Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 23, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.
Abstract

This document describes an extension to the naming mechanism of disruption tolerant networks (RFC4838) to support intentional naming. Intentional naming is a means by which a source node specifies the destination node(s) for a bundle in terms of predicates on attributes of the node(s), instead of by a canonical endpoint identifier (EID) of the node. Intentional naming is closely tied to the concept of binding, as described in RFC 4838. Since information required to route an intentionally named bundle may not be available at the source node, this information must be supplied at one or more subsequent nodes along the bundle’s path toward its destination(s). The architecture required for an intentional naming capability in a DTN must support the notion that a bundle can make progress toward its destination(s) in the absence of complete binding information. In this document we describe a framework for intentional naming in a DTN, propose a syntax for intentional names, and describe a distributed procedure for late or partial binding. We also present sample use cases for late binding and a notional name binding algorithm, called GRAIN, that can deliver bundles to intentional names with geographic and role attributes, e.g. "first responders within a kilometer of a specified location." Finally, we discuss the limitations in our current ability to field an ideal intentional naming system (i.e., one that can support generic intentional names), and we suggest a somewhat restrictive framework that is both useful and feasible to deploy.
Table of Contents

1. Introduction ........................................... 4
   1.1. Use Cases for Intentional Naming .................. 4
   1.2. Binding: Early and Late, Complete and Partial ...... 6
   1.3. Organization of this Document ..................... 7
2. Expressiveness of Intentional Names ........................ 8
   2.1. Namespaces and Ontologies ......................... 9
   2.2. Expressiveness of Names ............................ 9
   2.3. Expressiveness of Ontology ........................ 11
   2.4. One, Any, All .................................... 11
   2.5. Conflicts and Disambiguation ....................... 12
3. Binding of Intentional Names and Routing .................... 14
   3.1. A Note on Tunneling ................................. 16
4. Architecture ............................................. 17
   4.1. Name Knowledge Base ................................ 17
   4.2. Dissemination ...................................... 18
   4.3. Resolution .......................................... 19
   4.4. Routing ............................................. 19
5. Implementing Intentional Names on DTN ........................ 20
   5.1. Metadata Extension Block for Late Binding .......... 20
   5.2. Interaction with DTN Routing ....................... 21
   5.3. New Interfaces ...................................... 24
   5.4. Application Delivery ................................ 25
   5.5. Syntax .............................................. 25
6. GRAIN: Gradient-Based Algorithm for Intentional Naming .... 27
   6.1. Resolution and Routing Policies .................... 28
   6.2. Disseminating Location Information ................. 29
   6.3. Predicate-scoped Epidemic Flooding ................ 29
7. Deploying an Intentional Naming System ........................ 30
   7.1. Scalability ......................................... 30
   7.2. Expressiveness of Intentional Names - Revisited due to Scalability Considerations .......... 31
   7.3. Security Considerations ............................. 32
8. Acknowledgments .......................................... 34
9. Informative References ..................................... 35
Appendix A. GRAIN Pseudocode ................................. 37
   A.1. Main loop ........................................... 38
   A.2. Message processing and state management ............ 38
   A.3. Bundle processing .................................... 39
   A.4. deliver_locally_if_appropriate ...................... 39
   A.5. change_from_stem_to_flood_if_appropriate ........... 40
   A.6. do_stem_forwarding .................................. 40
   A.7. do_flood_forwarding .................................. 41
   A.8. Location dissemination ............................... 41
Authors’ Addresses ........................................... 42
1. Introduction

Delay/Disruption-Tolerant Networking (DTN) is an architecture and a set of protocols that enable communication in environments with intermittent connectivity and long delays. An architecture for DTNs [RFC4838] and a bundle protocol [RFC5050] have been previously defined. This document describes a framework for intentional naming, an extension to the naming mechanism of DTNs. In a DTN, nodes and services can appear, move, and disappear dynamically; not only is it impossible for a node to have complete information about the state of the names, addresses, and routes in the network, but the intended recipient of a bundle may not be part of the network—or even yet exist—when the bundle is created. Given such a flexible and dynamic network, it is valuable to be able to denote nodes in terms of attributes of the nodes, as well as by canonical DTN endpoint identifiers (EID).

[This is not a new idea; for example X.500 naming [ISO.9594-1.1988] is expressed in terms of attribute-value pairs. Also, MIT’s Intentional Naming System has proposed similar ideas [INS]. However, the application of these ideas to DTNs is new.]

We use the term "canonical EID of a DTN node" to refer to the EID of a bundle processing entity (typically a DTN routing module executing on that node) that is capable of receiving bundles addressed to that EID from other DTN nodes. Every DTN node is expected to possess a canonical EID.

An intentional name [INS] is a name defined in terms of predicates on node attributes.

Since intentional naming is so closely tied to the concept of binding, we will carefully define that and related concepts. For now we note a major difference between a DTN and a conventional (e.g. IPv4) network. In a conventional network, the work of mapping from a name to an address, and the process of routing a packet to that address, are separable. That is not typically possible in the dynamic environment of most DTNs.

1.1. Use Cases for Intentional Naming

The following are some example use cases for intentional naming.

1. Addressing by heterogeneous network names. There are several namespaces currently in use in the public Internet as well as in other internets. These include the IP address space (IPv4 or IPv6), the DNS namespace, Email addresses, naming for non-IP based internets (telephone numbers, interplanetary internet,
wireless sensor networks), and naming used by other commercial or military networks and overlays. Since DTN could be used for bridging heterogeneous networks, a sender may want to address a bundle to "the node with IPv6 address A" even if the sender is not running an IPv6 stack or if a portion of the path to the destination is not an IPv6 network.

2. Location-based addressing. Destinations can be named in terms of spatial coordinates, e.g. (latitude, longitude, altitude). Resolving such a name may require nodes to be aware of both their own location and the location of other nodes. This form of addressing would be useful when notifying all users (via their nodes) in a region of an emergency, or for advertising a service available in a particular area.

3. Descriptive location-based addressing. Names can be expressed in terms of physical addresses, or proximity to a well-known object or feature, e.g., "node at 10 Moulton Street, Cambridge, MA", "node within 500 meters of bridge at (latitude, longitude)". In addition to node location information, resolving such a name would require a geographical information system or other cartographic information. This name scheme would be useful like the previous example, but has the added advantage of using a descriptive name.

4. Destination names can also be expressed in terms of membership in a group ("all soldiers that are members of Charlie company") or some combination of a role and a location ("all first responders within one kilometer of (latitude, longitude)", "all with rank lieutenant or higher in building 17"). In order to resolve such names, access to group membership information, and knowledge of the military hierarchy is necessary. If membership in a particular group implies the ability or function of performing a particular task, this naming scheme may be described as "role-based".

5. Authorization based addressing. Names may be of the form "all personnel with Top Secret clearance." Similar to role-based addressing, this example has additional implications for security. To resolve such a name we need access to security clearance or access control information at some point in the delivery process.

6. Node-attribute based addressing. Names can be of the form "all nodes with more than 100GB of free storage", "any node with SATCOM capability", "node with temperature > 45 degrees C", or "node with temperature > max safe operating temperature". In order to resolve such names, we need access to information about
the capabilities of the nodes. This naming scheme is similar to role-based naming, but instead of basing the delivery on the node owner’s role, delivery is based on the attributes of the node itself. In each of these cases we want to deliver the bundle to a node that has particular attributes (location, temperature, authorization) without knowing the node’s canonical EID.

