LTP-T: A Generic Delay Tolerant Transport Protocol

Stephen Farrell, Vinny Cahill,
Distributed Systems Group, Department of Computer Science,
Trinity College Dublin, Ireland.
stephen.farrell@cs.tcd.ie

Abstract

This report proposes a generic delay tolerant transport protocol constructed as an extension of the Licklider Transmission Protocol, a delay tolerant point-to-point protocol currently under development which is primarily aimed at deep space applications. The report justifies and describes the proposed protocol and the work that is planned to validate its properties.

Introduction

Deep space networking is (and perhaps always will be) in its infancy when compared to the terrestrial Internet. This is so for a combination of very good reasons, but also perhaps due to some networking factors where improvements are possible. Amongst the very good reasons are the physics of the situation which means that there's nothing that can be done about light trip times, occultation, etc., but there are also very good engineering reasons why networking is harder in this context, ultimately reducing to the lack of power, as transmitting or receiving bits at deep space distances consumes a lot of power and providing that power is hard.

However, current deep space communications are also quite simplistic (compared to the terrestrial Internet) in ways that we can perhaps more easily improve, for example, its not at all clear that there is really a need to manually schedule each and every communications contact. Similarly it ought to be easier to arrange to route data via intermediate nodes, especially in cases where there are lander and orbiter craft involved.

In fact, work on addressing some of these issues has been underway for some time, particularly under the auspices of the Internet Research Task Force's (IRTF's) delay tolerant networking (DTN) research group (the DTNRG [1]). That group are developing two sets of specifications for delay tolerant protocols. The first, called bundling [2,3], focuses on an overlay network approach and the second, called the Licklider Transmission Protocol (LTP [4,5]) which is a point-to-point protocol aimed mainly at deep space long-haul links. Within the DTNRG LTP is generally seen as a protocol that can underpin bundling, so that bundles are transported over LTP on long-haul links.

There are however, also cases where LTP could be suitable for use on all links in a path. In particular LTP could be used for lander to orbiter communications, and then also between the orbiter and an earthstation. In such cases, there ought to be efficiency gains (compared to, for example, layering bundles over LTP) in deriving a transport protocol from LTP.

There are also terrestrial sensor networking cases, particularly where "data mules" are in use, where such a transport protocol can be useful - an example of such a use case is described in detail elsewhere [6]. For our present purposes it is perhaps sufficient to note that data-mules which physically traverse a field of sensor nodes are highly analogous to earthstations in a deep space network as can be seen from the table below which compares such networks.

	Deep space network	Sensor network
"Edge" nodes	Spacecraft	Sensors
"Middle" nodes	Earth stations	Data mules
"Sink" nodes	Internet host	Internet host
Edge/Middle Connectivity	Via schedule and ephemeris	Via schedule and radio range
Middle/End connectivity	Via application layer gateway	Scheduled/ when mule "at home"
Main network constraint	Spacecraft power	Sensor node power
Application traffic profile	Telemetry and science payloads	Telemetry and sensor data

Table 1: Deep space/sensor network analogy

As with all analogies, this ultimately breaks down, perhaps primarily in terms of the numbers of sensors expected versus the number of spacecraft, however, it is worth noting that current sensor networks tend to involve limited numbers of nodes and that the

total number of spacecraft supported by the DSN today is perhaps not that dissimilar to a typical sensor network.

In this paper we propose a way to use the LTP extension mechanism [7] to create such an end-to-end capable transport protocol, which we call "LTP Transport" or LTP-T.

Note that although there will be cases where the LTP-T is more efficient than the overlay network approach, bundling is applicable in a broader range of networking environments, so that LTP-T is not "competing" with bundling, but represents a compatible approach to delay tolerant networking.

Background

The difficulties of deep space networking ([2,4]) are such that current popular network protocols cannot be directly used. In particular, the well known issues with TCP in high bandwidth-delay product cases [8] particularly affect deep space communications.

