Computer Communications 32 (2009) 1724–1730

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

The design and implementation of a session layer for delay-tolerant networks

Michael Demmer [a,*], Kevin Fall [b]

[a] University of California Berkeley, Computer Science Division, 387 Soda Hall, Berkeley, CA 94720, USA
[b] Intel Research Berkeley, 2150 Shattuck Ave, Penthouse Suite, Berkeley, CA 94704, USA

ARTICLE INFO

Article history:
Available online 20 February 2009

Keywords:
Session layer
Multicast routing protocol
Design
Implementation
Applications

ABSTRACT

The DTN architecture is based around sender-initiated unicast communication that is insufficient or inconvenient to meet the needs of many applications. To address these limitations, we define a DTN session layer and associated extensions to the DTN bundle protocol that more naturally support receiver-driven applications and multicast communication. Within a session, we provide mechanisms allowing applications to convey ordering relationships between successive transmissions that can be used by the network to help ensure a distributed application's delivery ordering expectations are met. We also extend the bundle protocol's expiration procedures to support more efficient network utilization by allowing in-network deletion of obsolete messages. We present the design rationale and describe our implementation of these mechanisms and discuss their advantages in meeting the needs of several popular types of applications.

1. Introduction

It is well known that many application protocols designed for the Internet architecture cannot operate well in challenged network environments that lack the relatively low-latency, fully-connected characteristics required for TCP-based communication. The Delay-Tolerant Networking (DTN) architecture [1,2] and its supporting Bundle Protocol (BP) [3] remedy many of these problems by providing an alternative, asynchronous service model based on application-defined data units (called *bundles*), medium-term storage within the network to deal with temporary link outages, and a wide diversity of network transport protocols. These and other factors allow the bundle protocol to offer significant advantages over TCP-based protocols when running in challenged network environments [4].

Despite these advantages, many applications utilize design patterns that cannot be naturally and efficiently adapted to the basic DTN service model, due to three key limitations. First, the BP treats each bundle independently, so there is no way for an application to convey to the network that a group of bundles are related. Put another way, the BP, and the DTN architecture more generally, has no concept of a communication *session* that may span multiple individual bundles. This limitation constrains the expressiveness for applications that may want to convey inter-bundle ordering or timing information, and means that the network cannot make routing or provisioning decisions based on knowledge of relationships between multiple bundles.

The second limitation (in part a consequence of the first) is that bundle communication is purely sender-initiated. There is no generic way for an application to *request* a particular piece of content or to receive a bundle transmission. Consequently, when adapting receiver-driven applications from Internet protocols (such as HTTP GET) to DTN protocols, proxy applications must be constructed that first send a request as one bundle, then send a corresponding reply in another. Because the DTN network is unaware of this communication pattern, it cannot treat the bundles in the request/response communication pattern as related. In contrast, having greater semantic knowledge of the communication pattern could provide for more efficient network operation. For example, such knowledge might allow a routing algorithm to set up dynamic forwarding state when it sees the content request, so that the response can be routed back to the requester.

The third limitation is the BP's lack of support for efficient group-based communication. Although there has been some prior work on defining the protocol needs and semantic models for multicast communication in DTNs [5,6], there is as of yet no group membership protocol or fully-specified service model for multicast communication in DTN environments. This limitation can lead to obvious inefficiencies in bandwidth-constrained environments where multiple downstream clients may be interested in the same content, but the current transfer protocol requires separate bundle transmissions for each destination endpoint. Although DTN can make use of lower-layer multicast-capable protocols, it is nonetheless desirable to implement group-based communication at the

* Corresponding author.
E-mail addresses: demmer@cs.berkeley.edu (M. Demmer), kfall@intel.com (K. Fall).

0140-3664/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2009.02.010

DTN layer itself, because efficient distribution of content in a DTN environment requires management not only of network paths but of storage along each path, a requirement most underlying multicast protocols fail to provide.

