Internet Research Task Force (IRTF)
Request for Comments: 6693
Category: Experimental
ISSN: 2070-1721

A. Lindgren
SICS
A. Doria
Technicalities
E. Davies
Folly Consulting
S. Grasic
Lulea University of Technology

August 2012

Probabilistic Routing Protocol for Intermittently Connected Networks

Abstract

This document is a product of the Delay Tolerant Networking Research Group and has been reviewed by that group. No objections to its publication as an RFC were raised.

This document defines PRoPHET, a Probabilistic Routing Protocol using History of Encounters and Transitivity. PRoPHET is a variant of the epidemic routing protocol for intermittently connected networks that operates by pruning the epidemic distribution tree to minimize resource usage while still attempting to achieve the best-case routing capabilities of epidemic routing. It is intended for use in sparse mesh networks where there is no guarantee that a fully connected path between the source and destination exists at any time, rendering traditional routing protocols unable to deliver messages between hosts. These networks are examples of networks where there is a disparity between the latency requirements of applications and the capabilities of the underlying network (networks often referred to as delay and disruption tolerant). The document presents an architectural overview followed by the protocol specification.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Research Task Force (IRTF). The IRTF publishes the results of Internet-related research and development activities. These results might not be suitable for deployment. This RFC represents the consensus of the Delay Tolerant Networking Research Group of the Internet Research

Task Force (IRTF). Documents approved for publication by the IRSG are not a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6693.

Table of Contents

1. Introduction

The Probabilistic Routing Protocol using History of Encounters and Transitivity (PRoPHET) algorithm enables communication between participating nodes wishing to communicate in an intermittently connected network where at least some of the nodes are mobile.

One of the most basic requirements for "traditional" (IP) networking is that there must exist a fully connected path between communication endpoints for the duration of a communication session in order for communication to be possible. There are, however, a number of scenarios where connectivity is intermittent so that this is not the case (thus rendering the end-to-end use of traditional networking protocols impossible), but where it still is desirable to allow communication between nodes.

Consider a network of mobile nodes using wireless communication with a limited range that is less than the typical excursion distances over which the nodes travel. Communication between a pair of nodes at a particular instant is only possible when the distance between the nodes is less than the range of the wireless communication. This means that, even if messages are forwarded through other nodes acting as intermediate routes, there is no guarantee of finding a viable continuous path when it is needed to transmit a message.

One way to enable communication in such scenarios is by allowing messages to be buffered at intermediate nodes for a longer time than normally occurs in the queues of conventional routers (cf. Delay-Tolerant Networking [RFC4838]). It would then be possible to exploit the mobility of a subset of the nodes to bring messages closer to their destination by transferring them to other nodes as they meet. Figure 1 shows how the mobility of nodes in such a scenario can be used to eventually deliver a message to its destination. In this figure, the four sub-figures (a) - (d) represent the physical positions of four nodes (A, B, C, and D) at four time instants, increasing from (a) to (d). The outline around each letter represents the range of the radio communication used for communication by the nodes: communication is only possible when the ranges overlap. At the start time, node A has a message -- indicated by an asterisk (*) next to that node -- to be delivered to node D, but there does not exist a path between nodes A and D because of the limited range of available wireless connections. As shown in sub-figures (a) - (d), the mobility of the nodes allows the message to first be transferred to node B, then to node C, and when finally node C moves within range of node D, it can deliver the message to its final destination. This technique is known as "transitive networking".

Mobility and contact patterns in real application scenarios are likely to be non-random, but rather be predictable, based on the underlying activities of the higher-level application (this could, for example, stem from human mobility having regular traffic patterns based on repeating behavioral patterns (e.g., going to work or the market and returning home) and social interactions, or from any number of other node mobility situations where a proportion of nodes are mobile and move in ways that are not completely random over time but have a degree of predictability over time). This means that if a node has visited a location or been in contact with a certain node several times before, it is likely that it will visit that location or meet that node again.

PRoPHET can also be used in some networks where such mobility as described above does not take place. Predictable patterns in node contacts can also occur among static nodes where varying radio conditions or power-saving sleeping schedules cause connection between nodes to be intermittent.

In previously discussed mechanisms to enable communication in intermittently connected networks, such as Epidemic Routing [vahdat_00], very general approaches have been taken to the problem at hand. In an environment where buffer space and bandwidth are infinite, epidemic routing will give an optimal solution to the problem of routing in an intermittently connected network with regard to message delivery ratio and latency. However, in most cases, neither bandwidth nor buffer space is infinite, but instead they are rather scarce resources, especially in the case of sensor networks.

PRoPHET is fundamentally an epidemic protocol with strict pruning. An epidemic protocol works by transferring its data to each and every node it meets. As data is passed from node to node, it is eventually passed to all nodes, including the target node. One of the advantages of an epidemic protocol is that by trying every path, it is guaranteed to try the best path. One of the disadvantages of an epidemic protocol is the extensive use of resources with every node needing to carry every packet and the associated transmission costs. PRoPHET's goal is to gain the advantages of an epidemic protocol without paying the price in storage and communication resources incurred by the basic epidemic protocol. That is, PRoPHET offers an alternative to basic epidemic routing, with lower demands on buffer space and bandwidth, with equal or better performance in cases where those resources are limited, and without loss of generality in scenarios where it is suitable to use PRoPHET.

In a situation where PRoPHET is applicable, the patterns are expected to have a characteristic time (such as the expected time between encounters between mobile stations) that is in turn related to the expected time that traffic will take to reach its destination in the part of the network that is using PRoPHET. This characteristic time provides guidance for configuration of the PRoPHET protocol in a network. When appropriately configured, the PRoPHET protocol effectively builds a local model of the expected patterns in the network that can be used to optimize the usage of resources by reducing the amount of traffic sent to nodes that are unlikely to lead to eventual delivery of the traffic to its destination.

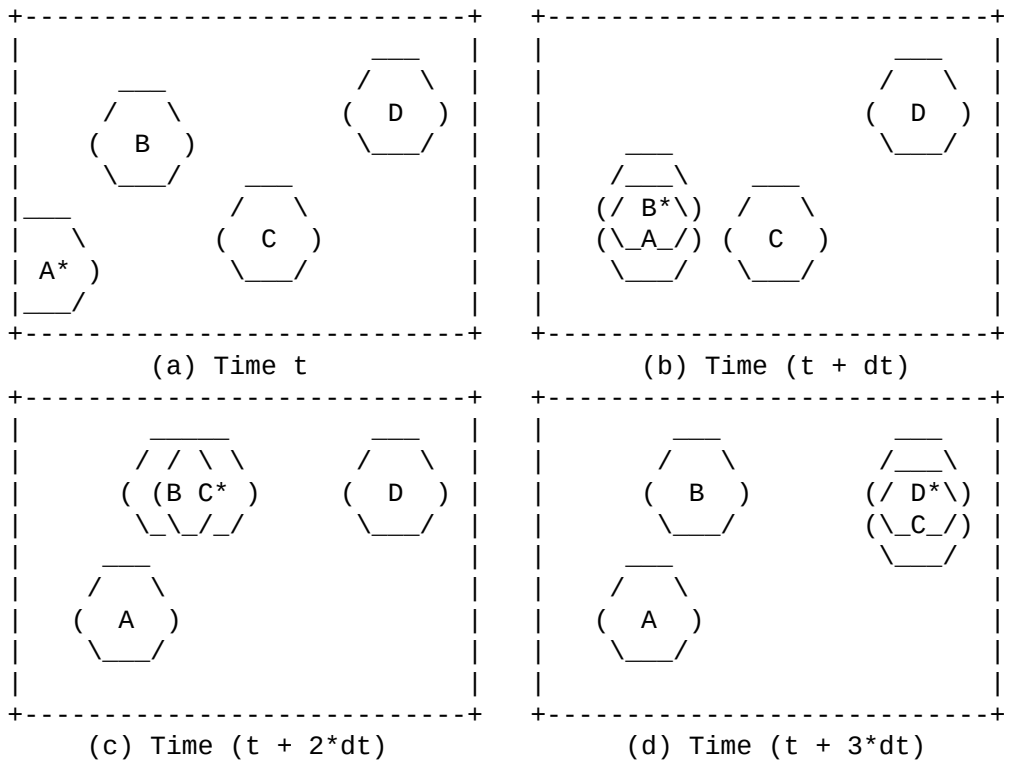


Figure 1: Example of transitive communication

This document presents a framework for probabilistic routing in intermittently connected networks, using an assumption of non-random mobility of nodes to improve the delivery rate of messages while keeping buffer usage and communication overhead at a low level. First, a probabilistic metric called delivery predictability is defined. The document then goes on to define a probabilistic routing protocol using this metric.

1.1. Relation to the Delay-Tolerant Networking Architecture

The Delay-Tolerant Networking (DTN) architecture [RFC4838] defines an architecture for communication in environments where traditional communication protocols cannot be used due to excessive delays, link outages, and other extreme conditions. The intermittently connected networks considered here are a subset of those covered by the DTN architecture. The DTN architecture defines routes to be computed based on a collection of "contacts" indicating the start time, duration, endpoints, forwarding capacity, and latency of a link in the topology graph. These contacts may be deterministic or may be

derived from estimates. The architecture defines some different types of intermittent contacts. The ones called "opportunistic" and "predicted" are the ones addressed by this protocol.

Opportunistic contacts are those that are not scheduled, but rather present themselves unexpectedly and frequently arise due to node mobility. Predicted contacts are like opportunistic contacts, but, based on some information, it might be possible to draw some statistical conclusion as to whether or not a contact will be present soon.

The DTN architecture also 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 also provides end-to-end addressing and acknowledgments. PRoPHET is specifically intended to provide routing services in a network environment that uses bundles as its data transfer mechanism but could be also be used in other intermittent environments.

1.2. Applicability of the Protocol

The PRoPHET routing protocol is mainly targeted at situations where at least some of the nodes are mobile in a way that creates connectivity patterns that are not completely random over time but have a degree of predictability. Such connectivity patterns can also occur in networks where nodes switch off radios to preserve power. Human mobility patterns (often containing daily or weekly periodic activities) provide one such example where PRoPHET is expected to be applicable, but the applicability is not limited to scenarios including humans.

In order for PRoPHET to benefit from such predictability in the contact patterns between nodes, it is expected that the network exist under similar circumstances over a longer timescale (in terms of node encounters) so that the predictability can be accurately estimated.

The PRoPHET protocol expects nodes to be able to establish a local TCP link in order to exchange the information needed by the PRoPHET protocol. Protocol signaling is done out-of-band over this TCP link, without involving the bundle protocol agent [RFC5050]. However, the PRoPHET protocol is expected to interact with the bundle protocol agent to retrieve information about available bundles as well as to request that a bundle be sent to another node (it is expected that the associated bundle protocol agents are then able to establish a link (probably over the TCP convergence layer [CLAYER]) to perform this bundle transfer).

TCP provides a reliable bidirectional channel between two peers and guarantees in-order delivery of transmitted data. When using TCP, the guarantee of reliable, in-order delivery allows information exchanges of each category of information to be distributed across several messages without requiring the PRoPHET protocol layer to be concerned that all messages have been received before starting the exchange of the next category of information. At most, the last message of the category needs to be marked as such. This allows the receiver to process earlier messages while waiting for additional information and allows implementations to limit the size of messages so that IP fragmentation will be avoided and memory usage can be optimized if necessary. However, implementations MAY choose to build a single message for each category of information that is as large as necessary and rely on TCP to segment the message.

While PRoPHET is currently defined to run over TCP, in future versions the information exchange may take place over other transport protocols, and these may not provide message segmentation or reliable, in-order delivery. The simple message division used with TCP MUST NOT be used when the underlying transport does not offer reliable, in-order delivery, as it would be impossible to verify that all the messages had arrived. Hence, the capability is provided to segment protocol messages into submessages directly in the PRoPHET layer. Submessages are provided with sequence numbers, and this, together with a capability for positive acknowledgements, would allow PRoPHET to operate over an unreliable protocol such as UDP or potentially directly over IP.

Since TCP offers reliable delivery, it is RECOMMENDED that the positive acknowledgment capability is not used when PRoPHET is run over a TCP transport or similar protocol. When running over TCP, implementations MAY safely ignore positive acknowledgments.

Whatever transport protocol is used, PRoPHET expects to use a bidirectional link for the information exchange; this allows for the 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.

In a large Delay- and Disruption-Tolerant Network (DTN), network conditions may vary widely, and in different parts of the network, different routing protocols may be appropriate. In this specification, we consider routing within a single "PRoPHET zone", which is a set of nodes among which messages are routed using PRoPHET. In many cases, a PRoPHET zone will not span the entire DTN, but there will be other parts of the network with other characteristics that run other routing protocols. To handle this, there may be nodes within the zone that act as gateways to other

nodes that are the destinations for bundles generated within the zone or that insert bundles into the zone. Thus, PRoPHET is not necessarily used end-to-end, but only within regions of the network where its use is appropriate.

1.3. PRoPHET as Compared to Regular Routing Protocols

While PRoPHET uses a mechanism for pruning the epidemic forwarding tree that is similar to the mechanism used in metric-based vector routing protocols (where the metric might be distance or cost), it should not be confused with a metric vector protocol.

In a traditional metric-based vector routing protocol, the information passed from node to node is used to create a single non-looping path from source to destination that is optimal given the metric used. The path consists of a set of directed edges selected from the complete graph of communications links between the network nodes.

In PRoPHET, that information is used to prune the epidemic tree of paths by removing paths that look less likely to provide an effective route for delivery of data to its intended destination. One of the effects of this difference is that the regular notions of split horizon, as described in [RFC1058], do not apply to PRoPHET. The purpose of split horizon is to prevent a distance vector protocol from ever passing a packet back to the node that sent it the packet because it is well known that the source does not lie in that direction as determined when the directed path was computed.

In an epidemic protocol, where that previous system already has the data, the notion of passing the data back to the node is redundant: the protocol can readily determine that such a transfer is not required. Further, given the mobility and constant churn of encounters possible in a DTN that is dominated by opportunistic encounters, it is quite possible that, on a future encounter, the node might have become a better option for reaching the destination. Such a later encounter may require a re-transfer of the data if resource constraints have resulted in the data being deleted from the original carrier between the encounters.

The logic of metric routing protocols does not map directly onto the family of epidemic protocols. In particular, it is inappropriate to try to assess such protocols against the criteria used to assess conventional routing protocols such as the metric vector protocols; this is not to say that the family of epidemic protocols do not have weaknesses but they have to be considered independently of traditional protocols.

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. Architecture

2.1. PRoPHET

This section presents an overview of the main architecture of PRoPHET, a Probabilistic Routing Protocol using History of Encounters and Transitivity. The protocol leverages the observations made on the non-randomness of mobility patterns present in many application scenarios to improve routing performance. Instead of doing blind epidemic replication of bundles through the network as previous protocols have done, it applies "probabilistic routing".

To accomplish this, a metric called "delivery predictability", $0 <= P_(A,B) <= 1$, is established at every node A for each known destination B. This metric is calculated so that a node with a higher value for a certain destination is estimated to be a better candidate for delivering a bundle to that destination (i.e., if $P_(A,B)>P_(C,B)$, bundles for destination B are preferable to forward to A rather than C). It is later used when making forwarding decisions. As routes in a DTN are likely to be asymmetric, the calculation of the delivery predictability reflects this, and $P_(A,B)$ may be different from $P_(B,A)$.

The delivery predictability values in each node evolve over time both as a result of decay of the metrics between encounters between nodes and due to changes resulting from encounters when metric information for the encountered node is updated to reflect the encounter and metric information about other nodes is exchanged.

When two PRoPHET nodes have a communication opportunity, they initially enter a two-part Information Exchange Phase (IEP). In the first part of the exchange, the delivery predictabilities for all destinations known by each node are shared with the encountered node. The exchanged information is used by each node to update the internal delivery predictability vector as described below. After that, the nodes exchange information (including destination and size) about the bundles each node carries, and the information is used in conjunction with the updated delivery predictabilities to decide which bundles to request to be forwarded from the other node based on the forwarding strategy used (as discussed in Section 2.1.4). The forwarding of bundles is carried out in the latter part of the Information Exchange Phase.

2.1.1. Characteristic Time Interval

When an application scenario makes PRoPHET applicable, the mobility pattern will exhibit a characteristic time interval that reflects the distribution of time intervals between encounters between nodes. The evolution of the delivery predictabilities, which reflects this mobility pattern, should reflect this same characteristic time interval. Accordingly, the parameters used in the equations that specify the evolution of delivery predictability (see Section 2.1.2) need to be configured appropriately so that the evolution reflects a model of the mobility pattern.

2.1.2. Delivery Predictability Calculation

As stated above, PRoPHET relies on calculating a metric based on the probability of encountering a certain node, and using that to support the decision of whether or not to forward a bundle to a certain node. This section describes the operations performed on the metrics stored in a node when it encounters another node and a communications opportunity arises. In the operations described by the equations that follow, the updates are being performed by node A, P_(A,B) is the delivery predictability value that node A will have stored for the destination B after the encounter, and P_(A,B)_old is the corresponding value that was stored before the encounter. If no delivery predictability value is stored for a particular destination B, P_(A,B) is considered to be zero.

As a special case, the metric value for a node itself is always defined to be 1 (i.e., P_(A,A)=1).

The equations use a number of parameters that can be selected to match the characteristics of the mobility pattern in the PRoPHET zone where the node is located (see Section 2.1.1). Recommended settings for the various parameters are given in Section 3.3. The impact on the evolution of delivery predictabilities if encountering nodes have different parameter setting is discussed in Section 2.1.2.1.

The calculation of the updates to the delivery predictabilities during an encounter has three parts.

When two nodes meet, the first thing they do is to update the delivery predictability for each other, so that nodes that are often encountered have a high delivery predictability. If node B has not met node A for a long time or has never met node B, such that P_(A,B) < P_first_threshold, then P_(A,B) should be set to P_encounter_first. Because PRoPHET generally has no prior knowledge about whether this is an encounter that will be repeated relatively frequently or one that will be a rare event, P_encounter_first SHOULD

be set to 0.5 unless the node has extra information obtained other than through the PRoPHET protocol about the likelihood of future encounters. Otherwise, P_(A,B) should be calculated as shown in Equation 1, where 0 <= P_encounter <= 1 is a scaling factor setting the rate at which the predictability increases on encounters after the first, and delta is a small positive number that effectively sets an upper bound for P_(A,B). The limit is set so that predictabilities between different nodes stay strictly less than 1. The value of delta should normally be very small (e.g., 0.01) so as not to significantly restrict the range of available predictabilities, but it can be chosen to make calculations efficient where this is important.

P_(A,B) =
P_(A,B)_old + (1 - delta - P_(A,B)_old) * P_encounter (Eq. 1)

There are practical circumstances where an encounter that is logically a single encounter in terms of the proximity of the node hardware and/or from the point of view of the human users of the nodes results in several communication opportunities closely spaced in time. For example, mobile nodes communicating with each other using Wi-Fi ad hoc mode may produce apparent multiple encounters with a short interval between them but these are frequently due to artifacts of the underlying physical network when using wireless connections, where transmission problems or small changes in location may result in repeated reconnections. In this case, it would be inappropriate to increase the delivery predictability by the same amount for each opportunity as it would be increased when encounters occur at longer intervals in the normal mobility pattern.

In order to reduce the distortion of the delivery predictability in these circumstances, P_encounter is a function of the interval since the last encounter resulted in an update of the delivery predictabilities. The form of the function is as shown in Figure 2.

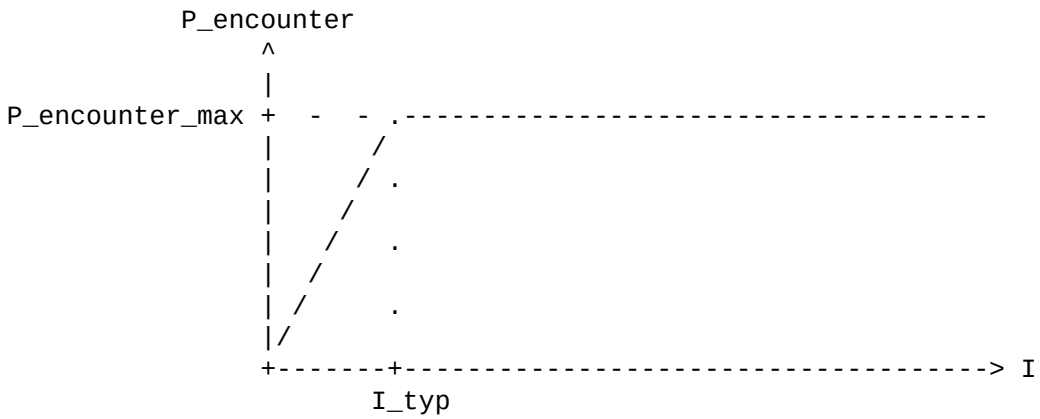


Figure 2: P_encounter as function of time interval, I, between updates

The form of the function is chosen so that both the increase of P_(A,B) resulting from Equation 1 and the decrease that results from Equation 2 are related to the interval between updates for short intervals. For intervals longer than the "typical" time (I_typ) between encounters, P_encounter is set to a fixed value P_encounter_max. The break point reflects the transition between the "normal" communication opportunity regime (where opportunities result from the overall mobility pattern) and the closely spaced opportunities that result from what are effectively local artifacts of the wireless technology used to deliver those opportunities.

P_encounter_max is chosen so that the increment in P_(A,B) provided by Equation 1 significantly exceeds the decay of the delivery predictability over the typical interval between encounters resulting from Equation 2.

Making P_encounter dependent on the interval time also avoids inappropriate extra increments of P_(A,B) in situations where node A is in communication with several other nodes simultaneously. In this case, updates from each of the communicating nodes have to be distributed to the other nodes, possibly leading to several updates being carried out in a short period. This situation is discussed in more detail in Section 3.2.2.

