DTN: An Architectural Retrospective

Kevin Fall, *Senior Member, IEEE,* and Stephen Farrell

***Abstract*—We review the rationale behind the current design of the Delay/Disruption Tolerant Networking (DTN) Architecture and highlight some remaining open issues. Its evolution, from a focus on deep space to a broader class of heterogeneous networks that may suffer disruptions, affected design decisions spanning naming and addressing, message formats, data encoding methods, routing, congestion management and security. Having now achieved relative stability with the design, additional experience is required in long-running operational environments in order to fine tune our understanding of DTN concepts and the types of capabilities that are worth the investment in implementation complexity. We expect key management, handling of congestion, multicasting capability, and routing to remain active areas of research and development, and that DTN may continue to be an active research endeavor for at least the next few years.**

***Index Terms*—Delay Tolerant Networking, Disruption Tolerant Networking, network architecture, protocols**

I. Introduction

In the last few years, Delay- and Disruption-Tolerant Networking (both known by the abbreviation DTN) have grown from relatively obscure research activities to a healthy research topic attracting both network designers and application developers. DTN is now a recognized area in networking research, due in part to practical experiences with mobile ad-hoc networks (MANETs) that are required to operate in situations where continuous end-to-end connectivity may not be possible.

While the architectural principles for DTN were synthesized and collected together about a half-decade ago, only recently have these principles been reviewed by a larger community and put to the test in a number of real-world pilots. With renewed interest in network architecture research, it appears timely to examine the DTN architecture retrospectively, highlighting some of its more unusual and controversial aspects, with the goal of providing concrete suggestions for capabilities applicable to network architectures that might be considered in evolving the DTN architecture or other networking architectures.

In this paper we review many of the principles of the DTN architecture [1], highlighting design decisions that have persevered through repeated analyses, along with those that have been updated or replaced. Some of this evolution can also be seen with the recent publication of two Internet RFCs (RFC 4838 [2] and RFC 5050 [3]), and a book on this subject [4].

Manuscript received April 15, 2007; revised January 10, 2008.

Kevin Fall is with Intel Research, Berkeley CA 94704 USA (e-mail: kfall@intel.com, http://www.cs.berkeley.edu/~kfall).

Stephen Farrell is with Trinity College, Dublin 2, Ireland (e-mail: stephen.farrell@cs.tcd.ie, https://www.cs.tcd.ie/Stephen.Farrell/).

Digital Object Identifier 10.1109/JSAC.2008.080609.

II. DTN's Capabilities

At its inception, the concepts behind the DTN architecture were primarily targeted at tolerating long delays and predictably-interrupted communications over long distances (i.e., in deep space). At this point in time, the work was an architecture for the *Interplanetary Internet* (IPN). By March 2003, when the first draft of the eventual RFC 4838 was published, one of the authors had coined the term *Delay Tolerant Networking* suggesting the intention to extend the IPN concept to other types of networks, specifically including terrestrial wireless networks. Terrestrial wireless networks also suffer disruptions and delay, and the DTN architectural emphasis grew from scheduled connectivity in the IPN case to include other types of networks and patterns of connectivity (e.g., opportunistic mobile ad-hoc networks with nodes that remain off for significant periods of time).

At roughly the same time, there was growing interest in wireless sensor networks (WSNs), a topic which itself has spawned numerous conferences, journals, theses, and some commercial activities [5]. Much of the activity in WSNs has been devoted to power management, routing, and other tasks such as software updates and programming environments. Common to most WSN systems is a form of *gateway*—a communication node that often implements an application layer gateway able to effectively translate Internet (TCP/IP) protocols to the specialized protocols used within WSNs. It is typical for such gateways to participate in routing domains on both the Internet and within the WSN. Some of these gateways also possess storage, used to hold data collected from the sensor network until consumed by an application. Such gateways were being constructed in an ad-hoc manner, specific to each WSN, limiting interoperability between them.

The DTN architecture was designed to accommodate not only network connection disruption, but also to provide a framework for dealing with the sort of heterogeneity found at sensor network gateways (and other gateways, more generally). Whereas the Internet (IP protocol) model supports heterogeneity as well, it does so by requiring every node to use a common network layer host identifier (IP address), packet format with universally-obeyed semantics, and routing methodology that assumes a connected routing graph in order to achieve interoperability. Supporting other addressing formats or semantics in conjunction with IP has resulted in widespread use of *overlay* networks, where the IP protocol is essentially used as a link protocol. As we shall see in more detail later, DTN uses naming, layering, encapsulation, and persistent storage to interconnect heterogeneous portions of a larger network, irrespective of formal layer.

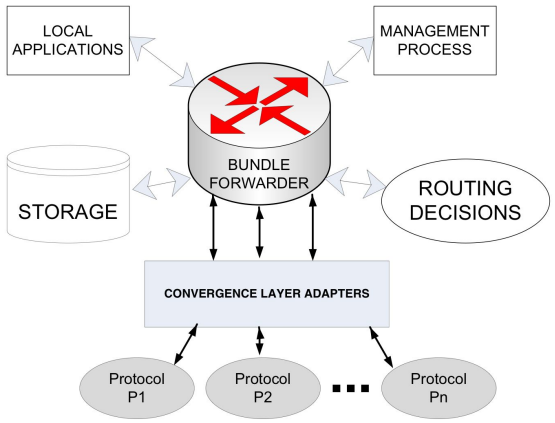

Fig. 1. An example implementation architecture shows how a bundle forwarder interacts with storage, routing decisions, and convergence layer adapters (forming the convergence layer) to utilize various protocols for delivery.