1.2. Binding: Early and Late, Complete and Partial

We use the term binding as it is defined in RFC4838: "Binding means interpreting the SSP [scheme-specific part] of an EID for the purpose of carrying an associated message towards a destination."

In an IP network binding occurs at two layers: i) the translation (using DNS) of a network name (e.g. yahoo.com) to an IP address, which is performed at the source node (The term early binding is often applied to this process.) and ii) the translation at each hop (using ARP) of the next hop’s IP address to a MAC address. Both of these binding operations result in the encapsulation of the packet and moving to a lower layer of the network stack. As noted above, these translation processes are completely separate from the routing processes (which occur at the next lower layers of the stack). This type of binding occurs within a DTN as well, as bundles are conveyed across a convergence layer to a DTN neighbor.

But intentional names (and other EIDs as well) may require additional information beyond the traditional cross-layer binding information (address) supplied to the lower layer router. We want to provide routing information at successive hops that will be used by the DTN router to deliver the bundle. At some point the node-specific EID(s) of the destinations may be completely known. At that time we say complete binding has occurred. If that happens at the source we call it early binding. If it occurs at a later node along the bundle’s path, we call it late binding. We anticipate that in many cases complete binding will never occur, and may frequently occur after a bundle is at (or within a single hop of) its final destination.

Node attributes (such as roles, location, or sensed values) may change during the lifetime of a bundle, and the knowledge of these attributes may be highly distributed and not universally known. Therefore, the nodes that match the intentional name predicate(s) may change during the bundle’s lifetime. We sometimes use the term "name resolution" as a synonym for "name binding". In order to route the bundle to its destination, we have to progressively resolve (i.e. provide routing information related to) the intentional name. When a node receives a bundle addressed to an intentional name, the node examines the intentional name, and uses its knowledge of the name to provide hints for DTN routing. This notion is referred to as partial
resolution or partial binding. We use any portion of the intentional name from which we can extract routing information, record that information, and hand the bundle to a DTN router located on the node. As it is progressively determined, routing information will be recorded in the bundle’s header (metadata extension block) and used by the DTN router. Frequently this information takes the form of next-hop nodes (sometimes called "care-of" nodes). The routing information provided at various nodes is intended to move the bundle "closer" to its final destination or to a node that can perform additional resolution.

A key (and novel) benefit of late binding of intentional names in a DTN is persistent bundle delivery to descriptively named groups. Since DTN nodes persistently store and deliver information, bundles intended for recipients that match a description can be delivered to those recipients, even if the recipients did not satisfy the description at the time the bundle was created. For example, a node may transmit a bundle with a one-day expiration period, addressed to all nodes in a specified region (e.g. building or neighborhood); the bundle is delivered to all nodes in the region currently, as well as all nodes that enter the region during the next day (until the bundle expires).

1.3. Organization of this Document

In the remainder of this document we describe the requirements and a framework for intentional name services in a DTN. We first discuss namespaces and ontologies, and motivate the degree of expressiveness needed for intentional names. We discuss how name binding and routing of bundles will work together, and describe an architecture for achieving that. We then describe how intentional naming will integrate into the DTN architecture, and give an example (GRAIN) of a special type of intentional name and a resolution algorithm for such names which has been prototyped on the DTN reference implementation. We conclude with an examination of some practical limitations to the successful deployment of an intentional naming capability in a DTN, and we suggest a more limited framework, whose deployment is both feasible and useful.
2. Expressiveness of Intentional Names

Intentional naming provides a means of describing a destination whose name we do not [perhaps cannot] know. It is, therefore, desirable for intentional names to be as expressive as possible, within the constraints of the system which implements them. As we argue below for greater expressiveness we should keep in mind the possible constraints that may eventually limit our choices. We list a few of those below.

1. Computational power. Expressiveness comes at the cost of computations. DTN nodes with greater computational capability will be able to handle more expressive intentional names. For a given name complexity, network nodes with adequate computational power must be present in sufficient numbers and locales.

2. Storage. Nodes must be able to deal with potentially large amounts of data - often in a persistent manner [e.g. through loss of power].

3. Network capacity. Conveying information about intentional names will necessarily consume some of the network’s capacity. Low capacity networks will have less capacity to devote for supporting intentional naming. Same is true for networks with large capacity and equally large data transmission demands.

4. Routing complexity. Interpreting intentional names (binding) must be closely tied to the DTN routing strategy. All nodes must be able to support a routing strategy that is consistent with the complexity of the intentional name space.

5. Locality of intentional naming information. To the extent that information about intentional names is spread across a large network, more demands are placed on the DTN in all of the above areas - especially as it relates to scaling.

6. Connectedness. Nodes in networks that are not well connected are less likely to be able to obtain useful binding information, or get to use it when available.

7. Security and privacy. Complex intentional names may affect how we handle security issues. And conversely, security considerations may constrain the domain of feasible intentional names. Some general privacy concerns include what nodes, etc, have the right to access the information needed to resolve an intentional name, how long is the information kept, how is it distributed etc.?
With those concerns in mind, we now suggest the desire for a very expressive set of intentional names, while keeping in mind our likely need to be more restrictive when it comes time to think about deploying such a system. We will discuss such tradeoffs when we define an implementation strategy for intentional naming.

2.1. Namespaces and Ontologies

An ontology is a representation of a set of concepts within a domain and the relationships between those concepts. In terms of node names and descriptions, an ontology includes both the attributes that can be used to describe a node, the rules obeyed by those attributes, and the operations that can be performed on those attributes. An ontology is richer than a simple namespace -- a node’s name is one of the attributes that can be part of an ontology. A node may have descriptions in multiple ontologies simultaneously; for example, a node may have IPv4, IPv6, and DNS names (attributes), as well as be described by a location ontology (which includes latitude, longitude, altitude attributes), and an ownership ontology (which might include the notions of individuals, corporations, and how a node might belong to a corporation and be assigned to an individual). This leads us to the question of expressiveness; how powerful a naming system do we need?

2.2. Expressiveness of Names

Our goal is for intentional names to be able to express both explicitly asserted attributes of a node (e.g. the node’s location), and derived attributes of the node (e.g. the distance between the node and a particular location). The syntax that we choose for intentional names, as well as the underlying name interpretation and matching algorithms we deploy, must be powerful enough to capture and manipulate these attributes. Examples of the kinds of attributes we envision include:

1. Attributes and literal values: Using the examples above, we see that we would like to be able to specify that the destination node has an attribute (is-soldier), the attribute value matches a literal value (company == ‘Charlie’), or holds a relationship to a literal value (temperature > 45 degrees C).

2. Attribute relationships: In many cases it is valuable to be able to express relationships between attributes of node (e.g. "temperature > safe operating temperature", where the safe operating temperature itself is a node attribute and hence may differ from node to node).
3. Term combinations: We want to be able to express conjunctions ("a AND b"), disjunctions ("a OR b"), and negation ("NOT c") in intentional names.

4. Derived or inferred attributes: At times we may wish to identify the destination(s) in terms of values derived or inferred from node attributes, e.g. within-distance-of(500m,BrooklynBridge). In order to determine whether this predicate is satisfied we need to know how to perform the derivation, either in terms of primitive operations available at each node (e.g. \( \sqrt{\sqrt{\text{my.x-bridge.x}^2} + \sqrt{\text{my.y-bridge.y}^2}} < 500 \)), or in terms of predicates that are included in the ontology (an ontology that allows specification of location is likely to also include an operation for computing distance between locations).

5. Groups of nodes with distributed attributes: One may want to summon a disaster relief team located nearest to the location of the disaster. Such a team may be composed of a fire-fighting unit, medical personnel, and police. A bundle addressed to the intentional "nearest disaster relief team" should reach the nearest available nodes in each class. This requires that the above relationship rules be captured in the appropriate ontology.