If a pair of nodes do not encounter each other during an interval, they are less likely to be good forwarders of bundles to each other, thus the delivery predictability values must age, being reduced in the process. The second part of the updates of the metric values is application of the aging equation shown in Equation 2, where $0 \leq \gamma \leq 1$ is the aging constant, and K is the number of time units that have elapsed since the last time the metric was aged. The

time unit used can differ and should be defined based on the application and the expected delays in the targeted network.

P_(A,B) = P_(A,B)_old * gamma^K (Eq. 2)

The delivery predictabilities are aged according to Equation 2 before being passed to an encountered node so that they reflect the time that has passed since the node had its last encounter with any other node. The results of the aging process are sent to the encountered peer for use in the next stage of the process. The aged results received from node B in node A are referenced as P_(B,x)_recv.

The delivery predictability also has a transitive property that is based on the observation that if node A frequently encounters node B, and node B frequently encounters node C, then node C probably is a good node to which to forward bundles destined for node A. Equation 3 shows how this transitivity affects the delivery predictability, where 0 <= beta <= 1 is a scaling constant that controls how large an impact the transitivity should have on the delivery predictability.

P_(A,C) = MAX(P_(A,C)_old, P_(A,B) * P_(B,C)_recv * beta) (Eq. 3)

Node A uses Equation 3 and the metric values received from the encountered node B (e.g., P_(B,C)_recv) in the third part of updating the metric values stored in node A.

2.1.2.1. Impact of Encounters between Nodes with Different Parameter Settings

The various parameters used in the three equations described in Section 2.1.2 are set independently in each node, and it is therefore possible that encounters may take place between nodes that have been configured with different values of the parameters. This section considers whether this could be problematic for the operation of PRoPHET in that zone.

It is desirable that all the nodes operating in a PRoPHET zone should use closely matched values of the parameters and that the parameters should be set to values that are appropriate for the operating zone. More details of how to select appropriate values are given in Section 3.3. Using closely matched values means that delivery predictabilities will evolve in the same way in each node, leading to consistent decision making about the bundles that should be exchanged during encounters.

Before going on to consider the impact of reasonable but different settings, it should be noted that malicious nodes can use inappropriate settings of the parameters to disrupt delivery of bundles in a PRoPHET zone as described in Section 6.

Firstly and importantly, use of different, but legitimate, settings in encountering nodes will not cause problems in the protocol itself. Apart from P_encounter_first, the other parameters control the rate of change of the metric values or limit the range of valid values that will be stored in a node. None of the calculations in a node will be invalidated or result in illegal values if the metric values received from another node were calculated using different parameters. Furthermore, the protocol is designed so that it is not possible to carry delivery predictabilities outside the permissible range of 0 to 1.

A node MAY consider setting received values greater than (1 - delta) to (1 - delta) if this would simplify operations. However, there are some special situations where it may be appropriate for the delivery predictability for another node to be 1. For example, if a DTN using PRoPHET has multiple gateways to the continuously connected Internet, the delivery predictability seen from PRoPHET in one gateway for the other gateway nodes can be taken as 1 since they are permanently connected through the Internet. This would allow traffic to be forwarded into the DTN through the most advantageous gateway even if it initially arrives at another gateway.

Simulation work indicates that the update calculations are quite stable in the face of changes to the rate parameters, so that minor discrepancies will not have a major impact on the performance of the protocol. The protocol is explicitly designed to deal with situations where there are random factors in the opportunistic nature of node encounters, and this randomness dominates over the discrepancies in the parameters.

More major discrepancies may lead to suboptimal behavior of the protocol, as certain paths might be more preferred or more deprecated inappropriately. However, since the protocol overall is epidemic in nature, this would not generally lead to non-delivery of bundles, as they would also be passed to other nodes and would still be delivered, though possibly not on the optimal path.

2.1.3. Optional Delivery Predictability Optimizations

2.1.3.1. Smoothing

To give the delivery predictability a smoother rate of change, a node MAY apply one of the following methods:

1. Keep a list of NUM_P values for each destination instead of only a single value. (The recommended value is 4, which has been shown in simulations to give a good trade-off between smoothness and rate of response to changes.) The list is held in order of acquisition. When a delivery predictability is updated, the value at the "newest" position in the list is used as input to the equations in Section 2.1.2. The oldest value in the list is then discarded and the new value is written in the "newest" position of the list. When a delivery predictability value is needed (either for sending to a peering PRoPHET node, or for making a forwarding decision), the average of the values in the list is calculated, and that value is then used. If less than NUM_P values have been entered into the list, only the positions that have been filled should be used for the averaging.

2. In addition to keeping the delivery predictability as described in Section 2.1.2, a node MAY also keep an exponential weighted moving average (EWMA) of the delivery predictability. The EWMA is then used to make forwarding decisions and to report to peering nodes, but the value calculated according to Section 2.1.2 is still used as input to the calculations of new delivery predictabilities. The EWMA is calculated according to Equation 4, where 0 <= alpha <= 1 is the weight of the most current value.

P_ewma = P_ewma_old * (1 - alpha) + P * alpha (Eq. 4)

The appropriate choice of alpha may vary depending on application scenario circumstances. Unless prior knowledge of the scenario is available, it is suggested that alpha is set to 0.5.

2.1.3.2. Removal of Low Delivery Predictabilities

To reduce the data to be transferred between two nodes, a node MAY treat delivery predictabilities smaller than P_first_threshold, where P_first_threshold is a small number, as if they were zero, and thus they do not need to be stored or included in the list sent during the Information Exchange Phase. If this optimization is used, care must be taken to select P_first_threshold to be smaller than delivery predictability values normally present in the network for destinations for which this node is a forwarder. It is possible that

P_first_threshold could be calculated based on delivery predictability ranges and the amount they change historically, but this has not been investigated yet.

2.1.4. Forwarding Strategies and Queueing Policies

In traditional routing protocols, choosing where to forward a message is usually a simple task; the message is sent to the neighbor that has the path to the destination with the lowest cost (often the shortest path). Normally, the message is also sent to only a single node since the reliability of paths is relatively high. However, in the settings we envision here, things are radically different. The first possibility that must be considered when a bundle arrives at a node is that there might not be a path to the destination available, so the node has to buffer the bundle, and upon each encounter with another node, the decision must be made whether or not to transfer a particular bundle. Furthermore, having duplicates of messages (on different nodes, as the bundle offer/request mechanism described in Section 4.3.5 ensures that a node does not receive a bundle it already carries) may also be sensible, as forwarding a bundle to multiple nodes can increase the delivery probability of that bundle.

Unfortunately, these decisions are not trivial to make. In some cases, it might be sensible to select a fixed threshold and only give a bundle to nodes that have a delivery predictability over that threshold for the destination of the bundle. On the other hand, when encountering a node with a low delivery predictability, it is not certain that a node with a higher metric will be encountered within a reasonable time. Thus, there can also be situations where we might want to be less strict in deciding who to give bundles to. Furthermore, there is the problem of deciding how many nodes to give a certain bundle to. Distributing a bundle to a large number of nodes will of course increase the probability of delivering that particular bundle to its destination, but this comes at the cost of consuming more system resources for bundle storage and possibly reducing the probability of other bundles being delivered. On the other hand, giving a bundle to only a few nodes (maybe even just a single node) will use less system resources, but the probability of delivering a bundle is lower, and the delay incurred is high.

When resources are constrained, nodes may suffer from storage shortage, and may have to drop bundles before they have been delivered to their destinations. They may also wish to consider the length of bundles being offered by an encountered node before accepting transfer of the bundle in order to avoid the need to drop the new bundle immediately or to ensure that there is adequate space to hold the bundle offered, which might require other bundles to be dropped. As with the decision as to whether or not to forward a

bundle, deciding which bundles to accept and/or drop to still maintain good performance might require different policies in different scenarios.

Nodes MAY define their own forwarding strategies and queueing policies that take into account the special conditions applicable to the nodes, and local resource constraints. Some default strategies and policies that should be suitable for most normal operations are defined in Section 3.6 and Section 3.7.

2.2. Bundle Protocol Agent to Routing Agent Interface

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

This specification defines the details of the PRoPHET routing agent, but the interface defines a more general interface that is also applicable to alternative routing protocols.

To enable the PRoPHET routing agent to operate properly, it must be aware of the bundles stored at the node, and it must also be able to tell the bundle protocol agent of that node to send a bundle to a peering node. Therefore, the bundle protocol 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
: Makes the bundle protocol agent send a specified bundle.

Accept Bundle
: Gives the bundle protocol agent a new bundle to store.

Bundle Delivered
: Tells the bundle protocol agent that a bundle was delivered to its destination.

Drop Bundle Advice
: Advises the bundle protocol agent that a specified bundle should not be offered for forwarding in future and may be dropped by the bundle protocol agent if appropriate.

Route Import
: Can be used by a gateway node in a PRoPHET zone to import reachability information about endpoint IDs (EIDs) that are external to the PRoPHET zone. Translation functions dependent on the external routing protocol will be used to set the appropriate delivery predictabilities for imported destinations as described in Section 2.3.

Route Export
: Can be used by a gateway node in a PRoPHET zone to export reachability information (destination EIDs and corresponding delivery predictabilities) for use by routing protocols in other parts of the DTN.

Implementation Note: Depending on the distribution of functions in a complete bundle protocol agent supporting PRoPHET, reception and delivery of bundles may not be carried out directly by the PRoPHET module. In this case, PRoPHET can inform the bundle protocol agent about bundles that have been requested from communicating nodes. Then, the Accept Bundle and Bundle Delivered functions can be implemented as notifications of the PRoPHET module when the relevant bundles arrive at the node or are delivered to local applications.

2.3. PRoPHET Zone Gateways

PRoPHET is designed to handle routing primarily within a "PRoPHET zone", i.e., a set of nodes that all implement the PRoPHET routing scheme. However, since we recognize that a PRoPHET routing zone is unlikely to encompass an entire DTN, there may be nodes within the zone that act as gateways to other nodes that are the destinations for bundles generated within the zone or that insert bundles into the zone.

PRoPHET MAY elect to export and import routes across a bundle protocol agent interface. The delivery predictability to use for routes that are imported depends on the routing protocol used to manage those routes. If a translation function between the external routing protocol and PRoPHET exists, it SHOULD be used to set the delivery predictability. If no such translation function exists, the delivery predictability SHOULD be set to 1. For those routes that are exported, the current delivery predictability will be exported with the route.

2.4. Lower-Layer Requirements and Interface

PRoPHET can be run on a large number of underlying networking technologies. To accommodate its operation on all kinds of lower layers, it requires the lower layers to provide the following functionality and interfaces.

Neighbor discovery and maintenance
: A PRoPHET node needs to know the identity of its neighbors and when new neighbors appear and old neighbors disappear. Some wireless networking technologies might already contain mechanisms for detecting neighbors and maintaining this state. To avoid redundancies and inefficiencies, neighbor discovery is thus not included as a part of PRoPHET, but PRoPHET relies on such a mechanism in lower layers. The lower layers MUST provide the two functions listed below. If the underlying networking technology does not support such services, a simple neighbor discovery scheme using local broadcasts of beacon messages could be run in between PRoPHET and the underlying layer. An example of a simple neighbor discovery mechanism that could be used is in Appendix B.

  New Neighbor
  : Signals to the PRoPHET agent that a new node has become a neighbor. A neighbor is defined here as another node that is currently within communication range of the wireless networking technology in use. The PRoPHET agent should now start the Hello procedure as described in Section 5.2.

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

Local Address
: An address used by the underlying communication layer (e.g., an IP or Media Access Control (MAC) address) that identifies the sender address of the current message. This address must be unique among the nodes that can currently communicate and is only used in conjunction with an Instance Number to identify a communicating pair of nodes as described in Section 4.1. This address and its format is dependent on the communication layer that is being used by the PRoPHET layer.

3. Protocol Overview

The PRoPHET protocol involves two principal phases:

- o becoming aware of new neighbors that implement the protocol and establishing a point-to-point connection between each pair of encountering nodes, and
- o using the connection for information exchange needed to establish PRoPHET routing and to exchange bundles.

3.1. Neighbor Awareness

Since the operation of the protocol is dependent on the encounters of nodes running PRoPHET, the nodes must be able to detect when a new neighbor is present. The protocol may be run on several different networking technologies, and as some of them might already have methods available for detecting neighbors, PRoPHET does not include a mechanism for neighbor discovery. Instead, it requires the underlying layer to provide a mechanism to notify the protocol of when neighbors appear and disappear as described in Section 2.4.

When a new neighbor has been detected, the protocol starts to set up a link with that node through the Hello message exchange as described in Section 5.2. The Hello message exchange allows for negotiation of capabilities between neighbors. At present, the only capability is a request that the offering node should or should not include bundle payload lengths with all offered bundles rather than just for fragments. Once the link has been set up, the protocol may continue to the Information Exchange Phase (see Section 3.2). Once this has been completed, the nodes will normally recalculate the delivery predictabilities using the equations and mechanisms described in Sections 2.1.2 and 2.1.3.

As described in Section 2.1.2, there are some circumstances in which a single logical encounter may result in several actual communication opportunities. To avoid the delivery predictability of the encountered node being increased excessively under these circumstances, the value of P_encounter is made dependent on the interval time between delivery predictability updates when the interval is less than the typical interval between encounters, but it is a constant for longer intervals.

In order to make use of this time dependence, PRoPHET maintains a list of recently encountered nodes identified by the Endpoint Identifier (EID) that the node uses to identify the communication session and containing the start time of the last communication session with that node. The size of this list is controlled because

nodes that are not in contact and that started their last connection more than a time I_typ before the present can be dropped from the list. It also maintains a record of the time at which the decay function (Equation 2) was last applied to the delivery predictabilities in the node.

3.2. Information Exchange Phase

The Information Exchange Phase involves two parts:

- establishing the Router Information Base (RIB Exchange Sub-Phase), and
- exchanging bundles using this information (Bundle Passing Sub-Phase).

Four types of information are exchanged during this process:

- Routing Information Base Dictionary (RIB Dictionary or RIBD),
- Routing Information Base (RIB),
- Bundle Offers, and
- Bundle Responses.

During a communication opportunity, several sets of each type of information may be transferred in each direction as explained in the rest of this section. Each set can be transferred in one or more messages. When (and only when) using a connection-oriented reliable transport protocol such as TCP as envisaged in this document, a set can be partitioned across messages by the software layer above the PRoPHET protocol engine.

In this case, the last message in a set is flagged in the protocol. This allows the higher-level software to minimize the buffer memory requirements by avoiding the need to build very large messages in one go and allows the message size to be controlled outside of PRoPHET. However, this scheme is only usable if the transport protocol provides reliable, in-order delivery of messages, as the messages are not explicitly sequence numbered and the overall size of the set is not passed explicitly.

The specification of PRoPHET also provides a submessage mechanism and retransmission that allows large messages specified by the higher level to be transmitted in smaller chunks. This mechanism was originally provided to allow PRoPHET to operate over unreliable transport protocols such as UDP, but can also be used with reliable

transports if the higher-level software does not want to handle message fragmentation. However, the sequencing and length adds overhead that is redundant if the transport protocol already provides reliable, in-order delivery.

The first step in the Information Exchange Phase is for the protocol to send one or more messages containing a RIB Dictionary TLV (Type-Length-Value message component) to the node with which it is peering. This set of messages contain a dictionary of the Endpoint Identifiers (EIDs) of the nodes that will be listed in the Routing Information Base (RIB); see Section 3.2.1 for more information about this dictionary. After this, one or more messages containing a Routing Information Base TLV are sent. This TLV contains a list of the EIDs that the node has knowledge of, and the corresponding delivery predictabilities for those nodes, together with flags describing the capabilities of the sending node. Upon reception of a complete set of these messages, the peer node updates its delivery predictability table according to the equations in Section 2.1.2. The peer node then applies its forwarding strategy (see Section 2.1.4) to determine which of its stored bundles it wishes to offer the node that sent the RIB; that node will then be the receiver for any bundles to be transferred.

After making this decision, one or more Bundle Offer TLVs are prepared, listing the bundle identifiers and their destinations for all bundles the peer node wishes to offer to the receiver node that sent the RIB. As described in [RFC5050], a bundle identifier consists of up to five component parts. For a complete bundle, the identifier consists of

- source EID,
- creation timestamp - time of creation, and
- creation timestamp - sequence number.

Additionally, for a bundle fragment, the identifier also contains

- offset within the payload at which the fragment payload data starts, and
- length of the fragment payload data.

If any of the Bundle Offer TLVs lists a bundle for which the source or destination EID was not included in the previous set of RIBD information sent, one or more new RIBD TLVs are sent next with an incremental update of the dictionary. When the receiver node has a dictionary with all necessary EIDs, the Bundle Offer TLVs are sent to

it. The Bundle Offer TLVs also contain a list of PRoPHET ACKs (see Section 3.5). If requested by the receiver node during the Hello phase, the Bundle Offer TLV will also specify the payload length for all bundles rather than for just fragments. This information can be used by the receiving node to assist with the selection of bundles to be accepted from the offered list, especially if the available bundle storage capacity is limited.

The receiving node then examines the list of offered bundles and selects bundles that it will accept according to its own policies, considering the bundles already present in the node and the current availability of resources in the node. The list is sorted according to the priority that the policies apply to the selected bundles, with the highest priority bundle first in the list. The offering node will forward the selected bundles in this order. The prioritized list is sent to the offering node in one or more Bundle Response TLVs using the same EID dictionary as was used for the Bundle Offer TLV.

When a new bundle arrives at a node, the node MAY inspect its list of available neighbors, and if one of them is a candidate to forward the bundle, a new Bundle Offer TLV MAY be sent to that node. If two nodes remain connected over a longer period of time, the Information Exchange Phase will be periodically re-initiated to allow new delivery predictability information to be spread through the network and new bundle exchanges to take place.

The Information Exchange Phase of the protocol is described in more detail in Section 5.3.

3.2.1. Routing Information Base Dictionary

To reduce the overhead of the protocol, the Routing Information Base and Bundle Offer/Response TLVs utilize an EID dictionary. This dictionary maps variable-length EIDs (as defined in [RFC4838]), which may potentially be quite long, to shorter numerical identifiers, coded as Self-Delimiting Numeric Values (SDNVs -- see Section 4.1. of RFC 5050 [RFC5050]), which are used in place of the EIDs in subsequent TLVs.

This dictionary is a shared resource between the two peering nodes. Each can add to the dictionary by sending a RIB Dictionary TLV to its peer. To allow either node to add to the dictionary at any time, the identifiers used by each node 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 in the Information Exchange Phase and the new identifiers can then be used in subsequent TLVs until the dictionary is re-initialized.

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 then 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. Implementers can choose 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 a RIB Dictionary TLV containing an identifier that is already in use, the node MUST confirm that the EID referred to is identical to the EID in the existing entry. Otherwise, the node must send an error response to the message with the TLV containing the error and ignore the TLV containing the error. If a node receives a RIB, Bundle Offer, or Bundle Response TLV that uses an identifier that is not in its dictionary, the node MUST send an error response and ignore the TLV containing the error.

3.2.2. Handling Multiple Simultaneous Contacts

From time to time, a mobile node may, for example, be in wireless range of more than one other mobile node. The PRoPHET neighbor awareness protocol will establish multiple simultaneous contacts with these nodes and commence information exchanges with each of them.

When updating the delivery predictabilities as described in Section 2.1.2 using the values passed from each of the contacts in turn, some special considerations apply when multiple contacts are in progress:

SC1 When aging the delivery predictabilities according to Equation 2, the value of K to be used in each set of calculations is always the amount of time since the last aging was done. For example, if node Z makes contact with node A and then with node B, the value of K used when the delivery predictabilities are aged in node Z for the contact with node B will be the time since the delivery predictabilities were aged for the contact with node A.

SC2 When a new contact starts, the value of P_encounter used when applying Equation 1 for the newly contacted node is always selected according to the time since the last encounter with that node. Thus, the application of Equation 1 to update P_(Z,A) when the contact of nodes Z and A starts (in the aging example just given) and the updating of P_(Z,B) when the contact of nodes Z and B starts will use the appropriate value of P_encounter according to how long it is since node Z previously encountered node A and node B, respectively.

SC3 If, as with the contact between nodes Z and B, there is another active contact in progress, such as with node A when the contact with node B starts, Equation 1 should *also* be applied to P_(z,x) for all the nodes "x" that have ongoing contacts with node Z (i.e., node A in the example given). However, the value of P_encounter used will be selected according to the time since the previous update of the delivery predictabilities as a result of information received from any other node. In the example given here, P_(Z,A) would also have Equation 1 applied when the delivery predictabilities are received from node B, but the value of P_encounter used would be selected according to the time since the updates done when the encounter between nodes Z and A started rather than the time since the previous encounter between nodes A and Z.

If these simultaneous contacts persist for some time, then, as described in Section 3.2, the Information Exchange Phase will be periodically rerun for each contact according to the configured timer interval. When the delivery predictability values are recalculated during each rerun, Equation 1 will be applied as in special consideration SC3 above, but it will be applied to the delivery predictability for each active contact using the P_encounter value selected according to the time since the last set of updates were performed on the delivery predictabilities, irrespective of which nodes triggered either the previous or current updates. This means that, in the example discussed here, P_(Z,A) and P_(Z,B) will be updated using the same value of P_encounter whether node A or node B initiated the update while the three nodes remain connected.

The interval between reruns of the information exchange will generally be set to a small fraction of the expected time between independent encounters of pairs of nodes. This ensures that, for example, the delivery predictability information obtained by node Z from node A will be passed on to node B whether or not nodes A and B can communicate directly during this encounter. This avoids problems that may arise from peculiarities of radio propagation during this sort of encounter, but the scaling of the P_encounter factor according to the time between updates of the delivery predictabilities means that the predictabilities for the nodes that are in contact are not increased excessively as would be the case if each information exchange were treated as a separate encounter with the value of P_encounter_max used each time. When several nodes are in mutual contact, the delivery predictabilities in each node stabilize after a few exchanges due to the scaling of P_encounter as well as the form of Equation 3 where a "max" function is used. This has been demonstrated by simulation.

