictionary TLV containing the error. If a node receives EID Dictionary metadata that uses an unknown identifier (i.e., not in the dictionary), the node MUST send an ACK TLV (i.e., EID ACK - unknown EID) message and ignore the TLV containing the error.

### 3.2.2. Operation in the presence of multiples neighbors

As a node may find itself in the range of more than one potential next forwarder, the neighbor sensing mechanism may establish multiple information exchanges with each of them.

If these simultaneous contacts persist for some time, then the information exchange process will be periodically rerun for each contact according to the configured timer interval, which means that different Hello TLVs will be exchanged at different times.

Based on the receipt time of these Hello TLVs at the sending node, it will establish the order for sending out the bundles, considering the storage capacity of the different neighbors.

## 3.3. Bundle Reception Policies

### 3.3.1. Queueing policy

Because of limited buffer resources, bundles may need to be dropped at some nodes. Although dLife evaluation based on simulations have shown little consumption due to limiting replication based on social strength, a scheme MUST be used upon an exhaustion of buffer space. Hence, each node MUST operate a queueing policy that determines which bundles should be available for forwarding.

This section defines a few basic queueing policies, inline with what

was proposed for PRoPHET. However, nodes MAY use other policies if desired. If not chosen differently due to the characteristics of the deployment scenario, nodes SHOULD choose FIFO as the default queueing policy.

FIFO
: Handle the queue in a First In First Out (FIFO) order. The bundle that was first entered into the queue is the first bundle to be dropped.

FLNT
: The bundle that has been forwarded the largest number of times is the first to be dropped. For this effect, dLife SHOULD keep track of the number of times each bundle has been forwarded to other nodes.

STTL
: The bundle that has shortest time-to-live is dropped first. As described in [RFC5050], each bundle has a timeout value specifying when it no longer is meaningful to its application and should be deleted. Since bundles with short remaining time to live will soon be dropped anyway, this policy decides to drop the bundle with the shortest remaining life time first. To successfully use a policy like this, there needs to be some form of time synchronization between nodes so that it is possible to know the exact lifetimes of bundles.

DLSW
: The bundle that has a destination with low social weight is dropped first. A low social weight means that the carrier may not be the best forwarder to this bundle. However, such bundle can only be dropped if it was already forwarded for at least a Minimum Bundle Forward (MBF) times, which is a minimum number of forwards that a bundle must have been forwarded before being dropped (if such a bundle exists).

More than one queueing policy MAY be combined in an ordered set, where the first policy is used primarily, the second only being used if there is a need to tie-break between bundles given the same eviction priority by the primary policy, and so on. It is worth noting that obviously nodes MUST NOT drop bundles for which it has custody unless the lifetime expires.

### 3.3.2. Custody Policy

The concept of custody transfer can be found in [RFC4838]. In general terms, the transmission of bundles with the Custody Transfer Requested option involves moving bundles "closer" (in terms of some

routing metric) to their ultimate destination(s) with reliability. The nodes receiving these bundles along the way (and agreeing to accept the reliable delivery responsibility) are called "custodians". The movement of a bundle (and its delivery responsibility) from one node to another is called a "custody transfer".

The reliability requirement that a custodian accepts can be instantiated in different ways: i) deleting the bundle after getting a confirmation of a successful custody transfer, which may require retransmissions over a reliable transport protocol, such as TCP. In this case, a bundle has normally one custodian in a moment in time; ii) deleting the bundle only after getting an acknowledgment that the bundle was delivered to the destination. In this case, a bundle can have more than one custodian, being the bundle replicated among custodians over a non-reliable transport protocol, such as UDP.

dLife takes no responsibilities for making custody decisions. Such decisions should be made by a higher layer. However, dLife insures that custodian nodes do not drop bundles for which it has custody unless the lifetime expires, or an acknowledge message is received for that bundle.

### 3.3.3. Destination Policy

When a bundle reaches its final destination, the Bundle Agent sends a notification to the routing agent (cf. Acked Bundle Notification in Section 2.4.1), being that information (e.g., Bundle, deliveryTime) stored in the ackedBundleList by the routing agent. When nodes exchange Social message TLVs, bundles that have been ACKed are also listed. The node that receives this list updates its own list of ACKed bundles to be the union of its previous list and the received list. To prevent the list of ACKed bundles growing indefinitely, the ackedBundleList is periodically checked and bundles are removed following the configured queueing policy (c.f. Section 3.3.1) if the size of the list is bigger than a predefined threshold.When a node receives a notification for a bundle it is carrying, it MUST delete that bundle from its queue, since the notification indicates that a bundle has been delivered to its final destination.

Nodes MAY keep track of which nodes they have sent Bundle ACKs for certain bundles to, and MAY in that case refrain from sending multiple Bundle ACKs for the same bundle to the same node.

## 4. Message Formats

This section defines the message formats of the dLife routing protocol. In order to allow for variable length fields, many numeric

fields are encoded as SDNVs, defined in [RFC5050], as in PRoPHET. Since many of the fields are coded as SDNVs, the size and alignment of fields indicated in many of the specification diagrams below are indicative rather than prescriptive.

The basic message format shown in Figure 1 consists of a header (see Section 4.1) followed by a sequence of one or more Type-Length-Value components (TLVs) taken from the specifications in Section 4.2.