The above desiderata make it clear that, say, a simple string-matching scheme is insufficient for our needs, and that something richer is needed. We propose that an intentional name be described using fragments of a Horn clause \([\text{Horn}]\) as used in logic programming. A Horn clause is of the form "term1 AND term2 AND ... termK implies term"; here "term1", "term2", etc. constitute the body and "term" constitutes the head which is true if the body evaluates to true. Logic programming syntax used to represent Horn clauses is as follows: term :- term1, term2, ..., termK.

A name (query) can be denoted by the head of a Horn clause which is a formula of the form "f(v1,v2,...,vK)"; or it could also be denoted by the body of a Horn clause, i.e., a formula which is a conjunction (AND) of logical terms, e.g., "f(v1),g(v2,v3),h(v4)". Note that v1, v2, etc. may be literals, attributes, or other terms. Also, basic terms such as "X == 2" or "V > 4" are isomorphic to "equals(X,2)" and "greaterthan(V,4)".

An ontology typically contains base facts denoted by single terms that capture relations between various attributes. Rules in an ontology will be denoted by Horn clauses.

This structure is familiar to most programmers as well as sufficiently expressive for the intentional names we envision being used. Another advantage is that a Horn clause fragment can be
expressed using a variety of syntactic forms, including an SQL-like query, a Prolog-like goal, a SPARQL query, and a C-like expression. We describe the exact name syntax in Section 5.5.

2.3. Expressiveness of Ontology

The expressiveness question applies to ontologies as well since they are responsible for expressing rules that encode the relationships between various concepts in the ontology. Simple examples of rules include:

- A large cache is one which has maximum storage > 100GB. This can be encoded as the following rule: `largeCache(C) :- maxStorageGB(C,S), S>100.`
- X is near Y if they are not more than 100m apart and there are no obstacles between them
- A disaster relief team must contain fire fighters, paramedics, and police.
- A person ready to board a flight to Dallas will request content about the weather there.
- A local resident of a city is someone who has stayed in that city for at least a year.

In general we intend to support the Description Logic (DL) subset of First Order Predicate Logic (FOPL) plus an added feature called Negation As Failure (NAF), which is a standard and useful feature in logic programming languages such as Prolog. NAF means that if a statement cannot be proved from the current knowledge base of facts (union of all knowledge bases in the network), then it must be false. This is the so-called closed world assumption.

2.4. One, Any, All

An intentional name may specify a singleton or a group, depending on how many reachable destinations meet the criteria. In such cases the sender of the bundle may wish to specify whether the bundle should be delivered to one, any, or all matching recipients. Hence, this is similar to the concept of minimum reception group (MRG) defined in [RFC4838] and [RFC5050]. We propose that the default behavior should be to deliver to all matching recipients (within bundle expiration time), and that the sender can override this behavior by adding a modifier to the intentional name.

The following examples show the difference between one, any, and all,
using notional syntax.

- one (president(BBN)): deliver the bundle to the current President of BBN. Once a copy of the bundle has been delivered to a particular node that meets the criteria, forwarding of the bundle can stop. Note that if a DTN routing protocol replicated the bundle prior to this delivery, copies of the bundle may be existing elsewhere in the network. Hence this semantics is to ensure certain local behavior.

- any (business == SushiBar AND within(1km,FenwayPark)): deliver the bundle to some (but not necessarily all) sushi bars within one kilometer of Fenway Park. The bundle should be forwarded to one or more nodes that meet the criteria. Similar to the previous example, once a bundle is delivered to a matching node, further forwarding should stop.

- all (business == SushiBar AND within(1km,FenwayPark)): deliver the bundle to all sushi bars within one kilometer of Fenway Park. The bundle should be forwarded to all nodes that meet the criteria. In this case, after delivering the bundle to a matching node, the resolution+routing algorithm may choose to continue forwarding it to other nodes in the network.

The first example brings up interesting questions about delivery semantics; the President of BBN can change over time and hence the delivery semantics here are that of ultimate late binding (i.e., if X is the current president of BBN and X receives the bundle, then delivery is legitimate. Also if Y was a president of BBN in the past when the bundle was sourced but is no longer the president when he receives the bundle, then Y is not a legitimate recipient of the bundle).

Note that because bundles may be replicated in the network (to increase the likelihood of delivery) we cannot guarantee that only one copy of the bundle is delivered. The difference between any and one is, therefore, somewhat fuzzy. An intermediate node can stop forwarding an any-bundle after sending only one, and might forward multiple replicas of a bundle marked with a one-address. In this way, one and any are in some sense advisory.

2.5. Conflicts and Disambiguation

In the examples above we implicitly used a number of different ontologies, including the IPv4 namespace, a namespace and ontology for location and distance, an ontology for temperature, and an ontology for an organization and ranks. When working with multiple namespaces, the issue of conflicts and disambiguation arises. There
are at least three approaches we can take to handle attribute name conflicts:

- **Enforced uniqueness**: all attribute and predicate names are restricted to use no more than one ontology. This requires some sort of central authority managing the overall attribute and predicate namespace, e.g. ensuring that there is only one "rank" attribute defined across all ontologies. In a closed network this might be workable, but as a DTN is, by definition, the union of multiple disparate networks, enforced uniqueness is not tenable.

- **Implicit disambiguation**: More like human language, an attribute or predicate name may be used in multiple ontologies, the only requirement being that it be clear from context how to interpret the attributes and predicates used in a name. Once again, this does not appear to be workable in a DTN. A geographic ontology might define location(latitude, longitude, altitude), and an organizational ontology might define location(campus, building, office), or define location() in terms of an individual’s position in the hierarchy.

- **Explicit disambiguation**: Each use of an attribute or predicate is explicitly linked to its ontology. This might be done in any number of ways, such as by prepending the name of the ontology to the rest of the name string, using a lexical scoping mechanism such as the following: geographic:within-distance(500m,location(-40.1,+30.7,0)) Here, the term "geographic" refers to the name of the geographic ontology that contains definitions of rules such as "within-distance". Given the above, we believe that explicit disambiguation is the best solution, and that is what we propose. (We will still need to disambiguate between ontology names, but this can be done using well-known techniques such as including the creator identity and timestamp in the ontology name).
3. Binding of Intentional Names and Routing

An intentional name is a description of a destination (which may be more than one node). The goal of routing is to deliver a bundle to the destination. For the router to do its job, it must be able to use the intentional name (more precisely the information derived from that name at this or previous nodes), along with other local information, to make an informed decision about how to forward the bundle. Thus routing must be "in on the game" of intentional naming. The simplest solution appears to be the early binding case, wherein the source node identifies one or more canonical EIDs to which this bundle shall be sent. Typically, the relevant EIDs will be placed in the metadata extension block, and we assume the router will be able to deliver the bundle once it knows the EID.

Curiously, it may not always be the case that completely resolved intentional names are easier to route than unresolved names. Consider the case of a bundle headed for a node at a specific location (so described with an intentional name). If I know recent locations of several network nodes, geographic routing [getting it to a closer location than me] may well be more feasible than routing based on a node name - especially if I haven’t heard from that node recently.

The resolution process is performed on each node by a system entity called a resolver. The router and resolver must work together to determine the disposition of each bundle with an intentional name. Under certain circumstances it may well be useful to complete a binding of an intentional name (to a canonical EID) to make the routing decision simpler. But, as we have seen above, it may not always be possible or useful to complete the binding at source. How the router/resolver ought to treat an intentionally named bundle is, perhaps, the most complicated part of the intentional naming process. And we will find that it drives most of the decisions about how to deploy an intentional naming system. We list below several questions the router/resolver duo may need to address.