The effect of the updates of the delivery predictabilities when there are multiple simultaneous contacts is that the information about good routes on which to forward bundles is correctly passed between sets of nodes that are simultaneously in contact through the transitive update of Equation 3 during each information exchange, but the delivery predictabilities for the direct contacts are not exaggerated.

3.3. Routing Algorithm

The basic routing algorithm of the protocol is described in Section 2.1. The algorithm uses some parameter values in the calculation of the delivery predictability metric. These parameters are configurable depending on the usage scenario, but Figure 3 provides some recommended default values. A brief explanation of the parameters and some advice on setting appropriate values is given below.

I_typ
: I_typ provides a fundamental timescale for the mobility pattern in the PRoPHET scenario where the protocol is being applied. It represents the typical or mean time interval between encounters between a given pair of nodes in the normal course of mobility. The interval should reflect the "logical" time between encounters and should not give significant weight to multiple connection events as explained in Section 2.1.2. This time interval informs the settings of many of the other parameters but is not necessarily directly used as a parameter. Consideration needs to be given to the higher statistical moments (e.g., standard deviation) as well as the mean (first

moment) of the distribution of intervals between encounters and the nature of that distribution (e.g., how close to a normal distribution it is). There is further discussion of this point later in this section and in Appendix C.

P_encounter_max

P_encounter_max is used as the upper limit of a scaling factor that increases the delivery predictability for a destination when the destination node is encountered. A larger value of P_encounter_max will increase the delivery predictability faster, and fewer encounters will be required for the delivery predictability to reach a certain level. Given that relative rather than absolute delivery predictability values are what is interesting for the forwarding mechanisms defined, the protocol is very robust to different values of P_encounter as long as the same value is chosen for all nodes. The value should be chosen so that the increase in the delivery predictability resulting from using P_encounter_max in Equation 1 more than compensates for the decay of the delivery predictability resulting from Equation 3 with a time interval of I_typ.

P_encounter(intvl)

As explained in Section 2.1.2, the parameter P_encounter used in Equation 1 is a function of the time interval "intvl". The function should be an approximation to

```
P_encounter(intvl) =
P_encounter_max * (intvl / I_typ) for 0<= intvl <= I_typ
P_encounter_max for intvl > I_typ
```

The function can be quantized and adapted to suit the mobility pattern and to make implementation easier. The overall effect should be that be that if Equation 1 is applied a number of times during a long-lived communication opportunity lasting I_typ, the overall increase in the delivery predictability should be approximately the same as if there had been two distinct encounters spaced I_typ apart. This second case would result in one application of Equation 1 using P_encounter_max.

P_first_threshold

As described in Section 2.1.2, the delivery predictability for a destination is gradually reduced over time unless increased as a result of direct encounters or through the transitive property. If the delivery predictability falls below the value P_first_threshold, then the node MAY discard the delivery predictability information for the destination and treat subsequent encounters as if they had never encountered the node previously. This allows the node to reduce the storage needed

for delivery predictabilities and decreases the amount of information that has to be exchanged between nodes; otherwise, the reduction algorithm would result in very small but non-zero predictabilities being maintained for nodes that were last encountered a long time ago.

P_encounter_first
: As described in Section 2.1.2, PRoPHET does not, by default, make any assumptions about the likelihood that an encountered node will be encountered repeatedly in the future or, alternatively, that this is a one-off chance encounter that is unlikely to be repeated. During an encounter where the encountering node has no delivery predictability information for the encountered destination node, either because this is really the first encounter between the nodes or because the previous encounter was so long ago that the predictability had fallen below P_first_threshold and therefore had been discarded, the encountering node sets the delivery predictability for the destination node to P_encounter_first. The suggested value for P_encounter_first is 0.5: this value is RECOMMENDED as appropriate in the usual case where PRoPHET has no extra (e.g., out-of-band) information about whether future encounters with this node will be regular or otherwise.

alpha
: The alpha parameter is used in the optional smoothing of the delivery predictabilities described in Section 2.1.3.1. It is used to determine the weight of the most current P-value in the calculation of an EWMA.

beta
: The beta parameter adjusts the weight of the transitive property of PRoPHET, that is, how much consideration should be given to information about destinations that is received from encountered nodes. If beta is set to zero, the transitive property of PRoPHET will not be active, and only direct encounters will be used in the calculation of the delivery predictability. The higher the value of beta, the more rapidly encounters will increase predictabilities through the transitive rule.

gamma
: The gamma parameter determines how quickly delivery predictabilities age. A lower value of gamma will cause the delivery predictability to age faster. The value of gamma should be chosen according to the scenario and environment in which the protocol will be used. If encounters are expected to be very frequent, a lower value should be chosen for gamma than if encounters are expected to be rare.

delta
: The delta parameter sets the maximum value of the delivery predictability for a destination other than for the node itself (i.e., P_(A,B) for all cases except P_(A,A)) as (1 - delta). Delta should be set to a small value to allow the maximum possible range for predictabilities but can be configured to make the calculation efficient if needed.

To set an appropriate gamma value, one should consider the "average expected delivery" time I_aed in the PRoPHET zone where the protocol is to be used, and the time unit used (the resolution with which the delivery predictability is being updated). The I_aed time interval can be estimated according to the average number of hops that bundles have to pass and the average interval between encounters I_typ. Clearly, if bundles have a Time To Live (TTL), i.e., the time left until the expiry time stored in the bundle occurs, that is less than I_aed, they are unlikely to survive in the network to be delivered to a node in this PRoPHET zone. However, the TTL for bundles created in nodes in this zone should not be chosen solely on this basis because they may pass through other networks.

After estimating I_aed and selecting how much we want the delivery predictability to age in one I_aed time period (call this A), we can calculate K, the number of time units in one I_aed, using K = (I_aed / time unit). This can then be used to calculate gamma as gamma = K'th-root(A).

I_typ, I_aed, K, and gamma can then be used to inform the settings of P_encounter_first, P_encounter_max, P_first_threshold, delta, and the detailed form of the function P_encounter(intvl).

First, considering the evolution of the delivery predictability P_(A,B) after a single encounter between nodes A and B, P_(A,B) is initially set to P_encounter_first and will then steadily decay until it reaches P_first_threshold. The ratio between P_encounter_first and P_first_threshold should be set so that P_first_threshold is reached after a small multiple (e.g., 3 to 5) of I_aed has elapsed, making it likely that any subsequent encounter between the nodes would have occurred before P_(A,B) decays below P_first_threshold. If the statistics of the distribution of times between encounters is known, then a small multiple of the standard deviation of the distribution would be a possible period instead of using a multiple of I_aed.

Second, if a second encounter between A and B occurs, the setting of P_encounter_max should be sufficiently high to reverse the decay that would have occurred during I_typ and to increase P_(A,B) above the value of P_encounter_first. After several further encounters,

P_(A,B) will reach (1 - delta), its upper limit. As with setting up P_first_threshold, P_encounter_max should be set so that the upper limit is reached after a small number of encounters spaced apart by I_typ have occurred, but this should generally be more than 2 or 3.

Finally, beta can be chosen to give some smoothing of the influence of transitivity.

These instructions on how to set the parameters are only given as a possible method for selecting appropriate values, but network operators are free to set parameters as they choose. Appendix C goes into some more detail on linking the parameters defined here and the more conventional ways of expressing the mobility model in terms of distributions of times between events of various types.

Recommended starting parameter values when specific network measurements have not been done are below. Note: There are no "one size fits all" default values, and the ideal values vary based on network characteristics. It is not inherently necessary for the parameter values to be identical at all nodes, but it is recommended that similar values are used at all nodes within a PRoPHET zone as discussed in Section 2.1.2.1.

Parameter	Recommended value
P_encounter_max	0.7
P_encounter_first	0.5
P_first_threshold	0.1
alpha	0.5
beta	0.9
gamma	0.999
delta	0.01

Figure 3: Default parameter settings

3.4. Bundle Passing

Upon reception of the Bundle Offer TLV, the node inspects the list of bundles and decides which bundles it is willing to store for future forwarding or that it is able to deliver to their destinations. This

decision has to be made using local policies and considering parameters such as available buffer space and, if the node requested bundle lengths, the lengths of the offered bundles. For each such acceptable bundle, the node sends a Bundle Response TLV to its peering node, which responds by sending the requested bundle. If a node has some bundles it would prefer to receive ahead of others offered (e.g., bundles that it can deliver to their final destination), it MAY request the bundles in that priority order. This is often desirable as there is no guarantee that the nodes will remain in contact with each other for long enough to transfer all the acceptable bundles. Otherwise, the node SHOULD assume that the bundles are listed in a priority order determined by the peering node's forwarding strategy and request bundles in that order.

3.4.1. Custody

To free up local resources, a node may give custody of a bundle to another node that offers custody. This is done to move the retransmission requirement further toward the destination. The concept of custody transfer, and more details on the motivation for its use can be found in [RFC4838]. PRoPHET takes no responsibilities for making custody decisions. Such decisions should be made by a higher layer.

3.5. When a Bundle Reaches Its Destination

A PRoPHET ACK is only a confirmation that a bundle has been delivered to its destination in the PRoPHET zone (within the part of the network where PRoPHET is used for routing, bundles might traverse several different types of networks using different routing protocols; thus, this might not be the final destination of the bundle). When nodes exchange Bundle Offer TLVs, bundles that have been ACKed are also listed, having the "PRoPHET ACK" flag set. 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, each PRoPHET ACK should have a timeout that MUST NOT be longer than the timeout of the bundle to which the ACK corresponds.

When a node receives a PRoPHET ACK for a bundle it is carrying, it MAY delete that bundle from its storage, unless the node holds custody of that bundle. The PRoPHET ACK only indicates that a bundle has been delivered to its destination within the PRoPHET zone, so the reception of a PRoPHET ACK is not a guarantee that the bundle has been delivered to its final destination.

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

If necessary in order to preserve system resources, nodes MAY drop PRoPHET ACKs prematurely but SHOULD refrain from doing so if possible.

It is important to keep in mind that PRoPHET ACKs and bundle ACKs [RFC5050] are different things. PRoPHET ACKs are only valid within the PRoPHET part of the network, while bundle ACKs are end-to-end acknowledgments that may go outside of the PRoPHET zone.

3.6. Forwarding Strategies

During the Information Exchange Phase, nodes need to decide on which bundles they wish to exchange with the peering node. Because of the large number of scenarios and environments that PRoPHET can be used in, and because of the wide range of devices that may be used, it is not certain that this decision will be based on the same strategy in every case. Therefore, each node MUST operate a _forwarding strategy_ to make this decision. Nodes may define their own strategies, but this section defines a few basic forwarding strategies that nodes can use. Note: If the node being encountered is the destination of any of the bundles being carried, those bundles SHOULD be offered to the destination, even if that would violate the forwarding strategy. Some of the forwarding strategies listed here have been evaluated (together with a number of queueing policies) through simulations, and more information about that and recommendations on which strategies to use in different situations can be found in [lindgren_06]. If not chosen differently due to the characteristics of the deployment scenario, nodes SHOULD choose GRTR as the default forwarding strategy.

The short names applied to the forwarding strategies should be read as mnemonic handles rather than as specific acronyms for any set of words in the specification.

We use the following notation in our descriptions below. A and B are the nodes that encounter each other, and the strategies are described as they would be applied by node A. The destination node is D. P_(X,Y) denotes the delivery predictability stored at node X for destination Y, and NF is the number of times node A has given the bundle to some other node.

GRTR
: Forward the bundle only if P_(B,D) > P_(A,D).

  When two nodes meet, a bundle is sent to the other node if the delivery predictability of the destination of the bundle is higher at the other node. The first node does not delete the bundle after sending it as long as there is sufficient buffer space available (since it might encounter a better node, or even the final destination of the bundle in the future).

GTMX
: Forward the bundle only if P_(B,D) > P_(A,D) && NF < NF_max.

  This strategy is like the previous one, but each bundle is given to at most NF_max other nodes in addition to the destination.

GTHR
: Forward the bundle only if
  P_(B,D) > P_(A,D) OR P_(B,D) > FORW_thres,
  where FORW_thres is a threshold value above which a bundle should always be given to the node unless it is already present at the other node.

  This strategy is similar to GRTR, but among nodes with very high delivery predictability, bundles for that particular destination are spread epidemically.

GRTR+
: Forward the bundle only if Equation 5 holds, where P_max is the largest delivery predictability reported by a node to which the bundle has been sent so far.

  P_(B,D) > P_(A,D) && P_(B,D) > P_max (Eq. 5)

  This strategy is like GRTR, but each node forwarding a bundle keeps track of the largest delivery predictability of any node it has forwarded this bundle to, and only forwards the bundle again if the currently encountered node has a greater delivery predictability than the maximum previously encountered.

GTMX+
: Forward the bundle only if Equation 6 holds.

  P_(B,D) > P_(A,D) && P_(B,D) > P_max && NF < NF_max (Eq. 6)

  This strategy is like GTMX, but nodes keep track of P_max as in GRTR+.

GRTRSort
: Select bundles in descending order of the value of P_(B,D) - P_(A,D).
Forward the bundle only if P_(B,D) > P_(A,D).

This strategy is like GRTR, but instead of just going through the bundle queue linearly, this strategy looks at the difference in delivery predictabilities for each bundle between the two nodes and forwards the bundles with the largest difference first. As bandwidth limitations or disrupted connections may result in not all bundles that would be desirable being exchanged, it could be desirable to first send bundles that get a large improvement in delivery predictability.

GRTRMax
: Select bundles in descending order of P_(B,D).
Forward the bundle only if P_(B,D) > P_(A,D).

This strategy begins by considering the bundles for which the encountered node has the highest delivery predictability. The motivation for doing this is the same as in GRTRSort, but based on the idea that it is better to give bundles to nodes with high absolute delivery predictabilities, instead of trying to maximize the improvement.

3.7. Queueing Policies

Because of limited buffer resources, nodes may need to drop some bundles. As is the case with the forwarding strategies, which bundle to drop is also dependent on the scenario. Therefore, each node MUST also operate a queueing policy that determines how its bundle queue is handled. This section defines a few basic queueing policies, but nodes MAY use other policies if desired. Some of the queueing policies listed here have been evaluated (together with a number of forwarding strategies) through simulations. More information about that and recommendations on which policies to use in different situations can be found in [lindgren_06]. If not chosen differently due to the characteristics of the deployment scenario, nodes SHOULD choose FIFO as the default queueing policy.

The short names applied to the queueing policies should be read as mnemonic handles rather than as specific acronyms for any set of words in the specification.

FIFO - First In First Out.
: The bundle that was first entered into the queue is the first bundle to be dropped.

MOFO - Evict most forwarded first.
In an attempt to maximize the delivery rate of bundles, this policy requires that the routing agent keep track of the number of times each bundle has been forwarded to some other node. The bundle that has been forwarded the largest number of times is the first to be dropped.

MOPR - Evict most favorably forwarded first.
Keep a variable FAV for each bundle in the queue, initialized to zero. Each time the bundle is forwarded, update FAV according to Equation 7, where P is the predictability metric that the node the bundle is forwarded to has for its destination.

FAV_new = FAV_old + (1 - FAV_old) * P (Eq. 7)

The bundle with the highest FAV value is the first to be dropped.

Linear MOPR - Evict most favorably forwarded first; linear increase.
Keep a variable FAV for each bundle in the queue, initialized to zero. Each time the bundle is forwarded, update FAV according to Equation 8, where P is the predictability metric that the node the bundle is forwarded to has for its destination.

FAV_new = FAV_old + P (Eq. 8)

The bundle with the highest FAV value is the first to be dropped.

SHLI - Evict shortest life time 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 lifetime 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. However, this is not specific to this routing protocol, but a more general DTN problem.

LEPR - Evict least probable first.
Since the node is least likely to deliver a bundle for which it has a low delivery predictability, drop the bundle for which the node has the lowest delivery predictability, and that has been forwarded at least MF times, where MF 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. As an example, one could select the queueing policy to be {MOFO; SHLI; FIFO}, which would start by dropping the bundle that has been forwarded the largest number of times. If more than one bundle has been forwarded the same number of times, the one with the shortest remaining lifetime will be dropped, and if that also is the same, the FIFO policy will be used to drop the bundle first received.

It is worth noting that a node MUST NOT drop bundles for which it has custody unless the bundle's lifetime expires.

4. Message Formats

This section defines the message formats of the PRoPHET routing protocol. In order to allow for variable-length fields, many numeric fields are encoded as Self-Delimiting Numeric Values (SDNVs). The format of SDNVs is defined in [RFC5050]. 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. Where SDNVs and/or text strings are used, the octets of the fields will be packed as closely as possible with no intervening padding between fields.

Explicit-length fields are specified for all variable-length string fields. Accordingly, strings are not null terminated and just contain the exact set of octets in the string.

The basic message format shown in Figure 4 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.

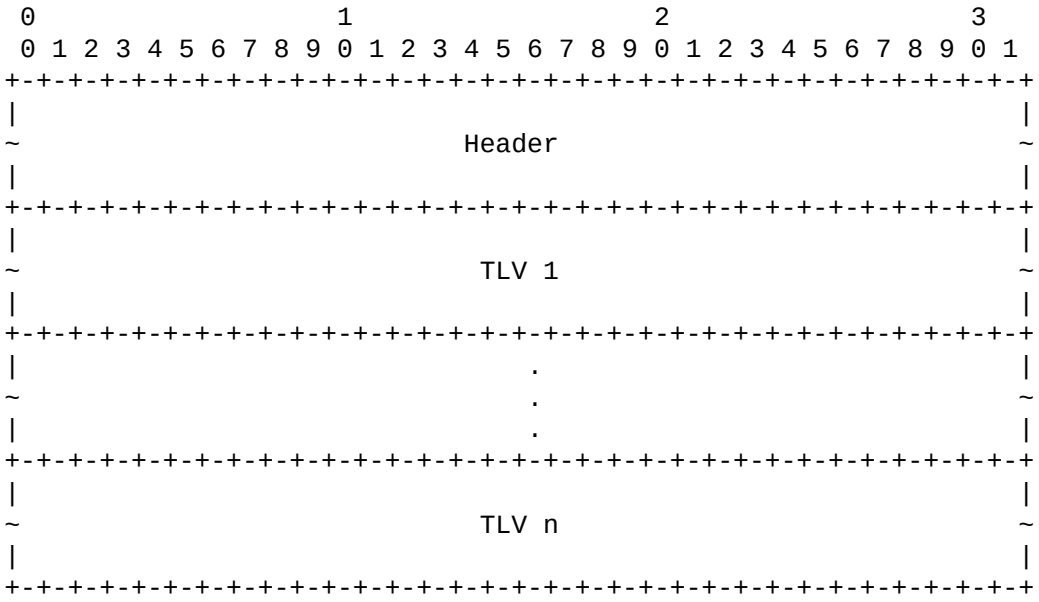


Figure 4: Basic PRoPHET Message Format

4.1. Header

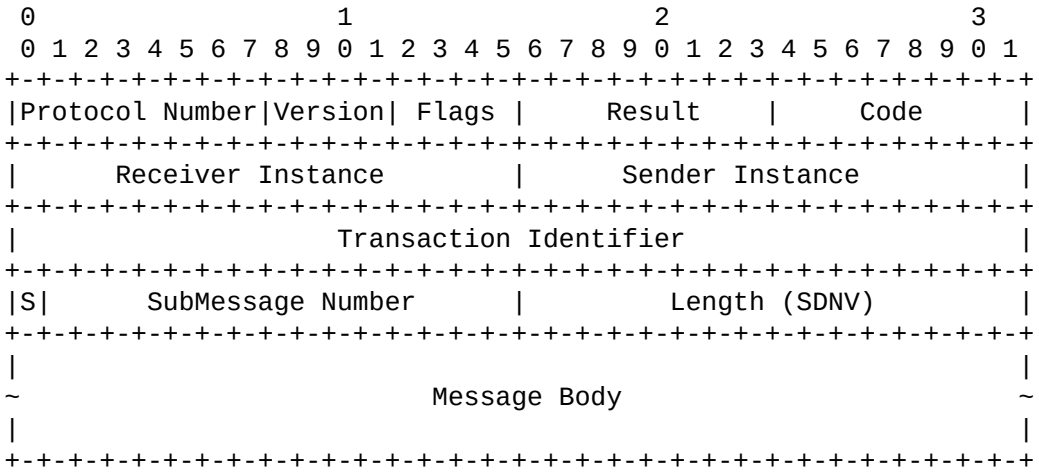


Figure 5: PRoPHET Message Header

Protocol Number
: The DTN Routing Protocol Number encoded as 8-bit unsigned integer in network bit order. The value of this field is 0. The PRoPHET header is organized in this way so that in principle PRoPHET messages could be sent as the Protocol Data Unit of an IP packet if an IP protocol number was allocated for PRoPHET.

At present, PRoPHET is only specified to use a TCP transport for carriage of PRoPHET packets, so that the protocol number serves only to identify the PRoPHET protocol within DTN. Transmitting PRoPHET packets directly as an IP protocol on a public IP network such as the Internet would generally not work well because middleboxes (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 private networks for experimentation or in situations where all communications are between isolated pairs of nodes. Also, in the future, other protocols that require transmission of metadata between DTN nodes could potentially use the same format and protocol state machinery but with a different Protocol Number.

Version
: The version of the PRoPHET Protocol. Encoded as a 4-bit unsigned integer in network bit order. This document defines version 2.

Flags
: Reserved field of 4 bits.

Result
: Field that is used to indicate whether a response is required to the request message if the outcome is successful. A value of "NoSuccessAck" indicates that the request message does not expect a response if the outcome is successful, and a value of "AckAll" indicates that a response is expected if the outcome is successful. In both cases, a failure response MUST be generated if the request fails. If running over a TCP transport or similar protocol that offers reliable in order delivery, deployments MAY choose not to send "Success" responses when an outcome is successful. To achieve this, the Result field is set to the "NoSuccessAck" value in all request messages.

  In a response message, the result field can have two values: "Success" and "Failure". The "Success" result indicates a success response. All messages that belong to the same success response will have the same Transaction Identifier. The "Success" result indicates a success response that may be contained in a single message or the final message of a success response spanning multiple messages.