In this work, we aim to remedy these limitations by proposing several BP extensions to support session-based communication using a publish/subscribe group membership service model and protocol. We then develop and evaluate these extensions within our version of the DTN reference implementation [7]. More specifically, we offer the following contributions:

- We present a framework for supporting DTN *communication sessions*, including a method for applications to designate a logical partial ordering among multiple bundles belonging to the same session.
- We introduce the notion of a *session custodian*, which is a DTN application that commits to storing and making available, now or in the future, bundles that comprise a communication session.
- We describe a new publish/subscribe service model for DTN communication, whereby applications can register as publishers, subscribers, and/or custodians for a session.
- We use bundle partial ordering to allow applications to designate that a particular bundle renders previously-transmitted bundles obsolete. This provides an optimization opportunity for the underlying network to purge in-transit stored bundles that have become obsolete.
- Finally, we discuss ways in which these mechanisms can efficiently support several applications in DTN environments, including content syndication, periodic data transmission, and distributed shared storage systems.

The remainder of this paper proceeds as follows. In Section 2 we motivate our work through a description of several application use cases. We present the design rationale for these additions in Section 3, followed by a discussion of implementation details in Section 4. We briefly survey related work in Section 5 and conclude in Section 6.

2. Motivations

We are motivated by two main characteristics shared by several example applications: (i) their application protocols fail to perform well in challenged network environments and (ii) the existing sender-initiated unicast service model of the basic DTN bundle protocol is insufficient or inconvenient to implement them.

Many popular Internet applications, including the ones we consider below, are designed around a basic assumption that two hosts are able to communicate with relatively low-latency and high bandwidth at any time. When these assumptions do not hold, the applications and their supporting protocols do not function well (or at all). Often this is due to non-robust handling of network connection failures or overly-optimistic timeouts that are not well matched to networks with high delays.

DTN is generally proposed for use in-network environments characterized by frequent interruption, limited network bandwidth, plus long and/or variable delay. At any given time, two hosts may or may not have a traditional network path between them. Bundle protocol agents ameliorate these problems by storing bundles temporarily during network partitions and outages. For some communicating host pairs, there may never be a traditional end-to-end contemporaneous network path so some approach other than conventional Internet-style networking would be required in such settings. Yet as we now discuss, there are several applications of interest that cannot be mapped to the basic bundle protocol service easily, motivating our interest in developing a session layer for DTN and the associated mechanisms to support it.

First, we consider the case of implementing *syndicated content distribution* in a challenged network environment. In well-connected Internet environments, content providers often distribute information using protocols such as RSS [8] or ATOM [9] over HTTP. In these protocols, clients periodically fetch an XML document that contains a list of current *items* for a particular content *feed* (identified by a URL). Each item is described by a unique identifier, a title and summary, a link to additional information, and timestamp information that indicates when the item was published and optionally when it was updated (re-published with new content). A client-side application periodically retrieves the list and compares the per-item timestamps with the corresponding timestamps of items in its local cache. New or updated items are displayed to the user.

In cases where the underlying network fabric is intermittent, the feed may not be available when a client wants to fetch the current list of items. One way to remedy the problem would be to deploy a *gateway proxy* at a well-connected site that periodically refreshes the current list and proactively distributes it to all interested clients. Yet this proxy would require some means of knowing which clients are interested in which content feeds, either through manual configuration, or through a custom subscription protocol. Also, in many cases only one item out of several on the feed is updated, leading to inefficiencies when retransmitting the whole XML document. Finally, in cases where multiple clients may share a constrained network link, unicast-only transmission wastes network resources.

As a second example, consider an application that distributes regularly-updated information such as weather forecasts, commodity prices, or traffic reports to a set of clients that may be intermittently connected. As in the syndication example, this application would benefit from a multicast service model to better utilize constrained network links. Furthermore, there is a design challenge relating to the frequency at which messages are generated. If messages are generated too often, then a network outage in the middle of the (DTN) network would cause several updates to queue up on one side of the network partition, waiting for connectivity to be restored. Even though the earlier updates are superseded by the more recent information conveyed in the later transmissions, the network has no way of knowing this. Rather, it delivers redundant updates that consume valuable network bandwidth when the partition heals and they are transmitted to the destination.

Finally, in the case of distributed shared storage systems, existing protocols like NFS rely heavily on continuously-available low-latency connectivity. In previous work, we have designed an alternative approach called TierStore [10] that provides an opportunistically-consistent shared filesystem as a framework for building information distribution applications in challenged networks. Although TierStore uses the bundle protocol for inter-node communication, it relies on manual configuration of a distribution topology among nodes in the network and has no way of adjusting this topology in response to changes in the network conditions.