- Am I able to completely resolve this intentional name?
- Can I get the bundle closer to its destination(s) without performing a complete binding? [i.e. Can I use information already existing in the bundle header?]
- Can I partially resolve the name beyond what has already been accomplished at prior nodes?
- Can I send the bundle to a node that may be able to perform a complete or partial binding? [These are sometimes called care-of
nodes]

- How expensive is each of the above operations?
- If more than one of the above are possible, which shall I do?
- What do I do if none of these options are available?

What might it mean in practice to "partially resolve" a name? Consider the following scenario: a military commander wishes to send a message to some officer within a half-kilometer of the Harvard Bridge. Using notional syntax, the name might be of the form rank="Officer" and location=within(500m, "Harvard Bridge (North)"").

Since it is unlikely that the source will know the EID(s) of "officers" that satisfy the above location constraints, it is unlikely that we can perform early binding. [And, as we have noted, it may be unwise to do so.] Hence, In order to resolve the name we need to first determine the location of the north end of the Harvard Bridge, route the bundle to nodes within a half-kilometer of that location, and deliver the bundle to nodes belonging to officers. The initial resolution, of the geographic portion of the name, can be performed at any node with geographic information (e.g. the commander’s node, or a node in the command headquarters). This would result in a physical location, but still no canonical EIDs. rank="Officer" and location=within(500m, loc(42.356931,-71.092527))

We would thus attempt to choose as the next hop a care-of node that was at or near the target location (or at least closer to the target location than we are). We consult a node location database and find that we have a link to a node close to the target, select that as the care-of node, and route the bundle there.

In this example, the care-of node is within the target region and is in radio contact with all nodes in the region. It consults its node location database, finding all nodes within the region. It can then focus on resolving the first part of the intentional name, i.e., rank="Officer".

The care-of node forwards the bundle to the target nodes. Each target node compares its rank (the rank of its owner) and either delivers or discards the bundle.

Alternatively, nodes in the close vicinity of the care-of node may have shared their "rank" information with the latter (such information should be signed and encrypted by appropriate security keys). In this case, the care-of node can resolve the entire intentional name to the EIDs of a selected subset of nodes that
shared their rank information earlier, specifically the ones with rank="Officer".

In this way late binding, incremental resolution, and routing are interleaved.

The goals of the proposed name resolution framework are in some ways similar to those of a wide-area peer-to-peer publish-subscribe system with the important additional requirement of disruption tolerance. Each node is both a publisher of, and subscriber to name binding information (i.e., name attributes and ontologies). The framework provides a distributed matching service that connects the subscribers to the publishers in a disruption-prone network environment. While several networking applications use the publish-subscribe paradigm at the application layer, we argue that this service should be available in the DTN network layer, i.e. alongside routing -- otherwise each intentional naming application would be expected to perform its own routing and therefore, potentially waste precious connectivity opportunities or network bandwidth for dissemination of redundant name binding information. If this service is supported in the network, then such dissemination can happen once and can be optimized for the DTN environment.

3.1. A Note on Tunneling

Although the discussion above assumes that all nodes are participating in the intentional naming scheme, it is possible that not all nodes in a DTN will be capable of resolving intentional names or routing intentionally named bundles.

In such a situation, a bundle may need to traverse several DTN hops from one resolver node to the next (and each DTN hop in turn may constitute several hops in the underlay networks). Routing of a bundle between two name resolvers that are not DTN-neighbors of each other can be achieved by tunneling, i.e. encapsulating the original user bundle in a control bundle that is de-capsulated once it reaches the next resolver node. Tunneling can be achieved by using the bundle-in-bundle encapsulation feature [I-D.irtf-dtnrg-bundle-encapsulation]. Tunneling is necessary in this scenario because the primary block of the bundle is immutable (otherwise it will fail the end-to-end security checks [I-D.irtf-dtnrg-bundle-security]), and hence there is no direct means of forcing the bundle to be routed to the next resolver EID without modifying the destination field of the bundle that stores the intentional name. But with the bundle-in-bundle feature, one can keep the original bundle unmodified and address the "envelope bundle" to the next resolver EID.
4. Architecture

The previous section discusses how names may be resolved, and how bundles with intentionally-named destinations can be routed. In this section we discuss the architectural components and procedures followed by nodes participating in the intentional naming system.

The major functional components of the architecture are:

- Name Knowledge Base Storage
- Knowledge Base Dissemination
- Resolution
- Routing [Although a DTN Router needs to be present even without intentional names, its function is greatly influenced by the design of the Intentional Naming System.]

4.1. Name Knowledge Base

Each node maintains a knowledge base (KB) of ontology and attribute information. A KB can contain historical, current, and future information about bindings of intentional name attributes to locally registered application EIDs, or to remote care-of EIDs, or other intentional names. The knowledge base can range from a simple look-up table, mapping intentional names to canonical EIDs of nodes capable of resolving names in that ontology, to a powerful deductive database engine (such as Prolog) that can infer complex derived attributes by execution of rules on base facts.

Nodes with greater computational power and richer network connectivity [advantaged nodes] will be more successful at resolving names than resource-strapped and ill-connected nodes. Advantaged nodes are able to manage large name KBs and support high volumes of resolution queries; these nodes would reasonably be provisioned as gateways between different DTN networks. However, nodes with high computational power but with little connectivity (e.g., powerful notebook computers in a MANET) may also perform useful resolution tasks, serving as a resolver for nearby nodes.

The range of possible knowledge base implementations is broad and is likely to be the subject of significant research and experimentation. We offer the following features as the basis for an initial design:

1. A simple lookup table mapping ontology names (e.g. geographic) to nodes that have advertised the corresponding ontology. When an ontology advertisement arrives it is inserted into the table.
The node uses the table to determine where to forward bundles addressed to names described in that ontology.

2. A lookup table mapping name attributes to canonical EIDs of nodes. The knowledge base can perform simple matches and comparisons to improve its forwarding (choice of care-of node) decisions. This mapping is possible if the corresponding ontology rules already reside on the current node.

3. A deductive database that stores facts about name attributes as well as rules used for attribute derivation. This kind of KB can be used for table matching, as well as execute a potentially complex rule to infer the result from a given fact base.

Note that the lookup tables may be simple in-memory structures, or backed by persistent relational databases, depending on the nature of the node and the requirements of the deployment. A deductive database might be implemented specifically for this application, or an existing inference engine (e.g. GNU Prolog, XSB [XSB], Flora-2 [Flogic]) might be used.

4.2. Dissemination

Even though all rules in an ontology are well-known at a certain node, all base facts that are necessary for successful resolution may not be present in the local KB. Hence a dissemination scheme is required for distributing those facts in the DTN.

A node that has the definition for an ontology advertises that fact (with appropriate version numbering and creation/modification times); other nodes can request a copy of the ontology definition, cache the fact that the ontology is available at the publisher node, or ignore the information. Nodes also publish their intentional name attributes, and forward along ontology advertisements and attributes provided to them.

In certain application scenarios (especially involving highly disruptive networks) it may be desirable for a node to actually publish the ontology rules rather than just the fact that it possesses some specified version of the ontology rules. In this scenario, all nodes will cache the ontology rules and forward them to other nodes.