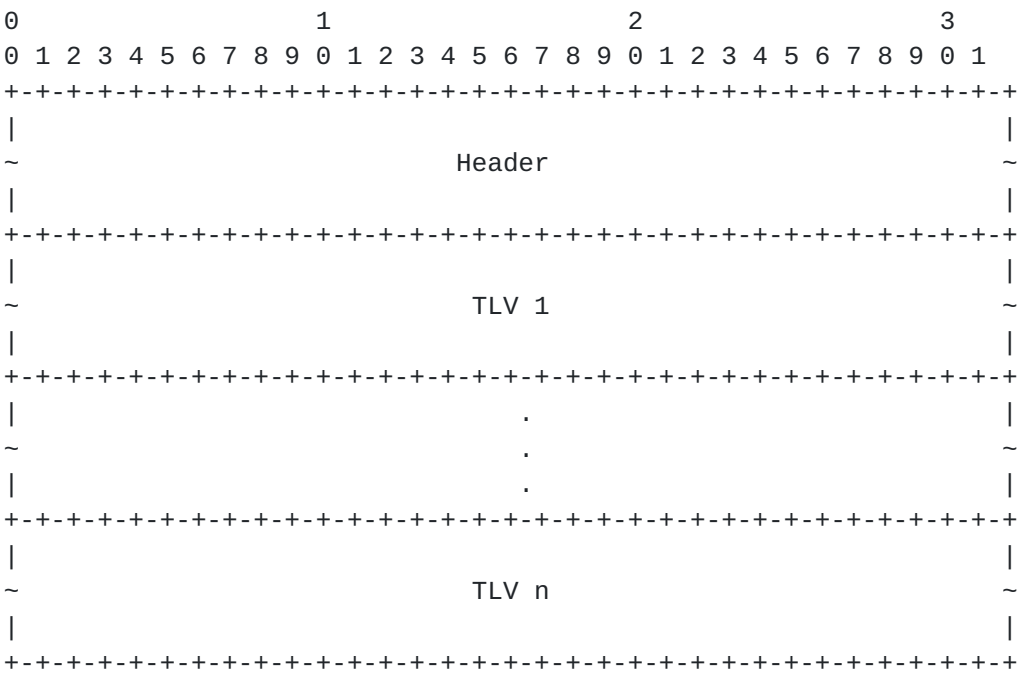


Figure 1: Basic dLife Message Format

## 4.1. Header

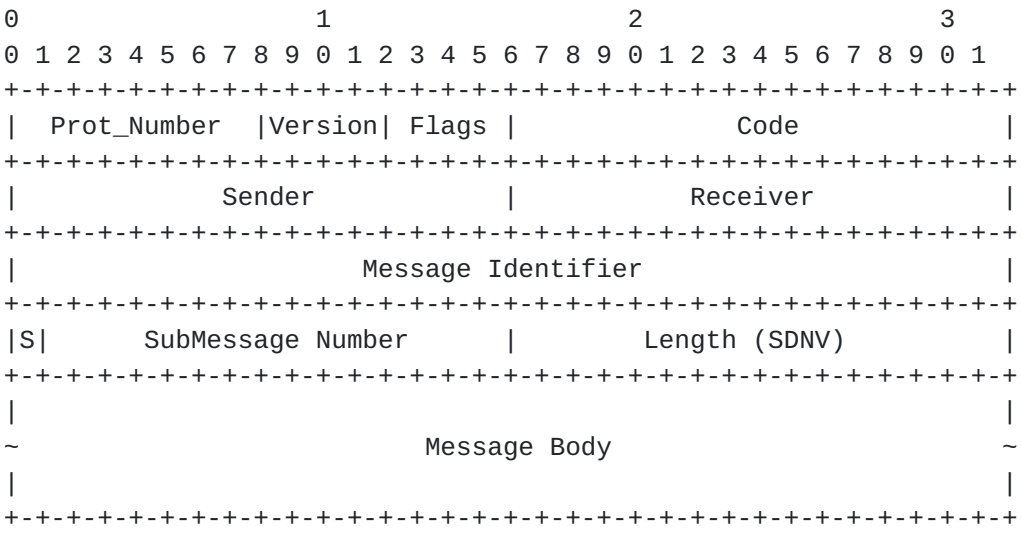


Figure 2: dLife Message Header

Prot_Number (Protocol Number)

The DTN Routing Protocol Number is encoded as 8-bit unsigned integer in network bit order. The value of this field is 0. The dLife header is organized in this way so that dLife messages MAY be sent as the Protocol Data Unit of an IP packet if an IP protocol number was allocated for dLife. Transmitting dLife packets directly as an IP protocol on a public IP network such as the Internet would generally not work well because middle boxes such as firewalls and NAT boxes would be unlikely to allow the protocol to pass through and the protocol does not provide any congestion control. However, it could be so used on opportunistic wireless networks, which is the goal of dLife. The use of a light transport protocol such as UDP, in opposition to TCP, ensures a better exploitation of the presumable short contact opportunities between peers in a DTN. Due to the lack of reliability of UDP, dLife specify an acknowledgement procedure to transmit metadata. Hence, dLife is prepared to use UDP over Wi-Fi, while over Bluetooth dLife uses the RFCOMM and L2CAP protocols. In both cases message acknowledgement is made by the dLife mechanism.

Version

The Version of the dLife Protocol. Encoded as a 4-bit unsigned integer in network bit order. This document defines version 1.

Flags

The flags field is encoded as a 4-bit unsigned integer in network bit order. The following values are currently defined:

- Success 0
- Failure 1

Code

This field gives further information concerning the flag in a response message. It is mostly used to pass an error code in a failure response, but can also be used to give further information in a success response message. In a request message, the code field is not used and is set to zero.

If the Code field indicates that the ACK TLV is included in the message, further information on the successful or failure of the message will be found in the ACK TLV, which MUST be the first TLV after the header.

The Code field is encoded as an 8-bit unsigned integer in network bit order with 5 unused and reserved bits, which were included to allow future extension of the protocol. The following values are defined:

- o Generic Response 0x00
- o Submessage Received 0x01
- o ACK TLV in message 0x02
- o Unexpected Error 0x03

The "generic response" and the "submessage receiver" values tell us that the success or failure indicated in the Flag field is related to the all message or a submessage, respectively.

Sender

This field is the instance number for the link of the sender of the dLife message. This instance number identifies a link between peering nodes and lasts until the link goes down or when the identity of the entity at the other end of the link changes. It is randomly generated. This number is a 16-bit number that is guaranteed to be unique within the recent past and to change when the link or node comes back up after going down. Zero is not a valid instance number. Messages sent throughout all the communication phases (i.e., Sensing, Hello, Information exchange) should use the sender's instance number. The Sender Instance is encoded as a 16-bit unsigned integer in network bit order.

Receiver

This field is the instance number for the link of the receiver of the dLife message. If the sender of the message does not know the current number of the receiver, this field MUST be set to zero. Messages sent throughout all the communication phases (i.e., Sensing, Hello, Information exchange) should use the receiver's instance number. The receiver number is encoded as a 16-bit unsigned integer in network bit order.

Message Identifier

Used to associate a message with its response message. This should be set in request messages to a value that is unique for the sending host within the recent past. Response messages contain the Message Identifier of the request they are responding to. The Message Identifier is a 32 bit pattern.

S-flag

If S is set (value 1) then the SubMessage Number field indicates the total number of SubMessage segments that compose the entire message. If it is not set (value 0) then the SubMessage Number field indicates the sequence number of this SubMessage segment within the whole message. The S field will only be set in the first sub-message of a sequence.

SubMessage Number

When a message is segmented because it exceeds the MTU of the link

layer or otherwise, each segment will include a SubMessage Number to indicate its position. Alternatively, if it is the first sub-message in a sequence of sub-messages, the S flag will be set and this field will contain the total count of SubMessage segments. The SubMessage Number is encoded as a 15-bit unsigned integer in network bit order. The SubMessage number is zerobased, i.e., for a message divided into n sub-messages, they are numbered from 0 to (n - 1). For a message that it is not divided into sub-messages the single message has the S-flag cleared (0) and the SubMessage Number is set to 0 (zero).

Length
Length in octets of this message including headers and message body. If the message is fragmented, this field contains the length of this SubMessage. The Length is encoded as an SDNV.

Message Body
The Message Body consists of a sequence of one or more of the TLVs specified in Section 4.2.

## 4.2. TLV Structure

All TLVs have the following format, and can be nested.

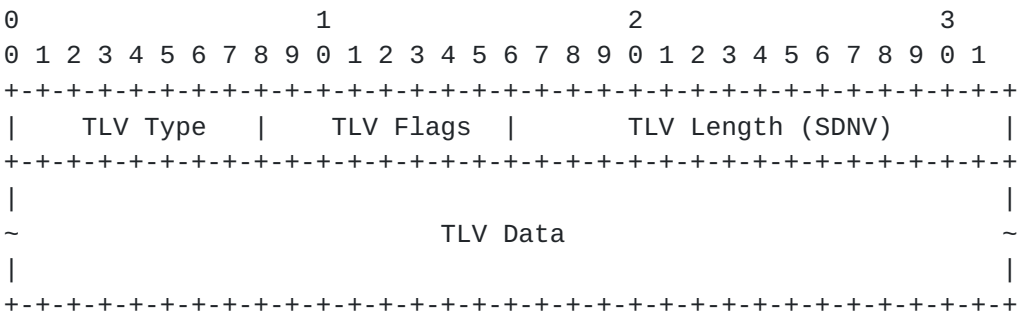


Figure 3: TLV Format

Type
Specific TLVs are defined in Section 4.3. The TLV Type is encoded as an 8-bit unsigned integer in network bit order.

TLV Flags
These are defined per TLV type. Any flags which are specified as reserved in specific TLVs SHOULD be transmitted as 0 and ignored on receipt.