The main approach being taken by the DTNRG is to define an overlay network which is felt to be the most generally applicable approach, since many DTNs may be run over different types of network. A typical use case given might involve an on-planet sensor network communicating with a lander using some mobile ad-hoc style protocol (e.g. AODV [9]), where the lander communicates with an orbiter using the CCSDS proximity-1 [10] protocol, the orbiter then communicates with an Earth station using LTP, which forwards application data over the terrestrial Internet. An important concept defined in the bundling specification is that co-called "custody" can be taken of bundles, so that retransmission does not require end-to-end signalling.

The LTP protocol is primarily designed to be run over single links with extremely high latency, and is therefore highly stateful, which is easily justifiable in this case since antenna pointing and ephemeris handling are similarly stateful and are required for any communication at these distances. LTP handles the transmission of a block of data, which may be split into segments that match the maximum transmission unit for the link in question. All of the data and control segments required for successful transmission of a block of data are part of the same LTP session. LTP is assumed to be layered over some link layer where the LTP engine is somehow given "cues" as to when it can communicate to a given peer. The final LTP concept required to understand LTP-T is the red/green concept – LTP allows some initial segments of the block to be marked as "red", the remaining segments being "green". Red segments must be actively acknowledged, whereas green segments need not be acknowledged.

Why a transport layer approach?

Given that the DTNRG is already working on both overlay and point-to-point approaches to DTN, we must first justify the proposed transport layer approach.

Firstly, there is an argument related to ease-of-use: programmers are familiar with developing applications which sit upon a transport layer and the more we can mimic a more "normal" network, the easier it will be to develop and deploy applications. The ideal, as we see it, would be to provide an API which is as easy to use as UDP sockets, but which provides transparent support for delay, intermittent connectedness, reliability, congestion controls and security. While this argument may not appear compelling from an architectural point of view, practically, it makes an enormous difference.

Secondly, though the overlay approach seemingly eliminates the need to handle congestion, it can, in fact, only disguise the problem. In any DTN node storage congestion is always possible (and is even likely), so mechanisms for handling this need to be provided. While the "custody" approach can handle this, properly combining custody with security mechanisms and bundle fragmentation is very difficult and potentially leads to inefficiency when handling secured, fragmented bundles. A satisfactory method of handling some of these cases has, in fact, yet to be demonstrated though work on that is on-going.

Thirdly, there will be networks which are homogeneous in terms of requiring the use of delay tolerant networking at each "hop", and there is no a-priori reason why different hop-by-hop DTN protocols should be used for each hop. Since LTP is suitable for use on each of these hops, it ought to be possible, and more efficient, to define a "multi-hop" analog to LTP (which is what we're doing here). Note that the converse of this argument is also valid – there will be heterogeneous networks, so there is no contradiction in defining both overlay and transport approaches – the specific

network on which the applications are to run will determine which is more appropriate to use.

Fourthly, we consider that most DTNs will at some point, and perhaps frequently, if intermittently, be connected to the terrestrial Internet, so defining an approach which can use standard naming and addressing at the application layer, and standard routing at the network layer, is beneficial. The bundling approach has to use a new and more complex naming scheme based on URIs in order to provide routing at both the "bundling" layer and within the underlying network. LTP-T is designed to use standard IP addressing and routing where possible. One could also use other naming and addressing if LTP-T were to be run over some other network. There is also a pragmatic argument here that the IP stack is ubiquitously available, even if sometimes in a limited form and that developing new transport protocols tailored to be run over IP is a well-understood task.

LTP-T

In this section we propose a concrete set of LTP extensions and processing rules which together define LTP-T. The description assumes that the reader is familiar with LTP.

Our aims here are that an existing LTP implementation ought to be able to simultaneously work in either point-to-point or transport mode; that it ought to be relatively easy to implement LTP-T, given an LTP implementation and that the protocol ought to satisfy the ease-of-use requirement stated above by being implementable below a sockets API[1].

As a point-to-point protocol LTP need not carry a destination identifier, doesn't have to consider fragmentation and has no need to consider congestion at intermediate nodes on a path. A transport protocol must of course consider these issues and in LTP-T these are handled using the LTP extension mechanism, which was originally defined to handle the addition of authentication fields to LTP.