We do not mandate a specific algorithm for attribute and ontology dissemination, although we believe that a geographically-bounded (or hop-count-bounded) epidemic dissemination will work well with the applications that we envision. See Section 6.3 for an example of such a technique.
4.3. Resolution

Name resolution (also known as binding) is the process of mapping an intentional name to the canonical EIDs of one or more destination nodes, or to the canonical EIDs of one or more care-of nodes, or to another intentional name, or to some additional information that will help in the routing process. [Note that the notion of care-of and bundle custody transfer are orthogonal in principle, even though practical implementations may decide to tie them together.]

This resolution process occurs as a result of a lookup into the local name knowledge base (either by a simple lookup into a file or an RDBMS, or by rule-based inference as mentioned earlier).

The architecture also supports the possibility of remote resolution of names; a DTN node may query a remote DTN resolver node (e.g. if both nodes are in a well-connected component of the network) to determine the appropriate next care-of node, and then forward the bundle on.

The results of the resolution process at a given node are typically added to the bundle’s metadata extension block.

4.4. Routing

Routing in a DTN has been (and will continue to be) the subject of numerous investigations. We note here that the inclusion of intentional names as legitimate bundle destinations places several requirements on the DTN routing functionality. In particular, the router must be able to identify and process the results of the resolution process performed on the current node and on prior nodes in the bundle’s path. Conversely, the capabilities of the DTN routers employed will place limits on the ability of the overall system to use intentional names.
5. Implementing Intentional Names on DTN

We now detail our model for implementing intentional naming on DTN. The important points to cover are:

1. What additional information will be stored in intentionally-named bundles, and where it will be stored.

2. How routing will be affected.

3. What new APIs will be needed.

4. How applications will register for, submit, and receive, intentionally-named bundles.

5. A syntax for intentional naming.

We have implemented the following in a prototype extension to DTN2, the DTNRG reference implementation. We note that the scope of this I-D does not address several issues of privacy, security, and trust.

5.1. Metadata Extension Block for Late Binding

In the DTN Bundle Protocol, the destination field (of the primary block) is set when the bundle is created and not modified thereafter because the Bundle Security Protocol requires that the contents of the primary block not be changed en-route; otherwise, the bundle will fail end-to-end authentication checks. When a bundle is addressed to an intentional name, the intentional name is stored in the destination field that is immutable. However, as a bundle passes through the network, care-of nodes may wish to decorate the bundle with resolution information (up to and including the canonical EID of the destination). This information will be stored in a metadata extension block in the bundle. This metadata block will be included in the hop-by-hop authentication checks (e.g., mandatory BAB-HMAC ciphersuite as described in [I-D.irtf-dtnrg-bundle-security]) but not in the end-to-end checks (e.g., the mutable canonicalization scheme defined in [I-D.irtf-dtnrg-bundle-security]).

Now we propose a metadata ontology for achieving progressive resolution and deferred binding. The metadata field in the extension block [I-D.symington-dtnrg-bundle-metadata-block] will have the following format:

1. Next resolver EID (SDNV): The field denotes the node that the bundle will be sent to next. The binding algorithm is responsible for computing this and populating this field. This field could be empty as well.
2. (optional) Flag denoting the current state of the bundle (SDNV):
   Currently the state indicates whether the bundle should be sent
   "point-to-point" (likely if the Next resolver EID field is non-
   empty) or should it be "flooded" (in this case, the Next resolver
   EID is likely to be empty)

3. (optional) Flag denoting conditions for scoping the transmission
   of the bundle (SDNV): In simple situations this flag will contain
   a time-to-live value. In more advanced implementations, it could
   contain predicate expressions that can govern the spatial scope
   of transmission (e.g. to contain a flood).

Note that in the current version of the Internet-Draft, the metadata
ontology information is assumed to be residing at every node that
intends to participate in the deferred binding protocol. In future
versions, we will address how such ontology information can be
disseminated or updated on-the-fly, how access to such information
will be controlled etc.

5.2. Interaction with DTN Routing

The steps for the interaction between routing and late binding
modules are outlined in Figure 1 and Figure 2. The first figure
outlines the general process of dissemination of intentional name
attributes and the second figure outlines the process of name
resolution.

In these figures, BPA refers to the core bundle protocol agent that
is responsible for implementing the bundle protocol and the bundle
security protocol. BPA interacts with other decision plane modules
(Epidemic Router and Resolver), convergence layer adaptor (CLA), DTN
applications (APP) and data storage (DS). These modules can be
implemented as a single process or as separate processes that
communicate using IPC.

Dissemination: An application registers itself with the BPA as it
normally would, adding intentional attributes and values (using the
extended API discussed below); these attributes are installed in the
local naming knowledge base. Attributes can also be received from
the LB (late binding/intentional naming node) knowledge base of
another node. It is also possible for the BPA to infer some of the
attributes that are not specified by an application (e.g., location
may be known to the BPA by other means) or determine them.
Periodically, the state of this node’s KB is disseminated (using the
epidemic router in our implementation) to other nodes. Note that we
use the special EID "dtn:nbrcast" to denote all DTN neighbors of a
particular node during the lifetime of the bundle.
Figure 1: Steps in Dissemination of Intentional Name Attributes

(1a): APP Registration events

(1b): Receive Link events or control data (from APP or remote LB)

(1c): Update KB with received info

(2): Extract/filter resolver state (S) to disseminate

(3): RequestDissemination(S, dest: dtn:nbrcast)

(4): Create bundle B(src=myEID, dest=dtn:nbrcast, payload=S)

(5): RequestInjectBundle(B) on link L

(6): Send bundle B on link L

Resolution: When a bundle arrives (in this figure, from a local application, although it could arrive from a remote node) it is first stored in the local data store (as are all bundles) and an EventBundleReceived event is triggered. This event is handled both by the local router and the resolver. The router directly queries the resolver and is returned one or more canonical EIDs for next-hop care-of nodes. The router determines the appropriate links to reach these nodes and requests that the bundle be sent to these nodes.
Not pictured is the case where there are nodes that do not understand the intentional naming extension are located on the path between this node and a care-of node. In this case the bundle will be encapsulated (using bundle-in-bundle) and sent to the care-of node.

![Diagram of bundle resolution process](image)

**Figure 2: Steps in the Resolution of an Intentional Name**

(1): Bundle B(dest: dtn:intent#...)

(2): Store B persistently

(3): EventBundleReceived(B, extension block MEB)

(4): Query local NameKB (dest, MEB)

(4a) Result: canonical EID List C; app EID List A; MEB’

(5): EventIntentionalNameResolved(C,A,MEB’)

(6): Compute links {L} for reaching C

(7a): DeliverBundleToApp(B,A)

(7b): RequestSendBundle (B,L)

(8a): Deliver bundle B to applications {A}
(8b): Send bundle B on links (L)

5.3. New Interfaces

The late binding API consists of the following parts:

Methods for name resolution:

- Local resolution: this method queries the local name KB with the DTN EID in the bundle destination field and the LB metadata extension block in the bundle (as specified in Section 5.1) as inputs. The query execution could be a simple database lookup or could invoke a more complex rule execution in a deductive database. If the query into the local KB returns a matching application EID registration then bundle delivery to that application is attempted. This process is referred to as application resolution. Since an intentional name could be bound to multiple DTN EIDs (multicast semantics), local resolution to a registered application may be inadequate. Therefore, if the name KB has additional intentional name to care-of EID bindings, the resolution query returns this list of care-of EIDs, and then the resolver populates the LB metadata extension block with these EIDs.

- Delivery to application: this method is used by the late binding module to request the BPA to deliver a bundle with an intentional name to a matching application registration. The matching is performed by the resolution component of the LB module.