DTN can use a multitude of different delivery protocols including TCP/IP, raw Ethernet, serial lines, or hand-carried storage drives for delivery. As each of these protocols provide somewhat different semantics, a collection of protocol-specific *convergence layer adapters* (CLAs) provide the functions necessary to carry DTN protocol data units (called *bundles*) on each of the corresponding protocols. Figure 1 gives the relative position of the convergence layer adapters in a conceptual implementation architecture:

In this conceptual implementation architecture, a central forwarder is responsible for moving bundles between applications, CLAs, and storage according to decisions made by routing algorithms. Arrows indicate interfaces, which may carry either bundles (in the case of storage, CLAs, and applications) or directives (routing decisions, management, applications). In some cases, implementing these interfaces using inter-process communication facilities rather than conventional procedure call has been useful to promote the ability to develop system components independently.

III. Naming, Addressing and Binding

Naming and addressing are some of the most fundamental aspects of a network architecture, and one of the most tricky aspects to get right. Generally, naming has been thought of as something useful to people or organizations while addressing is more useful for network operations and routing. Names are generally expected to be variable-length strings while addresses are expected to be fixed-length identifiers. Some form of mapping or binding function is used to convert names into addresses. In the case of the Internet, this is the domain name system (DNS). In the case of various overlay network systems (e.g., Chord), it may be a locally-executed hash function.

In the evolution of the DTN architecture, nodes have always had identifiers. These are used in the context of the *bundle protocol* [3], which provides the basic message delivery service for DTN. Originally, identifiers in the bundle protocol were constructed as a 3-tuple of the form *(region, host, application)*, which was able to not only identify a host, but also an application of interest on the host. A region was a portion of the network topology, and in the original IPN design was generally assumed to represent a well-connected area surrounding a planet. Routing decisions were thus relatively straightforward, based first on region, and then on host identifier, somewhat similar to the way routing is arranged in IP networks where aggressive CIDR address prefix aggregation is performed. After some consideration of the application portion of the identifier, it was merged into the host identifier, forming an aggregate demultiplexing identifier where the partitioning between host and application was determined within a region.

After extended consideration of the tie between the region portion of an identifier and its required association with the network topology, the region construct was significantly modified. This decision was based primarily on the observation that nodes may have multiple network interfaces and may also be mobile, so additional flexibility was required in how they are named. It became more important to support multiple namespaces with differing naming semantics than coupling an identifier to a location in the topology to aid routing. With multiple namespaces, hosts may have multiple identifiers and these may be either assigned by users, or imposed by the networks to which nodes become connected. This began to blur the distinction of name and address. Blurring seems to be an attractive direction, as precisely distinguishing between the two has become increasingly challenging (e.g., consider vanity telephone numbers).

In recognizing that nodes may require multiple identifiers and even multiple *types* of identifiers, a naming structure was sought that is capable of encoding names or addresses from multiple different name spaces (and thereby also requiring a way to identify the namespaces from which the identifier had been allocated). Fortunately, work in the IETF had already been accomplished in the area of generalized naming systems, in the form of *Universal Resource Identifiers* (URIs) [6]. Although URIs are somewhat more complicated than required by the bundle protocol, they have a few important properties:

- Allocated Name Spaces – Each URI is fundamentally of the form `<scheme>:<scheme-specific-part>`, where the scheme is a string allocated from a set of well-known and administered scheme names (e.g., `http`, `sip`, `file`)
- Variable Length – Although bounded to a relatively large size, URIs are essentially free-form except for a few reserved characters that have special semantics
- Structured Semantics – URIs obey a general syntax and semantics, but a new scheme may define its own special additional semantics, subject to general rules that apply to all URIs

Using URIs as identifiers brings several advantages. First, they can encode names or addresses taken from many namespaces. For example, we might refer to a host by its Ethernet address as `ether://00-12-33-fe-22-31` but also refer to it using some distinguished hierarchical name like `dns://myhost.foo.com`. While in the Internet, the scheme specifier also tends to suggest the protocol stack

used (e.g., `http` is typically http/TCP/IP) to contact remote node(s), this need not be the case for DTN; we can use the bundle protocol, or some other combination of protocols. URIs as used in DTN are referred to as *endpoint identifiers* or EIDs.

Next, using strings for representing EIDs opens up the possibility of creating interesting DTN forwarding policies using string matching. For example, a wildcarded string match could be be used in directing a DTN forwarder to cause any traffic destined for Columbia University to be directed to some particular next hop:

```
dtn://*.columbia.edu.dtn ->
 ether://00-12-33-fe-22-31
```

This example illustrates that the addressing format for a DTN "next hop" (at a DTN forwarding node) need not be of the same scheme as that of the source or destination in the bundle. This is in contrast to Internet routing entries, where next hops are generally expressed using the same address format.

Using URIs can also help to support application layer gateways by piggybacking on a number of pre-existing URI schemes. For example, the URIs:

```
http://www.slashdot.org
dtn:http://www.slashdot.org
```

are syntactically legitimate bundle protocol EIDs. The full extent to which this capability may be useful remains to be seen, as few such application layer gateways using existing schemes have been constructed, but the naming compatibility is clear. Understanding the precise semantics of the `http` scheme when carried by the bundle protocol, for example, remains to be explored.

Finally, the generality of the URI format is also applicable for routing systems in which the "destination" address of a message is really a function of its contents. Proposals along the lines of the DONA [7] could make use of this flexibility by constructing URIs of the following form:

$$\texttt{dona://}a_1 = X_1 \& a_2 = X_2 \& \ldots \& a_k = X_k$$

where each (a_i, X_i) is an attribute/value pair.