ReturnReceipt is a value of the result field used to indicate that an acknowledgement is required for the message. The default for messages is that the controller will not acknowledge responses. In the case where an acknowledgement is required, it will set the Result Field to ReturnReceipt in the header of the Message.

The result field is encoded as an 8-bit unsigned integer in network bit order. The following values are currently defined:

NoSuccessAck:	Result = 1
AckAll:	Result = 2
Success:	Result = 3
Failure:	Result = 4
ReturnReceipt	Result = 5

Code

This field gives further information concerning the result 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 or an event message. In a request message, the code field is not used and is set to zero.

If the Code field indicates that the Error TLV is included in the message, further information on the error will be found in the Error 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. Separate number code spaces are used for success and failure response messages. In each case, a range of values is reserved for use in specifications and another range for private and experimental use. For success messages, the following values are defined:

Generic Success	0x00
Submessage Received	0x01
Unassigned	0x02 - 0x7F
Private/Experimental Use	0x80 - 0xFF

The Submessage Received code is used to acknowledge reception of a message segment. The Generic Success code is used to acknowledge receipt of a complete message and successful processing of the contents.

For failure messages, the following values are defined:

Reserved	0x00 - 0x01
Unspecified Failure	0x02
Unassigned	0x03 - 0x7F
Private/Experimental Use	0x80 - 0xFE
Error TLV in message	0xFF

The Unspecified Failure code can be used to report a failure for which there is no more specific code or Error TLV value defined.

Sender Instance

For messages during the Hello phase with the Hello SYN, Hello SYNACK, and Hello ACK functions (which are explained in Section 5.2), it is the sender's instance number for the link. It is used to detect when the link comes back up after going down or when the identity of the entity at the other end of the link changes. The instance 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. For the RSTACK function (also explained in detail in Section 5.2), the Sender Instance field is set to the value of the Receiver Instance field from the incoming message that caused the RSTACK function to be generated. Messages sent after the Hello phase is completed should use the sender's instance number for the link. The Sender Instance is encoded as a 16-bit unsigned integer in network bit order.

Receiver Instance

For messages during the Hello phase with the Hello SYN, Hello SYNACK, and Hello ACK functions, it is what the sender believes is the current instance number for the link, allocated by the entity at the far end of the link. If the sender of the message does not know the current instance number at the far end of the link, this field MUST be set to zero. For the RSTACK message, the Receiver Instance field is set to the value of the Sender Instance field from the incoming message that caused the RSTACK message to be generated. Messages sent after the Hello phase is completed should use what the sender believes is the current instance number for the link, allocated by the entity at the far end of the link. The Sender Instance is encoded as a 16-bit unsigned integer in network bit order.

Transaction 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. Reply messages contain the Transaction Identifier of the request to which they are responding. The Transaction Identifier is a bit pattern of 32 bits.

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 submessage 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 submessage in a sequence of submessages, 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 zero-based, i.e., for a message divided into n submessages, they are numbered from 0 to (n - 1). For a message that is not divided into submessages, the single message has the S-flag cleared (value 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
: As specified in Section 4, the Message Body consists of a sequence of one or more of the TLVs specified in Section 4.2.

The protocol also requires extra information about the link that the underlying communication layer MUST provide. This information is used in the Hello procedure described in more detail in Section 5.2. Since this information is available from the underlying layer, there is no need to carry it in PRoPHET messages. The following values are defined to be provided by the underlying layer:

Sender Local Address
: An address that is used by the underlying communication layer as described in Section 2.4 and identifies the sender address of the current message. This address must be unique among the nodes that can currently communicate, and it is only used in conjunction with the Receiver Local Address, Receiver Instance, and Sender Instance to identify a communicating pair of nodes.

Receiver Local Address
: An address that is used by the underlying communication layer as described in Section 2.4 and identifies the receiver address of the current message. This address must be unique among the nodes that can currently communicate, and is only used in conjunction with the Sender Local Address, Receiver Instance, and Sender Instance to identify a communicating pair of nodes.

When PRoPHET is run over TCP, the IP addresses of the communicating nodes are used as Sender and Receiver Local Addresses.

4.2. TLV Structure

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

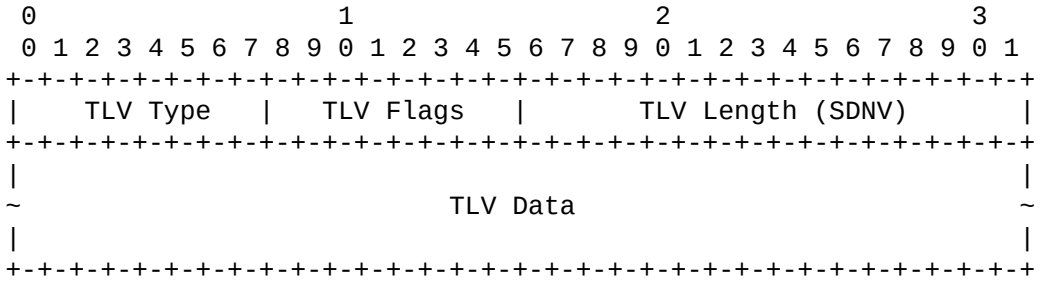


Figure 6: TLV Format

TLV Type
: Specific TLVs are defined in Section 4.3. The TLV Type is encoded as an 8-bit unsigned integer in network bit order. Each TLV will have fields defined that are specific to the function of that TLV.

TLV Flags
: These are defined per TLV type. Flag n corresponds to bit 15-n in the TLV. Any flags that 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. Note that TLVs are not padded to any specific alignment unless explicitly required in the description of the TLV. No TLVs in this document specify any padding.

4.3. TLVs

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

4.3.1. Hello TLV

The Hello TLV is used to set up and maintain a link between two PRoPHET nodes. Hello messages with the SYN function are transmitted periodically as beacons or keep-alives. The Hello TLV is the first TLV exchanged between two PRoPHET nodes when they encounter each other. No other TLVs can be exchanged until the first Hello sequence is completed.

Once a communication link is established between two PRoPHET nodes, the Hello TLV will be sent once for each interval as defined in the interval timer. If a node experiences the lapse of HELLO_DEAD Hello intervals without receiving a Hello TLV on a connection in the INFO_EXCH state (as defined in the state machine in Section 5.1), the connection SHOULD be assumed broken.

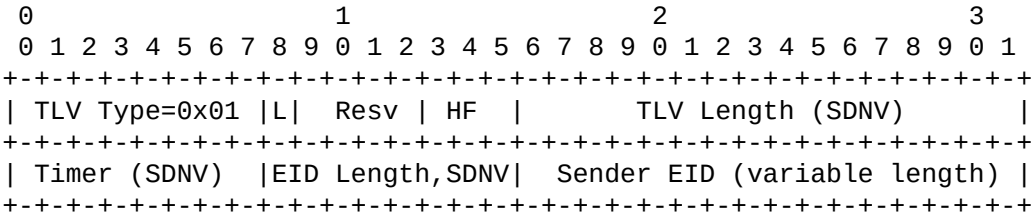


Figure 7: Hello TLV Format

TLV Flags
: The TLV Flags field contains two 1-bit flags (S and L) and a 3-bit Hello Function (HF) number that specifies one of four functions for the Hello TLV. The remaining 3 bits (Resv) are unused and reserved:

HF
: TLV Flags bits 0, 1, and 2 are treated as an unsigned 3-bit integer coded in network bit order. The value of the integer specifies the Hello Function (HF) of the Hello TLV. Four functions are specified for the Hello TLV.

  The encoding of the Hello Function is:

```
SYN:     HF = 1
SYNACK:  HF = 2
ACK:     HF = 3
RSTACK:  HF = 4
```

The remaining values (0, 5, 6 and 7) are unused and reserved. If a Hello TLV with any of these values is received, the link should be reset.

Resv
: TLV Flags bits 3, 4, 5, and 6 are reserved. They SHOULD be set to 0 on transmission and ignored on reception.

L
: The L bit flag (TLV Flags bit 7) is set (value 1) to request that the Bundle Offer TLV sent during the Information Exchange Phase contains bundle payload lengths for all bundles, rather than only for bundle fragments as when the L flag is cleared (value 0), when carried in a Hello TLV with Hello Function SYN or SYNACK. The flag is ignored for other Hello Function values.

TLV Data

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.

EID Length
: The EID Length field is used to specify the length of the Sender EID field in octets. If the Endpoint Identifier (EID) has already been sent at least once in a message with the current Sender Instance, 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 DTN endpoint identifier (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 PRoPHET routing. This field is of variable length.

4.3.2. Error TLV

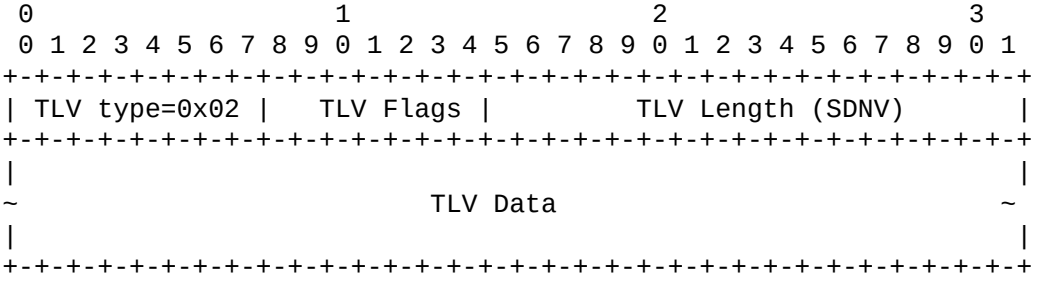


Figure 8: Error TLV Format

TLV Flags
: For Error TLVs, the TLV Flags field carries an identifier for the Error 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 Error TLV types are defined:

Dictionary Conflict	0x00
Bad String ID	0x01
Reserved	0x02 - 0x7F
Private/Experimental Use	0x80 - 0xFF

TLV Data
: The contents and interpretation of the TLV Data field are specific to the type of Error TLV. For the Error TLVs defined in this document, the TLV Data is defined as follows:

  Dictionary Conflict
  : The TLV Data consists of the String ID that is causing the conflict encoded as an SDNV followed by the EID string that conflicts with the previously installed value. The Endpoint Identifier is NOT null terminated. The length of the EID can be determined by subtracting the length of the TLV Header and the length of the SDNV containing the String ID from the TLV Length.

Bad String ID
The TLV Data consists of the String ID that is not found in the dictionary encoded as an SDNV.

4.3.3. Routing Information Base Dictionary TLV

The Routing Information Base Dictionary includes the list of endpoint identifiers used in making routing decisions. The referents remain constant for the duration of a session over a link where the instance numbers remain the same and can be used by both the Routing Information Base messages and the bundle offer/response messages. The dictionary is a shared resource (see Section 3.2.1) built in each of the paired peers from the contents of one or more incoming TLVs of this type and from the information used to create outgoing TLVs of this type.

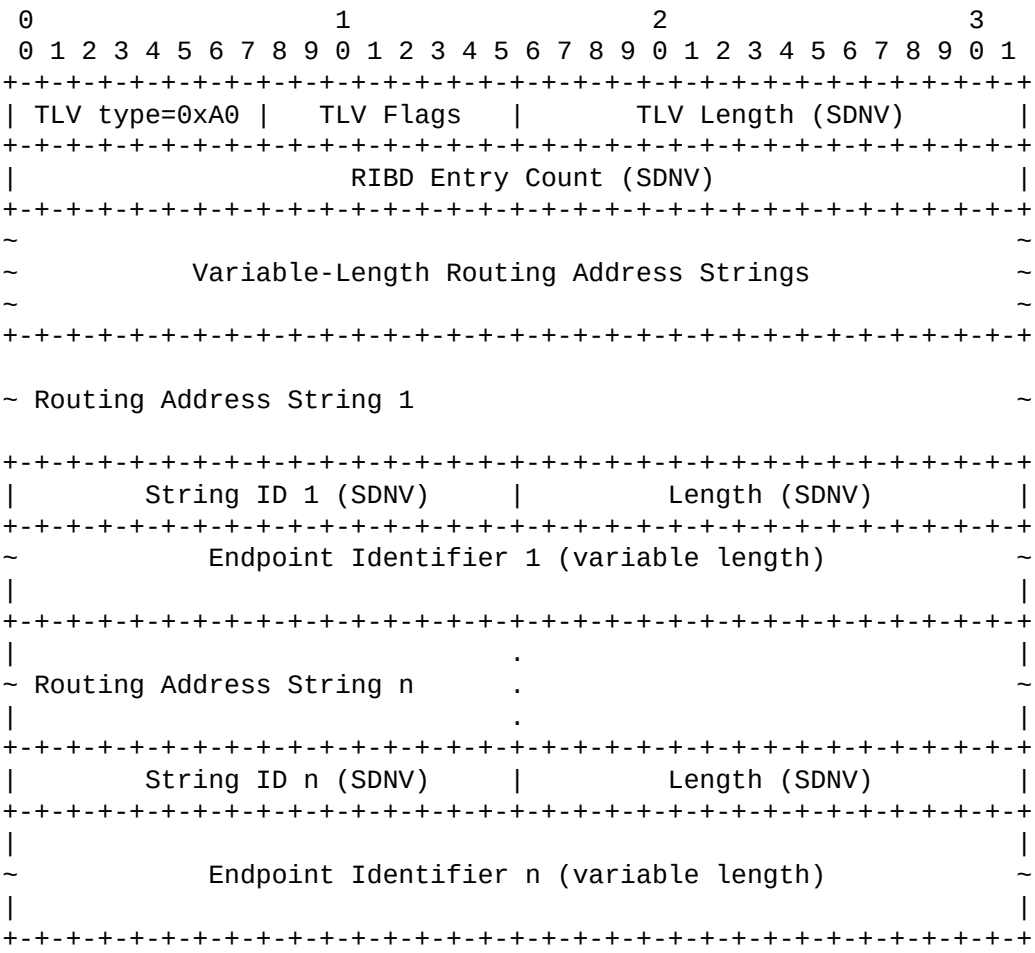


Figure 9: Routing Information Base Dictionary TLV Format

TLV Flags

The encoding of the Header flag field relates to the capabilities of the source node sending the RIB Dictionary:

Flag 0: Sent by Listener	0b1
Flag 1: Reserved	0b1
Flag 2: Reserved	0b1
Flag 3: Unassigned	0b1
Flag 4: Unassigned	0b1
Flag 5: Unassigned	0b1
Flag 6: Unassigned	0b1
Flag 7: Unassigned	0b1

The "Sent by Listener" flag is set to 0 if this TLV was sent by a node in the Initiator role and set to 1 if this TLV was sent by a node in the Listener role (see Section 3.2 for explanations of these roles).

TLV Data

RIBD Entry Count
: Number of entries in the database. Encoded as SDNV.

String ID
: SDNV identifier that is constant for the duration of a session. String ID zero is predefined as the node that initiates the session through sending the Hello SYN message, and String ID one is predefined as the node that responds with the Hello SYNACK 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 SYN 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 SYNACK 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
: 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. Routing Information Base TLV

The Routing Information Base lists the destinations (endpoints) a node knows of and the delivery predictabilities it has associated with them. This information is needed by the PRoPHET algorithm to make decisions on routing and forwarding.

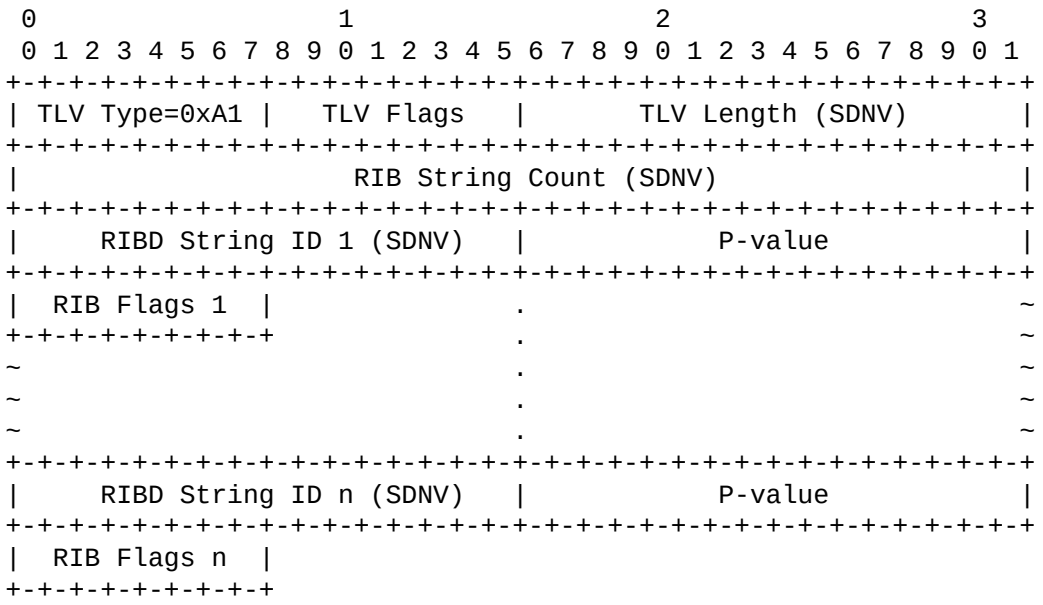


Figure 10: Routing Information Base TLV Format

TLV Flags

The encoding of the Header flag field relates to the capabilities of the Source node sending the RIB:

Flag 0: More RIB TLVs	0b1
Flag 1: Reserved	0b1
Flag 2: Reserved	0b1
Flag 3: Unassigned	0b1
Flag 4: Unassigned	0b1
Flag 5: Unassigned	0b1
Flag 6: Unassigned	0b1
Flag 7: Unassigned	0b1

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

TLV Data

RIB String Count
Number of routing entries in the TLV. Encoded as an SDNV.

RIBD String ID
String ID of the endpoint identifier of the destination for which this entry specifies the delivery predictability as predefined in a dictionary TLV. Encoded as an SDNV.

P-value
Delivery predictability for the destination of this entry as calculated from previous encounters according to the equations in Section 2.1.2, encoded as a 16-bit unsigned integer. The encoding of this field is a linear mapping from [0,1] to [0, 0xFFFF] (e.g., for a P-value of 0.75, the mapping would be 0.75*65535=49151=0xBFFF; thus, the P-value would be encoded as 0xBFFF).

RIB Flag
The encoding of the 8-bit RIB Flag field is:

Flag 0: Unassigned	0b1
Flag 1: Unassigned	0b1
Flag 2: Unassigned	0b1
Flag 3: Unassigned	0b1
Flag 4: Unassigned	0b1
Flag 5: Unassigned	0b1
Flag 6: Unassigned	0b1
Flag 7: Unassigned	0b1

4.3.5. Bundle Offer and Response TLVs (Version 2)

After the routing information has been passed, the node will ask the other node to review available bundles and determine which bundles it will accept for relay. The source relay will determine which bundles to offer based on relative delivery predictabilities as explained in Section 3.6.

Note: The original versions of these TLVs (TLV Types 0xA2 and 0xA3) used in version 1 of the PRoPHET protocol have been deprecated, as they did not contain the complete information needed to uniquely identify bundles and could not handle bundle fragments.

Depending on the bundles stored in the offering node, the Bundle Offer TLV might contain descriptions of both complete bundles and bundle fragments. In order to correctly identify bundle fragments, a

bundle fragment descriptor MUST contain the offset of the payload fragment in the bundle payload and the length of the payload fragment. If requested by the receiving node by setting the L flag in the SYN or SYNACK message during the neighbor awareness phase, the offering node MUST include the length of the payload in the descriptor for complete bundles. The appropriate flags MUST be set in the B_flags for the descriptor to indicate if the descriptor contains the payload length field (set for fragments in all cases and for complete bundles if the L flag was set) and if the descriptor contains a payload offset field (fragments only).

The Bundle Offer TLV also lists the bundles for which a PRoPHET acknowledgement has been issued. Those bundles have the PRoPHET ACK flag set in their entry in the list. When a node receives a PRoPHET ACK for a bundle, it SHOULD, if possible, signal to the bundle protocol agent that this bundle is no longer required for transmission by PRoPHET. Despite no longer transmitting the bundle, it SHOULD keep an entry for the acknowledged bundle to be able to further propagate the PRoPHET ACK.

The Response TLV format is identical to the Offer TLV with the exception of the TLV Type field. Bundles that are being accepted from the corresponding Offer are explicitly marked with a B_flag. Specifications for bundles that are not being accepted MAY either be omitted or left in but not marked as accepted. The payload length field MAY be omitted for complete bundles in the Response message even if it was included in the Offer message. The B_flags payload length flag MUST be set correctly to indicate if the length field is included or not. The Response message MUST include both payload offset and payload length fields for bundle fragments, and the B_flags MUST be set to indicate that both are present.

