Delay-Tolerant Network Research Group
Internet-Draft
Intended status: Informational
Expires: September 12, 2011

G. Clark
Ohio University

March 11, 2011

Initial Requirements for Remote Network Management in Delay-Tolerant Networks
draft-clark-dtnrg-netman-req-00

Abstract

This document describes an initial set of requirements and considerations for use when designing a network management protocol to be used in a DTN.

Requirements Language

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

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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

This Internet-Draft will expire on September 12, 2011.

Table of Contents

1. Introduction

1.1. Overview

In traditional networks, it's often assumed that network latencies and loss rates are both relatively low, bandwidth is plentiful in both directions, and that time can be easily synchronized and agreed upon between devices. Additionally, physical connections are largely static, logical connections may persist for long periods of time without incident, and major routes change relatively infrequently. Few (if any) of these things are generally true when dealing with DTNs.

For example, when communicating with nodes in space, links are often highly asymmetric. Additionally, there's often a relatively large propagation delay involved with all transmissions to / from the target node. Further, given that most celestial bodies are only visible for limited periods of time (dependent upon the rotation of the Earth, etc.), it becomes difficult (if not impossible) to maintain a steady connection; many links of this type provide only short bursts of connectivity between large periods of scheduled downtime.

Another potential application of DTN can be found in animal tracking. In an attempt to save money and power, it's often preferable to have devices avoid using direct satellite links when possible. Since routes and existing nodes are not known, tracking devices form a variant of a MANET with the goal of eventually delivering information about each animal to a given destination via e.g. a vehicle which drives through the area once a week.

Unfortunately, many traditional network monitoring and / or management protocols seem to be poorly suited for DTN environments. NETCONF (as defined in RFC 4741 [RFC4741]), for example, requires a persistent connection to be maintained to the managed node; this seems to make the protocol a poor choice for deployment in a DTN environment. SNMP, as another example, relies largely on regular requests to retrieve relatively complex pieces of data from a remote node . When polling a node where the propagation delay can be hours or days, sending an explicit request for every piece of data leads to responses with data that is far older than it would be if a different model were used. Further, SNMP tends to be relatively inefficient when monitoring the status of a set of OIDs; each update must be requested, and each request is essentially the same, leading to a large number of redundant requests; when the uplink is smaller than the downlink, this can have a negative impact on the network and / or limit the frequency and size of requests.

One alternative to sending requests at regular intervals is the publish / subscribe model. In this model, each monitored node is considered to be a publisher. To request information from a publisher, a node (subscriber) sends a subscription to that publisher. The subscription contains exactly what it is the subscriber wants to know. Once the publisher receives this subscription, it processes it and provides the information to the subscriber as described within the subscription. Unfortunately, publish / subscribe suffers from its own set of problems. For example, what happens when a subscriber disappears without unsubscribing from a resource? What happens when too many subscribers want the same resource and overwhelm the publisher?

One way to address some of the issues with publish / subscribe involves introducing a proxy between the subscriber and the provider. All subscribers who wish to subscribe to a resource instead send their subscriptions to the proxy, which consolidates, caches, and forwards subscriptions (and corresponding updates) as it sees fit. Information is then provided back to the proxy on a regular basis as described in the subscription, which then forwards the information to interested subscribers.

1.2. Environment

When considering the DTN environment in general, the following conditions often apply:

- High latency
- Opportunistic routing
- Periodic disconnection
- Loss rate has huge variance (depending on the network)
- RTT has huge variance (depending on the network)

Additionally, since this document assumes the use of the bundle protocol (as described in RFC 5050[RFC5050]), the following holds true:

- Notification of data delivery status is possible (via end-to-end acknowledgment)
- Consistency checking is not performed
- Successful bundle delivery is not guaranteed (although custody helps)

- Bundle order is not preserved
- Packet order can only be preserved within the context of a single bundle
- No theoretical upper limit on the size of a bundle
- Time is not absolute (clock synchronization is actually kind of hard)
- Nodes have some persistent storage (necessary for store + forward)

2. Deploying Existing Protocols in DTNs

It is beyond the scope of this document to recommend best practices for deploying each individual network management protocol in a DTN. Instead, this section is designed to describe how DTN conditions might map to existing protocols in the abstract.

2.1. Overview

Where possible, a bundle protocol transport should be implemented for network management protocols before they are deployed in a DTN environment. In the event that a network management protocol does not define a native bundle protocol transport, however, it is possible for the bundle protocol to emulate a UDP transport to some extent by encapsulating packets into bundles and tunneling them between two or more endpoints.