For a message containing DTN URIs comprising symbolic names, (i.e., URIs using namespaces apart from standard address formats), some *binding*[1] step is performed by one or more nodes along the delivery path. Such binding may be performed anywhere along the delivery path. In the Internet, this happens at multiple layers and at multiple locations. When DNS is invoked at a sending node, this is a form of *early binding*, which is used immediately in mapping a DNS name to an IP address. Subsequent mappings are performed on the IP address in delivering its containing packet toward its destination.

DTN supports direct forwarding based on symbolic names (or intentional names [8]), so the early binding typical of DNS in the Internet is not generally required. Instead, messages are passed along toward their destinations based on forwarding entries present at DTN routing nodes that match against the name. This is known in the DTN literature as *late binding*. DTN supports both late and early binding, depending on the scheme used. The extent to which late-binding scales to networks of many routers will be interesting to see as DTN deployments scale up.

[1]There is another use of the term binding to mean associating a name or address with a receiving application, a function performed by the `bind()` socket API call. We instead use the term *registration* to refer to the state created by that operation, which can be persistent for DTN applications.

IV. DTN PDUs: BUNDLES

Applications in the DTN architecture operate on messages carried in variable-length protocol PDUs called bundles. The name "bundle" derives from considering protocols that attempt to minimize the number of round-trip exchanges required to complete a protocol transaction, and dates back to the original IPN work. By "bundling" together all information required to complete a transaction (e.g., protocol options and authentication data), the number of exchanges can be reduced, which is of considerable interest if the round trip time is hours, days or weeks.

Bundles comprise a collection of typed *blocks*. Each block contains meta-data; some also contain application data. For much of the evolution of the DTN architecture and bundle protocol, meta-data blocks were simply called headers, but after it became apparent that the bundle security protocol (see Section VII) required the ability to append meta-data (e.g., a MAC) to a bundle, the term block was adopted. Blocks are chained together as extension headers are in IPv6.

The extensibility of chained blocks has been key to supporting experimentation with the bundle protocol. For example, a bit indicates whether receiving a block of unknown type causes the containing bundle to be discarded or whether the block can instead be processed unaltered (i.e., as opaque data). This is expected to be of use, for example, as new authorization (e.g., capabilities) or routing functions (e.g., source routing) are investigated.

A. Blocks

The first or *primary* block of each bundle, illustrated in Figure 2, contains the DTN equivalents of the data typically found in an IP header on the Internet: version, source and destination EIDs, length, processing flags, and (optional) fragmentation information. It also contains some additional fields, more specific to the bundle protocol: report-to EID, current custodian EID, creation timestamp and sequence number, lifetime and a *dictionary*. Strings are placed in the dictionary, and offsets are used as pointers to the beginnings of strings in an effort to reduce space that would otherwise be devoted to duplicate strings. Most fields are variable in length, and use a relatively compact notation called *self-delimiting numerical values* (SDNVs) [3]. Early designs for the primary bundle block used more fixed-length fields, but the relative merit of choosing a fixed-length field for simplicity was ultimately found to be less compelling than the flexibility offered by SDNVs. SDNVs are discussed in more detail in Section IV-C.

The bundle processing control flags indicate a number of special circumstances associated with the containing bundle: fragmentation condition (fragmented, allowed), type (regular or administrative), special requests (custody, acknowledgment

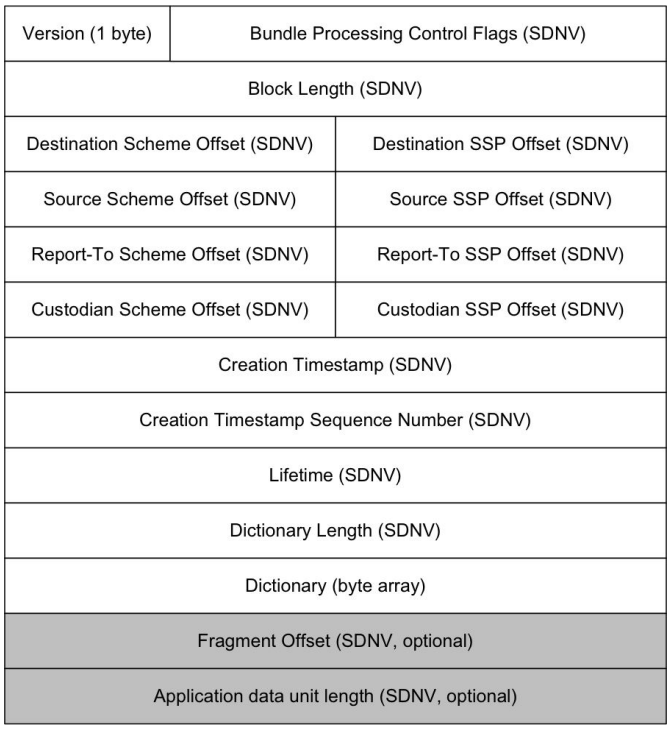

Fig. 2. The structure of the primary block of a bundle contains a fixed-length version field followed by a collection of variable-length fields encoded as self-delimiting numeric values (SDNVs).

generation, delivery status), class of service indication, and if the destination endpoint is known to be a *singleton* (that is, a single entity as opposed to a multicast endpoint). This last indicator is used when forwarding using custody transfer to alert a custodian that multiple nodes may be responding with custody transfer acknowledgments (see Section VI).