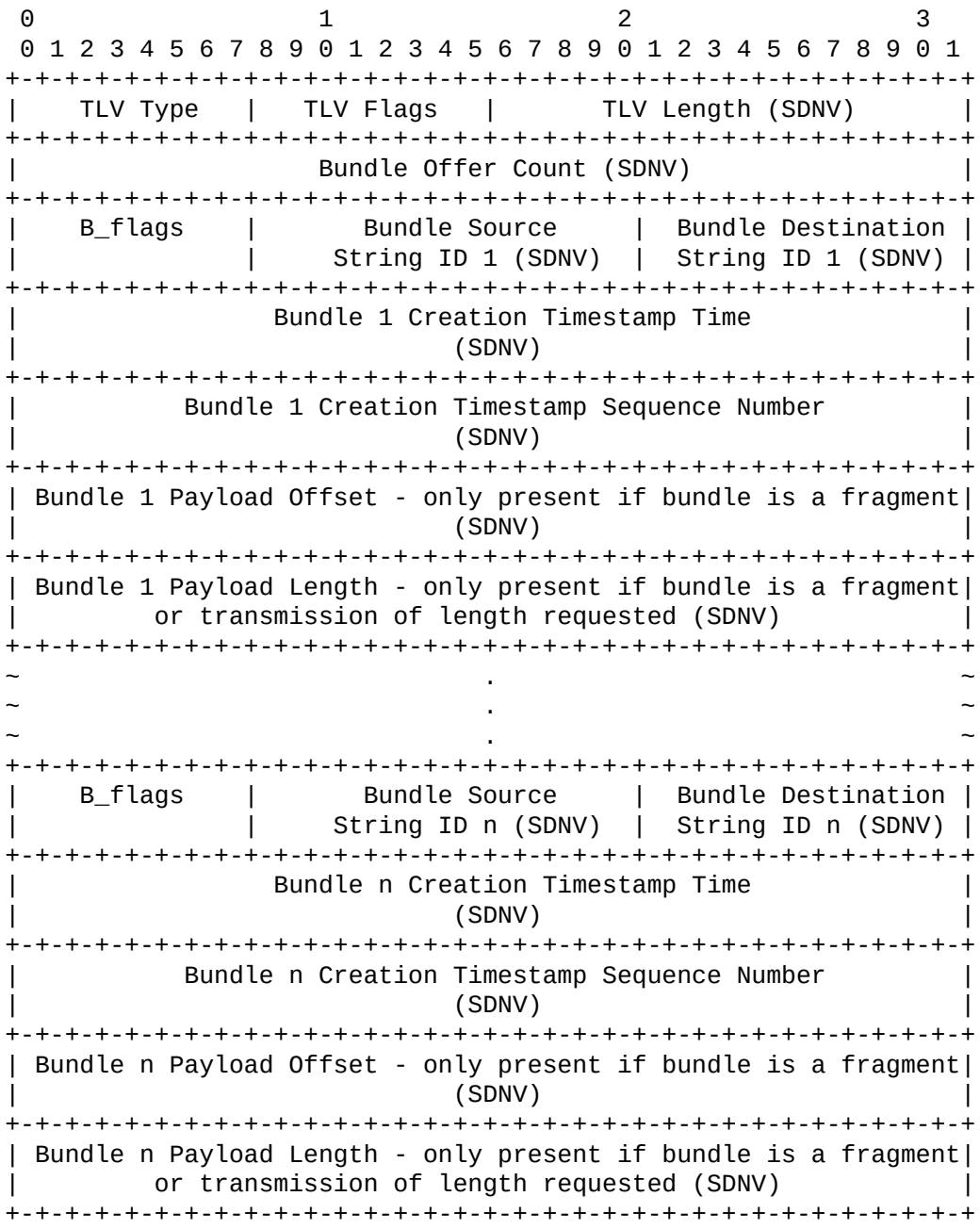


Figure 11: Bundle Offer and Response TLV Format

TLV Type

The TLV Type for a Bundle Offer is 0xA4. The TLV Type for a Bundle Response is 0xA5.

TLV Flags

The encoding of the Header flag field relates to the capabilities of the source node sending the RIB:

Flag 0:	More Offer/Response TLVs Following	0b1
Flag 1:	Unassigned	0b1
Flag 2:	Unassigned	0b1
Flag 3:	Unassigned	0b1
Flag 4:	Unassigned	0b1
Flag 5:	Unassigned	0b1
Flag 6:	Unassigned	0b1
Flag 7:	Unassigned	0b1

If the Bundle Offers or Bundle Responses are divided between several TLVs, the "More Offer/Response TLVs Following" flag MUST be set to 1 in all but the last TLV in the sequence where it MUST be set to 0.

TLV Data

Bundle Offer Count

Number of bundle offer/response entries. Encoded as an SDNV. Note that 0 is an acceptable value. In particular, a Bundle Response TLV with 0 entries is used to signal that a cycle of information exchange and bundle passing is completed.

B Flags

The encoding of the B Flags is:

Flag 0:	Bundle Accepted	0b1
Flag 1:	Bundle is a Fragment	0b1
Flag 2:	Bundle Payload Length included in TLV	0b1
Flag 3:	Unassigned	0b1
Flag 4:	Unassigned	0b1
Flag 5:	Unassigned	0b1
Flag 6:	Unassigned	0b1
Flag 7:	PRoPHET ACK	0b1

Bundle Source String ID

String ID of the source EID of the bundle as predefined in a dictionary TLV. Encoded as an SDNV.

Bundle Destination String ID
: String ID of the destination EID of the bundle as predefined in a dictionary TLV. Encoded as an SDNV.

Bundle Creation Timestamp Time
: Time component of the Bundle Creation Timestamp for the bundle. Encoded as an SDNV.

Bundle Creation Timestamp Sequence Number
: Sequence Number component of the Bundle Creation Timestamp for the bundle. Encoded as an SDNV.

Bundle Payload Offset
: Only included if the bundle is a fragment and the fragment bit is set (value 1) in the bundle B Flags. Offset of the start of the fragment payload in the complete bundle payload. Encoded as an SDNV.

Bundle Payload Length
: Only included if the bundle length included bit is set (value 1) in the bundle B Flags. Length of the payload in the bundle specified. This is either the total payload length if the bundle is a complete bundle or the bundle fragment payload length if the bundle is a fragment. Encoded as an SDNV.

5. Detailed Operation

In this section, some more details on the operation of PRoPHET are given along with state tables to help in implementing the protocol.

As explained in Section 1.2, it is RECOMMENDED that "Success" responses should not be requested or sent when operating over a reliable, in-order transport protocol such as TCP. If in the future PRoPHET were operated over an unreliable transport protocol, 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 if running over an unreliable protocol, but guaranteed by TCP or another reliable protocol otherwise. 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 gives high-level state tables for the operation of PRoPHET. The following sections will describe each part of the operation in more detail (including state tables for the internal states of those procedures).

The following main or high-level states are used in the state tables:

WAIT_NB This is the state all nodes start in. Nodes remain in this state until they are notified that a new neighbor is available. At that point, the Hello procedure should be started with the new neighbor, and the node transitions into the HELLO state. Nodes SHOULD be able to handle multiple neighbors in parallel, maintaining separate state machines for each neighbor. This could be handled by creating a new thread or process during the transition to the HELLO state that then takes care of the communication with the new neighbor while the parent remains in state WAIT_NB waiting for additional neighbors to communicate. In this case, when the neighbor can no longer be communicated with (described as "Neighbor Gone" in the tables below), the thread or process created is destroyed and, when a connection-oriented protocol is being used to communicate with the neighbor, the connection is closed. The current version of the protocol is specified to use TCP for neighbor connections so that these will be closed when the neighbor is no longer accessible.

HELLO Nodes are in the HELLO state from when a new neighbor is detected until the Hello procedure is completed and a link is established (which happens when the Hello procedure enters the ESTAB state as described in Section 5.2; during this procedure, the states ESTAB, SYNSENT, and SYNRCVD will be used, but these are internal to the Hello procedure and are not listed here). If the node is notified that the neighbor is no longer in range before a link has been established, it returns to the WAIT_NB state, and, if appropriate, any additional process or thread created to handle the neighbor MAY be destroyed.

INFO_EXCH After a link has been set up by the Hello procedure, the node transitions to the INFO_EXCH state in which the Information Exchange Phase is done. The node remains in this state as long as Information Exchange Phase TLVs (Routing RIB, Routing RIB Dictionary, Bundle Offer, Bundle Response) are being received. If the node is notified that the neighbor is no longer in range before all information and bundles have been exchanged, any associated connection is closed and the node

returns to the WAIT_NB state to await new neighbors. The Timer(keep_alive) is used to ensure that the connection remains active.

In the INFO_EXCH state, the nodes at both ends of the established link are able to update their delivery predictability information using data from the connected peer and then make offers of bundles for exchange which may be accepted or not by the peer. To manage these processes, each node acts both as an Initiator and a Listener for the Information Exchange Phase processes, maintaining subsidiary state machines for the two roles. The Initiator and Listener terms refer to the sending of the Routing RIB information: it is perhaps counterintuitive that the Listener becomes the bundle offeror and the Initiator the bundle acceptor during the bundling passing part.

The protocol is designed so that the two exchanges MAY be carried out independently but concurrently, with the messages multiplexed onto on a single bidirectional link (such as is provided by the TCP connection). Alternatively, the exchanges MAY be carried out partially or wholly sequentially if appropriate for the implementation. The Information Exchange Phase is explained in more detail in Section 3.2.

When an empty Bundle Response TLV (i.e., no more bundles to send) is received, the node starts the Timer(next_exchange). When this timer expires, assuming that the neighbor is still connected, the Initiator reruns the Information Exchange Phase. If there is only one neighbor connected at this time, this will have the effect of further increasing the delivery predictability for this node in the neighbor, and changing the delivery predictabilities as a result of the transitive property (Equation 3). If there is more than one neighbor connected or other communication opportunities have happened since the previous information exchange occurred, then the changes resulting from these other encounters will be passed on to the connected neighbor. The next_exchange timer is restarted once the information exchange has completed again.

If one or more new bundles are received by this node while waiting for the Timer(next_exchange) to expire and the delivery predictabilities indicate that it would be appropriate to forward some or all of the bundles to the connected node, the bundles SHOULD be immediately offered to the connected neighbor and transferred if accepted.

State: WAIT_NB

Condition	Action	New State
New Neighbor	Start Hello procedure for neighbor Keep waiting for more neighbors	HELLO WAIT_NB

State: HELLO

Condition	Action	New State
Hello TLV rcvd		HELLO
Enter ESTAB state	Start Information Exchange Phase	INFO_EXCH
Neighbor Gone		WAIT_NB

State: INFO_EXCH

Condition	Action	New State
On entry	Start Timer(keep-alive) Uses Hello Timer interval	INFO_EXCH
Info Exch TLV rcvd	(processed by subsidiary state machines)	INFO_EXCH
No more bundles	Start Timer(next_exchange)	INFO_EXCH
Keep-alive expiry	Send Hello SYN message	INFO_EXCH
Hello SYN rcvd	Record reception Restart Timer(keep-alive)	INFO_EXCH
Neighbor Gone		WAIT_NB

The keep-alive messages (messages with Hello SYN TLV) are processed by the high-level state machine in the INFO_EXCH state. All other messages are delegated to the subsidiary state machines of the Information Exchange Phase described in Section 5.3. The receipt of keep-alive messages is recorded and may be used by the subsidiary machines to check if the peer is still functioning. The connection will be aborted (as described in Section 4.3.1) if several keep-alive messages are not received.

5.2. Hello Procedure

The Hello procedure is described by the following rules and state tables. In this section, the messages sent consist of the PRoPHET header and a single Hello TLV (see Figure 4 and Section 4.3.1) with the HF (Hello Function) field set to the specified value (SYN, SYNACK, ACK or RSTACK).

The state of the L flag in the latest SYN or SYNACK message is recorded in the node that receives the message. If the L flag is set (value 1), the receiving node MUST send the payload length for each bundle that it offers to the peer during the Information Exchange Phase.

The rules and state tables use the following operations:

- The "Update Peer Verifier" operation is defined as storing the values of the Sender Instance and Sender Local Address fields from a Hello SYN or Hello SYNACK function message received from the entity at the far end of the link.

- The procedure "Reset the link" is defined as:

  When using TCP or other reliable connection-oriented transport:
  Close the connection and terminate any separate thread or process managing the connection.

  Otherwise:

  1. Generate a new instance number for the link.

  2. Delete the peer verifier (set to zero the values of Sender Instance and Sender Local Address previously stored by the Update Peer Verifier operation).

  3. Send a SYN message.

  4. Transition to the SYNSENT state.

- The state tables use the following Boolean terms and operators:

  A The Sender Instance in the incoming message matches the value stored from a previous message by the "Update Peer Verifier" operation.

  B The Sender Instance and Sender Local Address fields in the incoming message match the values stored from a previous message by the "Update Peer Verifier" operation.

  C The Receiver Instance and Receiver Local Address fields in the incoming message match the values of the Sender Instance and Sender Local Address used in outgoing Hello SYN, Hello SYNACK, and Hello ACK messages.

  SYN A Hello SYN message has been received.

  SYNACK A Hello SYNACK message has been received.

  ACK A Hello ACK message has been received.

  && Represents the logical AND operation

  || Represents the logical OR operation

  ! Represents the logical negation (NOT) operation.

- A timer is required for the periodic generation of Hello SYN, Hello SYNACK, and Hello ACK messages. The value of the timer is announced in the Timer field. To avoid synchronization effects, uniformly distributed random jitter of +/-5% of the Timer field SHOULD be added to the actual interval used for the timer.

  There are two independent events: the timer expires, and a packet arrives. The processing rules for these events are:

```
Timer Expires:  Reset Timer
                If state = SYNSENT Send SYN message
                If state = SYNRCVD Send SYNACK message
                If state = ESTAB   Send ACK message
```

```
            Packet Arrives:
                If incoming message is an RSTACK message:
                    If (A && C && !SYNSENT) Reset the link
                    Else discard the message.
                If incoming message is a SYN, SYNACK, or ACK message:
                    Response defined by the following State Tables.
                If incoming message is any other PRoPHET TLV and
                    state != ESTAB:
                    Discard incoming message.
                    If state = SYNSENT Send SYN message(Note 1)
                    If state = SYNRCVD Send SYNACK message(Note 1)

           Note 1: No more than two SYN or SYNACK messages should be
           sent within any time period of length defined by the timer.
```

- A connection across a link is considered to be achieved when the protocol reaches the ESTAB state. All TLVs, other than Hello TLVs, that are received before synchronization is achieved will be discarded.

5.2.1. Hello Procedure State Tables

State: SYNSENT

Condition	Action	New State
SYNACK && C	Update Peer Verifier; Send ACK message	ESTAB
SYNACK && !C	Send RSTACK message	SYNSENT
SYN	Update Peer Verifier; Send SYNACK message	SYNRCVD
ACK	Send RSTACK message	SYNSENT

State: SYNRCVD

Condition	Action	New State
SYNACK && C	Update Peer Verifier; Send ACK message	ESTAB
SYNACK && !C	Send RSTACK message	SYNRCVD
SYN	Update Peer Verifier; Send SYNACK message	SYNRCVD
ACK && B && C	Send ACK message	ESTAB
ACK && !(B && C)	Send RSTACK message	SYNRCVD

State: ESTAB

Condition	Action	New State
SYN || SYNACK	Send ACK message (notes 2 and 3)	ESTAB
ACK && B && C	Send ACK message (note 3)	ESTAB
ACK && !(B && C)	Send RSTACK message	ESTAB

Note 2: No more than two ACK messages should be sent within any time period of length defined by the timer. Thus, one ACK message MUST be sent every time the timer expires. In addition, one further ACK message may be sent between timer expirations if the incoming message is a SYN or SYNACK. This additional ACK allows the Hello functions to reach synchronization more quickly.

Note 3: No more than one ACK message should be sent within any time period of length defined by the timer.

5.3. Information Exchange Phase

After the Hello messages have been exchanged, and the nodes are in the ESTAB state, the Information Exchange Phase, consisting of the RIB Exchange and Bundle Passing Sub-Phases, is initiated. This section describes the procedure and shows the state transitions

necessary in these sub-phases; the following sections describe in detail the various TLVs passed in these phases. On reaching the ESTAB state in the high-level HELLO state, there is an automatic transition to the INFO_EXCH high-level state.

PRoPHET runs over a bidirectional transport as documented in Section 1.2 so that when a pair of nodes (A and B) have reached the ESTAB state, they are able to perform the Information Exchange Phase processes for both the A-to-B and B-to-A directions over the link that has just been established. In principle, these two processes are independent of each other and can be performed concurrently. However, complete concurrency may not be the most efficient way to implement the complete process. As explained in Section 3.2.1, the Routing Information Base Dictionary is a shared resource assembled from a combination of information generated locally on each node and information passed from the peer node. Overlaps in this information, and hence the amount of information that has to be passed between the nodes, can be minimized by sequential rather than concurrent operation of the dictionary generation and update processes. It may also be possible to reduce the number of bundles that need to be offered by the second offeror by examining the offers received from the first offeror -- there is no need for the second offeror to offer a bundle that is already present in the first offeror’s offer list, as it will inevitably be refused.

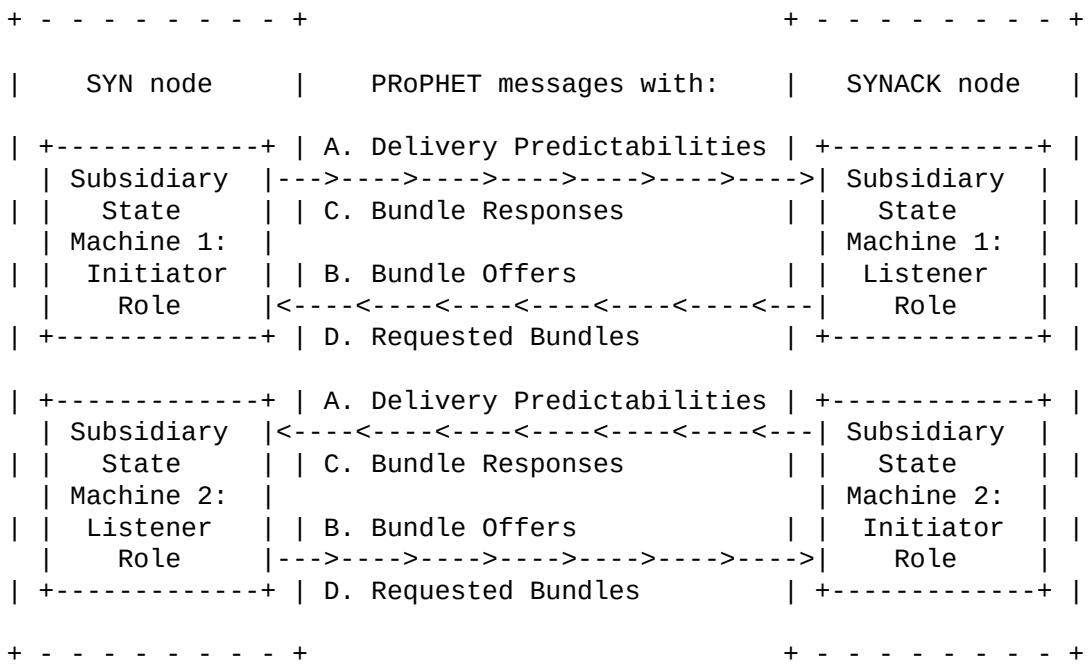
All implementations MUST be capable of operating in a fully concurrent manner. Each implementation needs to define a policy, which SHOULD be configurable, as to whether it will operate in a concurrent or sequential manner during the Information Exchange Phase. If it is to operate sequentially, then further choices can be made as to whether to interleave dictionary, offer, and response exchange parts, or to complete all parts in one direction before initiating the other direction.

Sequential operation will generally minimize the amount of data transferred across the PRoPHET link and is especially appropriate if the link is half-duplex. However it is probably not desirable to postpone starting the information exchange in the second direction until the exchange of bundles has completed. If the contact between the nodes ends before all possible bundles have been exchanged, it is possible that postponing the start of bundle exchange in the second direction can lead to bundle exchange being skewed in favor of one direction over the other. It may be preferable to share the available contact time and bandwidth between directions by overlapping the Information Exchange Phases and running the actual bundle exchanges concurrently if possible. Also, if encounters expected in the current PRoPHET zone are expected to be relatively short, it MAY not be appropriate to use sequential operation.

One possible interleaving strategy is to alternate between sending from the two nodes. For example, if the Hello SYN node sends its initial dictionary entries while the Hello SYNACK node waits until this is complete, the Hello SYNACK node can then prune its proposed dictionary entries before sending in order to avoid duplication. This approach can be repeated for the second tranche of dictionary entries needed for the Bundle Offers and Responses, and also for the Bundle Offers, where any bundles that are offered by the Hello SYN node that are already present in the Hello SYNACK node need not be offered to the Hello SYN node. This approach is well suited to a transport protocol and physical medium that is effectively half-duplex.

At present, the decision to operate concurrently or sequentially is purely a matter of local policy in each node. If nodes have inconsistent policies, the behavior at each encounter will depend on which node takes the SYN role; this is a matter of chance depending on random timing of the start of communications during the encounter.

To manage the information transfer, two subsidiary state machines are created in each node to control the stages of the RIB Exchange Sub-Phase and Bundle Passing Sub-Phase processes within the INFO_EXCH high-level state as shown in Figure 12. Each subsidiary state machine consists of two essentially independent components known as the "Initiator role" and the "Listener role". One of these components is instantiated in each node. The Initiator role starts the Information Exchange Phase in each node and the Listener role responds to the initial messages, but it is not a passive listener as it also originates messages. The transition from the ESTAB state is a "forking" transition in that it starts both subsidiary state machines. The two subsidiary state machines operate in parallel for as long as the neighbor remains in range and connected.



The letters (A - D) indicate the sequencing of messages.

Figure 12: Information Exchange Phase Subsidiary State Machines

These subsidiary state machines can be thought of as mirror images: for each state machine, one node takes on the Initiator role while the other node takes on the Listener role. TLVs sent by a node from the Initiator role will be processed by the peer node in the Listener role and vice versa. As indicated in Figure 12, the Initiator role handles sending that node's current set of delivery predictabilities for known destinations to the Listener role node. The Listener role node uses the supplied values to update its delivery predictabilities according to the update algorithms described in Section 2.1.2. It then decides which bundles that it has in store should be offered for transfer to the Initiator role node as a result of comparing the local predictabilities and those supplied by the Initiator node. When these offers are delivered to the Initiator role node, it decides which ones to accept and supplies the Listener role node with a prioritized list of bundles that it wishes to accept. The Listener role node then sends the requested bundles.

These exchanges are repeated periodically for as long as the nodes remain in contact. Additionally, if new bundles arrive from other sources, they may be offered, accepted, and sent in between these exchanges.

The PRoPHET protocol is designed so that in most cases the TLV type determines the role in which it will be processed on reception. The only exception to this is that both roles may send RIB Dictionary TLVs: the Initiator role sends dictionary entries for use in the subsequent RIB TLV(s), and the Listener role may send additional dictionary entries for use in subsequent Bundle Offer TLVs. The two cases are distinguished by a TLV flag to ensure that they are processed in the right role context on reception. If this flag was not provided, there are states where both roles could accept the RIB Dictionary TLV, making it impossible to ensure that the correct role state machine accepts the RIB Dictionary TLV. Note that the correct updates would be made to the dictionary whichever role processed the TLV and that the ambiguity would not arise if the roles are adopted completely sequentially, i.e., if the RIB Exchange Sub-Phase and associated Bundle Passing Sub-Phase run to completion in one direction before the process for the reverse direction is started.