- Remote resolution: these methods are for sending/receiving resolution requests and responses to/from a remote DTN node (since the process is asynchronous). Note that the local resolution API will be exercised at the remote resolver nodes before they send a resolution response to the requester. Methods for distributed name KB management:

  - Name update: this method is used by a DTN application to register (or publish) its intentional name attributes to a local name KB. Any subsequent changes to the name attributes (such as locations, roles etc.) are also updated using this API. This information is used by the local application resolution procedure during name resolution.

  - Name KB dissemination: these methods are used by DTN resolver nodes to send/receive portions of their name KBs to/from other DTN resolver nodes. While in the most primitive version, all records in the local name KB can be packed in a bundle and disseminated to a remote resolver, more advanced logic can be
used to pack and send records from the name KB selectively. This information is used by the resolution module for mapping an intentionally named bundle destination to a list of care-of EIDs in the DTN. Conflict resolution or reconciliation at the remote node is also an open topic and is domain specific.

5.4. Application Delivery

In the DTN late binding architecture, an application may register with the bundle protocol agent (BPA) with one or more intentional name attributes (e.g. role=Lieutenant). Any bundle with an intentional name query that matches these attributes should be delivered to this application. Normally, the BPA performs a simple lookup of the incoming bundle’s destination field in its registration table (this includes regular expression matches in the reference implementation). However, in the DTN late binding architecture, flat matches or even regular expression matches are insufficient for matching intentional names. Hence, the matching needs to be more general, since resolution rules in the name KB could map the incoming bundle destination EID to a specific application EID registered on this node. If such an event happens, the intentional naming module instructs the BPA to deliver the bundle to the appropriate application registration.

5.5. Syntax

EIDs follow URI syntax (scheme:scheme-specific-part) [RFC3986][RFC5050]. The "dtn:" scheme has been provisionally registered with IANA. The scheme-specific part (SSP) of a DTN intentional name should be capable of denoting an individual DTN endpoint (node or a service) or a group of nodes, with potentially multiple attributes. In the scheme-specific-part, we propose qualifiers such as "intent:uni#", "intent:any#", and "intent:all#" for denoting unicast, anycast and multicast semantics respectively. The default is "intent#" which denotes multicast. Note that these semantics are in line with the semantics of the minimum reception group (MRG) for the respective cases defined in [RFC4838] and [RFC5050].

As an example, we first describe an ontology that denotes the space of descriptive names with location and role based attributes. The ontology has two attributes, role (a string literal) and location (an (x, y) pair) and a predicate, within, which takes two locations and a distance. (All ontologies have an EID attribute.)

Using an SQL-like syntax, the node entries would look like

create table node (EID string, role string, location xy_pair)
Assume we wish to communicate with all coffee houses within 100 distance units of dtn://pbasu.bbn. (We have skipped units in these examples for reducing clutter.) This can be represented as a database query, which using SQL-like syntax, would look like the following:

```
select EID from node where role=coffee and within(100, (1,3), location)
```

Instead of an SQL-like syntax, we propose a Prolog-like syntax, in which attributes of the node are described as relationships. With this syntax, the facts would be in the form:

```
role(dtn://pbasu.bbn, researcher), location(dtn://pbasu.bbn, (1,3))

role(dtn://dunkin-donuts.cambridge, coffee), location(dtn://dunkin-donuts.cambridge, (3, 47))

role(dtn://starbucks.cambridge, coffee), location(dtn://starbucks.cambridge, (0, 50))

role(dtn://peets.cambridge, coffee), location(dtn://peets.cambridge, (-17, 270))
```

Our query would then be expressed in the following way (using the Horn clause syntax proposed in Section 2.2):

```
role(E, coffee), location(E, Loc), within(100, (1,3), Loc)
```

This query amounts to finding some EID E where there is a fact (assertion) for role(E, coffee), there is a location (E, Loc), and the distance between Loc and (1,3) is less than 100. Note that E denotes the mandatory variable in the intentional name whereas Loc is an internal variable which is necessary for specifying the desired rule. We express this fact by adding a question mark in front of the mandatory variable. Hence the actual intentional name that gets put in the destination field of the bundle looks like the following:

```
dtn:intent#role(?E, coffee), location(?E, Loc), within(100, (1,3), Loc)
```
6. GRAIN: Gradient-Based Algorithm for Intentional Naming

In this section we describe GRAIN, a Gradient-based Resolution Algorithm for Intentional Names, for resolving a class of intentional names and delivering bundles to those EIDs. We focus on name-queries for individuals or groups with certain roles located within a specified distance of a specified coordinate.

In the following, we assume that all DTN nodes in the network are running a copy of GRAIN and they each have access to their current location information (e.g. via a GPS device). An intentional name query supported by GRAIN can have role, location, and distance attributes, e.g. company commanders (role) located within 1 km (distance) of 42.390501N, 71.147987W (location).

GRAIN does not resolve the entire name at the source node; instead it performs progressive resolution as the bundle moves through the network to determine the ultimate destination endpoint(s). At the initial stages of resolution (near the source node), GRAIN uses the location attribute to determine one or more nodes closer to the eventual destination, and forwards the bundle to those nodes. This process continues until the bundle is within the region defined by the location and distance attributes. At that point, GRAIN begins using the balance of the attributes to determine how to route and deliver the bundle.

A bundle can be in one of two states, STEM or FLOOD. In the STEM state, GRAIN routes the bundle toward the eventual destination while trying to use as few network resources as possible. When the bundle arrives within the circle defined by (location, distance), GRAIN switches the bundle’s state to FLOOD and is more liberal with the use of network resources. When in FLOOD state the bundle is routed using predicate-scoped epidemic flooding, a variant of epidemic routing.

Epidemic routing forwards a bundle to all nodes in the network eventually. In predicate-scoped epidemic flooding we want the bundle to eventually spread to nodes only located inside the (location, distance) circle and not those outside it. This constraint can be imposed by applying a predicate filter, "within(location, distance)" , on all the forwarding paths determined by epidemic routing. The bundle is only forwarded to nodes that meet the predicate.

In our implementation of GRAIN, routing is controlled by carrying the above state of the bundle in its metadata extension block (MEB).
6.1. Resolution and Routing Policies

GRAIN supports three resolution and routing policies for progressive resolution:

**DTN-ROUTE**: If this policy is enabled and active, then at the source node GRAIN determines the intermediate node closest to the target location. (This is possible because of the prior dissemination of location information, described below). The bundle is routed using a standard bundle routing protocol (such as APLS, described above). Once the bundle reaches the intermediate node, which is just outside the circle defined by (location, distance), its state is switched from STEM to FLOOD and it is flooded to nodes within the region by the use of predicate-scoped epidemic flooding (defined in Section 6.3). This ensures that we get full coverage within the region of interest while diminishing resource consumption to reach the region of interest.

**GEOGRAPHIC_NEAREST_NEIGHBOR**: In order for DTN-ROUTE to work efficiently it needs up-to-date global node location information, which may not be readily available in disrupted networks. GEOGRAPHIC_NEAREST_NEIGHBOR uses local (rather than global) node location information to make resolution-plus-routing decisions. The source node resolves the geographic part of the intentional name to the canonical EID of the neighbor that is closest to the target location. GRAIN then forwards the bundle (using a standard bundle routing algorithm) to this neighbor and the resolution steps are repeated. Since all nodes are expected to have timely location updates from their neighbors, this policy is expected to make forward progress toward the target location until the state of the bundle changes to FLOOD. Subsequently, the same steps are taken to flood the bundle using predicate-scoping.