[1] Our current LTP implementation [6] provides a sockets API, using a model where sockets are used and discarded by the application, with delay tolerance being handled inside the network stack. Amongst other things, this requires a garbage collection model for handling sessions after the application has discarded the socket.

The LTP extension mechanism allows for the addition of both header and trailer extensions, up to a maximum of 16 (of each). We therefore start by defining the following set of extensions to LTP:

Source address: this header extension contains a single SDNV[2] that represents the address of the originating LTP host. Note that this extension need not be present for the first hop, where the session ID contains the required value. This information is required when propagating congestion information back upstream.

Destination address: this header extension contains a single SDNV that represents the address of the final LTP host and is required for forwarding and routing.

Estimated block size: this header extension contains a single SDNV that specifies (an estimate for) the number or bytes in the block to be transmitted, so as to allow for congestion control for intermediate hosts. Note that this extension is an estimate since the exact size may not be available, for example, if the application deals with streaming content. It is not an error for the actual block to be bigger or smaller than this value.

Port: this header extension contains a single SDNV which represents the port at the destination to which this data is targeted. It is analogous to the client service identifier currently defined in LTP.

Hop count: this header extension has as its value a single byte containing the number of further hops allowed before segments from this session should be dropped. When a packet arrives when the session hop count is zero then, if the current host cannot communicate directly with the destination, the segment is dropped. This extension is used to avoid looping and other potential transport problems.

Via: This header extension contains a sequence of LTP address/time tuples representing the path taken by the data so far. The encoding is an initial SDNV which contains the number of elements in the sequence followed by the sequence of pairs of SDNVs representing the host and timestamp.

[2] An SDNV is a self-describing numeric value, basically an encoded positive, multi-precision integer. The SDNV encoding scheme is defined in LTP and is also used by the bundling protocol.

Any local time format may be used. If a host is configured to accumulate tracing information as segments traverse a path, then it includes a via extension in any LTP segment. This first via must have one entry representing the sender and the time the segment is transmitted (note: not the time at which the segment was enqueued). Subsequent hosts must include an updated via extension in all segments (including fragments) containing data which was found in that first segment. Updating the via extension value is done by adding this host's identifier and timestamp to the end of the previous value, that is, the value of the extension lists the hosts in the order in which they were traversed. This extension can be used for testing and debugging.

End-to-end authentication: This defines how to calculate an authentication field which can be validated end-to-end. The existing syntax for the LTP authentication extension is used, the difference being that the input bytes consist only of those bytes of the segment that will be the same end-to-end. Only the final destination host is required to verify this extension's value, intermediate hosts may verify the value if the ciphersuite allows and if they cache/digest all the required bytes of the block. The input bytes for the end-to-end authentication digest calculation comprise the following, in the order stated:

- The number of extensions included in the calculation
- The source address, even though this is not explicitly present in the first hop session
- The destination address
- The estimated block size, if present
- The port number, if present
- The bytes of the block

Note that this does not include hop-by-hop extensions; nor the hop-count, nor the via extensions, which all change at each hop. In future, we may define an end-to-end confidentiality extension, but that is omitted for the present.

Congestion notification: The value of this extension is a sequence of pairs of SDNVs, the first being a host identifier and the second of each pair being the number of (real, not punctuated) seconds from the time of transmission of the segment during which it is believed the host in question will be suffering from storage congestion. If the time value is unknown (e.g. if the congestion timer described below expires) then this field should be zero. This extension is further described below.