If sequential operation is selected, the node that sent the Hello SYN function message MUST be the node that sends the first message in the Information Exchange Phase process. This ensures that there is a well-defined order of events with the Initiator role in the Hello SYN node (i.e., the node identified by String ID 0) starting first. The Hello SYNACK node MAY then postpone sending its first message until the Listener role state machine in the Hello SYNACK node has reached any of a number of points in its state progression according to locally configured policy and the nature of the physical link for the current encounter between the nodes as described above. If concurrent operation is selected, the Hello SYNACK node can start sending messages immediately without waiting to receive messages from the peer.

The original design of the PRoPHET protocol allowed it to operate over unreliable datagram-type transports as well as the reliable, in-order delivery transport of TCP that is currently specified. When running over TCP, protocol errors and repeated timeouts during the Information Exchange Phase SHOULD result in the connection being terminated.

5.3.1. State Definitions for the Initiator Role

The state machine component with the Initiator role in each node starts the transfer of information from one node to its peer during the Information Exchange Phase. The process from the Initiator's point of view does the following:

- The Initiator role determines the set of delivery predictabilities to be sent to the peer node and sends RIB dictionary entries necessary to interpret the set of RIB predictability values that

are sent after the dictionary updates. On second and subsequent executions of this state machine during a single session with the same peer, there may be no RIB Dictionary entries to send. Either an empty TLV can be sent or the TLV can be omitted.

- The Initiator then waits to receive any RIB Dictionary updates followed by bundle offers from the Listener role on the peer node.

- The Initiator determines which of the bundle offers should be accepted and, if necessary, reorders the offers to suit its own priorities. The possibly reordered list of accepted bundles is sent to the peer node using one or more bundle responses.

- The peer then sends the accepted bundles to the Initiator in turn.

- Assuming that the link remains open during the bundle sending process, the Initiator signals that the Bundle Passing Sub-Phase is complete by sending a message with an empty Bundle Response TLV (i.e, with the Bundle Offer Count set to 0 and no bundle offers following the TLV header).

- When the bundle transfer is complete, the Initiator starts the Timer(next_exchange). Assuming that the connection to the neighbor remains open, when the timer expires, the Initiator restarts the Information Exchange Phase. During this period, Hello SYN messages are exchanged as keep-alives to check that the neighbor is still present. The keep-alive mechanism is common to the Initiator and Listener machines and is handled in the high-level state machine (see Section 5.1.

A timer is provided that restarts the Initiator role state machine if Bundle Offers are not received after sending the RIB. If this node receives a Hello ACK message containing an Error TLV indicating there has been a protocol problem, then the connection MUST be terminated.

The following states are used:

CREATE_DR
: The initial transition to this state from the ESTAB state is immediate and automatic for the node that sent the Hello SYN message. For the peer (Hello SYNACK sender) node, it may be immediate for nodes implementing a fully concurrent process or may be postponed until the corresponding Listener has reached a specified state if a sequential process is configured in the node policy.

The local dictionary is initialized when this state is entered for the first time from the ESTAB state. The initial state of the dictionary contains two entries: the EID of the node that sent the Hello SYN (String ID 0) and the EID of the node that sent the Hello SYNACK (String ID 1). If the peer reports via a Hello ACK message containing an Error TLV reporting a Dictionary Conflict or Bad String ID error, then the connection MUST be terminated.

The CREATE_DR state will be entered in the same way from the REQUEST state when the Timer(next_exchange) expires, signaling the start of a new round of information exchange and bundle passing.

When in this state:

* Determine the destination EIDs for which delivery predictabilities will be sent to the peer in a RIB TLV, if any. Record the prior state of the local dictionary (assuming that String IDs are numbers allocated sequentially, the state information needed is just the highest ID used before this process started) so that the process can be restarted if necessary. Update the local dictionary if any new EIDS are required; format one or more RIB Dictionary TLVs and one or more RIB TLVs and send them to the peer. If there are no dictionary entries to send, TLVs with zero entries MAY be sent, or the TLV can be omitted, but an empty RIB TLV MUST be sent if there is no data to send. The RIB Dictionary TLVs generated here MUST have the Sent by Listener flag set to 0 to indicate that they were sent by the Initiator.

* If an Error TLV indicating a Dictionary Conflict or Bad String ID is received during or after sending the RIB Dictionary TLVs and/or the RIB TLVs, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* Start a timer (known as Timer(info)) and transition to the SEND_DR state.

Note that when (and only when) running over a transport protocol such as TCP, both the RIB Dictionary and RIB information MAY be spread across multiple TLVs and messages if required by known constraints of the transport protocol or to reduce the size of memory buffers. Alternatively, the information can be formatted using a single RIB Dictionary TLV and a single RIB TLV. These TLVs may be quite large, so it may be necessary to segment the message either using the PRoPHET submessage capability or, if the transport protocol has appropriate capabilities, using those inherent capabilities. This discussion of segmentation applies to

the other states and the bundle offer and bundle response messages and will not be repeated.

If more than one RIB TLV is to be used, all but the last one have the "More RIB TLVs" flag set to 1 in the TLV flags. It is not necessary to distinguish the last RIB Dictionary TLV because the actions taken at the receiver are essentially passive (recording the contents), and the sequence is ended by the sending of the first RIB TLV.

SEND_DR

In this state, the Initiator node expects to be receiving Bundle Offers and sending Bundle Responses. The Initiator node builds a list of bundles offered by the peer while in this state:

* Clear the set of bundles offered by the peer on entry to the state.

* If the Timer(info) expires, re-send the RIB Dictionary and RIB information sent in the previous CREATE_DR state using the stored state to re-create the information. The RIB dictionary update process in the peer is idempotent provided that the mappings between the EID and the String ID in the re-sent RIB Dictionary TLVs are the same as in the original. This means that it does not matter if some of the RIB Dictionary TLVs had already been processed in the peer. Similarly, re-sending RIB TLVs will not cause a problem.

* If a message with a RIB Dictionary TLV marked as sent by a Listener is received, update the local dictionary based on the received TLV. If any of the entries in the RIB Dictionary TLV conflict with existing entries (i.e., an entry is received that uses the same String ID as some previously received entry but the EID in the entry is different), send a Response message with an Error TLV containing a Dictionary Conflict indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer. Note that in some circumstances no dictionary updates are needed, and the first message received in this state will carry a Bundle Offer TLV.

* If a message with a Bundle Offer TLV is received, restart the Timer(info) if the "More Offer/Response TLVs Following" flag is set in the TLV; otherwise, stop the Timer(info). Then process any PRoPHET ACKs in the TLV by informing the bundle protocol agent, and add the bundles offered in the TLV to the set of bundles offered. If the "More Offer/Response TLVs Following" flag is set in the TLV, wait for further Bundle Offer TLVs. If a Bundle Offer TLV is received with a String ID that is not in

the dictionary, send a message with an Error TLV containing a Bad String ID indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* If the "More Offer/Response TLVs Following" flag is clear in the last Bundle Offer TLV received, inspect the set of bundles offered to determine the set of bundles that are to be accepted using the configured queueing policy. Record the set of bundles accepted so that reception can be checked in the Bundle Passing Sub-Phase. Format one or more Bundle Response TLVs flagging the accepted offers and send them to the peer. If more than one Bundle Response TLV is sent, all but the last one should have the "More Offer/Response TLVs Following" flag set to 1. At least one Bundle Response TLV MUST be sent even if the node does not wish to accept any of the offers. In this case, the Bundle Response TLV contains an empty set of acceptances.

* If an Error TLV indicating a Bad String ID is received during or after sending the Bundle Response TLVs, abort any in-progress Initiator or Listener process, re-initialize the local dictionary, and terminate the connection to the peer.

* Restart the Timer(info) timer in case the peer does not start sending the requested bundles.

* Transition to state REQUEST.

REQUEST

In this state, the Initiator node expects to be receiving the bundles accepted in the Bundle Response TLV(s):

* Keep track of the bundles received and delete them from the set of bundles accepted.

* If the Timer(info) expires while waiting for bundles, format and send one or more Bundle Response TLVs listing the bundles previously accepted but not yet received. If more than one Bundle Response TLV is sent, all but the last one should have the "More Offer/Response TLVs Following" flag set to 1.

* If an Error TLV indicating a Bad String ID is received during or after sending the Bundle Response TLVs, abort any in-progress Initiator or Listener process, re-initialize the local dictionary, and terminate the connection to the peer.

* Restart the Timer(info) timer after each bundle is received in case the peer does not continue sending the requested bundles.

* When all the requested bundles have been received, format a Bundle Response TLV with the Bundle Offer Count set to zero and with the "More Offer/Response TLVs Following" flag cleared to 0 to signal completion to the peer node. Also, signal the Listener in this node that the Initiator has completed. If the peer node is using a sequential policy, the Listener may still be in the initial state, in which case, it needs to start a timer to ensure that it detects if the peer fails to start the Initiator state machine. Thereafter, coordinate with the Listener state machine in the same node: when the Listener has received the completion notification from the peer node and this Initiator has sent its completion notification, start Timer(next_exchange).

* If the Timer(next_exchange) expires, transition to state CREATE_DR to restart the Information Exchange Phase.

Note that if Timer(info) timeout occurs a number of times (configurable, typically 3) without any bundles being received, then this SHOULD generally be interpreted as the problem that the link to the peer is no longer functional and the session should be terminated. However, some bundles may be very large and take a long time to transmit. Before terminating the session, this state machine needs to check if a large bundle is actually being received although no new completed bundles have been received since the last expiry of the timer. In this case the timer should be restarted without sending the Bundle Response TLV. Also, if the bundles are being exchanged over a transport protocol that can detect link failure, then the session MUST be terminated if the bundle exchange link is shut down because it has failed.

5.3.2. State Definitions for the Listener Role

The state machine component with the Listener role in each node initially waits to receive a RIB Dictionary update followed by a set of RIB delivery predictabilities during the Information Exchange Phase. The process from the point of view of the Listener does the following:

- Receive RIB Dictionary updates and RIB values from the peer. Note that in some circumstances no dictionary updates are needed, and the RIBD TLV will contain no entries or may be omitted completely.

- When all RIB messages have been received, the delivery predictability update algorithms are run (see Section 2.1.2) using the values received from the Initiator node and applying any of the optional optimizations configured for this node (see Section 2.1.3).

- Using the updated delivery predictabilities and the queueing policy and forwarding strategy configured for this node (see Section 2.1.4) examine the set of bundles currently stored in the Listener node to determine the set of bundles to be offered to the Initiator and order the list according to the forwarding strategy in use. The Bundle Offer TLVs are also used to notify the peer of any PRoPHET ACKs that have been received by the Listener role node.

- Send the list of bundles in one or more bundle offers, preceded if necessary by one or more RIB dictionary updates to add any EIDs required for the source or destination EIDs of the offered bundles. These updates MUST be marked as being sent by the Listener role so that they will be processed by the Initiator role in the peer.

- Wait for the Initiator to send bundle responses indicating which bundles should be sent and possibly a modified order for the sending. Send the accepted bundles in the specified order. The bundle sending will normally be carried out over a separate connection using a suitable DTN convergence layer.

- On completion of the sending, wait for a message with an empty Bundle Response TLV indicating correct completion of the process.

- The Listener process will be notified if any new bundles or PRoPHET ACKs are received by the node after the completion of the bundle sending that results from this information exchange. The forwarding policy and the current delivery predictabilities will then be applied to determine if this information should be sent to the peer. If it is determined that one or more bundles and/or ACKs ought to be forwarded, a new set of bundle offers are sent to the peer. If the peer accepts them by sending bundle responses, the bundles and/or ACKS are transferred as previously.

- Periodically, the Initiator in the peer will restart the complete information exchange by sending a RIB TLV that may be, optionally, preceded by RIB Dictionary entries if they are required for the updated RIB.

Timers are used to ensure that the Listener does not lock up if messages are not received from the Initiator in a timely fashion. The Listener is restarted if the RIB is not received, and a Hello ACK message is sent to force the Initiator to restart. If bundle response messages are not received in a timely fashion, the Listener re-sends the bundle offers and associated dictionary updates. The following states are used:

WAIT_DICT

The Listener subsidiary state machine transitions to this state automatically and immediately from the state ESTAB in both peers. This state will be entered in the same way if the Timer(next_exchange) expires in the peer, signaling the start of a new round of information exchange and bundle passing. This will result in one or more RIB TLVs being sent to the Listener by the peer node's Initiator.

* When a RIB Dictionary TLV is received, use the TLV to update the local dictionary, start or (if it is running) restart the Timer(peer) and transition to state WAIT_RIB. If any of the entries in the RIB Dictionary TLV conflict with existing entries (i.e., an entry is received that uses the same String ID as some previously received entry, but the EID in the entry is different), send a Response message with an Error TLV containing a Dictionary Conflict indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* If a Hello ACK message is received from the peer node, transition to state WAIT_DICT and restart the process.

If multiple timeouts occur (configurable, typically 3), assume that the link is broken and terminate the session. Note that the RIB Dictionary and RIB TLVs may be combined into a single message. The RIB TLV should be passed on to be processed in the WAIT_RIB state.

WAIT_RIB

In this state, the Listener expects to be receiving one or more RIB TLVs and possibly additional RIB Dictionary TLVs.

* On entry to this state, clear the set of received delivery predictabilities.

* Whenever a new message is received, restart the Timer(peer) timer.

* If a RIB dictionary TLV is received, use it to update the local dictionary and remain in this state. If any of the entries in the RIB Dictionary TLV conflict with existing entries (i.e., an entry is received that uses the same String ID as some previously received entry, but the EID in the entry is different), send a message with an Error TLV containing a Dictionary Conflict indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* If a RIB TLV is received, record the received delivery predictabilities for use in recalculating the local delivery predictabilities. If a delivery predictability value is received for an EID that is already in the set of received delivery predictabilities, overwrite the previously received value with the latest value. If a delivery predictability value is received with a String ID that is not in the dictionary, send a message with an Error TLV containing a Bad String ID indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* When a RIB TLV is received with the "More RIB TLVs" flag cleared, initiate the recalculation of delivery predictabilities and stop the Timer(peer). Use the revised delivery predictabilities and the configured queueing and forwarding strategies to create a list of bundles to be offered to the peer node.

* Record the state of the local dictionary in case the offer procedure has to be restarted. Determine if any new dictionary entries are required for use in the Bundle Offer TLV(s). If so, record them in the local dictionary, then format and send RIB Dictionary entries in zero or more RIB Dictionary TLV messages to update the dictionary in the peer if necessary.

* Format and send Bundle Offer TLV(s) carrying the identifiers of the bundles to be offered together with any PRoPHET ACKs received or generated by this node. If more than one Bundle Offer TLV is sent, all but the last Bundle Offer TLV sent MUST have the "More Offer/Response TLVs Following" flag set to 1.

* When all Bundle Offer TLVs have been sent, start the Timer(info) and transition to state OFFER.

* If the Timer(peer) expires, send a Hello ACK TLV to the peer, restart the timer, and transition to state WAIT_DICT.

* If an Error TLV indicating a Dictionary Conflict or Bad String ID is received during or after sending the RIB Dictionary TLVs and/or the Bundle Offer TLVs, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* If a Hello ACK message is received from the peer node, transition to state WAIT_DICT and restart the process.

OFFER

In this state, the Listener expects to be receiving one or more Bundle Response TLVs detailing the bundles accepted by the Initiator node. The ordered list of accepted bundles is communicated to the bundle protocol agent, which controls sending them to the peer node over a separate connection.

* When a Bundle Response TLV is received with a non-zero count of Bundle Offers, extract the list of accepted bundles and send the list to the bundle protocol agent so that it can start transmission to the peer node. Ensure that the order of offers from the TLV is maintained. Restart the Timer(info) unless the last Bundle Response TLV received has the "More Offer/ Response TLVs Following" flag set to 0. If a Bundle Response TLV is received with a String ID that is not in the dictionary, send a message with an Error TLV containing a Bad String ID indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

* After receiving a Bundle Response TLV with the "More Offer/ Response TLVs Following" flag set to 0 stop the Timer(info) and transition to state SND_BUNDLE.

* If the Timer(info) expires, send a Hello ACK TLV to the peer, restart the timer and transition to state WAIT_DICT.

* If a Hello ACK message is received from the peer node, transition to state WAIT_DICT and restart the process.

SND_BUNDLE

In this state the Listener monitors the sending of bundles to the Initiator peer node. In the event of disruption in transmission, the Initiator node will, if possible, re-send the list of bundles that were accepted but have not yet been received. The bundle protocol agent has to be informed of any updates to the list of bundles to send (this is likely to involve re-sending one or more bundles). Otherwise, the Listener is quiescent in this state.

* When a Bundle Response TLV is received with a non-zero count of Bundle Offers, extract the list of accepted bundles and update the list previously passed to the bundle protocol agent so that it can (re)start transmission to the peer node. Ensure that the order of offers from the TLV is maintained so far as is possible. Restart the Timer(info) unless the last Bundle Response TLV received has the "More Offer/Response TLVs Following" flag set to 0. If a Bundle Response TLV is received with a String ID that is not in the dictionary, send a message with an Error TLV containing a Bad String ID indicator, abort

any in-progress Initiator or Listener process, re-initialize the local dictionary, and restart the Information Exchange Phase as if the ESTAB state had just been reached.

* After receiving a Bundle Response TLV with the "More Offer/ Response TLVs Following" flag set to 0, stop the Timer(info) and wait for completion of bundle sending.

* If the Timer(info) expires, send a Hello ACK TLV to the peer, restart the timer, and transition to state WAIT_DICT.

* If a Hello ACK message is received from the peer node, transition to state WAIT_DICT and restart the process.

* When a Bundle Response TLV is received with a zero count of Bundle Offers, the Bundle Passing Sub-Phase is complete. Notify the Initiator that the Listener process is complete and transition to state WAIT_MORE.

As explained in the Initiator state REQUEST description, depending on the transport protocol (convergence layer) used to send the bundles to the peer node, it may be necessary during the bundle sending process to monitor the liveness of the connection to the peer node in the Initiator process using a timer.

WAIT_MORE

In this state, the Listener monitors the reception of new bundles that might be received from a number of sources, including

* local applications on the node,

* other mobile nodes that connect to the node while this connection is open, and

* permanent connections such as might occur at an Internet gateway.

When the Listener is notified of received bundles, it determines if they should be offered to the peer. The peer may also re-initiate the Information Exchange Phase periodically.

* When the bundle protocol agent notifies the Listener that new bundles and/or new PRoPHET ACKs have been received, the Listener applies the selected forwarding policy and the current delivery predictabilities to determine if any of the items ought to be offered to the connected peer. If so, it carries

out the same operations as are described in the WAIT_RIB state to build and send any necessary RIB Dictionary TLVs and RIB TLVs to the Initiator in the peer.

* When all Bundle Offer TLVs have been sent, start the Timer(info) and transition to state OFFER.

* If a RIB dictionary TLV is received, use it to update the local dictionary and transition to state WAIT_RIB. If any of the entries in the RIB Dictionary TLV conflict with existing entries (i.e., an entry is received that uses the same String ID as some previously received entry, but the EID in the entry is different), send a message with an Error TLV containing a Dictionary Conflict indicator, abort any in-progress Initiator or Listener process, and terminate the connection to the peer.

Note that the RIB Dictionary and RIB TLVs may be combined into a single message. The RIB TLV should be passed on to be processed in the WAIT_RIB state.

5.3.3. Recommendations for Information Exchange Timer Periods

The Information Exchange Phase (IEP) state definitions include a number of timers. This section provides advice and recommendations for the periods that are appropriate for these timers.

Both Timer(info) and Timer(peer) are used to ensure that the state machines do not become locked into inappropriate states if the peer node does not apparently respond to messages sent in a timely fashion either because of message loss in the network or unresponsiveness from the peer. The appropriate values are to some extent dependent on the speed of the network connection between the nodes and the capabilities of the nodes executing the PRoPHET implementations. Values in the range 1 to 10 seconds SHOULD be used, with a value of 5 seconds RECOMMENDED as default. The period should not be set to too low a value, as this might lead to inappropriate restarts if the hardware is relatively slow or there are large numbers of pieces of information to process before responding. When using a reliable transport protocol such as TCP, these timers effectively provide a keep-alive mechanism and ensure that a failed connection is detected as rapidly as possible so that remedial action can be taken (if possible) or the connection shut down tidily if the peer node has moved out of range.

Timer(next_exchange) is used to determine the maximum frequency of (i.e., minimum period between) successive re-executions of the information exchange state machines during a single session between a pair of nodes. Selection of the timer period SHOULD reflect the

trade-off between load on the node processor and desire for timely forwarding of bundles received from other nodes. It is RECOMMENDED that the timer periods used should be randomized over a range from 50% to 150% of the base value in order to avoid synchronization between multiple nodes. Consideration SHOULD be given to the expected length of typical encounters and the likelihood of encounters between groups of nodes when setting this period. Base values in the range of 20 to 60 seconds are RECOMMENDED.

5.3.4. State Tables for Information Exchange

This section shows the state transitions that nodes go through during the Information Exchange Phase. State tables are given for the Initiator role and for the Listener role of the subsidiary state machines. Both nodes will be running machines in each role during the Information Exchange Phase, and this can be done either concurrently or sequentially, depending on the implementation, as explained in Section 5.3. The state tables in this section should be read in conjunction with the state descriptions in Sections 5.3.1 and 5.3.2.

5.3.4.1. Common Notation, Operations and Events

The following notation is used:

nS Node that sent the Hello SYN message.

nA Node that sent the Hello SYNACK message.

The following events are common to the Initiator and Listener state tables:

ErrDC Dictionary Conflict Error TLV received.

ErrBadSI Bad String ID Error TLV received.