In the remainder of this paper, we show how the addition of a session layer with several key features remedies these shortcomings in the DTN service model and enables these and other applications to be constructed in a more robust, straightforward, and efficient manner.

3. A DTN session layer

With the above motivations in mind, we now describe the design in more detail. We begin with the semantics of the proposed

DTN session service model and how a session is named and manipulated. We then cover how a partial ordering of messages within a session can be expressed by an application and how this can be used to implement in-network discarding of obsolete bundles. We finish the discussion with an overview of how group membership is handled, comparing how the assumptions in IP multicast differ substantially from those in DTN multicast, leading to a different set of choices within the design space.

3.1. Session names and service model

We extend the existing DTN service model by defining a *session* as containing a group of bundles that have an application-defined relationship. Each session is named with a URI, as are all Endpoint Identifiers (EIDs) in the bundle protocol. Any legitimate URI can be used as the name of a session, making the system flexible enough to use a variety of naming formats, both new and existing. This design simplifies the construction of application proxies for use in DTN environments, as these proxies can in many cases simply use the identifiers used by the application protocol, thereby avoiding the necessity of managing a new namespace.

Sessions may span multiple senders and receivers over time, and include intermediary nodes devoted to storing the contents of a session for later retrieval. The underlying communications protocols utilize both single destination (unicast) or multiple destination (multicast) transports, and the structure or format of endpoint identifiers do not necessarily need to differ depending on the combination.

This approach to naming contrasts with the IP multicast addressing model [11], that utilizes a single common address space for both unicast and multicast communication. With the flexibility of URIs, it is not necessary to "carve out" a portion of the endpoint identifier space specifically for multicast use. In DTN, unicast sessions are treated as merely a simplified case of multicast communication with one member in the destination group.

In the existing bundle protocol service model, applications reference URIs as EIDs to *register* with a *bundle protocol agent* via an API to send and receive bundles. We extend this service interface with a set of options to allow an application to select one or more of its intended *roles* related to a communication session. By enhancing the network API to carry this type of information, certain underlying network provisioning tasks can be made more efficient. The three roles an application can play in a session are as follows:

- *Publisher* applications generate bundles and make them available for distribution on a communication session.
- *Subscriber* applications receive bundles that were generated by publishers.
- *Custodian* applications are responsible for receiving published bundles and making them available to new subscribers.

The first two roles are familiar concepts, as our session service model follows a classic publish/subscribe communication pattern [12]. In particular, publishers and subscribers need not be aware of each other's identity, and instead communicate by publishing content to or subscribing for content from a session identified by an abstract identifier (e.g., topic name). Also, subscribers and publishers need not interact with the network contemporaneously; one application can publish a bundle to a session and another can subscribe to retrieve the bundle later, assuming the bundles is not obsolete (see below) and its lifetime has not expired. Thus, our proposed session model provides both spatial and temporal decoupling which is intuitively compatible with the types of physical environments DTN is targeted for. As a consequence, however, the asynchronous interaction model requires some entity in the network to store transmitted bundles until they expire (or are rendered obsolete). This is the function of the custodian role.

More specifically, the custodian role is used to enable temporally-decoupled communication and/or receiver-initiated transfers in a DTN context. When an application elects to be a custodian for a session EID (or a set of EIDs), the network delivers a notification to the custodian when the first publisher and subscriber applications register on the session. The custodian is then responsible for storing published messages and making them available for subscribers that may join later.

In this way, the custodian acts similarly to a *broker* in a classic publish/subscribe interaction, as it matches publishers with subscribers. As we discuss below, however, DTN routers ordinarily store data within the network, so a DTN router is a natural point to instantiate a session custodian. Not surprisingly, this coincidence is not accidental.

The custodian mechanism also allows an application to generate content "on-demand" from exogenous sources in response to requests that occur at client nodes in the network. For example, when implementing an HTTP DTN proxy that imports web traffic into a DTN network, an infrastructure-based proxy application would first register as a custodian for a range of EIDs, say `http:*`[1]. Then when a subscriber expresses interest in a particular content URL, a client-side proxy would register as a subscriber for that URL, resulting in a notification to the custodian. The custodian would then download the requested web content and publish it to the requester via using a DTN session named with appropriate URI.