By setting various bits in the bundle processing control flags, the sender can request a report for any of the following events: receipt at destination node, custody acceptance at a node, bundle forwarded/deleted/delivered en route, and receipt by destination application. Clearly, these capabilities need to be used with caution because of the amount of report traffic they may generate, may be limited in live operations by policy, and are really intended for diagnosing network problems, as with ICMP in the Internet. While there is some hesitation as to the value of having such a rich set of report types, requiring support for them in protocol implementations has already proven itself extremely useful during interoperability test and debugging sessions held in conjunction with IETF.[2]

To the best of our knowledge, the report-to EID is unique to the bundle protocol. It allows a sender of a bundle requesting one or more status reports to have the reports directed to node(s) other than itself. This is in contrast to Internet ICMP messages which are specified to be sent back to the sender of the packets that caused the ICMP messages to be generated. This capability is useful in the DTN context because some senders may not be expected to exist beyond the time required to transmit a bundle they have sent. Examples include expendable sensor nodes that are lost or destroyed after reporting their sensor readings to a nearby DTN relay node.

The origination time in each bundle indicates the real time at which the bundle was sent from its origin. The lifetime is a positive offset of real time from the origination time. If a bundle is found to be queued at the end of its lifetime, it can be discarded. This is one of the ways excess bundles can be cleared from the network. It also provides a basis for implementing policy: a network operator could arrange for bundles beyond some age to be expired early (or late).

The use of real time in bundles imposes a requirement on each participating DTN node: that real time be synchronized, at least roughly. This requirement was considered repeatedly, as it represents a significant departure from common practice in the Internet today. To date, we have identified four reasons for imposing it. First, most applications for which DTN was designed are time sensitive; resources are consumed at particular points in space and time. A DTN node not knowing the time renders the DTN far less useful for most applications which themselves require time. Second, in most of the cases where DTN has been tested, and in most cases for which it is planned, access to real-time is already provided by some mechanism (including in deep space and underwater environments).[3] Third, routing using scheduled connectivity is inherently tied to link availability at a certain time. Fourth, network management tasks, including tracing and debugging are considerably easier when a common time reference is used throughout the network.

Other than the required primary block appearing at the beginning of a bundle, additional blocks are optional but use a common basic format. The common format includes an 8-bit block type (like the extension header type in IPv6), processing flags and block length. The processing flags indicate whether the block is to be copied in any fragment created, whether a status report should be issued if the block type is unknown to the node forwarding the bundle and whether the bundle should be dropped in this case. The indication to copy the block to each fragment is really designed for blocks carrying meta-data associated with delivery of the bundle contents such as handling restrictions, retention guidelines, digital rights management, or sensitivity labels. In the environments that require them, such meta-data are typically mandatorily bound close to the data they describe.

B. Fragmentation

The ability for bundles to be fragmented, either prior to transmission, or while in transit, has been an ongoing point of discussion since the original IPN work. At that time, fragmentation had been repeatedly adopted and abandoned. The motivation for bundle fragmentation was similar to that for IP fragmentation: to adapt relatively large bundles for transport using message-oriented protocols with finite limitations on message size. The argument against fragmentation was its complexity: in particular, the interaction of fragmentation

[2]http://www3.ietf.org/proceedings/06nov/slides/DTNRG-1/sld1.htm

[3]The common exception to this rule is when DTN has been placed in certain embedded systems that lack a real-time clock. In such cases, the system usually boots with a software clock set to the year 1970. This is expected to be a relatively minor problem, as more embedded systems become equipped with real time clocks and GPS.

with custody transfer. A key issue in this context is whether the granularity of a custody transfer acknowledgment could express a partial bundle and if fragments needed to be reassembled while in transit.

Early in the transition from the IPN to DTN architecture, and upon further understanding of routing requirements, the need for fragmentation became undeniable. In particular, understanding that a routing opportunity (contact) is not measured in bandwidth but instead measured in byte (storage) units (the product of a bandwidth and a time window of opportunity to use it), a way was needed to divide large bundles into appropriately-sized units to fill contacts. The term *proactive fragmentation* was therefore adopted to capture this case. Proactive fragmentation is performed ahead of time, before a contact of known duration and capacity becomes active. Proactive fragmentation can also be used when adaptation to lower-layer message-oriented transports is required. This is the case for which bundle fragmentation was originally considered.

Supporting fragmentation in the basic bundle protocol is not unusually difficult. A special header is used (similar to IPv6) to describe a fragment's offset and length relative to its original position in the bundle when it was first transmitted. As with IP fragmentation, bundle fragments are only required to be reassembled at the final receiver(s). Custody acknowledgments are expanded to be capable of describing partial bundles. Bundle fragments are identified as belonging to the same original bundle by a common identifier comprising a subset of the primary block (sender, receiver, origination timestamp).

Support for fragmentation in conjunction with encryption applied along a bundle's delivery path is a more challenging issue. Encryption can expand the size of a cleartext bundle when transformed into a cyphertext bundle, (e.g., when using typical cypher block chaining modes of encryption). Such encryption could be applied to a fragmentary bundle, and if routing selects different paths for different bundle fragments, the destination could eventually receive an overlapping mixture of cleartext and cyphertext fragments, making re-assembly challenging. Custody transfer can also be difficult; when different custody acknowledgments refer to the additional padding bytes created by the encryption, re-assembly can once again be challenging.

Fortunately, it is relatively easy to avoid such cases (e.g., by encrypting at the source) and algorithmic alternatives with equally strong security properties are available, such as *counter-mode* encryption [9]. Using this form of encryption, the cyptertext and cleartext versions of a bundle are of equal length, meaning the relevant fragmentation-related information (offset, length, and byte range descriptions) remain accurate whether or not bundles are encrypted.