HelloAck Hello ACK TLV received. This message is delivered to both Initiator and Listener roles in order to cause a restart of the Information Exchange Phase in the event of message loss or protocol problems.

InitStart	Sent by Listener role to Initiator role to signal the Initiator role to commence sending messages to peer. If the Listener instance is running in the node that sent the Hello SYN (nS), then InitStart is signaled immediately when the state is entered. For the node that sent the Hello SYNACK (nA), InitStart may be signaled immediately if the operational policy requires concurrent operation of the Initiator and Listener roles or postponed until the Listener role state machine has reached a state defined by the configured policy.
RIBnotlast	RIB TLV received with "More RIB TLVs" flag set to 1.
RIBlast	RIB TLV received with "More RIB TLVs" flag set to 0.
REQnotlast	Bundle Response TLV received with More Offer/Response TLVs Following flag set to 1.
REQlast	Bundle Response TLV received with More Offer/Response TLVs Following flag set to 0.
RIBDi	RIBD TLV received with Sent by Listener flag set to 0 (i.e., it was sent by Initiator role).
RIBDl	RIBD TLV received with Sent by Listener flag set to 1 (i.e., it was sent by Listener role).
Timeout(info)	The Timer(info) has expired.
Timeout(peer)	The Timer(peer) has expired.

Both the Initiator and Listener state tables use the following common operations:

- The "Initialize Dictionary" operation is defined as emptying any existing local dictionary and inserting the two initial entries: the EID of the node that sent the Hello SYN (String ID 0) and the EID of the node that sent the Hello SYNACK (String ID 1).

- The "Send RIB Dictionary Updates" operation is defined as:

  1. Determining what dictionary updates will be needed for any extra EIDs in the previously selected RIB entries set that are not already in the dictionary and updating the local dictionary with these EIDs. The set of dictionary updates may be empty if no extra EIDs are needed. The set may be empty even on the first execution if sequential operation has been

selected, this is the second node to start and the necessary EIDs were in the set previously sent by the first node to start.

2. Formatting zero or more RIBD TLVs for the set of dictionary updates identified in the "Build RIB Entries" operation and sends them to the peer. The RIBD TLVs MUST have the "Sent by Listener" flag set to 0 if the updates are sent by the Initiator role and to 1 if sent by the Listener role. In the case of the Initiator role, an empty RIBD TLV MUST be sent even if the set of updates is empty in order to trigger the Listener state machine.

- The "Update Dictionary" operation uses received RIBD TLV entries to update the local dictionary. The received entries are checked against the existing dictionary. If the String ID in the entry is already in use, the entry is accepted if the EID in the received entry is identical to that stored in the dictionary previously. If it is identical, the entry is unchanged, but if it is not a Response message with an Error TLV indicating Dictionary Conflict is sent to the peer in an Error Response message, the whole received RIBD TLV is ignored, and the Initiator and Listener processes are restarted as if the ESTAB state has just been reached.

- The "Abort Exchange" operation is defined as aborting any in-progress information exchange state machines and terminating the connection to the peer.

- The "Start TI" operation is defined as (re)starting the Timer(info) timer.

- The "Start TP" operation is defined as (re)starting the Timer(peer) timer.

- The "Cancel TI" operation is defined as canceling the Timer(info) timer.

- The "Cancel TP" operation is defined as canceling the Timer(info) timer.

5.3.4.2. State Tables for the Initiator Role

The rules and state tables for the Initiator role use the following operations:

- The "Build RIB Entries" operation is defined as:

  1. Recording the state of the local dictionary.

  2. Determining the set of EIDs for which RIB entries should be sent during this execution of the Initiator role state machine component. If this is a second or subsequent run of the state machine in this node during the current session with the connected peer, then the set of EIDs may be empty if no changes have occurred since the previous run of the state machine.

  3. Determining and extracting the current delivery predictability information for the set of EIDs selected.

- The "Send RIB Entries" operation formats one or more RIB TLVs with the set of RIB entries identified in the "Build RIB Entries" operation and sends them to the peer. If the set is empty, a single RIB TLV with zero entries is sent. If more than one RIB TLV is sent, all but the last one MUST have the "More RIB TLVs" flag set to 1; the last or only one MUST have the flag set to 0.

- The "Clear Bundle Lists" operation is defined as emptying the lists of bundles offered by the peer and bundles requested from the peer.

- The "Notify ACKs" operation is defined as informing the bundle protocol agent that PRoPHET ACKs has been received for one or more bundles in a Bundle Offer TLV using the Bundle Delivered interface (see Section 2.2).

- The "Record Offers" operation is defined as recording all the bundles offered in a Bundle Offer TLV in the list of bundles offers.

- The "Select for Request" operation prunes and sorts the list of offered bundles held into the list of requested bundles according to policy and the available resources ready for sending to the offering node.

- The "Send Requests" operation is defined as formatting one or more non-empty Bundle Response TLVs and sending them to the offering node. If more than one Bundle Offer TLV is sent, all but the last one MUST have the "More Offer/Response TLVs Following" flag set to 1; the last or only one MUST have the flag set to 0.

- The "Record Bundle Received" operation deletes a successfully received bundle from the list of requests.

- The "All Requests Done" operation is defined as formatting and sending an empty Bundle Offer TLV, with the "More Offer/Response TLVs Following" flag set to 0, to the offering node.

- The "Check Receiving" operation is defined as checking with the node bundle protocol agent if bundle reception from the peer node is currently in progress. This is needed in case a timeout occurs while waiting for bundle reception and a very large bundle is being processed.

- The "Start NE" operation is defined as (re)starting the Timer(next_exchange).

The following events are specific to the Initiator role state machine:

LastBndlRcvd Bundle received from peer that is the only remaining bundle in Bundle Requests List.

NotLastBndlRcvd Bundle received from peer that is not the only remaining bundle in Bundle Requests List.

OFRnotlast Bundle Offer TLV received with "More Offer/Response TLVs Following" flag set to 1.

OFRlast Bundle Offer TLV received with "More Offer/Response TLVs Following" flag set to 0

Timeout(next_exch) The Timer(next_exchange) has expired

State: CREATE_DR

Condition	Action	New State
On Entry	If previous state was ESTAB:	
	Initialize Dictionary	
	Always:	
	Build RIB Entries	
	Wait for Init Start	CREATE_DR
InitStart	Send RIB Dictionary Updates	
	Send RIB Entries	
	Start TI	SEND_DR
ErrDC	Abort Exchange	(finished)
ErrBadSI	Abort Exchange	(finished)
HelloAck	Abort Exchange	CREATE_DR

State: SEND_DR

Condition	Action	New State
On Entry	Clear Bundle Lists	SEND_DR
Timeout(info)	Send RIB Dictionary Updates Send RIB Entries (note 1)	SEND_DR
RIBDl received	Update Dictionary (note 2) If Dictionary Conflict found: Abort Exchange Else: Start TI	 CREATE_DR SEND_DR
OFRnotlast	Notify ACKs Record Offers Start TI	 SEND_DR
OFRlast	Cancel TI Notify ACKs Record Offers Select for Request Send Requests Start TI	 REQUEST
ErrDC	Abort Exchange	(finished)
ErrBadSI	Abort Exchange	(finished)
HelloAck	Abort Exchange	CREATE_DR

State: REQUEST

Condition	Action	New State
Timeout(info)	Check Receiving If bundle reception in progress: Start TI	REQUEST
	Otherwise: Send Requests Start TI (note 3)	REQUEST
NotLastBndlRcvd	Record Bundle Received Start TI	REQUEST
LastBndlRcvd	Cancel TI All Requests Done Start NE	REQUEST
Timeout(next_exch)		CREATE_DR
HelloAck	Abort Exchange	CREATE_DR

Note 1:
   No response to the RIB has been received before the timer expired, so we re-send the dictionary and RIB TLVs. If the timeout occurs repeatedly, it is likely that communication has failed and the connection MUST be terminated.

Note 2:
   If a Dictionary Conflict error has to be sent, the state machine will be aborted. If this event occurs repeatedly, it is likely that there is either a serious software problem or a security issue. The connection MUST be terminated.

Note 3:
   Remaining requested bundles have not arrived before the timer expired, so we re-send the list of outstanding requests. If the timeout occurs repeatedly, it is likely that communication has failed and the connection MUST be terminated.

5.3.4.3. State Tables for the Listener Role

The rules and state tables for the Listener role use the following operations:

- The "Clear Supplied RIBs" operation is defined as setting up an empty container to hold the set of RIBs supplied by the peer node.

- The "Record RIBs Supplied" operation is defined as:

  1. Taking the RIB entries from a received RIB TLV.

  2. Verifying that the String ID used in each entry is present in the dictionary. If not, an Error TLV containing the offending String ID is sent to the peer, and the Initiator and Listener processes are aborted and restarted as if the ESTAB state had just been reached.

  3. If all the String IDs are present in the dictionary, record the delivery predictabilities for each EID in the entries.

- The "Recalc Dlvy Predictabilities" operation uses the algorithms defined in Section 2.1.2 to update the local set of delivery predictabilities using the using the set of delivery predictabilities supplied by the peer in RIB TLVs.

- The "Determine Offers" operation determines the set of bundles to be offered to the peer. The local delivery predictabilities and the delivery predictabilities supplied by the peer are compared, and a prioritized choice of the bundles stored in this node to be offered to the peer is made according to the configured queueing policy and forwarding strategy.

- The "Determine ACKs" operation is defined as obtaining the set of PRoPHET ACKs recorded by the bundle protocol agent that need to be forwarded to the peer. The list of PRoPHET ACKs is maintained internally by the PRoPHET protocol implementation rather than the main bundle protocol agent (see Section 3.5).

- The "Determine Offer Dict Updates" operation is defined as determining any extra EIDs that are not already in the dictionary, recording the previous state of the local dictionary, and then adding the required extra entries to the dictionary.

- The "Send Offers" operation is defined as formatting one or more non-empty Bundle Offer TLVs, incorporating the sets of Offers and PRoPHET ACKs previously determined, and sending them to the peer node. If more than one Bundle Offer TLV is sent, all but the last one MUST have the "More Offer/Response TLVs Following" flag set to 1; the last or only one MUST have the flag set to 0.

- The "Record Requests" operation is defined as recording all the bundles offered in a Bundle Offer TLV in the list of bundles offers. Duplicates MUST be ignored. The order of requests in the TLVs MUST be maintained so far as is possible (it is possible that a bundle has to be re-sent, and this may result in out-of-order delivery).

- The "Send Bundles" operation is defined as sending, in the order requested, the bundles in the requested list. This requires the list to be communicated to the bundle protocol agent (see Section 2.2).

- The "Check Initiator Start Point" operation is defined as checking the configured sequential operation policy to determine if the Listener role has reached the point where the Initiator role should be started. If so, the InitStart notification is sent to the Initiator role in the same node.

The following events are specific to the Listener role state machine:

RIBnotlast RIB TLV received with "More RIB TLVs" flag set to 1.

RIBlast RIB TLV received with "More RIB TLVs" flag set to 0 and a non-zero count of RIB Entries.

REQnotlast Bundle Response TLV received with More Offer/Response TLVs Following flag set to 1.

REQlast Bundle Response TLV received with More Offer/Response TLVs Following flag set to 0 and a non-zero count of bundle offers.

REQempty Bundle Response TLV received with More Offer/Response TLVs Following flag set to 0 and a zero count of bundle offers.

State: WAIT_DICT

Condition	Action	New State
On Entry	Check Initiator Start Point	WAIT_DICT
RIBDi	Update Dictionary (note 1) If Dictionary Conflict found: Abort Exchange Else: Start TP	(finished) WAIT_RIB
HelloAck	Abort Exchange	WAIT_DICT

State: WAIT_RIB

Condition	Action	New State
On Entry	Clear Supplied RIBS	WAIT_RIB
RIBDi	Update Dictionary (note 1) If Dictionary Conflict found: Abort Exchange	(finished)
	Else: Start TP	WAIT_RIB
RIBnotlast	Record RIBS Supplied (note 2) If EID missing in dictionary: Abort Exchange	(finished)
	Else: Start TP	WAIT_RIB
RIBlast	Check Initiator Start Point Record RIBS Supplied (note 2) If EID missing in dictionary: Abort Exchange	(finished)
	Otherwise Recalc Dlvy Predictabilities Cancel TP Determine Offers Determine ACKs Determine Offer Dict Updates Send RIB Dictionary Updates Send Offers Start TI	OFFER
HelloAck	Abort Exchange	WAIT_DICT
Any Other TLV rcvd	Abort Exchange	(finished)
Timeout(peer)	Send RIB Dictionary Updates Send Offers Start TI (note 3)	OFFER

State: OFFER

Condition	Action	New State
REQnotlast	Send Bundles Start TI	OFFER
REQlast	Cancel TI Check Initiator Start Point Send Bundles	SND_BUNDLE
REQempty	Cancel TI Check Initiator Start Point	WAIT_MORE
HelloAck	Abort Exchange	WAIT_DICT
Timeout(info)	Send RIB Dictionary Updates Send Offers Start TI (note 3)	OFFER

State: SND_BUNDLE

Condition	Action	New State
REQnotlast	Send Bundles Start TI	SND_BUNDLE
REQlast	Cancel TI Send Bundles	SND_BUNDLE
REQempty	Cancel TI Check Initiator Start Point	WAIT_MORE
HelloAck	Abort Exchange	WAIT_DICT
Timeout(info)	Send RIB Dictionary Updates Send Offers Start TI (note 3)	OFFER

State: WAIT_MORE

Condition	Action	New State
More Bundles	Determine Offers Determine ACKs Determine Offer Dict Updates Send RIB Dictionary Updates Send Offers Start TI	OFFER
RIBDi	Update Dictionary (note 1) If Dictionary Conflict found: Abort Exchange Else: Start TP	(finished) WAIT_RIB
REQnotlast	Send Bundles Start TI	SND_BUNDLE
REQlast	Cancel TI Send Bundles	SND_BUNDLE
REQempty	Cancel TI Check Initiator Start Point	SND_BUNDLE
HelloAck	Abort Exchange	WAIT_DICT
Timeout(info)	Send RIB Dictionary Updates Send Offers Start TI (note 3)	OFFER

Note 1:
Both the dictionary and the RIB TLVs may come in the same PRoPHET message. In that case, the state will change to WAIT_RIB, and the RIB will then immediately be processed.

Note 2:
Send an ACK if the timer for the peering node expires. Either the link has been broken, and then the link setup will restart, or it will trigger the Information Exchange Phase to restart.

Note 3:
When the RIB is received, it is possible for the PRoPHET agent to update its delivery predictabilities according to Section 2.1.2. The delivery predictabilities and the RIB is then used together with the forwarding strategy in use to create a bundle offer TLV. This is sent to the peering node.

Note 4:
No more bundles are requested by the other node; transfer is complete.

Note 5:
No response to the bundle offer has been received before the timer expired, so we re-send the bundle offer.

5.4. Interaction with Nodes Using Version 1 of PRoPHET

There are existing implementations of PRoPHET based on draft versions of this specification that use version 1 of the protocol. There are a number of significant areas of difference between version 1 and version 2 as described in this document:

- In version 1, the delivery predictability update equations were significantly different, and in the case of the transitivity equation (Equation 3) could lead to degraded performance or non-delivery of bundles in some circumstances.

- In the current version , constraints were placed on the String IDs generated by each node to ensure that it was not possible for there to be a conflict if the IDs were generated concurrently and independently in the two nodes.

- In the current version, a flag has been added to the Routing Information Base Dictionary TLV to distinguish dictionary updates sent by the Initiator role and by the Listener role.

- In the current version, the Bundle Offer and Response TLVs have been significantly revised. The version 2 TLVs have been allocated new TLV Type numbers, and the version 1 TLVs (types 0xA2 and 0xA3) are now deprecated. For each bundle specifier, the source EID is transmitted in addition to the creation timestamp by version 2 to ensure that the bundle is uniquely identified. Version 2 also transmits the fragment payload offset and length when the offered bundle is a bundle fragment. The payload length can optionally be transmitted for each bundle (whether or not it is a fragment) to give the receiver additional information that can be useful when determining which bundle offers to accept.

- The behavior of the system after the first Information Exchange Phase has been better defined. The state machine has been altered to better describe how the ongoing operations work. This has involved the removal of the high-level state WAIT_INFO and the addition of two states in the Listener role subsidiary state machine (SND_BUNDLE and WAIT_MORE). The protocol on the wire has not been altered by this change to the description of the state machine. However, the specification of the later stages of operation was slightly vague and might have been interpreted differently by various implementers.

A node implementing version 2 of the PRoPHET protocol as defined in this document MAY ignore a communication opportunity with a node that sends a HELLO message indicating that it uses version 1, or it MAY partially downgrade and respond to messages as if it were a version 1 node. This means that the version field in all message headers MUST contain 1.

It is RECOMMENDED that the version 2 node use the metric update equations defined in this document even when communicating with a version 1 node as this will partially inhibit the problems with the transitivity equation in version 1, and that the version 2 node modify any received metrics that are greater than (1 - delta) to be (1 - delta) to avoid becoming a "sink" for bundles that are not destined for this node. Also version 1 nodes cannot be explicitly offered bundle fragments, and an exchange with a node supporting version 1 MUST use the, now deprecated, previous versions of the Bundle Offer and Response TLVs.

Generally, nodes using version 1 should be upgraded if at all possible because of problems that have been identified.

6. Security Considerations

Currently, PRoPHET does not specify any special security measures. As a routing protocol for intermittently connected networks, PRoPHET is a target for various attacks. The various known possible vulnerabilities are discussed in this section.

The attacks described here are not problematic if all nodes in the network can be trusted and are working towards a common goal. If there exist such a set of nodes, but there also exist malicious nodes, these security problems can be solved by introducing an authentication mechanism when two nodes meet, for example, using a public key system. Thus, only nodes that are known to be members of the trusted group of nodes are allowed to participate in the routing. This of course introduces the additional problem of key distribution, but that is not addressed here.

Where suitable, the mechanisms (such as key management and bundle authentication or integrity checks) and terminology specified by the Bundle Security Protocol [RFC6257] are to be used.

6.1. Attacks on the Operation of the Protocol

There are a number of kinds of attacks on the operation of the protocol that it would be possible to stage on a PRoPHET network. The attacks and possible remedies are listed here.

6.1.1. Black-Hole Attack

A malicious node sets its delivery predictabilities for all destinations to a value close to or exactly equal to 1 and/or requests all bundles from nodes it meets, and does not forward any bundles. This has two effects, both causing messages to be drawn towards the black hole instead of to their correct destinations.

1. A node encountering a malicious node will try to forward all its bundles to the malicious node, creating the belief that the bundle has been very favorably forwarded. Depending on the forwarding strategy and queueing policy in use, this might hamper future forwarding of the bundle and/or lead to premature dropping of the bundle.

2. Due to the transitivity, the delivery predictabilities reported by the malicious node will affect the delivery predictabilities of other nodes. This will create a gradient for all destinations with the black hole as the "center of gravity" towards which all bundles traverse. This should be particularly severe in connected parts of the network.

6.1.1.1. Attack Detection

A node receiving a set of delivery predictabilities that are all at or close to 1 should be suspicious. Similarly, a node that accepts all bundles and offers none might be considered suspicious. However, these conditions are not impossible in normal operation.

6.1.1.2. Attack Prevention/Solution

To prevent this attack, authentication between nodes that meet needs to be present. Nodes can also inspect the received metrics and bundle acceptances/offers for suspicious patterns and terminate communications with nodes that appear suspicious. The natural evolution of delivery predictabilities should mean that a genuine node would not be permanently ostracized even if the values lead to

termination of a communication opportunity on one occasion. The epidemic nature of PRoPHET would mean that such a termination rarely leads to non-delivery of bundles.

6.1.2. Limited Black-Hole Attack / Identity Spoofing

A malicious node misrepresents itself by claiming to be someone else. The effects of this attack are:

1. The effects of the black-hole attack listed above hold for this attack as well, with the exception that only the delivery predictabilities and bundles for one particular destination are affected. This could be used to "steal" the data that should be going to a particular node.

2. In addition to the above problems, PRoPHET ACKs will be issued for the bundles that are delivered to the malicious node. This will cause these bundles to be removed from the network, reducing the chance that they will reach their real destination.

6.1.2.1. Attack Detection

The destination can detect that this kind of attack has occurred (but it cannot prevent the attack) when it receives a PRoPHET ACK for a bundle destined to itself but for which it did not receive the corresponding bundle.

6.1.2.2. Attack Prevention/Solution

To prevent this attack, authentication between nodes that meet needs to be present.

6.1.3. Fake PRoPHET ACKs

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

6.1.3.1. Attack Prevention/Solution

If a public key cryptography system is in place, this attack can be prevented by mandating that all PRoPHET ACKs be signed by the destination. Similarly to other solutions using public key cryptography, this introduces the problem of key distribution.

6.1.4. Bundle Store Overflow

After encountering and receiving the delivery predictability information from the victim, a malicious node may generate a large number of fake bundles for the destination for which the victim has the highest delivery predictability. This will cause the victim to most likely accept these bundles, filling 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.

6.1.4.1. Attack Detection

If it is possible for the destination to figure out that the bundles it is receiving are fake, it could report that malicious actions are underway.

6.1.4.2. Attack Prevention/Solution

This attack could be prevented by requiring sending nodes to sign all bundles they send. By doing this, intermediate nodes could verify the integrity of the messages before accepting them for forwarding.

6.1.5. Bundle Store Overflow with Delivery Predictability Manipulation

A more sophisticated version of the attack in the previous section can be attempted. The effect of the previous attack was lessened since the destination node of the fake bundles existed. This caused fake bundles to be purged from the network when the destination was encountered. The malicious node may now use the transitive property of the protocol to boost the victim's delivery predictabilities for a non-existent destination. After this, it creates a large number of fake bundles for this non-existent destination and offers them to the victim. As before, these bundles will fill up the bundle storage of the victim. The impact of this attack will be greater as there is no probability of the destination being encountered and the bundles being acknowledged. Thus, they will remain in the bundle storage until they time out (the malicious node may set the timeout to a large value) or until they are evicted by the queueing policy.