TLV Length
Length of the TLV in octets, including the TLV header and any nested TLVs. Encoded as an SDNV.

### 4.3. TLVs

This section describes the various TLVs that can be used in dLife messages.

#### 4.3.1. Hello TLV

The Hello TLV is used to set up and maintain a link between two dLife nodes. Hello messages are the first TLVs exchanged between nodes when they are within range of communication and are used to inform neighbors about the EID, storage capacity, and current time of the node and a timer value.

The Hello sequence must be completed so other TLVs can be exchanged. After the Hello procedure dLife nodes will store the information about each other EIDs, capacities and considered timer. Such action is acknowledged by signaling that the communication has been established. If during the Hello procedure, an ACK is failed to be received, disconnection occurred and link should be assumed broken.

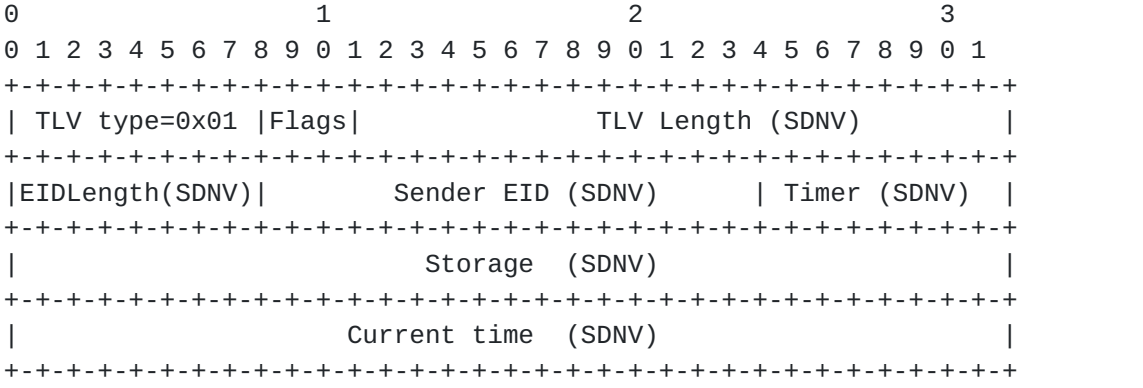


Figure 4: Hello TLV Format

TLV Flags

The TLV Flags field contains a three bit Hello Function (HF) number that specifies one of three functions for the Hello TLV. The encoding of the Hello Function is:

- HEL: HF = 1
- ACK: HF = 2

The HEL function is used by a node to send metadata needed for the dLife operation, namely the storage capacity of the node, its EID, current time, and a timer value. The ACK function in used by a node to acknowledge the reception of an HEL Hello TLV.

TLV Data

EID Length

The EID Length field is used to specify the length of the Sender EID field in octets. If the EID has already been sent at least once in a message, a node MAY choose to set this field to zero, omitting the Sender EID from the Hello TLV. The EID Length is encoded as an SDNV and the field is thus of variable length.

Sender EID

The Sender EID field specifies the EID of the sender that is to be used in updating routing information and making forwarding decisions. If a node has multiple EIDs, one should be chosen for dLife routing. This field is of variable length.

Timer

The Timer field is used to inform the receiver of the timer value used in the Hello processing of the sender. The timer specifies the nominal time between periodic Hello messages. It is a constant for the duration of a session. The timer field is specified in units of 100 ms and is encoded as an SDNV.

Storage

This field indicates the node's storage capacity. Used to inform the potential senders of the node's limitations in terms of successful reception of bundles. This field is encoded as an SDNV.

Current Time

This field specifies the current time in the peering node. It is given in seconds and counting starts in the year 2000. This field is encoded as an SDNV.

### 4.3.2. ACK TLV

This ACK TLV can be used by itself or nested in Hello TLV.

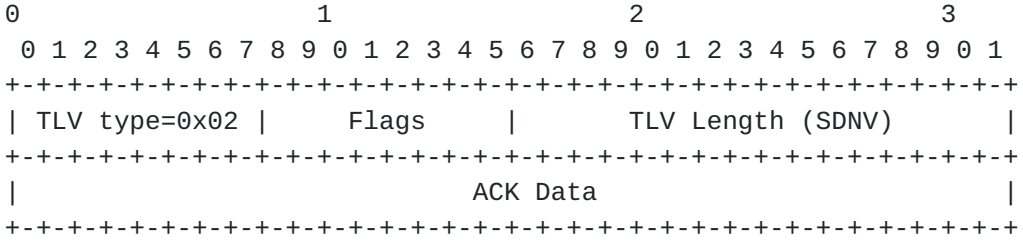


Figure 5: ACK TLV Format

TLV Flags

The TLV Flags field carries an identifier for the ACK TLV type as an 8-bit unsigned integer encoded in network bit order. A range of values is available for private and experimental use in addition to the values defined here. The following ACK TLV types are defined:

- Break ACK 0x00
  Used when a node wants to break the connection due to the fact that the neighbor has a storage capacity lower that its threshold to send bundles.

- Social ACK 0x01
  Report on the reception of Social TLVs (SWNI, bundleList and ackedBundleList).

- EID ACK (identifier/EID discrepancy) 0x02
  Report on the identifier/EID discrepancy error as mentioned in Section 3.2.1.

- EID ACK (unknown EID) 0x03
  Report on the unknown EID error as mentioned in Section 3.2.1.

- Reserved 0x04 - 0x7F

- Private/Experimental Use 0x80 - 0xFF

TLV Data

The contents and interpretation of the TLV Data field are specific to the type of ACK TLV. The ACK Type is defined as follows:

Break ACK
  This field is set to zero.

Social ACK
  This field is set to zero.

EID ACK (identifier/EID discrepancy)
  String ID causing the discrepancy and the EID string that differs the previous value.

EID ACK (unknown EID)
  String ID not found in the dictionary.

### 4.3.3. EID Dictionary TLV

The EID Dictionary TLV includes the list of EIDs used in making routing decisions and is a shared resource (cf. Section 3.2.1) built

in each of the paired peers. The dictionary can be updated as more EID Dictionary TLVs are received.


Figure 6: EID Dictionary TLV Format

Reserved

8 unused and reserved bits, which were included to allow future extension of the protocol

TLV Data

String/EID Count

Number of strings corresponding to the nodes' EIDs the sender node has encountered so far. Encoded as SDNV.

String ID n

SDNV identifier that is constant for the duration of a session. String ID zero is predefined as the node initiating the session through sending the Hello message, and String ID one is predefined as the node responding with the Hello ACK message. These entries do not need to be sent explicitly as the EIDs are exchanged during the Hello procedure.

In order to ensure that the String IDs originated by the two peers do not conflict, the String IDs generated in the node that sent the Hello HEL message MUST have their least significant bit set to 0 (i.e., are even numbers) and the String IDs generated in the node that responded with the Hello ACK message must have their least significant bit set to 1 (i.e., they are odd numbers).

Length
Length of Endpoint Identifier in this entry. Encoded as SDNV.

Endpoint Identifier n
Text string representing the Endpoint Identifier. Note that it is NOT null terminated as the entry contains the length of the identifier.

#### 4.3.4. Social TLV

This TLV provides the SWNI, bundleList and ackedBundleList information, i.e., a list of the nodes that the peer node has encountered up to that moment along with its social weight towards them, the importance of the peer node, as well as the lists containing bundles being carried by the peering node and acknowledgements for already delivered bundles (limited to the latest BUNDLE_DELIVERED number of bundles). This TLV allows dLife nodes to choose the bundles to be sent according to the forwarding strategies explained in Section 2.3.