3.2. Sequence identifiers and obsolete messages

The ability to group together related messages offered by the session model provides a first step in meeting the needs of some distributed applications. However, the ability to express the logical ordering relationships between bundles in a session is necessary for those applications that are sensitive to causality relationships. For example, in an application that transmits deltas on a shared database, applying the differences in the wrong order can corrupt the database. Because the arrival order of messages may differ from their transmission order due to in-network reordering or multiple-path routing, some additional mechanisms are required to establish and maintain application-specified bundle orderings.

A simple way to determine the bundle generation order and spacing would rely on the *creation timestamp* that is part of the basic DTN bundle protocol. For multiple message transmissions from the same node, comparing the creation timestamps indicates their transmission order. Even in cases of multiple senders, DTN nodes are expected to have (at least loose) time synchronization so some information can be gleaned by comparing the timestamps among messages from different sources. In the content syndication example presented above, different items on the same feed have no ordering requirement, so the network can deliver them and they can be displayed in any order.

Some applications, however, may require a richer mechanism than timestamps to express bundle ordering relationships. If an item is published and then subsequently updated, a temporal dependency is formed, necessitating maintenance of an appropriate ordering. Also, in multi-party communication in a distributed system (such as used in TierStore), it is critical for nodes to determine the set of other messages that had been received and processed earlier before a node sends a new message, in order to determine the causality relationship between the events [13].

[1] This and other examples assume an endpoint identifier pattern matching mechanism that uses a similar syntax to UNIX glob string matching function, as is used in the DTN reference implementation.

To address the need for more sophisticated ordering options, our session layer allows applications to attach a multi-purpose *sequence identifier* to a bundle destined for transmission. A sequence identifier may be a scalar or vector, and indicates a partial ordering relationship among two or more bundles based on time or sequence. We adopt a generalized mechanism based on *vector clocks* [14] that meets these requirements. Our goal in designing this mechanism is to support a range of use cases, including sequential sequence numbers assigned by a single sender (e.g., as in TCP), real-time timestamps with a hash-based validator (e.g., as in HTTP 1.1 [15]), as well as richer semantic identifiers used in multi-sender distributed systems contexts. In these contexts, it is often significantly more useful for applications to know whether one bundle was generated with knowledge of another bundle, as opposed to simply whether one bundle was generated before or after another in a global time context. Thus vector clocks are an effective way of conveying the causality relationships that are important to many distributed, multi-party applications. We discuss our use of vector clocks in more detail in Section 4.2.

Equipping the network with knowledge of an application's sequencing requirements allows it to make an optimization for discarding unwanted or obsolete data early. Recall the example from the previous section in which an application is periodically broadcasting updates from some information source such as a weather forecast, and wants to avoid the potential buildup of many redundant messages in case of a network partition. The application could transmit at a lower rate which would limit the queue buildup, but at the same time would reduce the update frequency for clients. Alternatively, it could set a shorter lifetime for the messages, which would also limit the queue buildup, but might mean that some clients never receive any updates if the end-to-end delay in message transfer exceeds the message lifetime.

To implement this optimization, we allow applications to designate an optional *obsoletes identifier* to convey that a set of other bundles have been rendered obsolete by the new message. The obsoletes identifier is a point in the sequence space where the notion of "older" is well-defined. Bundles older than the obsoletes identifier are considered obsolete, and the newer bundle is said to obsolete the older ones. This allows for controlled discarding of obsolete bundles as indicated by an application as opposed to a method chosen independely by the network. With this capability, when a DTN router queues a bundle, the bundle's obsoletes identifier is compared with the sequence identifiers of other queued session bundles and any obsolete bundles are discarded. Thus by using the obsoletes identifier, applications can transmit updated information at a rate that is most appropriate for the application's needs, without worrying about potential downstream queue buildup caused by its own previous messages.

3.3. Group membership and bundle state

The final component in our design discussion relates to the group membership protocol used to implement the session service model. Fundamentally, the goals of this protocol are similar to those used to implement IP multicast subscription interest (e.g., IGMP [16] and MLD [17]). We also require associated multicast routing protocols like PIM [18,19] or DVMRP [20] to build a distribution tree within the network with sufficient forwarding state to deliver packets (or bundles) to all subscribers. Nodes can join and leave the distribution tree, and the routing algorithms maintain the best path(s) through the network to deliver data to subscribed nodes.