C. Data Encodings

The bundle protocol uses two noteworthy approaches to its encoding of block fields. SDNVs are used to encode positive numeric values that may span a large range. This format is designed to be an efficient encoding scheme for relatively short positive integers (up to 56-bits long), but to also work for arbitrary length values (at the expense of some efficiency). The scheme uses the high order bit of each byte as a continuation flag, leaving seven bits remaining to carry information. This encoding scheme is also used by a lower layer delay tolerant link protocol called the *Licklider Transmission Protocol* (LTP) [10], which has been designed to act as a transport protocol for carrying bundles over high-delay private point-to-point paths.

Variable-length strings are grouped together in a the dictionary, located near the end of the primary block. A field referring to a string is then encoded as an offset to the beginning of the string within the dictionary. Strings make up the largest fraction of the byte overhead imposed by the bundle protocol; the primary header includes 4 variable-length EIDs, each of which are encoded using 2 variable-length strings (one for the scheme name, the rest for the scheme-specific part or SSP).

For the bundle protocol, the dictionary represents a convenient locus for reducing the otherwise large amount of space neeed to hold URI strings. The dictionary is similar to the string tables used by compilers when creating binaries including string literals. Any time a string literal is used more than once, the additional occurrences do not induce additional overhead. This is useful in the common case for DTN where the source and report-to EIDs of the primary bundle block (and possibly the current custodian) may all contain the same URI, but is also made available for *any* block of the bundle to reference using offsets.

D. Error Detection

The bundle protocol provides no bit-level error detection or correction mechanism apart from the message integrity checks associated with the bundle security mechanisms. If bundle security is not used, it is conceivable that bundles might have bits unintentionally modified in transit. Such modifications can occur either in application data or in bundle meta-data. This was a conscious design decision made by the designers, as the bundle protocol is intended for two primary uses. First, it can operate as an network layer, essentially replacing IP. In this case, error detection and correction are left to the higher layers based on similar reasoning.[4] Alternatively, the bundle protocol can be used above existing transport (or other) layer protocols, which commonly provide data integrity checks. This arrangement leaves bundle data potentially vulnerable to corruption if errors in the DTN forwarding engine or host occur.

In addition to the two use cases mentioned above that leave the question as to whether a bundle-layer integrity check is necessary unclear, there are applications where data with errors are valuable and where retransmissions are not desirable. For example, uncompressed image data from remote sensors, even if not error-free, may be valuable to deliver as soon as possible, especially if contact opportunities may be infrequent. The current design, therefore, leaves the task of bit-level error detection and repair up to the application.

[4]IP version 4 has both an IP-layer header checksum as well as transport-layer checksums covering some portions of the IP-layer header. The IP header checksum was removed when specifying the IPv6 header, leaving only the transport layer checksum for end-to-end error detection.

V. Routing

The subject of DTN routing has almost become an independent research area. There have been more than a dozen different routing schemes proposed, a few PhD theses, and a number of papers—mostly simulation studies that explore the particular features of one or more algorithms. This is perceived as a healthy situation, and was anticipated during the development of the DTN architecture. A survey of such schemes appears in a useful review by Zhang [11].

The DTN architecture claims applicability to a wide range of operating environments, and was therefore intended to support pluralism between the naming formats, routing algorithms, and network technology. The routing problem can be coarsely divided into whether the routing graph is assumed to be connected or not, with DTN typically aiming at the latter. In addition, methods for routing bundles may involve creation and deletion of single or multiple copies of a bundle, various degrees of knowledge about the topology and traffic pattern (e.g., past, current, and future contact, traffic load, and buffer occupancy), fragmentation, various levels of granularity in decision making, resource reservations, different routing for different class of service or custody bundles, and different options for the loci of the routing computation (e.g., at the edges with source-route forwarding versus routing computations at each node).

It is perhaps of little surprise that DTN concepts are beginning to find their way into the MANET literature, which has to date focused largely on routing in relatively dense mobile ad-hoc networks where end-to-end connectivity between any pair of nodes is possible. Combining DTN and MANET concepts together suggests that nodes may operate using some combination of simple forwarding and more delay-tolerant store-carry-forward operation. Some such networks go a step further and couple the routing system to node control systems. This coupling supports the ability to beneficially place nodes to act as routers or data ferries to enable communication even in otherwise sparse and partitioned networks. The DTN architecture and bundle protocol are careful to not restrict the type of routing that may be used in any particular operating environment.

Future DTN nodes will likely have to support a number of different routing strategies and protocols in order to operate efficiently in the vast diversity of environments in which the node may find itself. For example, a node may be well-connected whilst "at home," and can use standard Internet routing, but may then be subject to significant disruption (e.g., if it moves, or the environment changes) and so may have to switch over to a more complex routing scheme. Such transitions may even occur frequently for some nodes.

DTN routing may eventually involve not only path or next-hop selection toward a destination using a single metric of goodness, but possibly multiple routing solutions depending on the types of bundles being moved. For example, a long path including a reliable custodian may be preferable to a shorter path lacking such a custodian. In addition, DTN routing selects not only next hops at each forwarding node but also next protocol. Thus, a routing solution may involve not only a set of paths, but also a set of appropriate encapsulating protocols used to facilitate delivery using a heterogeneous set of transports.

VI. Custody and Congestion

DTN custody transfer is a service that may be optionally provided to a bundle as it is delivered through a DTN. When used, custody transfer keeps track of a current "responsible entity" or "custodian" for each bundle, and the custodian is required to keep the bundle safe in persistent memory until another custodian has received it successfully. Bundles may be moved from one custodian to another (nominally toward the bundle's destination), and an acknowledged transfer is accomplished for each. There are circumstances where this acknowledgment procedure can fail when the connection breaks during a transfer operation, or the network does not support bi-directional data transfer [12], [13]. These situations are expected to be relatively rare, but insufficient deployment experience leaves the question open at this time.