LTP-T operation

LTP-T basically consists of a sequence of more-or-less independent LTP "hops", with the following differences/additions:

1. The originating LTP host determines the red/green parts of the block and this distinction must be honoured by all intermediate hosts.
2. LTP-T handles errors and Automatic Retransmission reQuest (ARQ) on a hop-by-hop basis, so reports are used just as in LTP. Effectively this corresponds to the "custody" concept for bundles, with the difference being that each LTP host is required to accept custody of all blocks it successfully receives. Custody is therefore passed when a block is successfully received by the next host.
3. In many cases, an entire LTP-T block will be successfully transferred between two peers prior to any of the block's bytes being forwarded to a subsequent peer on the path to the destination. However, there may be cases where some of the block will be forwarded before the entire block has been received. In such cases, even bytes which have yet to be acknowledged to the originating peer may be forwarded to the next-hop.
4. An intermediate host may fragment segments if required to do so, for example if the MTU on the next hop is too small for this segment. Fragmentation must not shorten the red part of the block, but may extend it, otherwise hosts are free to fragment in any way.
5. Since each host is required to take custody of each segment's bytes, an LTP host may well suffer from congestion in terms of storage space. In order to reduce the likelihood of this occurrence, the estimated block size extension may be used to "reserve" space. A host that does not have sufficient space to handle an incoming block should cancel the session indicating that congestion has occurred Congestion is discussed further below.
6. Cookie handling and hop-by-hop authentication are handled on a per-hop basis – there is currently no end-to-end signalling related to either. Whether an end-to-end anti Denial-of-Service mechanisms is required is for future study.

We use the following terminology when discussing the use of these extensions: The "initial segment extensions" are the source (where necessary) and destination address, the block size, the port and the hop-count. We propose the following rules for handling initial segment extensions:

- The first segment of a session must contain the initial segment extensions; subsequent segments of the session may include them. LTP-T receivers must be able to handle the case where the initial segments for a session do not contain these extensions (or are not correctly received, at first). Including the initial segment extensions in a number of segments may be useful if there is a high probability of the first segment of the session being corrupted.
- Initial segments should be marked as red.
- The same values for the initial segment extensions must be used for all of the initial segments, that is, a host may not vary the values in these extensions within a single session.
- If end-to-end authentication is to be used, the initial segments must also contain the ciphersuite information, i.e. the end-to-end header extension.
- The source address is not required for the first hop (but may be included).

The last segment of the block must contain the end-to-end authentication trailer extension if the initial segments contained the end-to-end authentication header extension. Note that there is no requirement to retransmit (or request retransmission for) data that fails end-to-end authentication checking. Such failure should be recorded and made available to a receiving application, but the data may still be delivered. Similarly if the end-to-end authentication trailer extension was included in a green segment, then it may not arrive to the final destination. This case is to be treated as if the trailer arrived but was incorrect.

LTP-T clearly requires some routing scheme in order to work, however we do not define how to route blocks at this level, since that effectively has to be handled in the same way as lower layer cues, e.g. via a scheduling module, so we are assuming (for now) that the scheduling module also provides routing tables or equivalent. Characterising routing schemes which are generally suitable for LTP-T is for future study.

The scheme for handling congestion is currently quite simple – we assume that each LTP implementation has a (punctuated) congestion timer for "to be forwarded" blocks – that is, for the entire session (perhaps based on the estimated block size extension). The idea is that the entire block has to be transmitted and (if necessary) acknowledged before this timer expires. The congestion timer is started when the first outbound segment for the block is transmitted (not when it is enqueued). This timer applies regardless of the route taken, so that the timer is not reset if a fall-back route is attempted following a first failure. If this timer expires, the entire block is deleted and all state information associated with the block is also deleted (e.g. an inbound session might be cancelled).

If the congestion timer expires on an intermediate host, then that host should, at the next opportunity, signal the congested state to upstream hosts by using the congestion extension. Similarly, a host suffering storage congestion should signal this to relevant peers by including the congestion extension in a cancel segment.

Hosts further upstream should further report the congested state to peers whenever they are aware of it and where segments from that peer may transit the congested host. If a host has no reason to believe that this information will be useful (for example, if the upstream peer will never route data via the congested host), then the information need not be forwarded. So we are piggybacking congestion notifications and have no equivalent to an explicit congestion notification [11].

Hosts must keep account of the time elapsed and the scheduling involved and must never include an entry in a congestion notification that would be useless to the recipient. In a deep space context, if the expected congestion time remaining is less than the light trip time from the notifier to the peer and then back to the congested host there is no point in informing the peer, since the congestion event will be over before the information arrives at the peer. In this way, congestion notifications are bounded, in time and (equivalently) in space.