However, due to the nature of challenged network environments, our design is quite different from IP multicast approaches. Recall that our target environment is characterized by intermittent connectivity, highly variable round trip times, and/or constrained bandwidth links. We use storage within the network to combat these challenges, so any support for multicast must deal simultaneously with path availability and storage management.

Because a link may be down for a considerable period of time, each node conveys a *subscription lifetime interval* when it joins a distribution tree, and periodically updates its subscriptions before this interval elapses. That way if an upstream node has not heard from any downstream subscriber when the interval lapses, the subscriber subtree is pruned from the distribution tree. The duration of a subscription lifetime interval is controlled by a configuration parameter based on an upper limit of the expected time for network outages. We also have a mechanism allowing downstream nodes to proactively unsubscribe from sessions they are no longer interested in.

As suggested, our protocol does not treat a link outage as an event that necessarily alters the distribution tree. At any point in time, a session distribution tree may include links that happen to be down, with the expectation that those links will heal in the future. This design avoids unnecessary recomputation of a distribution tree in cases where links' connectivity fluctuates. In prior work, we applied a similar approach to the design of Delay-Tolerant Link State Routing (DTLSR) [21] in which the routing graph can include links that may be down at a certain point in time but are expected to come back up in the (near) future.

In DTN, nodes on a distribution tree for a session cache a copy of all unexpired, non-obsolete bundles on the session, even if such bundles have been delivered to all current downstream subscribers. This allows for late joiners (or earlier joiners who were in an isolated portion of the network due to partition) to still receive unexpired and non-obsolete session data. It is possible for network conditions to change such that a distribution tree should be altered for efficiency. For example, a child node may select a new parent, but because it was already a member of a particular session, it already has a copy of all known bundles on the session. As we discuss in more detail below, we include summary information that describes the set of bundles that are already stored for the session along with the subscription message, allowing the new parent to elide unnecessary transmission of bundles that are already stored at the child.

Note how our approach differs from dense mode IP multicast prune state [18], a form of soft-state which prunes downstream neighbors until the state is removed. Our state is essentially the opposite: it provides data flow to the subtree as long as the state is present. Our type of state is more similar to group state from protocols such as BIDIR-PIM [22].

In our design we favor decisions that conserve network bandwidth at the potential expense of storage and computation. We tend to favor hard (or "firm") state over soft-state, and we try to convey a relatively rich amount of application semantic intent to the network. These decisions helps to combat the relatively harsh operating environments we must tolerate for DTN networks. In contrast, IP multicast implementations make essentially the opposite choice, as IP routers do not store messages once they have been delivered to downstream subscribers and have essentially no knowledge of an application's intent, potentially consuming additional network bandwidth in cases where late subscribers need to receive previously-transmitted packets. This design disparity clearly reflects the different environmental assumptions behind DTN multicast protocols and Internet multicast protocols.

4. Implementation details

We now briefly describe the details of our session layer implementation. We used the DTN reference implementation [7] as the

framework for our extensions, because it supports evaluation using both simulation and real-world experimentation.

4.1. Session service interface

To support the ability for an application to register its role in a session and manipulate its bundle sequencing, we added extensions to the DTN reference implementation's API. First, we modified the registration service, used by applications to indicate interest in receiving bundles destined for a particular endpoint identifier, by adding options to select one or more of the session application roles (*publisher, subscriber, custodian*). The bundle protocol agent monitors these application registrations and then engages the relevant group membership and routing protocols to set up the required forwarding state.

A publisher application informs the bundle protocol agent that a bundle transmission is part of a session by first registering for a session EID in publisher mode and then passing the returned registration identifier along with the bundle transmission API call. To communicate this information to other nodes in the network, we defined a new bundle protocol extension block called the *session block*. This block contains the endpoint identifier of the session, which is encoded for efficiency using the dictionary mechanisms defined in the bundle protocol specification [3]. This encoding enables efficient transmission of the session EID in cases where it is the same as (or shares components with) the URIs used in the bundle's destination and/or source EIDs.