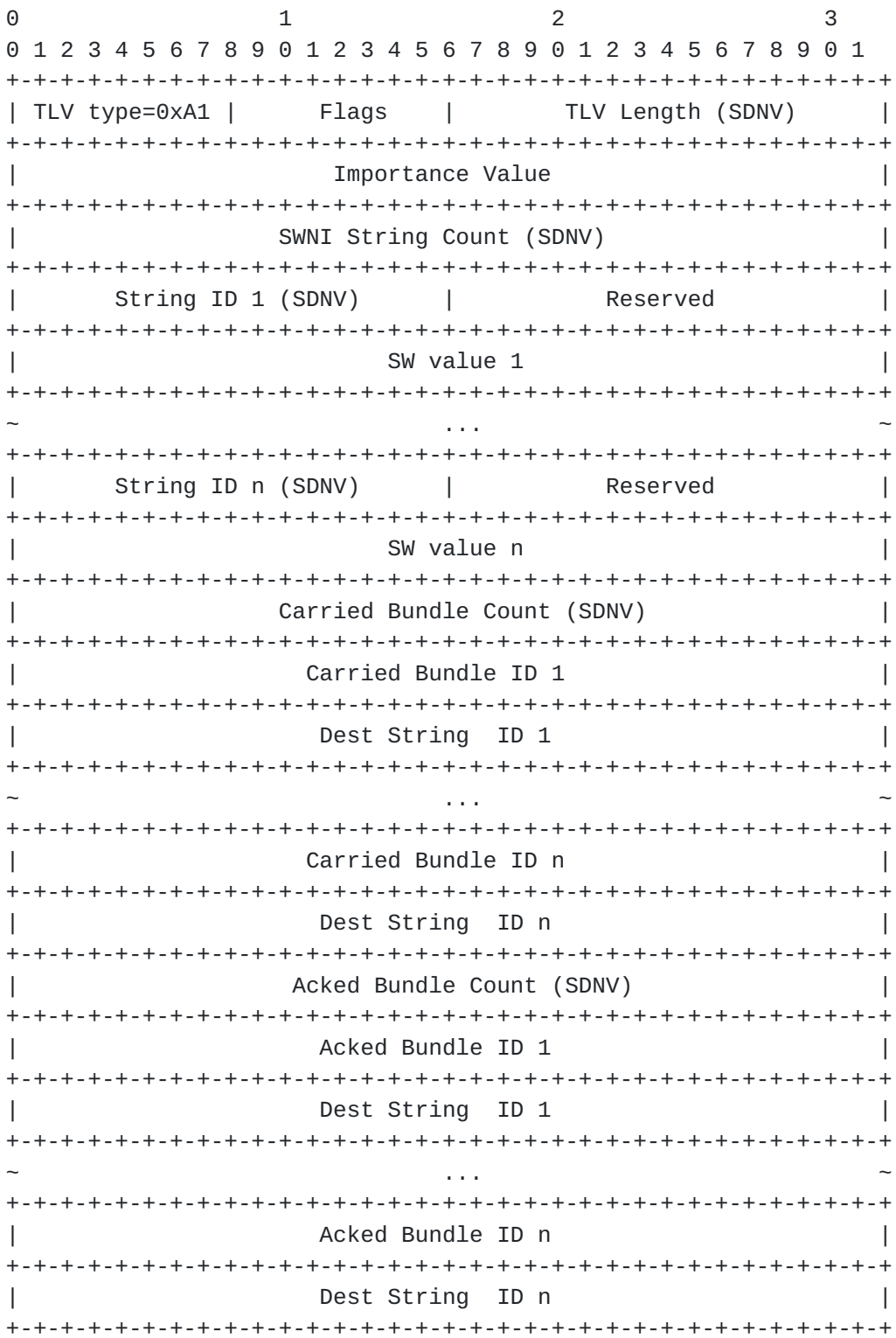


Figure 7: Social message TLV Format

TLV Flags

The encoding of the Header flag field relates to the capabilities

of the Source node sending the Social message.

- o Flag 0: More Social TLVs
- o Flag 1: Reserved
- o Flag 2: Reserved
- o Flag 3: Reserved
- o Flag 4: Reserved
- o Flag 5: Reserved
- o Flag 6: Reserved
- o Flag 7: Reserved

The "More Social TLVs" flag is set to 1 if the Social message requires more TLVs to be sent in order to be fully transferred. This flag is set to 0 if this is the final TLV.

TLV Data

Importance
: Importance of the node sending the Social message as a 32-bit unsigned integer encoded in network bit order.

SWNI String Count
: Number of entries regarding the SWNI information in the TLV. Encoded as SDNV.

String ID
: String ID of the endpoint identifier of the encountered node for which this entry specifies the social weight as predefined in a dictionary TLV. This field is of variable length. This field is followed by 16 unused and reserved bits, which were included to allow future extension of the protocol (e.g., use of another metric such as contact volume or signal strength to improve the forwarding choice). Encoded as SDNV.

SW value
: Social weight between peering node and that specific encountered node n as a 32-bit unsigned integer encoded in network bit order.

Carried Bundle Count
: Number of entries regarding the carried bundle information in the TLV. Encoded as SDNV.

Carried Bundle ID
: Identifiers of the bundles that the sender is currently carrying as a 32-bit unsigned integer.

Acked Bundle Count
: Number of entries regarding the acknowledged bundle information

in the TLV. Encoded as SDNV.

Acked Bundle ID
Identifiers of the bundles that the sender has received acknowledgements for as a 32-bit unsigned integer.

Dest String ID
String ID of the endpoint identifier of the destination for a carried or acknowledged bundle. This field is of variable length. Encoded as SDNV.

## 5. Detailed Operation

This section provides further details about the operation of dLife, including state tables. As explained before dLife aims to release any assumption about the reliability of the transport protocol, and so positive acknowledgements would be necessary to signal successful delivery of (sub)messages. In this section the phrase "send a message" should be read as *successful* sending of a message, signaled by receipt of the appropriate "Success" response. Hence the state descriptions below do not explicitly mention positive acknowledgements, whether they are being sent or not.

### 5.1. High Level State Tables

This section provides the high level state tables for the operation of dLife. The next section provides a more detailed view of each part of the protocol's operation. The following states are used to define the dLife operation:

SENSING
This is the state all nodes start in. Nodes remain in this state until a new contact opportunity arises. Once the routing agent has sensed the presence of a peer, via notification of the lower layer, it will start counting the contact duration. The Hello procedure will be triggered depending if the peer is a new contact or an already encountered peer (as explained in Section 3.1) by switching to the HELLO state. Since multiple contacts may happen, the node should also remain in the SENSING state in order to detect new contact opportunities. This is handled by creating a new thread or process during the transition to the HELLO state, which then takes care of the communication with the new peer while the parent process remains in state SENSING waiting for additional peer to communicate with. In the case when the neighbor is no longer available (described as 'neighbor gone notification rcvd' in the tables below), the thread or process created is destroyed.

HELLO

Nodes remain in the HELLO state from when a new contact opportunity arises until the Hello procedure is done and nodes are connected (which happens when the Hello procedure reaches the LINK_UP state as described in Section 5.3 - during this procedure, the state LINK_UP and WAIT_HELLO_HEL are used, but are not presented here since they are internal to the Hello procedure). Once the Hello procedure is done, the node starts the information exchange phase and transitions to the EXCHANGE state. If while in the HELLO state the node is notified that the neighbor is no longer in range by the lower layer, it returns to the SENSING state and, if appropriate, MAY destroys any additional process or thread created to handle the neighbor.