The delivery predictability for the fake destination may spread in the network due to the transitivity, but this is not a problem, as it will eventually age and fade away.

The impact of this attack could be increased if multiple malicious nodes collude, as network resources can be consumed at a greater speed and at many different places in the network simultaneously.

6.2. Interactions with External Routing Domains

Users may opt to connect two regions of sparsely connected nodes through a connected network such as the Internet where another routing protocol is running. To this network, PRoPHET traffic would look like any other application-layer data. Extra care must be taken in setting up these gateway nodes and their interconnections to make sure that malicious nodes cannot use them to launch attacks on the infrastructure of the connected network. In particular, the traffic generated should not be significantly more than what a single regular user end host could create on the network.

7. IANA Considerations

Following the policies outlined in "Guidelines for Writing an IANA Considerations Section in RFCs" (RFC 5226 [RFC5226]), the following name spaces are defined in PRoPHET.

- For fields in the PRoPHET message header (Section 4.1):
  - DTN Routing Protocol Number
  - PRoPHET Protocol Version
  - PRoPHET Header Flags
  - PRoPHET Result Field
  - PRoPHET Codes for Success and Codes for Failure
- Identifiers for TLVs carried in PRoPHET messages:
  - PRoPHET TLV Type (Section 4.2)
- Definitions of TLV Flags and other flag fields in TLVs:
  - Hello TLV Flags (Section 4.3.1)
  - Error TLV Flags (Section 4.3.2)
  - Routing Information Base (RIB) Dictionary TLV Flags (Section 4.3.3)
  - Routing Information Base (RIB) TLV Flags (Section 4.3.4)
  - Routing Information Base (RIB) Flags per entry (Section 4.3.4)
  - Bundle Offer and Response TLV Flags (Section 4.3.5)

* Bundle Offer and Response B Flags per offer or response (Section 4.3.5)

The following subsections list the registries that have been created. Initial values for the registries are given below; future assignments for unassigned values are to be made through the Specification Required policy. Where specific values are defined in the IANA registries according to the specifications in the subsections below, the registry refers to this document as defining the allocation.

7.1. DTN Routing Protocol Number

The encoding of the Protocol Number field in the PRoPHET header (Section 4.1) is:

Protocol	Value	Reference
PRoPHET Protocol	0x00	This document
Unassigned	0x01-0xEF	
Private/Experimental Use	0xF0-0xFF	This document

7.2. PRoPHET Protocol Version

The encoding of the PRoPHET Version field in the PRoPHET header (Section 4.1) is:

Version	Value	Reference
Reserved (do not allocate)	0x00	This document
PRoPHET v1	0x01	This document
PRoPHET v2	0x02	This document
Unassigned	0x03-0xEF	
Private/Experimental Use	0xF0-0xFE	This document
Reserved	0xFF	

7.3. PRoPHET Header Flags

The following Flags are defined for the PRoPHET Header (Section 4.1):

Meaning	Bit Position	Reference
Unassigned	Bit 0	
Unassigned	Bit 1	
Unassigned	Bit 2	
Unassigned	Bit 3	

7.4. PRoPHET Result Field

The encoding of the Result field in the PRoPHET header (Section 4.1) is:

Result Value	Value	Reference
Reserved	0x00	This document
NoSuccessAck	0x01	This document
AckAll	0x02	This document
Success	0x03	This document
Failure	0x04	This document
ReturnReceipt	0x05	This document
Unassigned	0x06 - 0x7F	
Private/Experimental Use	0x80 - 0xFF	This document

7.5. PRoPHET Codes for Success and Codes for Failure

The encoding for Code field in the PRoPHET header (Section 4.1) for "Success" messages is:

Code Name	Values	Reference
Generic Success	0x00	This document
Submessage Received	0x01	This document
Unassigned	0x02 - 0x7F	
Private/Experimental Use	0x80 - 0xFF	This document

The encoding for Code in the PRoPHET header (Section 4.1) for "Failure" messages is:

Code Name	Values	Reference
Reserved (do not allocate)	0x00 - 0x01	This document
Unspecified Failure	0x02	This document
Unassigned	0x03 - 0x7F	
Private/Experimental Use	0x80 - 0xFE	This document
Error TLV in Message	0xFF	This document

7.6. PRoPHET TLV Type

The TLV Types defined for PRoPHET (Section 4.2) are:

Type	Value	Reference
Reserved (do not allocate)	0x00	This document
Hello TLV	0x01	This document
Error TLV	0x02	This document
Unsassigned	0x03 - 0x9F	
RIB dictionary TLV	0xA0	This document
RIB TLV	0xA1	This document
Bundle Offer (deprecated)	0xA2	This document
Bundle Response (deprecated)	0xA3	This document
Bundle Offer (v2)	0xA4	This document
Bundle Response (v2)	0xA5	This document
Unassigned	0xA6 - 0xCF	
Private/Experimental Use	0xD0 - 0xFF	This document

7.7. Hello TLV Flags

The following TLV Flags are defined for the Hello TLV (Section 4.3.1). Flag numbers 0, 1, and 2 are treated as a 3-bit unsigned integer with 5 of the 8 possible values allocated, and the other 3 reserved. The remaining bits are treated individually:

Meaning	Value	Reference
	(Flags 0, 1, and 2)	
Reserved (do not allocate)	0b000	This document
SYN	0b001	This document
SYNACK	0b010	This document
ACK	0b011	This document
RSTACK	0b100	This document
Unassigned	0b101 - 0b111 (Flags 3 - 7)	
Unassigned	Flag 3	
Unassigned	Flag 4	
Unassigned	Flag 5	
Unassigned	Flag 6	
L Flag	Flag 7	This document

7.8. Error TLV Flags

The TLV Flags field in the Error TLV (Section 4.3.2) is treated as an unsigned 8-bit integer encoding the Error TLV number. The following values are defined:

Error TLV Name	Error TLV Number	Reference
Dictionary Conflict	0x00	This document
Bad String ID	0x01	This document
Unassigned	0x02 - 0x7F	
Private/Experimental Use	0x80 - 0xFF	This document

7.9. RIB Dictionary TLV Flags

The following TLV Flags are defined for the RIB Base Dictionary TLV (Section 4.3.3):

Meaning	Bit Position	Reference
Sent by Listener	Flag 0	This document
Reserved (do not allocate)	Flag 1	This document
Reserved (do not allocate)	Flag 2	This document
Unassigned	Flag 3	
Unassigned	Flag 4	
Unassigned	Flag 5	
Unassigned	Flag 6	
Unassigned	Flag 7	

7.10. RIB TLV Flags

The following TLV Flags are defined for the RIB TLV (Section 4.3.4):

Meaning	Bit Position	Reference
More RIB TLVs	Flag 0	This document
Reserved (do not allocate)	Flag 1	This document
Reserved (do not allocate)	Flag 2	This document
Unassigned	Flag 3	
Unassigned	Flag 4	
Unassigned	Flag 5	
Unassigned	Flag 6	
Unassigned	Flag 7	

7.11. RIB Flags

The following RIB Flags are defined for the individual entries in the RIB TLV (Section 4.3.4):

Meaning	Bit Position	Reference
Unassigned	Flag 0	
Unassigned	Flag 1	
Unassigned	Flag 2	
Unassigned	Flag 3	
Unassigned	Flag 4	
Unassigned	Flag 5	
Unassigned	Flag 6	
Unassigned	Flag 7	

7.12. Bundle Offer and Response TLV Flags

The following TLV Flags are defined for the Bundle Offer and Response TLV (Section 4.3.5):

Meaning	Bit Position	Reference
More Offer/Response TLVs Following	Flag 0	This document
Unassigned	Flag 1	
Unassigned	Flag 2	
Unassigned	Flag 3	
Unassigned	Flag 4	
Unassigned	Flag 5	
Unassigned	Flag 6	
Unassigned	Flag 7	

7.13. Bundle Offer and Response B Flags

The following B Flags are defined for each Bundle Offer in the Bundle Offer and Response TLV (Section 4.3.5):

Meaning	Bit Position	Reference
Bundle Accepted	Flag 0	This document
Bundle is a Fragment	Flag 1	This document
Bundle Payload Length Included in TLV	Flag 2	This document
Unassigned	Flag 3	
Unassigned	Flag 4	
Unassigned	Flag 5	
Unassigned	Flag 6	
PRoPHET ACK	Flag 7	This document

8. Implementation Experience

Multiple independent implementations of the PRoPHET protocol exist.

The first implementation is written in Java, and has been optimized to run on the Lego MindStorms platform that has very limited resources. Due to the resource constraints, some parts of the protocol have been simplified or omitted, but the implementation contains all the important mechanisms to ensure proper protocol operation. The implementation is also highly modular and can be run on another system with only minor modifications (it has currently been shown to run on the Lego MindStorms platform and on regular laptops).

Another implementation is written in C++ and runs in the OmNet++ simulator to enable testing and evaluation of the protocol and new features. Experience and feedback from the implementers on early versions of the protocol have been incorporated into the current version.

An implementation compliant to an Internet-Draft (which was posted in 2006 and eventually evolved into this RFC) has been written at Baylor University. This implementation has been integrated into the DTN2 reference implementation.

An implementation of the protocol in C++ was developed by one of the authors (Samo Grasic) at Lulea University of Technology (LTU) as part of the Saami Networking Connectivity project (see Section 9) and continues to track the development of the protocol. This work is now

part of the Networking for Communications Challenged Communities (N4C) project and is used in N4C testbeds.

9. Deployment Experience

During a week in August 2006, a proof-of-concept deployment of a DTN system, using the LTU PRoPHET implementation for routing was made in the Swedish mountains -- the target area for the Saami Network Connectivity project [ccnc07] [doria_02]. Four fixed camps with application gateways, one Internet gateway, and seven mobile relays were deployed. The deployment showed PRoPHET to be able to route bundles generated by different applications such as email and web caching.

Within the realms of the SNC and N4C projects, multiple other deployments, both during summer and winter conditions, have been done at various scales during 2007-2010 [winsdr08].

An implementation has been made for Android-based mobile telephones in the Bytewalla project [bytewalla].

10. Acknowledgements

The authors would like to thank Olov Schelen and Kaustubh S. Phanse for contributing valuable feedback regarding various aspects of the protocol. We would also like to thank all other reviewers and the DTNRG chairs for the feedback in the process of developing the protocol. The Hello TLV mechanism is loosely based on the Adjacency message developed for RFC 3292. Luka Birsa and Jeff Wilson have provided us with feedback from doing implementations of the protocol based on various preliminary versions of the document. Their feedback has helped us make the document easier to read for an implementer and has improved the protocol.

11. References

11.1. Normative References

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

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

11.2. Informative References

[CLAYER] Demmer, M., Ott, J., and S. Perreault, "Delay Tolerant Networking TCP Convergence Layer Protocol", Work in Progress, August 2012.

[RFC1058] Hedrick, C., "Routing Information Protocol", RFC 1058, June 1988.

[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.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

[bytewalla] Prasad, M., "Bytewalla 3: Network architecture and PRoPHET implementation", Bytewalla Project, KTH Royal Institute of Technology, Stockholm, Sweden, October 2010, <http://www.bytewalla.org/sites/bytewalla.org/files/Bytewalla3_Network_architecture_and_PRoPHET_v1.0.pdf>.

[ccnc07] Lindgren, A. and A. Doria, "Experiences from Deploying a Real-life DTN System", Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007), Las Vegas, Nevada, USA, January 2007.

[doria_02] Doria, A., Uden, M., and D. Pandey, "Providing connectivity to the Saami nomadic community", Proceedings of the 2nd International Conference on Open Collaborative Design for Sustainable Innovation (dyd 02), Bangalore, India, December 2002.

[lindgren_06] Lindgren, A. and K. Phanse, "Evaluation of Queueing Policies and Forwarding Strategies for Routing in Intermittently Connected Networks", Proceedings of COMSWARE 2006, January 2006.

[vahdat_00] Vahdat, A. and D. Becker, "Epidemic Routing for Partially Connected Ad Hoc Networks", Duke University Technical Report CS-200006, April 2000.

[winsdr08] Lindgren, A., Doria, A., Lindblom, J., and M. Ek, "Networking in the Land of Northern Lights - Two Years of Experiences from DTN System Deployments", Proceedings of the ACM Wireless Networks and Systems for Developing Regions Workshop (WiNS-DR), San Francisco, California, USA, September 2008.

Appendix A. PRoPHET Example

To help grasp the concepts of PRoPHET, an example is provided to give an understanding of the transitive property of the delivery predictability and the basic operation of PRoPHET. In Figure 13, we revisit the scenario where node A has a message it wants to send to node D. In the bottom right corner of subfigures a-c, the delivery predictability tables for the nodes are shown. Assume that nodes C and D encounter each other frequently (Figure 13a), making the delivery predictability values they have for each other high. Now assume that node C also frequently encounters node B (Figure 13b). Nodes B and C will get high delivery predictability values for each other, and the transitive property will also increase the value B has for D to a medium level. Finally, node B meets node A (Figure 13c), which has a message for node D. Figure 13d shows the message exchange between node A and node B. Summary vectors and delivery predictability information is exchanged, delivery predictabilities are updated, and node A then realizes that P_(b,d) > P_(a,d), and thus forwards the message for node D to node B.

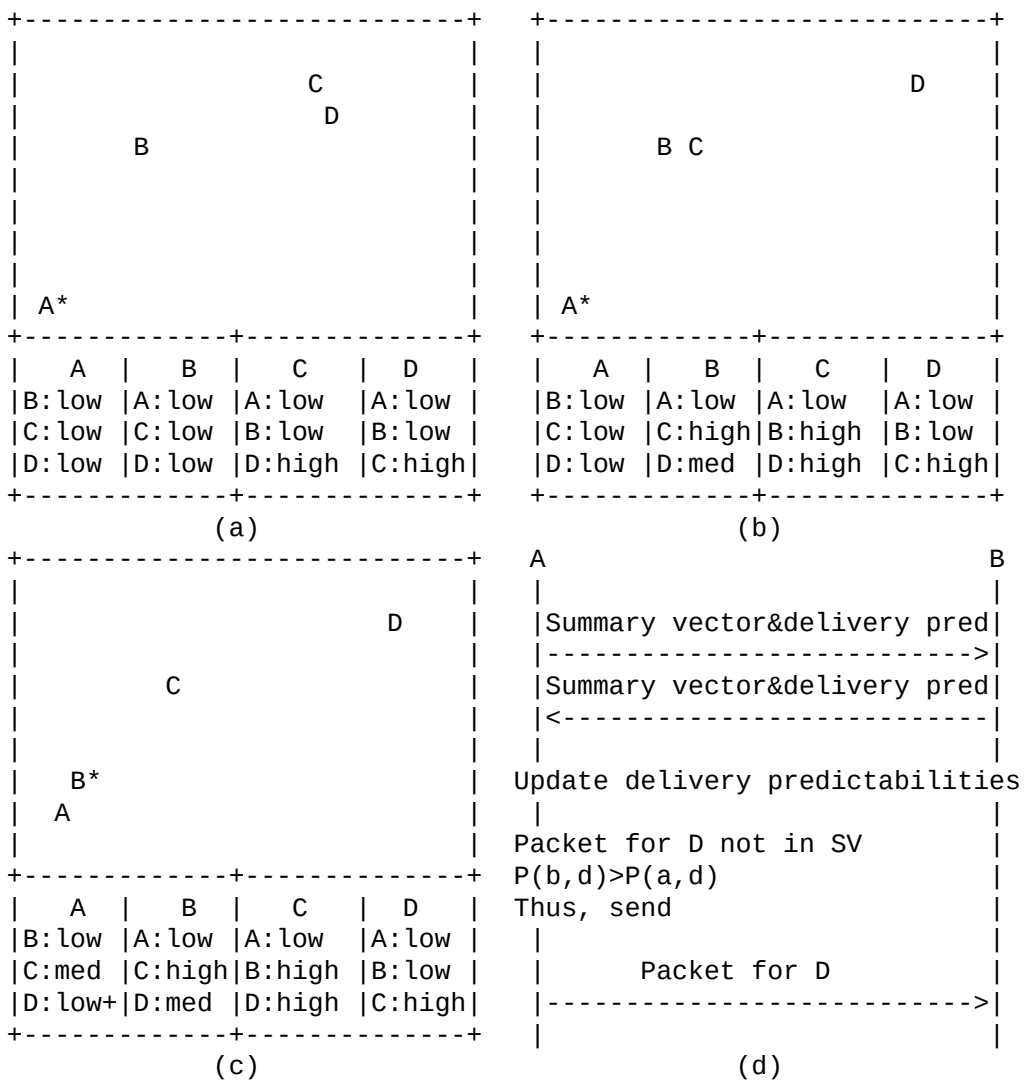


Figure 13: PRoPHET example

Appendix B. Neighbor Discovery Example

This section outlines an example of a simple neighbor discovery protocol that can be run in-between PRoPHET and the underlying layer in case lower layers do not provide methods for neighbor discovery. It assumes that the underlying layer supports broadcast messages as would be the case if a wireless infrastructure was involved.

Each node needs to maintain a list of its active neighbors. The operation of the protocol is as follows:

1. Every BEACON_INTERVAL milliseconds, the node does a local broadcast of a beacon that contains its identity and address, as well as the BEACON_INTERVAL value used by the node.

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

   A. The sending node is already in the list of active neighbors. Update its entry in the list with the current time, and update the node's BEACON_INTERVAL if it has changed.

   B. 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 and the node's BEACON_INTERVAL. Notify the PRoPHET agent that a new neighbor is available ("New Neighbor", as described in Section 2.4).

3. If a beacon has not been received from a node in the list of active neighbors within a time period of NUM_ACCEPTED_LOSSES * BEACON_INTERVAL (for the BEACON_INTERVAL used by that node), 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, and the PRoPHET agent should be notified that a neighbor has left ("Neighbor Gone", as described in Section 2.4).

Appendix C. PRoPHET Parameter Calculation Example

The evolution of the delivery predictabilities in a PRoPHET node is controlled by three main equations defined in Section 2.1.2. These equations use a number of parameters that need to be appropriately configured to ensure that the delivery predictabilities evolve in a way that mirrors the mobility model that applies in the PRoPHET zone where the node is operating.

When trying to describe the mobility model, it is more likely that the model will be couched in terms of statistical distribution of times between encounters and times to deliver a bundle in the zone. In this section, one possible way of deriving the PRoPHET parameters

from a more usual description of the model is presented. It should be remembered that this may not be the only solution, and its appropriateness will depend both on the overall mobility model and the distribution of the times involved. There is an implicit assumption in this work that these distributions can be characterized by a normal-type distribution with a well-defined first moment (mean). The exact form of the distribution is not considered here, but more detailed models may wish to use more specific knowledge about the distributions to refine the derivation of the parameters.

To characterize the model, we consider the following parameters:

P1 The time resolution of the model.

P2 The average time between encounters between nodes, I_typ, where the identity of the nodes is not taken into account.

P3 The average number of encounters that a node has between meeting a particular node and meeting the same node again.

P4 The average number of encounters needed to deliver a bundle in this zone.

P5 The multiple of the average number of encounters needed to deliver a bundle (P4) after which it can be assumed that a node is not going to encounter a particular node again in the foreseeable future so that the delivery predictability ought to be decayed below P_first_threshold.

P6 The number of encounters between a particular pair of nodes that should result in the delivery predictability of the encountered node getting close to the maximum possible delivery predictability (1 - delta).

We can use these parameters to derive appropriate values for gamma and P_encounter_max, which are the key parameters in the evolution of the delivery predictabilities. The values of the other parameters P_encounter_first (0.5), P_first_threshold (0.1), and delta (0.01), with the default values suggested in Figure 3, generally are not specific to the mobility model, although in special cases P_encounter_first may be different if extra information is available.

To select a value for gamma:
After a single, unrepeated encounter, the delivery predictability of the encountered node should decay from P_encounter_first to P_first_threshold in the expected time for P4 * P5 encounters. Thus:

P_first_threshold = P_encounter_first * gamma ^ ((P2 * P4 * P5)/P1)

which can be rearranged as

gamma =
exp(ln(P_first_threshold/P_encounter_first) * P1 / (P2* P4 * P5)).

Typical values of gamma will be less than 1, but very close to 1 (usually greater than 0.99). The value has to be stored to several decimal places of accuracy, but implementations can create a table of values for specific intervals to reduce the amount of on-the-fly calculation required.

Selecting a value for P_encounter_max:
Once gamma has been determined, the decay factor for the average time between encounters between a specific pair of nodes can be calculated:
Decay_typ = gamma ^ ((P2 * P3)/P1)

Starting with P_encounter_first, using Decay_typ and applying Equation 1 from Section 2.1.2 (P6 - 1) times, we can calculate the typical delivery predictability for the encountered node after P6 encounters. The nature of Equation 1 is such that it is not easy to produce a closed form that generates a value of P_encounter_max from the parameter values, but using a spreadsheet to apply the equation repeatedly and tabulate the results will allow a suitable value of P_encounter_max to be chosen very simply. The evolution is not very sensitive to the value of P_encounter_max, and values in the range 0.4 to 0.8 will generally be appropriate. A value of 0.7 is recommended as a default.

Once a PRoPHET zone has been in operation for some time, the logs of the actual encounters can and should be used to check that the selected parameters were appropriate and to tune them as necessary. In the longer term, it may prove possible to install a learning mode in nodes so that the parameters can be adjusted dynamically to maintain best congruence with the mobility model that may itself change over time.

Authors' Addresses

Anders F. Lindgren
Swedish Institute of Computer Science
Box 1263
Kista SE-164 29
SE

Phone: +46707177269
EMail: andersl@sics.se
URI: http://www.sics.se/~andersl

Avri Doria
Technicalities
Providence RI
US

EMail: avri@acm.org
URI: http://psg.com/~avri

Elwyn Davies
Folly Consulting
Soham
UK

EMail: elwynd@folly.org.uk

Samo Grasic
Lulea University of Technology
Lulea SE-971 87
SE

EMail: samo.grasic@ltu.se