We also added a new API call to notify custodian applications of new subscriber and/or publisher interest in a particular session EID. As implied by the example in Section 3.1, when an application registers as a custodian, it can supply a pattern that covers a range of session endpoint identifiers, using the glob-based wildcard syntax offered by the reference implementation. Thus a custodian application, upon receiving notification of subscription interest in a particular session EID that matches the wildcard pattern, would then create a new registration to transmit and/or receive bundles on the specific session.

4.2. Sequence identifiers and vector clocks

As mentioned above in Section 3.2, we developed an extension to the bundle protocol to attach a sequence identifier to a bundles using a format based on logical vector clocks [14]. Vector clocks are commonly used in distributed systems to express causality relationships between events that do not rely on fine-grained global time synchronization.

In a standard vector clock distributed system, time is expressed as a vector of (node, counter) tuples, one for each node in the system. Each node maintains a vector clock that represents its notion of the "current" time. Whenever an event occurs at a node, the node monotonically advances the counter in its own entry in the clock and includes a copy of the entire clock in any message it sends to other nodes. Upon receiving a message, a node updates its vector clock such that each entry will have the most recent (highest) corresponding counter value. This provides the node with a snapshot as to where all other nodes are in their sequence space (or were at the time the update messages were sent).

In this way, causality between two events $E1$ and $E2$ can be determined. If at least one entry in $E1$'s vector clock is greater than the corresponding entry in $E2$, and no entry in $E2$ is greater than the corresponding entry in $E1$, then $E1$ must have occurred after $E2$ was observed by the node that generated $E1$. In contrast, if one entry in $E2$ is greater than its counterpart in $E1$, yet another entry in $E1$ is greater than that in $E2$, then the two events occurred concurrently. Thus, for example, the clock <(A,5)(B,3)(C,6)> is greater than <(A,3)(B,3)(C,2)>, whereas <(A,5)(B,3)(C,6)> is concurrent with <(A,4)(B,5)(D,6)>. Two clocks are said to be equivalent if and only if all entries match.

We make one additional modification to the standard vector clock algorithm to increase its flexibility by allowing an application to designate that some elements of the vector clock contain *unordered* counters. These counters are ignored when comparing vectors elements for order, but checked when comparing them for equality. This is used, for example, with HTTP 1.1 strong validators [15] that consist of a (timestamp, entity validator) pair, in which the timestamp marks the time when the object was generated, and the validator is an implementation-defined content identifier (such as a hash of the content itself). The timestamps are compared when determining which object is more recent, but the entity tag is compared for equality, as when determining whether a cached object is equivalent to a server object. This approach is needed to handle cases where an object is updated multiple times within a single timestamp granularity.

We implement sequence identifiers using another bundle protocol extension block called the *sequence identifier block* to encode a vector clock and attach it to the bundle. Each column in the clock contains a URI and an integer value counter. We again use the dictionary mechanisms to efficiently encode the URI elements and we use Self-Describing Numeric Values (SDNVs) to encode the counters, as described in the bundle protocol specification [3]. To implement the message obsoleting feature, we add a second optional extension block (the *obsoletes identifier block*) to the bundle using the same vector clock encoding format as the sequence identifier block.

4.3. Session membership protocol

To support the dissemination of session membership interest, we implemented a new administrative protocol using bundles to convey subscription messages. This protocol is responsible for constructing and maintaining the session distribution tree, notifying custodian applications when a subscriber or publisher has registered for the session, and properly forwarding session bundles to all subscribers along the distribution tree.

To implement this protocol, we first defined a new EID scheme `dtn-session`, in which the scheme-specific-part contains another embedded EID identifying a session. Routers use this scheme for two new types of administrative bundles: SUBSCRIBE and UNSUBSCRIBE, analogous to join/leave packets in IGMP [16]. When a DTN application registers as a subscriber for a session to which the bundle protocol agent is not already subscribed (e.g., feed://www.dtnrg.org/hg/DTN2/rss-log.xml), the agent generates a new SUBSCRIBE message with a destination EID in the `dtn-session` scheme (e.g., dtn-session:feed://www.dtnrg.org/hg/DTN2/rss-log.xml).