A BP tunnel used by a network management protocol should respect the type of destination specified in the packet being tunneled. Specifically, if a packet, it should only be sent through a tunnel with a single destination endpoint. If the packet is anycast, it should be sent through a tunnel with one or more destination endpoints, where only a single destination receives the bundle. If the packet is multicast, it should be sent through a tunnel with one or more destination endpoints, where all destinations receive the bundle.

If packets are aggregated and encapsulated into a single bundle before being sent through a DTN, care should be taken to ensure that operations with side effects are only forwarded to a set of endpoints representative of each packet's original destination. It may be a good idea to only encapsulate more than one packet in a single bundle if all packets share a set of common destinations.

For practical reasons, broadcast packets should not be sent to a DTN.

Tunnels may classify broadcast packets as multicast packets for the purposes of aggregation and encapsulation.

The above set of requirements should allow many existing network management protocols which define a UDP transport layer (e.g. SNMP, IPFIX as defined in RFC 5101 [RFC5101]) to function in a DTN environment.

Unmodified versions of connection-oriented protocols (such as NETCONF) should not be deployed in a DTN.

Many network management protocols rely on either regular unsolicited updates (e.g. SNMP TRAPs sent every 30 seconds) or regular polling (SNMP GET) to maintain current information on a device. In a DTN, such regular updates could place a great deal of strain on the network; see Section 4 of this document for a more detailed discussion of retrieving data in managed DTN networks.

Care should be taken to ensure that requests are bundled to the greatest extent possible. Note that references to dynamic managed data (e.g. table indices) MUST NOT be used outside the scope of a single bundle.

3. Architectures

3.1. Overview

One straightforward approach to translating between two protocols involves routing packets through a gateway that speaks the protocols being converted from / to. This gateway converts from one protocol to the other, and then forwards the packet along to its original destination; an inverse mapping is performed for any replies that are destined for the client. This method is nice because it's simple, and can be implemented such that it's virtually invisible to the client / server.

An extension to this approach involves building a cache into the gateway. When the gateway receives a request destined for a remote node, it checks its local cache to see if it can fulfill part or all of the request locally before forwarding it on to the destination gateway. To do this, however, the gateway must have an idea of the effective lifetime of various pieces of network management information; the more dynamic the information, the more recent the cached information needs to be.

Note, however, that directly translating between protocols in this fashion (regardless of the caching mechanism used) would require a

1:1 mapping between the protocol being translated from and the protocol being translated to. Given that many DTN network management protocols will differ from network management protocols designed to be deployed in traditional networks, translating requests in this fashion may not be practical.

An alternative to using transparent gateways to translate individual requests involves using opaque proxies. Where a transparent gateway would forward requests / subscriptions on to the network, an opaque proxy would act as an arbiter between a DTN and a traditional network. The proxy would be responsible for fulfilling network management requests for a number of clients as it saw fit. This could lead to a great deal of flexibility in how that data was obtained (for example, the proxy could use a publish / subscribe protocol to obtain data from DTN nodes and offer that data over the traditional network via SNMP / NETCONF / etc. Additionally, it can be beneficial to publishers / servers to only have to deal with a single client (the proxy) as opposed to N clients (if all of the clients interested in the publisher's data subscribe to it directly).

Deploying opaque proxies does require some consideration, however. The proxy essentially is completely responsible for fulfilling and distributing information to the network; thus, any proxy must be completely trusted by all the nodes it serves. Additionally, the proxy must ensure that any security constraints defined on the nodes are communicated to and enforced at the proxy. Further, the proxy becomes a single point of failure for the entire network behind it. For these reasons (and others not enumerated here), security and redundancy must be carefully examined when deploying a proxy of this nature.

3.2. Proxies : Access Control

The following describes a few approaches to controlling access to managed DTNs behind proxies in general; these approaches should be considered complete nor necessarily even good ideas.

3.2.1. Node-Based Access Control

The node creates an access control list and publishes it at a known endpoint. Access controls are specified on a per-user basis for a particular set of endpoints (which would work very well with the DTN2 naming scheme, but not so much with IPN). It would be up to the proxy to perform authorization / authentication for those users.

Note that these access controls would apply only to what the proxy was allowed to do with the data it obtained. These resources would need to be shared only with the proxy, and the proxy would need to

take steps to validate the node it was talking to before it tried to do anything with this data.

3.2.2. Proxy-Based Access Control

The node has no input as to how a proxy can redistribute the information it provides. This is defined on the proxy on a per-node basis.