EXCHANGE

With the communication link set, nodes enter the EXCHANGE state in which the transmission of dLife metadata between peers is done. The node remains in this state as long as Information Exchange Phase TLVs (EID Dictionary, Social and ACK) are being exchanged.

In the EXCHANGE state both nodes are able to exchange their EID dictionaries and SWNI information. With dLife the exchange of information about dictionary, social weight and node importance MAY be carried out independently but concurrently with the messages multiplexed on a single bidirectional link, or alternatively, the exchanges MAY be carried out partially or wholly sequentially if appropriate for the implementation. The information exchange process is explained in more detail in Section 3.2.

When a Social TLV is received the node MUST notify the Bundle Agent about the bundles that SHOULD be forwarded to the peer node. If the "More Social TLV" flag is set to zero (i.e., no more bundles to send), the node returns to the SENSING state. If the routing agent is notified by the lower layer that the neighbor is no longer in range, the node switches to the SENSING state and, if appropriate, MAY destroy any additional process or thread created to handle the neighbor.

If one or more new bundles are received by this node and the TECD and TECDi metrics indicate that it would be appropriate to forward some or all of the bundles to the connected node(s), the bundles SHOULD be immediately transferred to the connected peer(s). As mentioned earlier, the lower layer is responsible in notifying the routing agent whether peers are still within communication range (cf. Section 2.4.2).

State: SENSING

| Condition | Action | New State |
|---|---|---|
| New Contact | Start contact duration count<br>Start Hello procedure | HELLO |
| Keep sensing for more contacts | | SENSING |
| Neighbor gone notification rcvd | | SENSING |

State: HELLO

| Condition | Action | New State |
|---|---|---|
| Hello TLV rcvd | | HELLO |
| Hello procedure done | Start Information Exchange Phase | EXCHANGE |
| Neighbor gone notification rcvd | | SENSING |

State: EXCHANGE

| Condition | Action | New State |
|---|---|---|
| On entry | Send metadata | EXCHANGE |
| EID Dict TLV rcvd | Update local EID Dictionary | EXCHANGE |
| Social TLV rcvd | Inform Bundle Agent | EXCHANGE |
| More Social TLV flag = 0 | | SENSING |
| New bundle | | EXCHANGE |
| Neighbor gone notification rcvd | | SENSING |

### 5.2. High Level Meta-Data Table

During its operation, dLife makes use of metadata locally stored as a consequence of the exchange of Social TLVs, Hello TLVs and local operations. The stored metadata is used in the computation of social weight towards other nodes and its own importance as well as to identify itself (cf. Section 2.2). Metadata can be persistent or temporary: the former MUST be kept for longer times and the latter is replaced as a new daily sample starts.

Persistent metadata includes the node's own importance (Imp), Average Duration (AD_peer) of contacts to peers, social weight to peer (w_peer), and last daily sample in the case of failure/shutdown (see Section 2.2). Since nodes receive information from neighbors, they must also store the peer's EID (EID_peer), importance of that peer (Imp_peer), and peer's current time (currTime_peer) (cf. Section 4.3.1). Additionally, nodes must keep track of number of times the bundles were forwarded (fwd_times) as to employ the queueing FLNT policy describe in Section 3.3.1.

The temporary metadata includes the node's own EID, storage capacity (StoCap), current time (currTime), EID Dictionary (EIDDict), contact duration (CD_peer) and total contact time (TCT_peer) for a given peer, and Last Encounter (LastEnc_peer). Temporary metadata received from peers are storage capacity (StoCap_peer), EID Dictionary (EIDDict_peer), and social weight list of that peer towards other nodes (SocList_peer).

In the SENSING state, each node MUST have its own EID, storage capacity, and current time ready for exchange in the case of a contact. When a contact is sensed, a node creates an entry for this potential peer (EID_peer) in metadata table and start counting the duration of this contact (CD_peer). Note that the peer EID will only be known in the HELLO state.

If the HELLO state is successfully concluded, the EID_peer is now known and new entries for the encountered peer are created (TCT_peer, AD_peer, w_peer, StoCap_peer, and currTime_peer) in the metadata table. LastEnc_peer is also initialized in the HELLO state and receives the time nodes encountered. This variable will tell a node if the social information to be exchanged is up-to-date.

When the EXCHANGE state starts, a node will receive from its peer its EIDDict_peer, SocList_peer and Imp_peer, which must be stored for later deciding in bundle forwarding.

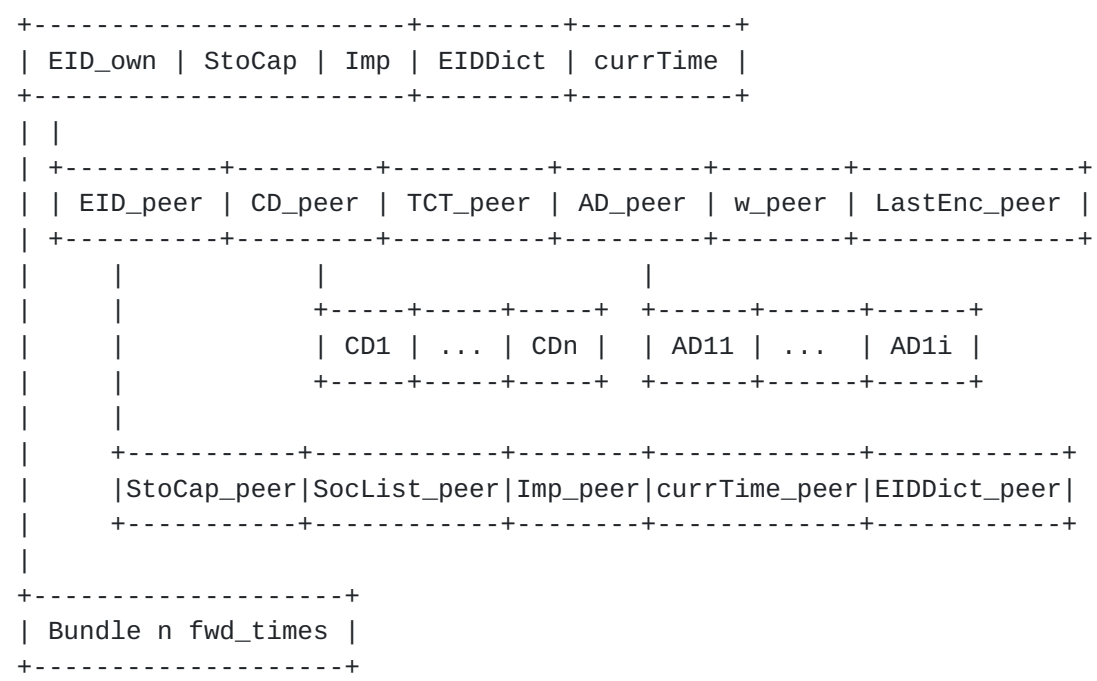


Figure 8: Meta-data Information

This metadata varies in size according to the type of information it stores:

- EID and EID_peer are coded as strings of variable size.
- StoCap and StoCap_peer are coded as int (32 bits).
- currTime, currTime_peer and LastEnc_peer are coded as int (32 bits).
- Imp, Imp_peer, AD_peer and w_peer are coded as floats (32 bits).
- CD_peer and TCT_peer are coded as long (64 bits).
- Fwd_times are coded as int (32 bits).
- EIDDict and EIDDict_peer are coded as a HashMap tuple with String ID encoded as SDNV and its equivalent EID of variable length (up to 32 bits).
- SocList_peer is coded as a HashMap tuple with String ID of the encountered nodes, encoded as SDNV, and the social weight of the peer towards these nodes.