The custody transfer model and use of persistent storage at intermediate nodes provides the ability to delegate the responsibility for reliable data transfers to portions of the network other than the original sender, without violating the guiding end-to-end principal in IP networks [14]. This is possible, and even necessary, in the DTN context because we assume the original source of data may become unreachable or inoperable (e.g., due to environmental factors) before transmitted data reaches its ultimate destination(s). By migrating all the state regarding the correctness of the data transfer to an intermediate node ("custodian"), the "end point" (in the sense of [14]) has merely been moved to another location; it is still ultimately responsible for the correct conclusion of a data transfer operation.

Note that the DTN approach does alter the context for interpreting Clark's "fate sharing" concept [15]. His argument suggests that placing critical connection state within intermediate nodes is unwise, as the ability to withstand partial network failures decreases. In the DTN setting, however, there is no connection state. There can be critical copies of network message fragments resident in the persistent storage at custodians, but DTN allows the set of potential custodians to be configured. Therefore, the amount and location of critical state can be carefully controlled, and limited to those nodes known to be highly reliable. This is especially important in the cases where DTN intermediate nodes (e.g., potential custodians) can be more reliable and have better connectivity than end nodes, such as sensors or robots.

Not every node in a DTN needs to offer custody transfer. A node may refuse to accept custody for messages for implementation or policy reasons, because not enough free storage space is currently available, or for other reasons. The importance of having custody transfer be truly optional seems, at present, to be unclear. Many users of DTN networks wish to lose no data, so every node and every bundle operates using custody transfer or some equivalent capability. This may be adequate for a stable network with sufficient storage resources, but is not when the source rate exceeds the network delivery rate beyond the network's buffering capability. This is, in essence, the main problem of DTN congestion.

Of course, congestion control is a major area of study in computer networking. It has been explored much less extensively in DTNs, with only a few papers having been published (see [16] and [17]). The DTN architecture specification [2] indicates congestion is still a topic "on which considerable debate ensues." DTN congestion occurs when storage resources become scarce due to the presence of too much bundle data or too many bundle fragments. A node experiencing these situations has several options to mitigate the situation, in the following order of preference: drop expired bundles, move bundles somewhere else, cease accepting bundles with custody transfer, cease accepting regular bundles, drop unexpired bundles, and drop unexpired bundles for which the node has custody.

Given that expired bundles are subject to being discarded prior to the onset of congestion, there may be no such bundles to discard. Moving bundles somewhere else may involve interaction with routing computations; this is a reasonable approach if storage exists near the congestion point, and is the subject of [16]. It is also straightforward to cease accepting bundles with custody. This amounts to a form of flow control operating at the (DTN) hop-by-hop layer, and can result in backlogs of custody transfers as they accumulate upstream of congested nodes. To cease accepting regular bundles, the node essentially disconnects from its neighbors for some period of time. DTN tolerates such disconnections, but doing so can once again result in upstream congestion. The last two options are the least attractive, with the very last being all but prohibited. Dropping unexpired bundles results in a less predictable network from an end-user perspective, as the bundle lifetime capability is essentially disabled. While some protocol could be developed to propagate the policy-based early expiration times implemented by certain nodes, this has received no attention to date. Discarding bundles for which a node has taken custody defeats much of the delay tolerant aspects of DTN (but not the heterogeneity support). DTN attempts to provide a delivery abstraction similar to a trusted mail delivery service; discarding custody bundles is clearly antithetic to this goal.

Even after several years of design, the value of custody transfer and behavior of DTN congestion remains to be fully understood. It is likely these will remain poorly understood until the DTN architecture is more widely deployed and carries significant traffic loads. This is not entirely surprising, as a similar story arose in the early history of the Internet. The original TCP protocol specification included no management of congestion, and the problem remained poorly understood (and largely unrecognized) until the late 1980's, more than 10 years after the first experiments with Internet technology were performed.

VII. Security

DTN security has evolved over the years. Initially, when designing for the IPN, most of the focus was on so-called "security policy gateways," that would roughly control access to the space-segment of the network. Controlling access to that part of the network was the most important security control point, but once traffic entered, it was presumed to be authorized and so there was little or no need for cryptographic mechanisms to be defined as part of the bundle protocol [1].

At around that time, the idea of cryptographic authentication protecting only the headers was proposed. The logic was that protection of the entire payload might be expensive (in CPU terms) and that once the header was protected then the bundle as a whole could be authenticated as being "wanted traffic" as opposed to unwanted traffic. However, while this would be reasonable for the space segment of an IPN, it ignored the existence of intermediate hosts that are not part of the DTN (e.g., IP routers) that, if subverted, could then modify the bundle payload. This demonstrated the need for additional work to define a more fully-featured set of security mechanisms.

Today, the DTN bundle security protocol specification [18] defines basic data integrity and confidentiality mechanisms for bundles. The approach defines two different data integrity blocks: one for end-to-end integrity, and a separate one for hop-by-hop integrity (between adjacent DTN nodes). The rationale for the separation is to provide for different types of canonicalization and key management that are likely to be used for hop-by-hop vs. end-to-end cryptographic services.

Some DTNs (e.g., wireless sensor networks) may involve nodes that are extremely challenged in CPU terms, or more likely, in key management terms, and so cannot themselves encrypt, decrypt, sign or verify bundles. In addition, there may be some DTNs in which portions of the physical network topology are contained in physically secured facilities. Cryptographic protection at the bundle layer may not be necessary in these network segments. For these reasons, DTN security allows for intermediate DTN nodes (between the source and destination) to apply or check the validity of the cryptographic credentials. The relevant nodes in these cases are referred to as the *security source* and *security destination*, respectively, which can differ from the bundle source and destination. Whether or not these features prove useful in future DTN pilots remains to be seen, but they do represent subtle differences from how cryptographic services are used in most networks today.