To set up the forwarding state, the router needs to notify an upstream node there is a new subscriber, so it forwards the subscribe message toward a custodian for the given session. The mechanism by which the custodian routes are distributed through the network depends on the particular routing algorithm in use in a particular deployment, and is not mandated by this group membership protocol. For example, we have added extensions to our implementation of DTLSR such that when applications register as a custodian for a session, a new route advertisement (in the `dtn-session:` scheme) is distributed through the network to allow subscription bundles to be properly routed towards the custodian. However, other routing mechanisms, both static and dynamic, could also be used to advertise the `dtn-session` routes. By using a new naming scheme for the subscription messages, the system can leverage existing routing implementations to distribute routes towards custodians.

When a session member receives a subscription request from a new node, it checks whether the node is already subscribed. If so,

the member adds the new subscriber to the distribution tree and determines which of the cached session messages need to be transmitted to the new node. If the new node was not previously subscribed to the session, then all cached messages need to be transmitted. However, if the new node was previously subscribed (implying that it is either refreshing its subscription or changing its upstream link to the session), then it includes a set of one or more sequence identifiers in the SUBSCRIBE bundle to summarize the set of messages it has already received. This allows the member to elide transmission of bundles that have already been seen. The subscription bundle thus eventually arrives at a node where an application has registered as a custodian for the session, triggering a notification to that application of the new subscriber.

Once all subscriber applications have relinquished their registrations and there are no other downstream subscriber nodes, a node leaves the distribution tree by generating an UNSUBSCRIBE message and transmitting it upstream. Also, as mentioned previously, routers need to periodically refresh their subscriptions by generating new SUBSCRIBE bundles and sending them upstream. A node indicates its desired subscription interval (i.e., how long before it will transmit a subscription refresh message) through the lifetime setting on the subscription bundle. Once a node receives a subscribe message, it starts a timer for the remaining lifetime of the subscription bundle. If that timer expires before a new subscription bundle is received, then the downstream node is assumed to be disconnected and is removed from the distribution tree.

5. Related work

We now briefly survey related work influencing our DTN session layer design. The service model and structure of the application interface are based on the wide range of publish/subscribe systems [12] that have been proposed in the literature and are used in practice. Also, in prior work [23], we proposed the use of the publish/subscribe model as the basis for a new general purpose network API in order to, among other things, make it easier to adapt applications to intermittent network environments. Here we extend that work and propose specific mechanisms needed to implement the publish/subscribe communication pattern in challenged network contexts.

The IP group membership and multicast routing protocols influenced our design for the service model and group membership protocol for the session layer. As noted above, IP multicast protocols assume that networks are available as needed, thus protocols for reliability and for message delivery to late joiners are based on retransmissions. In contrast, our protocols rely more on caching and place a higher premium on network transmissions to accommodate constrained bandwidth environments.

Although the distributed systems literature has several examples of systems based on vector clocks to represent causal ordering, our implementation was based primarily on our prior work in designing the TierStore distributed storage service [10].

In the context of multicast in DTNs, Zhao et al. [5] proposed a taxonomy of semantic models for multicast in DTNs, as well as some simulation results of classes of multicast routing algorithms. They proposed new semantic models of multicast communication and group membership based on different temporal constraints, intended to take into account the potentially lengthy delays between nodes. Our sequence identifiers and obsoletes identifiers serve a similar purpose, except that rather than relying solely on time as a discriminator, our identifiers are more flexible because they can be used for application-defined ordering purposes.

Symington et al. [6,24] explored the challenges involved in providing custodial bundle service to multicast destinations. In particular, they consider several implementation challenges that result when a node that does not accept custody still needs to branch a bundle transmission to multiple downstream nodes over time. While some of these concerns would need to be addressed in our session model, our approach requires all subscribed nodes to keep a cached copy of all the bundles that comprise the session until they expire or are obsoleted, making several of their mechanisms unnecessary or appropriate for our assumptions. However, some environments may not allow caching of all a session's bundles, and further investigation is appropriate for these cases.

6. Conclusions and future work

In conclusion, the addition of a session layer to DTN protocols and service model can effectively remedy several shortcomings in the DTN architecture, making it easy and natural to construct several useful information distribution applications that can function well in challenged network environments.