### 5.3 Hello Procedure

The hello procedure consists of the exchange of messages comprising the header TLV and a single Hello TLV (see Section 4.3.1) with the HF (Hello Function) field set to the specified value (HEL or ACK).

The rules and state tables for this procedure are shown below with the main states and actions to be taken upon the receipt of the required information:

- Hello HEL messages MUST always issued first, and SHOULD be followed by a Hello ACK.

- The link between peers is only considered available when the LINK_UP state is reached.

- Hello messages MAY be exchanged concurrently, but also in a sequential manner, depending on the nature of the communication medium (full- or half-duplex). In the case of the latter, the process MUST be completed in one direction prior to initiating in the other one.

Upon a contact, nodes exchange a Hello HEL message. After that nodes will have information about each other's EID, storage capacity, and current time. The current time is used to determine in which daily sample the peer is in order to facilitate any required updates concerning social weights and importances that may have happened since last encounter between them. Additionally, a timer value is exchanged so nodes know the periodicity of next hello messages in order to signal the arrival of the Hello HEL message.

At this moment, nodes MUST create entries for the peer (EID_peer) where the contact duration (CD_peer), total connected time (TCT_peer), average duration (AD_peer), social weight (w_peer), storage capacity (StoCap_peer), current time (currTime_peer), LastEnc_peer (initialized with currTime_peer), social weight list (SocList_peer), and the importance (Imp_peer) towards this specific peer are going to be maintained and updated. Additionally, a timer for receiving the Hello ACK message MUST be started. Up to this point, nodes are in the WAIT_HELLO_HEL state.

A second hello message with the ACK flag MUST be issued to inform nodes about the successful reception of the Hello HEL message. If Hello ACK message does not arrive within the specified time (Hello ACK timeout), the link is assumed lost, nodes are disconnected, contact duration count MUST stop and variables SHOULD be saved. After this, the start of a new hello procedure is required and the nodes remain at the WAIT_HELLO_HEL state.

The link is assumed active upon the receipt of the Hello ACK message. This means nodes are connected and ready to shift to the exchange information phase. At this point values should be saved in created entries and nodes move to the LINK_UP state.

Nodes will remain at the LINK_UP state until the routing agent receive a notification from the lower layer reporting that the peer is no more within communication range (cf. neighbor gone in Section 2.4.2). At this point nodes MUST stop the contact duration count with the value being saved to CD_peer (CD1 ... CDn) variables of each disconnected peer.

State: WAIT_HELLO_HEL

| Condition | Action | New State |
|---|---|---|
| Hello HEL rcvd | Start timer for Hello ACK receipt<br>Initialize variables | WAIT_HELLO_HEL |
| Hello ACK rcvd | Save variables | LINK_UP |
| Hello ACK timeout | Stop contact duration count | WAIT_HELLO_HEL |
| Neighbor gone notification rcvd | Save variables | |

State: LINK_UP

| Condition | Action | New State |
|---|---|---|
| Neighbor gone notification rcvd | Stop contact duration count<br>Save variables | WAIT_HELLO_HEL |

## 5.4 Information Exchange Phase

After the exchange of hello messages, the nodes are in the LINK_UP state, which allows the exchange of information. dLife is bidirectional and the information exchange processes between a pair of nodes (from A to B and vice-versa) are independent and expected to run almost entirely concurrently. This is because EID Dictionaries

SHOULD be synchronized to allow better performance of the protocol.

The information exchange phase consists of messages comprising the dLife header and EID Dictionary and Social TLVs (see Sections 4.3.3 and 4.3.4). The rules and state tables are shown below with the main states and actions to be taken upon the receipt of the required information.

- o No information SHALL be exchanged prior to the hello procedure.

- o Information messages MAY be exchanged concurrently, but also in a sequential manner, depending on the nature of the communication medium (full- or half-duplex). In the case of the latter, the process MUST be completed in one direction prior to initiating in the other one.

Once in the information exchange phase, as soon as nodes enter the WAIT_INFO state, they SHOULD send the EID Dictionary in order to synchronize the mapping between their identifiers (String ID) and EIDs of nodes they have encountered. As the EID dictionary is received, the node will shift to the BUILD_SOCIAL_TLV state and check whether or not problems with the mapping are detected. In the case of EID errors, an ACK TLV with EID ACK flag MUST be sent to inform the peer node about problems (cf. Sections 3.2.1 and 4.3.2) with the received identifiers (String ID).

Still in the WAIT_INFO state, if a node gets a Social TLV, it will obtain the SWNI, bundleList and ackedBundleList information from its peer node. With such information, the node will decide which bundles SHOULD be exchanged based on the forwarding strategies (cf. Section 2.3), will update its ackedBundleList, and will inform the Bundle Agent (cf. Send Bundle in Section 2.4.1) about the list of bundles that SHOULD be forwarded to its peering node.

The node will remain in the WAIT_INFO state after receiving an Acked Bundle Notification (cf. Section 2.4.1), which means that the bundles forwarded by the Bundle Agent arrived at the destination node, and the sending node can update its ackedBundleList; and also when receiving an ACK TLV with the Break ACK flag, which signals that the peering node decided to disconnect given its lack of storage capacity, and the sending node will stop the contact duration count and wait for information coming from other peers.

As soon as entering the BUILD_SOCIAL_TLV state, the node MUST build and send its Social TLV to the peering node and will remain in this state until it receives an ACK TLV with the Social ACK flag. The node will shift to the WAIT_INFO state upon the receipt of the Social ACK.

Since dLife updates SWNI information at every daily sample, this can influence in the next forwarding decisions. Thus, the node should report such updates to its peering node and shift to the BUILD_SOCIAL_TLV state.

Independently of the state the node finds itself, if receiving a notification from the lower layer reporting that the peer is no longer within communication range (cf. neighbor gone in Section 2.4.2), it MUST stop the contact duration count and wait for information coming from other peers.

State: WAIT_INFO

| Condition | Action | New State |
|---|---|---|
| On entry | Send EID Dictionary TLV | WAIT_INFO |
| EID Dictionary rcvd | Check identifier/EID mapping | BUILD_SOCIAL_TLV |
| EID error | Report problem<br>ACK TLV flag EID ACK | BUILD_SOCIAL_TLV |
| Social TLV rcvd | Get SWNI, bundleList and ackedBundleList information | WAIT_INFO |
| | Inform Bundle Agent (Send Bundle) | |
| | Update ackedBundleList | |
| Acked Bundle Notification rcvd | Update ackedBundleList | WAIT_INFO |
| Break ACK rcvd | Disconnect | WAIT_INFO |
| | Stop contact duration count | |
| SWNI updated | Report peer | BUILD_SOCIAL_TLV |
| Neighbor gone notification rcvd | Stop contact duration count | WAIT_INFO |

State: BUILD_SOCIAL_TLV

| Condition | Action | New State |
|---|---|---|
| On entry | Build and send Social TLV | BUILD_SOCIAL_TLV |
| Social ACK rcvd | | WAIT_INFO |
| Neighbor gone notification rcvd | Stop contact duration count | WAIT_INFO |

## 6. Security Considerations

Currently, dLife does not specify any special security measures. However, as a routing protocol for opportunistic networks, dLife may be a target for various attacks. Such attacks may not be problematic if all nodes in the network can be trusted and are working towards a common goal. If there is such a set of nodes, but there are also malicious nodes, consequent security problems can be solved by introducing an authentication mechanism when two nodes meet, for example using a Pretty Good Privacy (PGP) system. Thus, only nodes that are known to be members of the trusted group of nodes are allowed to participate in the dLife routing. This of course introduces the additional problem of key distribution, which is out-of-scope of this document.Examples of possible vulnerabilities are:

Black Hole Attack

A malicious node sets its social weights for all destinations to a very high value. This has two effects, both causing messages to be drawn towards the black hole, instead of to its correct destination: i) depending on queueing policy, this might lead to premature dropping of the bundle; ii) the social weights reported by the malicious node will affect the computation of the node importance. This could place the malicious use as the center of any communication.