3.2.3. Hybrid Approach

Each node publishes a resource assigning itself to a logical group, which the proxy uses to figure out A) which resources that node provides and B) how those resources should be shared. The groups and their corresponding offerings would likely need to be standardized.

3.2.4. Better Suggestions

Anyone?

3.3. Scaling

In general, proxies should be constructed such that they scale reasonably well not only with respect to the number of nodes in the DTN they represent, but also to the number of clients connecting to them at any given time.

3.4. Caching Methods

In many cases, it's known that the data on a remote node won't change more frequently than once every X seconds; in these cases, it makes sense to create a cache that can answer requests on behalf of a node without having to actually actively query a node. This is especially important when dealing with nodes that are available infrequently and / or have limited bandwidth; if at all possible, requests should be serviced at the node with the highest availability (both in terms of bandwidth and historical uptime).

This section explores different types of caching and their possible application in DTNs. It provides a simple overview of each type of caching, along with a very simple analysis of considerations when deploying the strategy in a DTN.

3.4.1. Cache Lifetime

When a client X requests a piece of data Y from a server, the server might respond with a time value that represents the maximum amount of time that piece of data can be stored and treated as valid by the

client. As a trivial example, say X requests the current day of the year; the server would respond with the current day, and also specify that the day wouldn't be changing for T seconds. Assuming that the next time the client wants to know the day is <= T seconds from its last request, it can consult its local cache to find the answer. This is a common technique employed by many protocols and applications (notably HTTP and web servers) to conserve client, server, and network resources.

This technique seems like it could be deployed in a DTN with an effectiveness related to the time the piece of data was designed to spend in the node's cache minus the time the data spent in transit. For example, it probably wouldn't be useful to specify that a piece of data was valid for thirty seconds beyond the time at which it was retrieved when the bundle containing the data was expected to spend at least five minutes in transit.

Note that specifying expiration dates for cached content as absolute dates (e.g. Friday, March 3rd, 2011, at 00:00:00 EST) could be useful in networks where widespread time synchronization were an option, and where links were expected to become available in a predictable fashion (as is the case in many space networks).

3.4.2. Validation Caching

This type of caching uses a tag generated by the server to uniquely identify the content of a given resource; this tag is most often generated by hashing the content of the resource with some kind of collision-resistant function (e.g. SHA-1). When the server receives a request, it responds with both the content of the given resource and that resource's tag. The next time the client requests the resource, it includes the tag it received from the server in the last response; if the tag for the current content on the server matches the tag the client received the last time it requested that resource, then the server tells the client that the data in the client's cache is still valid.

Some care should be taken when applying this type of caching to DTNs. Since the order in which bundles are delivered is not known, there is the possibility that a set of client / server interactions could overlap, resulting in the client mistakenly believing the data has changed. Thus, some kind of mechanism should be used to identify the order in which tags are received, and all tags but the most recent should be ignored.

3.4.3. Caching Resource Metadata

Yet another way to cache data involves caching metadata (e.g. field type) of a transaction, allowing the client and / or server to keep the amount of information they actually need to transmit to a minimum. ASN.1, for example, specifies the type of each object when it is defined. Assuming that the client and server both agree on a single ASN.1 specification, the type (and possibly length) information normally found in data encoded following the BER can be omitted, possibly leading to more efficient data encoding; this concept is described in detail in X.691-0207, which defines the Packed Encoding Rules.

One example of this type of caching being used in practice can be found in IPFIX (IP Flow Information eXchange), a publish / subscribe network monitoring protocol. This protocol defines resource metadata in templates, which are sent by the providers (publishers) to each collector (subscriber) when it connects. Subsequent updates each reference this original template ID, leading to very small update sizes.

This kind of caching works especially well when the structure of the data being sent is known to be relatively static. For example, when a DTN node wished to report its status at regular intervals, it seems like it would make sense for that node to preface its updates by providing a template describing the data it would be sending, then encode a reference to that metadata in each update.

Caching metadata this way seems well suited to network monitoring in certain kinds of DTNs. Satellite telemetry, for example, could benefit greatly from this kind of caching; satellites are unlikely to regularly modify the format of the updates they send, which means the type and order of each update could be cached and applied to updates as they arrived.

3.4.4. Caching Update / Response Rules

Network monitoring / management systems on Earth generally don't have to worry about automated responses to system errors and the like; if something breaks, it's usually pretty easy for a system administrator to see the notification and respond before the problem gets out of hand.