Future work

We have proposed LTP-T as a generic transport protocol for delay tolerant networks. However, this proposal is not done in a vacuum, but is rather based upon work done developing and, more importantly, implementing (in a concrete sensor network) the LTP protocol. We are planning (subject to

funding) to pilot this sensor network for a lake-water quality monitoring application, which will provide a real-world example of the use of LTP-T.

However, we also plan to validate LTP-T by integrating with a network simulator (most likely OMNET++ [12]). In order to make fair comparisons, we will also investigate integrating the current bundling reference implementation [13] with the same simulator. We will simulate the use of LTP-T for some model missions. Finally, we will compare the use of bundling and LTP-T for these missions.

While most of the extensions required for LTP-T are relatively obvious, the congestion handling mechanism is perhaps the most speculative and attention will be directed to validating its effectiveness.

As a proposed transport protocol which can run over IP, there are also a set of standard factors which must be evaluated for this protocol, for example, throughput, latency, fairness etc. We plan to evaluate these in the usual fashion via simulations and tests using the actual implementation. This will likely require using somewhat different metrics compared to those normally used – for example it is more relevant analyse throughput and latency in terms of punctuated time for many test scenarios.

In conclusion we expect that LTP-T will prove to be an efficient and usable transport protocol for a range of practical deep-space and terrestrial delay tolerant networks.

Acknowledgements

Work within Trinity College Dublin on the LTP protocol has mainly been carried out under the SeNDT project [6,14], funded by Enterprise Ireland under their Research Innovation Fund programme. In addition to authors, the collaborators in the SeNDT project included Dermot Geraghty, James Garland and Ivor Humphreys, all from Trinity College Dublin.

DTNRG work on LTP has involved collaboration with Manikantan Ramadas from the University of Ohio and Scott Burleigh from NASA JPL, who, however, should bear none of the blame for LTP-T!

References

[1] Internet Research Task Force Delay Tolerant Networking Research Group http://www.dtnrg.org/ http://www.irtf.org/charters/dtnrg.html

[2] V. Cerf et. al., "Delay Tolerant Network Architecture", Internet Draft, draft-irtf-dtnrg-arch-02.txt, Jul 2004, work-in-progress

[3] K. Scott, S. Burleigh, "Bundle Protocol Specification", Internet Draft, draft-irtf-dtnrg-bundle-spec-02.txt, September 2004, work-in-progress

[4] S. Burleigh et al, "Licklider Transmission Protocol – Motivation", Internet Draft, draft-irtf-dtnrg-ltp-motivation-00.txt, December 2004, work-in-progress

[5] M. Ramadas et al, "Licklider Transmission Protocol – Specification", Internet Draft, draft-irtf-dtnrg-ltp-02.txt, December 2004, work-in-progress

[6] SeNDT project web site: http://down.dsg.cs.tcd.ie/sendt/

[7] S. Farrell et al, "Licklider Transmission Protocol – Extensions", Internet Draft, draft-irtf-dtnrg-ltp-extensions-00.txt, December 2004, work-in-progress

[8] T. Lakshman, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss", IEEE/ACM Transactions on Networking, 1997

[9] C. Perkins et al. "Ad-hoc On-demand Distance Vector (AODV) Routing", Internet RFC 3561, July 2003.

[10] CCSDS Proximity-1 Space Link Protocol. Draft Recommendation for Space CCSDS 211.0-B-3, Blue Book, May 2004.

[11] K. Ramakrishnan et al, "The Addition of Explicit Congestion Notification (ECN) to IP", Internet RFC 3168, September 2001.

[12] A. Varga, "The OMNeT++ Discrete Event Simulation System", In the Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic.

[13] DTN2 reference implementation documentation, http://www.dtnrg.org/wiki/Dtn2Documentation

[14] SeNDT project web site: http://down.dsg.cs.tcd.ie/sendt/