There are a number of open issues in DTN security [4], some of which may be more tractable than others. First, the interaction of fragmentation and the application of cryptographic mechanisms can be challenging, as mentioned in Section IV-B. Given that support for cryptographic services is optional, then it is possible that a set of fragments could be reassembled where one of the fragments contains ciphertext. Clearly such combinations are a concern, and additional deployment experience will be required before we can confidently select between the various restrictions that might ameliorate these problematic situations. As discussed earlier, the current approach uses counter-mode ciphersuites only.

While the bundle security protocol defines cryptographic services, it does not (yet) provide any way to manage the required keys. Work on this is only really now beginning and various fairly standard approaches will have to be considered before some solutions are chosen. Of course, any solutions need to be appropriate for operation in DTN environments, where regular low-latency communication may be infrequent.

The last area of security that warrants further study is a

model for the authorization of traffic in DTNs that would be analogous to how the problem of authentication, authorization and accounting (AAA) is handled in the Internet today. Again, work here is just beginning, but in a sense this represents a full-circle: we now (almost) have sufficient basic mechanisms in place to finally tackle what was always going to be a major security problem in DTNs, as it is in Internet: the problem of unwanted traffic [19].

VIII. Future work

We have already discussed a number of areas where more work is required before we can consider the DTN architecture to be well-tested. These include routing, security, and congestion management. An additional area of concern is the API by which applications make use of the DTN architecture and its capabilities.

For routing, although a significant number of approaches have been discussed in the literature, relatively few have been demonstrated in live implementations (see [20] and [21] for notable exceptions), and none have been demonstrated at large scale in realistic environments. Combining traditional store and forward routing with store-carry-forward routing has also received relatively little attention beyond the academic literature.

In security, while we expect to see good progress on basic key management, we have yet to really see DTN security be shown to be robust, especially when faced with the types of attacks that are daily occurrences on the Internet. It will be interesting to see how well the DTN architecture, which supports a hop-by-hop security mechanism, holds up against such attacks, and in particular how robust DTN nodes will prove against denial-of-service attacks. When network capacity is scarce or connectivity is infrequent, the impact of denial-of-service attacks will likely be more devastating as compared with a well-connected Internet host.

Congestion management is an issue that has been raised since the earliest days of the IPN and DTN designs, yet has received relatively little attention. Perhaps this situation is the result of insufficient use (there is no significant congestion on links with low duty cycle), or perhaps this problem is so formidable that solutions remain elusive. Without the conventional type of low-latency feedback available in ordinary networks, congestion management and control can be exceedingly difficult. The approaches suggested to date tend to involve either allocated credits that are carried in packets in exchange for storage over time or mechanisms designed to offload congested nodes using storage available in local neighborhoods. The former approach requires some form of storage economy that seems independently challenging to construct, while the latter approach is useful only in cases where a sufficiently dense neighborhood to a congested node is available.

More work on the integration of DTN protocols with real world applications is needed to evaluate the success of the DTN approach. This need has motivated an investigation into asynchronous APIs that support not only DTN but also other recent network architecture proposals [22]. With a move toward time-decoupled interaction between clients and server, storage within the network becomes a central focus, and the behavior of custodians requires further investigation. Especially interesting may be how to manage the storage utilization at intermediate nodes or custodians among competing entities. Once again, economics may play a significant role here.

Methods for interfacing the bundle protocol with applications that assume current Internet naming and addressing schemes are also required. This suggests the need to construct a set of proxies, acting as both Internet and DTN applications, capable of translating the semantics of various existing protocols to and from DTN delay-tolerant environments. While constructing such proxies can range from the simple (email) to the nearly impossible (VoIP), a reasonable set of about half a dozen such proxies seems appropriate to begin vetting the basic ideas.

Finally, it remains to be determined whether some approach other than DTN is sufficient to handle the types of heterogeneity and disruption DTN is attempting to mitigate. For example, much work had gone into specific protocols for sensor networks that suffer from various problems (high loss, low power, low node capability, etc), yet a significant portion of that community is now changing its focus to supporting the IP/IPv6 protocols, but with accompanying routing changes to handle the high degree of loss and low power operation expected from networks of low-capability devices.

Beyond the areas known to require additional study and experience, there are mechanisms within the DTN architecture that could be causes for future problems. For example, while the DTN reporting mechanism has been found to be very useful in interoperability testing, lax use of this feature might overwhelm some disrupted links with excessive reports when deployed in real networks. In addition, the ability for DTN to store data for significant periods of time until network connectivity is re-established can lead to high-rate transmissions at the time of re-attachment. For sufficiently robust receivers, this is not a significant problem, but for others such bursty data transmissions can pose scheduling and resource allocation issues.

IX. Conclusion

Designing the DTN architecture was, in part, an exercise in recognizing one's personal assumptions about how networks operate in light of a broad set of operating environments to which most network designers are not accustomed. Given vast experience with the TCP/IP protocol architecture accumulated to date, it is easy to take many of its features as requirements for any network architecture, yet this is not always the appropriate course of action. For example, core IP assumptions about network connectivity and routing, use of relatively small packets, lack of persistent node storage, universal global addressing and end-to-end reliability have all been modified in designing the DTN architecture that must operate in a wider range of environments and with a potentially wide range of otherwise incompatible equipment.