When dealing with a network where the RTT is relatively large, however, updates generated by a network monitoring / management protocol can essentially be considered post-mortem; by the time they arrive, it can be far too late to fix them. Thus, it's often up to the device to automatically respond to an event in such a way that it

can survive until an operator is able to analyze the data it sent and correct the issue. While many simple responses can be hard-coded into the device, these simple responses can sometimes be unable to deal with unexpected situations. As such, a network monitoring / management protocol should provide the ability to define new methods of adapting to a situation (or perhaps of correcting old ones) without user intervention.

3.4.5. Distributed Monitoring / Implicit Caches

Another way to cache network management data involves configuring a set of nodes along the route of a network management transaction to keep a partial record of that transaction and / or support implicit caching (as described in "DTN-based Content Storage and Retrieval" [DTNCSR]). Sending an explicit request to a monitored node with a specification for a maximum age of relevant data would allow intermediate nodes to reply on behalf of the monitored node. Note that this technique could be particularly effective in distributed networks where data was shared among nodes and not especially time-sensitive. This technique could also allow the partial reconstruction of the last known status of nodes which were not actively responding to requests for some reason (e.g. crashed, sleeping, out of communications range, etc).

4. Retrieving Monitored Data

Many of the network management protocols in use today provide a mix of request / response and event notification capabilities. SNMP, for example, not only offers remote hosts the ability to actively poll an SNMP entity via the various flavors of the SNMP GET command set (e.g. GET BULK, GET NEXT), but also allows managed nodes to be configured to send unsolicited TRAP and INFORM messages when certain conditions are met (for example, when an interface comes up or goes down).

In general, polling individual devices for information has a neglible impact on most traditional networks: since bandwidth is plentiful and latency is low, the overhead involved with sending repeated requests for information has little impact on network performance. Furthermore, polling requires little (if any) state to be saved on the monitored node, and is straightforward to implement on most devices; this makes it possible to deploy SNMP onto just about anything that speaks UDP.

By contrast, event notification leads to more efficient network utilization in the general case and tends to scale far more effectively than polling does. Since data is only sent when a certain set of conditions are fulfilled, the management station does

not have to continually poll the device for its information. In larger networks, the resources (both network and CPU) this method can save the management station (and sometimes even the monitored node) are substantial.

Of these two methods, event-based notification seems to be much better suited to the DTN environment than polling does. One reason for this is the huge upper bound on the RTT between nodes of a DTN. While sending a request for information across a LAN incurs virtually no delay (a few milliseconds), a request sent across a DTN generally has a much higher RTT (minutes, hours, or days). This higher RTT combined with the inherent instability of DTNs often makes actively polling a remote node a largely ineffective means of obtaining any kind of information about a transient fault in a system; by the time the request arrives, enough time has passed that the fault could have passed completely (or have developed into a more serious issue that resulted in the failure of the node).

Note that polling a managed DTN node (via SNMP or otherwise) at regular intervals is not recommended. Given that the order / timing of bundle delivery is not guaranteed, there is no way to ensure that requests arrive at a managed node in order or, more importantly, with the frequency at which they were originally sent. There's a reasonable probability that multiple requests would be queued and immediately flood the monitored node for processing as soon as the node's link came up, resulting in a number of identical results coming back through the network. Not only could this saturate the link at the monitored node, it could also waste a great deal of storage at intermediate nodes.

5. Modeling Management Data

5.1. Overview

While debating and electing a certain representation for managed data is beyond the scope of this document, it is important to carefully consider the language used to model the structure and availability of data at a node. To that end, this document briefly examines three different modeling languages used by existing network management protocols. Structure of Management Information, (most recently SMIv2, defined in RFC 2578 [RFC2578]), YANG (defined in RFC 6020 [RFC6020]), and the Managed Object Format ([CIM]).

5.2. Structure of Management Information

Quoting RFC 2578 ([RFC2578]):

"Management information is viewed as a collection of managed objects, residing in a virtual information store, termed the Management Information Base (MIB). Collections of related objects are defined in MIB modules. These modules are written using an adapted subset of OSI's Abstract Syntax Notation One, ASN.1 (1988) [1]. It is the purpose of this document, the Structure of Management Information (SMI), to define that adapted subset, and to assign a set of associated administrative values."