In this case, a node should raise alert if the social weights and node importance that it receives from a new neighbor is much higher than the cumulative moving average of the information received from all previous nodes. This situation can be handle by implementing a trustworthy authentication mechanism for pervasive computing, allowing a node to get extra confidence that a neighbor will handle social weights and node importance in a trustworthy manner.

Identity Spoofing

With identity spoofing, a malicious node claims to be someone else. This could be used to "steal" the data that should be going to a particular node. This will cause these bundles to be removed from the network, reducing the chance that they will reach their real destination.

This can be prevented by using authentication between pervasive nodes.

Bundle Store Overflow

After encountering and receiving the social weights and node importance information from the victim, a malicious node may generate a large number of fake bundles to the destination for which the victim has the social weights. This will cause the victim to fill up its bundle storage, possibly at the expense of other, legitimate, bundles. This problem is transient as the messages will be removed when the victim meets the destination and delivers the messages.

This attack can be prevented by requiring sending nodes to sign all bundles they originate. This will allow intermediate nodes to verify the integrity of the messages before accepting them.

There are some typical vulnerabilities that are not potential problems with dLife such as:

Fake ACKS

In this typical situation a malicious node may issue fake ACKs for all bundles (or only bundles for a certain destination if the attack is targeted at a single node) carried by nodes it meets. The affected bundles will be deleted from the network, greatly reducing their probability of being delivered to the destination.

This situation does not occur with dLife since a node can only send an ACK to bundles that the current carrier decided to forward to it (based on local forwarding policies) and not for bundles that the potential malicious node asked to be forwarded.

## 7. Implementation Experience

The initial implementation of dLife is written in Java for the version 1.4.1 of the Opportunistic Network Emulator (ONE), named Dlife.java [Moreira12a], which implements the RoutingDecisionEngine interface to be used with the DecisionEngineRouter class. This implementation, which can be downloaded from the ONE web site, contains all the major mechanisms described in this document to

ensure proper protocol operation. There are however some parts that are only specified, such as the queuing policies, and other that still need specification, such as the security considerations. The implementation considered nodes with limited storage resources (2 MB) and restricted communication: WiFi and Bluetooth. By running on ONE, the goal of this first implementation wast to enable dLife to be tested in different scalable large pervasive scenarios (some based on real traces such as the one from Cambridge University) with other protocols: PRoPHET and Bubble Rap. The three key performance indicators that were studied were average message delay, probability of message delivery and protocol cost (number of duplicate messages in the network at the time of delivery). Experience and feedback from the implementers on early versions of the protocol have been incorporated into the current version.

A second implementation of dLife was done in Java using the Android API development. The class responsible for routing is known as dLifeRouter.java. This implementation follows a modular design to allow operation over multiple platforms. For that, a dLife library is being developed in Java (version 1.6+). This library comprises different classes that in turn include the components, messages, interfaces, and functionalities specified in this draft. The implemented classes are:

The SocialInformation class
: Comprises the SIG, SW, and IA components which are responsible for keeping track of the different contact between devices, computing the social weight among them and determining their importance.

The DlifeNeighbor class
: Include peer-specific information (e.g., EID_peer, Imp_peer, SocList_peer).

The DlifeTLV class
: Provides all TLVs and methods used by dLife to create and interpret such TLVs.

The main class (dLifeRouter)
: Includes the DM component that works along with other classes (i.e., components) based on the exchanged metadata and computed information to take routing decisions.

The dLifeRouter class includes the interfaces to the DTN architecture/Bundle Agent (e.g., get bundle list, send bundle) and to the Bluetooth Convergence Layer (e.g., neighbor sensing). The implementation of the Bundle Agent, Bluetooth Convergence Layer and dLife are presented as an Opportunistic Networking Module to the DTN architecture.

The DTN architecture/Bundle Agent class was implemented based on RFCs 4838 and 5050. Currently available implementations for Linux and Android devices such as DTN2, IBR-DTN and Bytewalla were studied to obtain enough implementation background.This class provides nodes with DTN core functionality, direct wireless communication, generic routing capabilities, and secure communications. Generic routing functionalities are provided by the implementation of a generic routing class, GenericRouting, which was extended to represent the dLife protocol. Secure communications are supported by the implementation of a security layer, BundleSecurity, implemented according to RFC 6257 - Bundle security Protocol Specification.

To allow direct wireless communication a common communication medium, the Bluetooth Convergence Layer (BCL), was created and implemented allowing direct exchange of bundles through the Bluetooth interface without the need of a structured WiFi network. The BCL class, which is not present in the available DTN architecture implementations, was implemented as general as possible to allow the interfacing between the Bundle Agent and the communication medium regardless of the used routing protocol and operational system running on the the devices. However, the current implementation was first developed for Android devices and thus the BCL takes advantages of some native Bluetooth functionality already implemented in this platform, such as the discovery of neighbors and the possibility of storing information about them. The implemented BCL (BluetoothConvergenceLayer.java) includes the Service Discovery Protocol (SDP) for sensing the medium and the Serial Port Profile (SPP) for data exchange. The BCL class uses the RFCOMM as transport protocol and run on top on the Logical link control and adaptation protocol (L2CAP), which interfaces with the Host Controller Interface (HCI). Additionally, the BCL provides a simple reliable data stream and supports multiple connections as this is expected to happen in the real world.

At the moment the Opportunistic Networking Module is partially functional on Android 2.3.6 Gingerbread (kernel version 2.6.35.7) devices, which are able to exchange information through their Bluetooth interfaces based on the current dLife specification. Currently devices can only manage one simultaneous Bluetooth connection. Actually the code is under revision to be improved in order to support multiple connections, as this is a common situation in urban scenarios. Version 1.0 of the Opportunistic Networking Module for Android devices will be made available for download once concluded.

## 8. Deployment Experience

In order perform implementation tests of the dLife protocol, a DTN testbed was created in the context of the DTN-Amazon project, having SITILabs/University Lusofona and Federal University of Para as current partners. The testbed has currently 10 devices (3 personal computers with Ubuntu 10.10 Maverick, 3 smartphones Android 2.3.6 Gingerbread, 4 wireless routers with OpenWrt 10.03.1).

The testbed was initially used to test three different DTN implementations: IBR-DTN, organized in a modular form, with a focus on embedded systems for easy portability; DTN2, incorporating all components of the DTN architecture, divided into modules such as Convergence Layers, Persistent Store, Bundle Router and more; Bytewalla (version 3, since version 5 was not yet fully functional with sporadic crashes).

This initial set of tests aimed to identify which implementation could be adopted in the DTN-Amazon project.

Both IBR-DTN and DTN2 were tested with two applications that were able to communicate based on two different routing protocols via WiFi interfaces configured in infrastructured mode: i) whisper chat application over PRoPHET (IBR-DTN testbed); and ii) the DTN-Amazon Android security application for surveillance of university campus over Epidemic and PRoPHET routing (DTN2).

Bytewalla was also deployed to allow the understanding of its functionalities. It was installed in Android devices with communication taking place through a wireless router, which could not interpret bundles and was only used to relay information.