We are currently implementing several example applications that leverage these session layer protocols and help to evaluate their efficacy and robustness, including a syndication proxy to distribute RSS/ATOM content, a web proxy to enable offline downloads of web pages, and a re-implementation of the TierStore distribution layer. Through the use of these applications, we hope to gain insights into the benefits (and shortcomings) of our session layer proposal from the perspective of the applications.

We are also exploring mechanisms to enable multiple custodians to register for the same communication session. Essentially, the idea is to provide an anycast-like service in which multiple distribution trees would be constructed for the different sets of subscribers, each rooted at the "nearest" custodian based on the routing topology. This addition requires another protocol to ensure that the custodians remain consistent with each other. Finally, we are exploring ways in which this session layer can leverage lower-layer multicast services (such as IP multicast) when they are available.

References

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, H. Weiss, Delay-tolerant networking architecture, RFC 4838, April 2007.
[2] K. Fall, A delay-tolerant network architecture for challenged internets, in: Proceedings of the ACM Symposium on Communications Architectures & Protocols (SIGCOMM), 2003.
[3] K. Scott, S. Burleigh, Bundle protocol specification, RFC 5050, November 2007.
[4] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, R. Patra, Implementing delay tolerant networking, Technical Report IRB-TR-04-020, Intel Research Berkeley, December 2004.
[5] W. Zhao, M. Ammar, E. Zegura, Multicasting in delay tolerant networks: semantic models and routing algorithms, in: Proceedings of the ACM SIGCOMM Workshop on Delay-Tolerant Networking (WDTN), 2005.
[6] S. Symington, R. Durst, K. Scott, Delay-Tolerant Networking Custodial Multicast Extensions, Internet Draft, `draft-symington-dtnrg-bundle-multicast-custodial-03.txt`, Work in Progress, November 2007.
[7] Delay Tolerant Networking Reference Implementation. URL: <http://www.dtnrg.org/wiki/Code>.
[8] D. Winer, RSS 2.0 Specification, July 2003. URL: <http://cyber.law.harvard.edu/rss/rss.html>.
[9] M. Nottingham, R. Sayre, The atom syndication format, RFC 4287, December 2005.
[10] M. Demmer, B. Du, E. Brewer, TierStore: a distributed file system for challenged networks in developing regions, in: Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST), 2008, pp. 35–48.
[11] Z. Albanna, K. Almeroth, D. Meyer, M. Schipper, IANA guidelines for IPv4 multicast address assignments, RFC 3171, August 2001.
[12] P. Eugster, P. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM Computing Surveys 35 (2) (2003) 114–131.
[13] L. Lamport, Time, clocks and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565.
[14] F. Mattern, Virtual time and global states of distributed systems, in: Proceedings of the Workshop on Parallel and Distributed Algorithms, 1989.
[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext transfer protocol (HTTP/1.1), RFC 2616, June 1999.
[16] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, Internet group management protocol, version 3, RFC 3376, October 2002.

[17] R. Vida, L. Costa (Eds.), Multicast Listener Discovery, version 2, (MLDv2) for IPv6, RFC 3810, June 2004.
[18] A. Adams, J. Nicholas, W. Siadak, Protocol independent multicast – dense mode (PIM-DM): protocol specification (revised), RFC 3973, Jaunary 2005.
[19] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, protocol independent multicast – sparse mode (PIM-SM): protocol specification (revised), RFC 4601 (Proposed Standard)w, August 2006.
[20] D. Waitzman, C. Partridge, S. Deering, Distance vector multicast routing protocol, RFC 1075, November 1988.
[21] M. Demmer, K. Fall, DTLSR: delay tolerant routing for developing regions, in: Proceedings of the SIGCOMM Workshop on Networked Systems in Developing Regions Workshop (NSDR), 2007.
[22] M. Handley, I. Kouvelas, T. Speakman, L. Vicisano, Bidirectional Protocol Independent Multicast (BIDIR-PIM), RFC 5015, October 2007.
[23] M. Demmer, K. Fall, T. Koponen, S. Shenker, Towards a modern communications API, in: Proceedings of the 6th Workshop on Hot Topics in Networks (HotNets), 2007.
[24] S. Symington, R. Durst, K. Scott, Custodial Multicast in Delay Tolerant Networks: Challenges and Approaches, Technical Report 06-1000, MITRE, 2007.