While the 1:1 compatibility SMI would provide for SNMP is attractive, SMI has a number of restrictions that may not make it the best choice for direct deployment in DTN environments. As reported in RFC 3535 [RFC3535], when a limited subset of network management operators and protocol designers were polled, concerns included: "the SMI language is hard to deal with and not very practical", "the lack of structured types and various RPC interactions (methods) make MIB modules much more complex to design and implement", and "the lack of query and aggregation capabilities (reduction of data) causes efficiency and scalability problems."

As such, SMI may be best deployed to define a subset of a node's management capabilities that could be accessed via traditional SNMP. The exact determination of such features along with a discussion of how such features might be mapped is beyond the scope of this document.

5.3. YANG

YANG is defined in RFC 6020 [RFC6020]to be "a language used to model data for the NETCONF protocol. A YANG module defines a hierarchy of data that can be used for NETCONF- based operations, including configuration, state data, Remote Procedure Calls (RPCs), and notifications. This allows a complete description of all data sent between a NETCONF client and server."

While YANG has been defined exclusively for use with NETCONF, it stands to reason that the language could be used as a general purpose approach to model data offered by a DTN.

5.4. Managed Object Format

This modeling language is defined as a part of the Common Information Model (CIM)[CIM], which states:

"The MOF syntax is a way to describe object definitions in textual form. It establishes the syntax for writing definitions. The main components of a MOF specification are textual descriptions of classes, associations, properties, references, methods and instance

declarations and their associated qualifiers. Comments are permitted. In addition to serving the need for specifying the managed objects, a MOF specification can be processed using a compiler. To assist the process of compilation, a MOF specification consists of a series of compiler directives."

For example, a description of a simple MOF class (which extends an arbitrary BasicObject class) might look like:

```
[Version(1.0), Description("This is an example class")]
class WidgetObject : BasicObject {

[MaxLen(256), Description("Who made the widget?")]
string widgetManufacturer;

[Units("Bytes"), Description("How much storage does the widget use?")]
uint64 widgetStorage;

[Description("Runs the widget")] uint32 runWidget(
[IN, Description("Describes how to run the widget")] string howToRun);

};
```

This structure could be used to define a set of schemas relevant to the DTN community.

6. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

7. Security Considerations

Trying to define a comprehensive security framework to deploy in a DTN is beyond the scope of this document.

In general, however, it is extremely difficult to identify a single set of network security requirements for network management protocols simply because DTNs can be deployed in so many different types of environments. For some devices, layered encryption and / or complex access control could be difficult. For others, physical security (e.g. an isolated group of nodes floating somewhere in the middle of the Atlantic operating at an exotic frequency) would probably limit the practical need for especially strong encryption.

As such, the following notes apply to designing network management protocols for deployment in DTNs

Nodes should enforce access controls locally

Proxies should be validated (e.g. via certificate of some kind)

Proxies should enforce any rules governing the redistribution of data

DTN network management protocols may elect to not provide authentication

DTN network management protocols may elect to not provide authorization

Note that if the network management protocol in use does not provide authentication or authorization, this support should be enabled elsewhere (e.g. via the Bundle Security Protocol [I-D.irtf-dtnrg-bundle-security]).

8. Acknowledgements

This document is a continuation of some of the work started in Will Ivancic's DTN network management requirements document (now an expired draft).

Thanks to Joshua Schendel, Zack Sims, Mithun Roy, Kevin Janowiecki, and Samuel Jero for reading the document and providing some invaluable feedback.

Observing the CoRE IETF working group develop protocols and specifications (specifically an HTTP mapping, an observe mechanism, and a proxy) has influenced the direction of this work.

9. References

9.1. Informative References

[RFC3535] Schoenwaelder, J., "Overview of the 2002 IAB Network Management Workshop", RFC 3535, May 2003.

[RFC4741] Enns, R., "NETCONF Configuration Protocol", RFC 4741, December 2006.

[RFC5101] Claise, B., "Specification of the IP Flow Information

Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information", RFC 5101, January 2008.

9.2. Normative References

[CIM] Distributed Management Task Force, Inc., "Common Information Model (CIM) Infrastructure Specification Version 2.3", October 2005.

[DTNCSR] Ott, J. and M. Pitkaenen, "DTN-based Content Storage and Retrieval", 2007.

[I-D.irtf-dtnrg-bundle-security] Symington, S., Farrell, S., Weiss, H., and P. Lovell, "Bundle Security Protocol Specification", draft-irtf-dtnrg-bundle-security-19 (work in progress), March 2011.

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

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, April 1999.

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

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

Author's Address

Gilbert Clark
Ohio University
Athens, Ohio 45701
USA

Email: gc355804@ohio.edu