Additionally, The DTN2 implementation was also used to analyze the behavior of a network environment with mobility, while the mule (wireless router) were carried by a vehicle to enable the exchange of information between two hosts. There were also attempts to deliver the message at a speed of 60 km/h, the limit considered by the scientific community so that the communication Wi-Fi still works.

As the idea was to exploit physical proximity (key aspect of dLife to determine different levels of social interaction among devices) between DTN-Amazon nodes, this initial set of tests showed that none of these available implementations were enough for the scenario of the project. Thus, they were studied to provide enough implementation knowledge to the project's designers resulting in the Opportunistic Networking Module.

For a proof of concept, initial deployment tests of the stable dLife

implementation over the Opportunistic Networking Module were carried out based on seven Android devices carried by students during their daily routine activities in the Federal University of Para campus for five days. A traffic generator was installed in each device to create a load of 6 messages/hour, towards the other six nodes used in the experience. Node storage was defined at 10MB and message size varied between 1KB and 1MB.

These tests aimed to: i)evaluate the BCL implementation and successfully generate the first contact traces based on it; and ii) test the behavior of dLife in terms of calculating the social weights between nodes and their importances.

Currently, the Opportunistic Networking Module is being finetuned and the next set of tests will analyze the performance of the dLife, when compared to behavior of other social-oblivious opportunistic routing solutions such as Epidemic and PROPHET, based on the following metrics: average message delay, probability of message delivery and the number of duplicate messages in the network at the time of delivery.

## 9. Differences from Previous Version

o Formula to determine the number of elapsed daily samples introduced (Section 2.2)

o Basic forwarding strategy updated (Section 2.3)

o New bundle agent functionality added (Section 2.4.1)

o Neighbor sensing phase updated (Section 3.1)

o Destination policy updated (Section 3.3.3)

o Message formats reviewed with fields being removed/added/updated (Section 4)

o Detailed operation reviewed and state charts updated (Section 5)

o Implementation and Deployment experiences updated (Sections 7 and 8)

o Section 9 included to summarize updates

## 10. References

### 10.1 Normative References

[Moreira12a] Moreira, W., Mendes, P., and Sargento, S., "Opportunistic Routing Based on Daily Routines," in Proceedings of the Sixth IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC 2012), (San Francisco, California, USA), June, 2012.

[Moreira12b] Moreira, W., Souza, M., Mendes, P., and Sargento, S., "Study on the Effect of Network Dynamics on Opportunistic Routing" in Proceedings of the Eleventh International Conference on Ad-Hoc Networks and Wireless (AdHoc Now 2012), (Belgrade, Serbia), July, 2012.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

### 10.2 Informative References

[Chaintreau06] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Impact of human mobility on the design of opportunistic forwarding algorithms," in Proceedings of INFOCOM, (Barcelona, Spain), April, 2006.

[Costa08] P. Costa, C. Mascolo, M. Musolesi, and G. P. Picco, "Socially-aware routing for publish-subscribe in delay-tolerant mobile ad hoc networks," Selected Areas in Communications, IEEE Journal on, vol. 26, pp. 748- 760, June, 2008.

[Daly07] E. M. Daly and M. Haahr, "Social network analysis for routing in disconnected delay-tolerant manets," in Proceedings of ACM MobiHoc, (Montreal, Canada), September, 2007.

[Eagle09] N. Eagle and A. Pentland, "Eigenbehaviors: identifying structure in routine," Behavioral Ecology and Sociobiology, vol. 63, pp. 1057-1066, May, 2009.

[Hossmann10] T. Hossmann, T. Spyropoulos, and F. Legendre, "Know thy neighbor: Towards optimal mapping of contacts to social graphs for dtn routing," in Proceedings of IEEE INFOCOM, (San Diego, USA), March, 2010.

[Hui07] P. Hui and J. Crowcroft, "How small labels create big improvements," in Proceedings of IEEE PERCOM Workshops, (White Plains, USA), March, 2007.

[Hui11] P. Hui, J. Crowcroft, and E. Yoneki, "Bubble rap: social-based forward- ing in delay tolerant networks," Mobile Computing, IEEE Transactions on, vol. 10, pp. 1576-1589, November, 2011.

[I-D.irtf-dtnrg-tcp-clayer] Demmer, M. and J. Ott, "Delay Tolerant Networking TCP Convergence Layer Protocol", draft-irtf-dtnrg-tcp-clayer-02 (work in progress), November 2008.

[I-D.irtf-dtnrg-udp-clayer] H. Kruse, S. Ostermann, "UDP Convergence Layers for the DTN Bundle and LTP Protocols", draft-irtf-dtnrg-udp-clayer-00 (work in progress), November 2008.

[Lindgren04] A. Lindgren, A. Doria, and O. Schelen, "Probabilistic routing in intermittently connected networks," in Service Assurance with Partial and Intermittent Resources, vol. 3126 of Lecture Notes in Computer Science, pp. 239--254, Springer Berlin / Heidelberg, 2004.

[Lindgren06] Lindgren, A. and K. Phanse, "Evaluation of Queueing Policies and Forwarding Strategies for Routing in Intermittently Connected Networks", Proceedings of COMSWARE 2006 , January 2006.

[Moreira11] W. Moreira and P. Mendes, "Survey on Opportunistic Routing for Delay/Disruption Tolerant Networks ," Tech. Rep. SITI-TR-11-02, SITI, University Lusofona, February 2011.

[Moreira_12c] Waldir Moreira, Paulo Mendes, Susana Sargento, "Assessment Model for Opportunistic Routing", IEEE Latin America Transactions, Vol 10 Issue 3 April 2012

[Mtibaa10] A. Mtibaa, M. May, M. Ammar, and C. Diot, "Peoplerank: Combining social and contact information for opportunistic forwarding," in Proceedings of INFOCOM, (San Diego, USA), March, 2010.

[Nelson09] S. Nelson, M. Bakht, and R. Kravets, "Encounter-based

routing in DTNs," in Proceedings of INFOCOM, (Rio de Janeiro, Brazil), April, 2009.

[RFC6257] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", RFC 6257, May 2011.

[Song07] L. Song and D. F. Kotz, "Evaluating opportunistic routing protocols with large realistic contact traces," in Proceedings of ACM MobiCom CHANTS, (Montreal, Canada), September, 2007.

[Vahdat00] Vahdat, A. and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks", Duke University Technical Report CS-200006, April 2000..in 3

Authors' Addresses

Waldir Moreira
SITILabs, Universidade Lusofona
Campo Grande, 376
1749-024 Lisboa
Portugal
Phone:
Email: waldir.junior@ulusofona.pt
URI: http://siti2.ulusofona.pt/~wjunior

Paulo Mendes
SITILabs, Universidade Lusofona
Campo Grande, 376
1749-024 Lisboa
Portugal
Phone:
Email: paulo.mendes@ulusofona.pt
URI: http://siti.ulusofona.pt/~pmendes

Ronedo Ferreira
ITEC, Universidade Federal do Para
Rua Augusto Correa, 01, Guama
66075-110 Belem-PA
Brasil
Phone:
Email: ronedo@aitinet.com
URI:

Douglas Cirqueira
ITEC, Universidade Federal do Para
Rua Augusto Correa, 01, Guama

66075-110 Belem-PA
Brasil
Phone:
Email: douglas.cirqueira@gmail.com
URI:

Eduardo Cerqueira
ITEC, Universidade Federal do Para
Rua Augusto Correa, 01, Guama
66075-110 Belem-PA
Brasil
Phone:
Email: cerqueira@ufpa.br
URI: http://www.gercom.ufpa.br/eduardo/