**GEOGRAPHIC_ALLNEARER_NEIGHBORS**: While using minimal network resources in the STEM phase, the GEOGRAPHIC_NEAREST_NEIGHBOR policy is more prone to failure under the ordering of link availability events. For example, it is more likely that a bundle is forwarded to a neighbor that constitutes a local minima in the geographic routing protocol. To reduce the probability of this happening, and for making routing more robust under disconnection and mobility, GEOGRAPHIC_ALLNEARER_NEIGHBORS resolves the intentional name to a list of neighbors that are nearer to the destination. The bundle is then forwarded, using the epidemic routing algorithm, to only those neighbors. (Note that the epidemic routing protocol delivers a bundle to a node at most once.) These neighbors continue the resolution process, routing the bundle toward the destination, until the state of the bundle changes to FLOOD. The bundle forwarding path in this case looks like a union of paths rather than a single path as
in the case of the previous policy. This policy works using just local information (unlike DTN-ROUTE), and is more robust under disruption than GEOGRAPHIC_NEAREST_NEIGHBOR, but consumes fewer resources than unconstrained epidemic flooding.

6.2. Disseminating Location Information

In order for the above resolution schemes to work, location information needs to be shared among the nodes in the network. Under the DTN-ROUTE policy, each node periodically broadcasts a location update bundle to all nodes in the network. As the location update bundle spreads through the network, nodes update their databases with (location, canonical EID) pairs. GEOGRAPHIC_NEAREST_NEIGHBOR and GEOGRAPHIC_ALLNEARER_NEIGHBORS, the two policies involving geographic routing, only send location updates to immediate neighbors. This is because the resolution algorithms corresponding to those policies do not use any information beyond one hop location information.

6.3. Predicate-scoped Epidemic Flooding

Normally, epidemic routing spreads a bundle to all nodes in the network eventually. When the bundle is in the FLOOD state, i.e., inside the circle defined by \((X,Y,D)\), we want the bundle to spread to nodes only located inside that circle and not outside. This can be achieved by applying a predicate filter for pruning the forwarding paths determined by epidemic routing, the predicate here being defined by "\(\text{within}(X,Y,D)\)". Since the vanilla version of epidemic routing does not have information about \((X,Y,D)\), this is achieved by the late binding module. In other words, the latter performs a lookup into its name KB and returns only those canonical EIDs that satisfy the predicate "\(\text{within}(X,Y,D)\)". The epidemic router then merely forwards the bundle to the appropriate neighbors. GRAIN supports three user selectable resolution/routing policies that help in progressive resolution. Numerous other policies are indeed possible, and these are provided as motivating examples for the need for a flexible naming system.
7. Deploying an Intentional Naming System

In the previous sections we made an effort to define an Intentional Naming framework and architecture in the broadest possible manner, so that it might encompass the most flexible and potentially useful capabilities. In those discussions we identified several research issues that require progress before a complete Intentional Naming System could be deployed. In this section we identify specific issues that currently limit our ability to deploy an Intentional Naming System, and we describe a more restrictive set of capabilities that we believe will allow a practical yet useful Intentional Naming System to be deployed.

The restrictions we adopt relate primarily to the expressiveness of intentional names, the locality of name binding information, and the scalability of the deployment.

7.1. Scalability

The DTN environment presents several challenges for the scalability of name binding algorithms. First, sharing up-to-date resolution information in an often-disconnected or intermittently connected network in a scalable and timely fashion is challenging. The uncertainty about the currency and validity of resolution information typically grows proportionally with the distance between source and destination nodes, hence the probability of resolving to the correct set of nodes in a dynamic DTN will likely decrease with the distance between the source and the destination.

In addition, descriptive intentional names are significantly richer than canonical DTN EIDs, hence the protocols need to scale not only with the size of the network but also with the size of the name–attribute space. Moreover an application registration may be dynamic, adding another dimension of complexity to the problem. Typically if a name–attribute space naturally lends itself to easy "aggregation" and there is a strong correlation between the distribution of node attributes and their topological location, it is more likely that we can develop scalable name binding algorithms. Examples include geographic attributes in our GRAIN algorithm, hierarchical IP addresses and DNS names, and static organizational hierarchies.

In general, however, it may not be possible to aggregate attribute-value pairs for all namespaces. In particular, if the name is a conjunction of two relations or predicates, e.g., "guardian(EID,S),student(S)", then the resolver nodes need to be able to perform a semi-join operation on the two relations "guardian" and "student". Since these two relations may be spatially distributed
over the entire DTN and there is no location attribute in the name, performing this operation in a scalable manner under potential disruption is not simple. This is a topic for our future research.

7.2. Expressiveness of Intentional Names - Revisited due to Scalability Considerations

In its most general form, an intentional name may include an arbitrary number of predicates which, together, will define the intended destination(s). We have presented examples of how such predicates may be bound to provide routing information which will assure bundle delivery. The presented examples do not, however, represent the most general case, and we now wish to make explicit what restrictions are necessary to assure reasonable name binding capability.

We first motivate these with an example. Consider an intentional name consisting of two predicates:

- IS Army PFC
- HAS SPOUSE BORN IN New York

We are addressing information to all PFCs who spouse was born in New York.

Consider what types of information would be required to allow this bundle to get closer to the desired targets, and where that information might reside. There may well be a database that identifies all PFCs. There is another that identifies all people born in New York. There are other databases that contain marriage (and divorce and death) information. Even if each of these databases was entirely contained in a single location (as opposed to being distributed); even if there were many replicas of each database, it would be very difficult to devise, on the fly, a strategy for assuring the bundle arrives at all [or even just one] desired destination(s). Decisions about what order to consider each of the predicates, how best to record intermediate results, and how to deal with databases that are [presumably temporarily] unreachable would have to be made at each hop.

If we further consider the fact that one of these predicates may actually change over time, we see that this is a difficult problem indeed. But we could make the problem much simpler if we add an additional predicate to the intentional name:

- IS WITHIN 100 yards of the Massachusetts State House
Why does adding a condition simplify the binding operation? First, it greatly reduces the number of potential recipients. But replacing "New York" with "Edgartown" would also do that. What the new condition does is give us a strategy for moving the bundle along: Send the bundle closer and closer to the State House, and when you get near enough, just ask everyone to self assess whether they meet the conditions, or, perhaps, ask some local record keeper to provide information about nearby people.

We generalize from this example by imposing the following two restrictions on the predicates we will allow in an intentional name:

1. One [and only one] of the predicates applies to an ontology that has either a well-defined distance function or a well-defined hierarchy and an end position [relative to distance or hierarchy]. This will be called the "primary ontology". All routing information [the result of partial bindings] will be aimed at getting the bundle closer to the stated end position.

2. Information regarding the truth of all other predicates must be more available the "closer" one gets to the stated end position.

The first of these allows routing progress to be made whenever a node is cognizant of the primary ontology. The second assures that making progress with respect to that primary predicate will eventually result in additional information about the others. The primary ontology/predicate will get the bundle to a neighborhood in which all other predicates can be evaluated.

These are very severe restrictions. But we believe they allow for a very useful subset of possible intentional names, as exemplified by the GRAIN example in Section 6 and Appendix A, which satisfies this restrictive property.

Note that these restrictions by no means constitute necessary conditions for scalable name binding.