The DTN architecture and its supporting bundle protocol, while having been developed over a number of years, are still relatively young. A few real-world deployments of the architecture have been tested successfully, but these have all

been temporary activities and have not been incorporated into mission-critical long-running applications. Perhaps surprisingly, after establishing most of DTN's core concepts (bundles, fragmentation, custody transfer, naming, etc), the architecture has undergone few radical changes, although modest changes have been made when authentication and confidentiality capabilities were brought into the model. It may yet be that further modest, or even major, ideological change will be required in fully implementing a delay tolerant multicasting capability, but whether this will have any profound effect on the bundle protocol remains to be seen and seems unlikely.

ACKNOWLEDGMENT

This paper reflects the work of many people involved in the DTN community and the DTNRG. The authors are especially indebted to Bob Durst, Keith Scott, and Susan Symington (MITRE), Scott Burleigh and Adrian Hooke (JPL), Vint Cerf (Google), Michael Demmer (UC Berkeley), and Jörg Ott (HUT).

REFERENCES

[1] K. Fall, "A Delay-Tolerant Network Architecture for Challenged Internets," in *Proc. ACM SIGCOMM '03*. New York, NY, USA: ACM Press, 2003, pp. 27–34.

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

[3] K. Scott and S. Burleigh, "Bundle Protocol Specification," Internet RFC 5050, Nov 2007.

[4] S. Farrell and V. Cahill, *Delay– and Disruption–Tolerant Networking*. Artech House Publishers, 2006, iSBN: 1-59693-063-2.

[5] I. Akyildiz, Y. S. W. Su, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393–422, 2002.

[6] T. Berners-Lee, T. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic syntax," Internet RFC 3986, Jan 2005.

[7] Y. Matsushita, M. Sakuma, H. Nishigaki, N. Miyazaki, and I. Yoshida, "An Overall Network Architecture Suitable for Implementation with either Datagram or Virtual Circuits Facilities," *SIGCOMM Comput. Commun. Rev.*, vol. 8, no. 3, pp. 5–24, 1978.

[8] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The Design and Implementation of an Intentional Naming System," in *Symposium on Operating Systems Principles*, December 1999.

[9] M. Dworkin, "Recommendation for Block Cipher Modes of Operation: Galois/Counter (GCM) and GMAC," *NIST Special Publication 800-38D*, November 2007. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[10] M. Ramadas, S. Burleigh, and S. Farrell, "Licklider Transmission Protcool," Internet-Draft draft-irtf-dtnrg-ltp-07.txt, October 2007, work in progress.

[11] Z. Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges," *IEEE Communications Surveys and Tutorials*, vol. 8(1), pp. 24–37, 2006. [Online]. Available: http://ieeexplore.ieee.org/iel5/9739/4116778/04116780.pdf

[12] K. Fall, W. Hong, and S. Madden, "Custody Transfer for Reliable Delivery in Delay Tolerant Networks," Intel Research Berkeley, Tech. Rep. IRB-TR-03-030, Jul 2003.

[13] E. Duros and W. Dabbous, "Supporting unidiretional links in the internet," in *Proceedings of the 1st International Workshop on Satellite-based Services*, 1996.

[14] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, no. 4, pp. 277–288, 1984.

[15] D. Clark, "The Design Philosophy of the DARPA Internet Protocols," in *Proc. ACM SIGCOMM*. Stanford, CA: ACM, Aug. 1988, pp. 106–114. [Online]. Available: citeseer.ist.psu.edu/clark88design.html

[16] M. Seligman, K. Fall, and P. Mundur, "Alternative custodians for congestion control in delay tolerant networks," in *Proc. ACM SIGCOMM Workshop on Challenged Networks*. New York, NY, USA: ACM Press, 2006, pp. 229–236.

[17] S. Burleigh, E. Jennings, and J. Schoolcraft, in *AIAA 9th International Conference on Space Operations (SpaceOps)*, 2006.

[18] S. Symington, S. Farrell, H. Weiss, and P. Lovell, "Bundle Security Protocol Specification," Internet-Draft, draft-irtf-dtnrg-bundle-security-04.txt, Sep 2007, work in progress.

[19] L. Andersson, E. Davies, and L. Zhang, "Report form the IAB Workshop on Unwanted Traffic March 9–10, 2006," Internet RFC 4948, Aug 2007.

[20] S. *et al*. Guo, "Design and Implementaiton of the KioskNet System (Extended Version)," University of Waterloo, Tech. Rep. CS-2007-40, Nov 2007.

[21] X. Zhang, J. Kurose, B. Levine, D. Towsley, and H. Zhang, "Study of a Bus-based Disruption-Tolerant Network: Mobility Modeling and Impact on Routing," in *ACM Mobicom*, September 2007.

[22] M. Demmer, K. Fall, T. Koponen, and S. Shenker, "Toward a Modern Communications API," in *Workshop on Hot Topics in Networks (HOTNETS)*, November 2007.

Kevin Fall Kevin Fall (S'87–M'99–SM'06) received the Ph.D. degree, in 1994, and M.S. degree, in 1992, from the University of California, San Diego in Computer Science and Engineering, and the B.A. degree in 1988 from the University of California, Berkeley in Computer Science.

He joined Intel Corporation's research lab in Berkeley, California in 2001, where he is currently a Principal Engineer. Since 2006, he has also been a visiting scholar at the Woods Hole Oceanographic Institution. His research interests include communication in unusual and stressed environments, network architecture and performance, simulation, and information security.

Stephen Farrell is a research fellow at Trinity College Dublin and Chief Technologist with NewBay Software. Stephen received a Joint Honours B.Sc. in Mathematics and Computer Science from University College Dublin in 1986. His research interests include security and communication in unusual and stressed environments,