7.3. Security Considerations

A primary security issue here is that of a denial of service attack. This is because the intentional name in the bundle destination field with multicast and anycast semantics could be bound to large groups. So a user could construct an intentional name such that a bundle is flooded throughout the network. Moreover, the persistent nature of DTN and the deferred binding feature of late binding make the situation worse since the bundle stays in the network until it expires and it gets replicated (over space and time) to nodes that either satisfy the name attributes or can progressively resolve.
Other security issues specifically pertaining to name resolution include:

1. How to ensure that the intentional name to canonical EID binding was created by a trusted party?

2. How to ensure that the name binding information is being advertised and modified only by authorized nodes?

3. How to verify that a specific name indeed matches the intentional name?

Other basic security concerns include confidentiality, authentication and integrity of intentional bundles. As long as the source of the bundle is a unique endpoint, it should be able to sign and encrypt the bundle and the mechanisms defined in the Bundle Security Protocol can be used to secure the intentionally named bundle. We leave this topic for future research.
8. Acknowledgments

We thank DARPA DTN program for supporting this work (DARPA/ATO contract number W15P7T-05-C-P211). We also thank our BBN colleagues Partha Pal, Mitch Tasman, and Ram Ramanathan for contributing to the initial discussions on the late binding architecture. We thank Chris Small for his help toward making this document more readable.
9. Informative References


Appendix A. GRAIN Pseudocode

We present the pseudo-code for GRAIN, which provides deferred binding to an intentional name that possesses geographic and role attributes. This constitutes of the following components: (1) robust and efficient geographic routing to near the final destination; (2) predicate-scoped flooding to reach nodes that satisfy the location constraints in the name; and (3) delivery to all registered applications that satisfy both role and location constraints in the intentional name.

Please note the following algorithmic details:

- The resolution/routing policy can be set to one of:
  * DTN_ROUTE
  * GEOGRAPHIC_NEAREST_NEIGHBOR
  * GEOGRAPHIC_ALLNEARER_NEIGHBORS

- Bundles carry route_state, initialized to STEM; target_location, target_radius, and attributes (in this case, a set of roles).

- Each bundle carries a unique BundleID.

- When a bundle with an intentional name is submitted, the source node receives an event, INTENTIONAL_BUNDLE_RECEIVED.

- Intentional names contain three components: target_location, target_radius, attribute_set

- The algorithm uses a data structure, node_table, which holds information about nodes, locations, and links.

- The algorithm maintains an pending_bundle_table that holds pending bundles.

- The algorithm maintains an application_delivery_table listing bundles that have been delivered to each application.
A.1. Main loop

GRAIN’s main loop consists of:

loop forever:
    receive_message
    process_bundles
end loop

A.2. Message processing and state management

This procedure is responsible for handling events as they come in, and managing GRAIN’s internal state. When a bundle arrives it is placed in the unresolved_bundle_table; bundle processing is handled in the following function.

procedure receive_message:
    accept message
    switch (message type)
        case LINK_AVAILABLE(Node:eid, Link:linkId):
            store (Node, Link) in node_table
        case LINK_UNAVAILABLE(Link:linkId):
            remove Link from node_table
        case LOCATION_DISSEMINATION(Node, loc):
            store (Node, loc) in node_table
        case APPLICATION_REGISTRATION(A:eid):
            store (A.name, A.attribute_set) in application_table
        case APPLICATION_DEREGISTRATION(A:eid):
            remove A from application_table
        case INTENTIONAL_BUNDLE_RECEIVED(B:bundle):
            store B in unresolved_bundle_table
        case BUNDLE_EXPIRED(B:bundle):
            remove B from unresolved_bundle_table
    end switch
end procedure
A.3. Bundle processing

We now describe how each pending bundle is processed. If appropriate, deliver a copy of the bundle to a local application. Then check to see if we should switch the bundle’s state from STEM to FLOOD. Last, we forward using the appropriate algorithm (STEM or FLOOD).

procedure process_bundles:
  for B in pending_bundle_table:
    deliver_locally_if_appropriate(B)
    change_from_stem_to_flood_if_appropriate(B)
    if bundle_route_state_metadata = STEM:
      do_stem_forwarding(B)
    else
      do_flood_forwarding(B)
    end if
  end for
end procedure

A.4. deliver_locally_if_appropriate

If we are within the target radius, and there are any applications registered that match the other attributes present in the address, deliver the bundle to the application.

procedure deliver_locally_if_appropriate(B)
  if dist(Me.location, B.target_location) < B.target_radius:
    for each application A in application_table:
      -- if the bundle has not yet been delivered, and matches
      -- the criteria for the application, deliver it
      if (A.AppId, B.BundleId) not in application_delivery_table
        and B.attribute_set ⊆ A.attribute_set:
          deliver B to application A
          store (AppId, B.BundleId) in application_delivery_table
        end if
    end for
  end if
end procedure
A.5. change_from_stem_to_flood_if_appropriate

This procedure determines whether we can reach any nodes within the
target region for the bundle. If so, we switch the bundle’s state to
FLOOD so that from this point on the bundle will be flooded to all
target nodes.

procedure change_from_stem_to_flood_if_appropriate(B)
    if B.route_state = STEM:
        for each Node in node_table:
            if dist(Node.location, B.target_location) < B.target_radius:
                B.route_state <- FLOOD
                return
        end if
    end if
end procedure

A.6. do_stem_forwarding

This procedure is invoked when we are outside the target region, and
we are working to move the bundle toward the target region.

procedure do_stem_forwarding(B)
    switch(B.routing_policy)
    case DTN_ROUTE
        forward B to known node closest to B.target_location
        remove B from unresolved_bundle_table
    case GEOGRAPHIC_NEAREST_NEIGHBOR:
        forward B to Nbr with smallest dist(Nbr, B.target_location)
        remove B from unresolved_bundle_table
    case GEOGRAPHIC_ALLNEARER_NEIGHBORS:
        for all Nbr where dist(Nbr, B.target_location)
            < dist(Me, B.target_location)
            forward B to Nbr
        -- Note: DO NOT remove B from unresolved_bundle_table
    end switch
end procedure
A.7. do_flood_forwarding

This procedure is invoked once we are within the target_radius for the bundle. We forward the bundle to all neighbors (nodes to which we have direct links) that are also within the target_radius.

procedure do_flood_forwarding(B):
    for each Node in node_table:
        if I have a direct link to Node
            and dist(Node.location, B.target_location) < B.target_radius:
                forward bundle B to Node
        end if
    end for
end Procedure

A.8. Location dissemination

This procedure, which in a separate thread concurrently with the bundle processing main loop, is responsible for sharing/updating node location information with neighbors.

N ← delay between updates (typically 10 < N < 60)

if (routing_policy = DTN_ROUTE)
    broadcast my location to all nodes every N seconds
    forward any location update bundles I receive to all neighbors
else if (routing_policy = GEOGRAPHIC_ALLNEARER_NEIGHBORS ||
    routing_policy = GEOGRAPHIC_NEAREST_NEIGHBOR)
    send my location to all neighbors every N seconds
    do not forward location update bundles to neighbors
end
Authors’ Addresses

Prithwish Basu  
BBN Technologies  
10 Moulton Street  
Cambridge, MA  02138  
US

Phone: +1 617 873 7742  
Email: pbasu@bbn.com  
URI:  http://www.ir.bbn.com/~pbasu/

Dan Brown  
BBN Technologies  
10 Moulton Street  
Cambridge, MA  02138  
US

Phone: +1 617 873 7560  
Email: dbrown@bbn.com

Stephen Polit  
BBN Technologies  
10 Moulton Street  
Cambridge, MA  02138  
US

Phone: +1 617 873 2159  
Email: spolit@bbn.com

Rajesh Krishnan  
Scientific Systems Company, Inc.

Email: krash@ieee.org