Evaluating LTP-T: A DTN-Friendly Transport Protocol

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

Abstract - **The Licklider Transmission Protocol (LTP) is a delay-tolerant point-to-point protocol being developed by the Delay-Tolerant Networking Research Group (DTNRG). LTP-Transport is an extension to LTP that provides end-to-end services and which is designed to be a generic Delay-Tolerant Network (DTN) friendly transport protocol. We describe our network emulation based test setup for these protocols and our evaluation of their performance for a few simple DTN scenarios. In particular we compare LTP-T with the other protocol being developed by the DTNRG, the bundle protocol (BP). Our results show that LTP-T can outperform the BP in some cases, though, as an overlay network protocol, the BP is more flexible in general. (Abstract)**

Keywords-component; Delay tolerant networking; DTN; LTP-T; transport; protocol evaluation

I. INTRODUCTION

Evaluating a DTN [1,2] protocol is somewhat different from a more typical transport protocol, (e.g. some variant of TCP) where we are often interested in throughput, fairness and other standard metrics. One reason for this is that DTN protocols are generally not currently in use in environments that involve much multiplexing of protocols, either with other DTN protocol flows, or with for example, TCP flows. So in this paper we describe a test setup that allows us to compare DTN protocols against one another in a meaningful and, we believe importantly, reproducible manner.

II. EMULATION VS. SIMULATION

Our approach has been to emulate, (rather than simulate), DTNs and to carry out various test runs of LTP-T [3] and the BP [4]. As will be seen, our conclusion is that emulation is preferable to simulation for evaluating DTN protocols (and generally!). But in order to explain this we first examine the options examined as we developed our test setup.

We first planned to use sensor nodes developed in Trinity College Dublin (TCD) for the SeNDT project[1] as the basis for evaluation. However, we revised this plan for a couple of reasons. The first was simple pragmatism – most of the operational hardware was in use for another pilot at the time that the protocol implementations were ready for evaluation. The second reason is that if the SeNDT nodes were actually in the laboratory for tests, then the fact that we're running on a particular processor[2] running arm-linux[3] doesn't really make any difference since we are, in this case, only emulating a network in any case.

Next we considered a space simulator, in particular the Satellite Took Kit[4] (STK), [5] which is the pre-eminent space simulator today, roughly in the place where ns-2 [6] is in network simulation. However, STK has been found not to have an interface usable for a network simulation or emulation [7], so it wasn't really a good option.

The next option therefore was to use a network simulator. The obvious choice here would be either ns-2, or maybe less obviously, OMNET++. [8] We ultimately rejected the idea of using a network simulator for reasons given below. Though there are a range of other open-source and commercial network simulators we didn't find any that were affordable and seemed like they would be significantly better than the two above.

There were three reasons for our rejection of network simulation in this case. Our first, though weakest, argument is that we share some of the skepticism [9,10] as to the usefulness and fidelity of network simulations, especially for a network setup like a DTN which is significantly different from typical simulated, or real, network settings.

While the developer of the simulation undoubtedly gains insight from the work, our feeling is that readers of the results gain much less insight and, in fact, may have trouble generating commensurate results. This can be due to issues with differing levels of detail, e.g., in wireless networks [11], where different simulations embody different decisions as to what is interesting to represent faithfully. Simulation results also need to be presented so that they can be validated, something that isn't always the case. [12]

From our own early work with OMNET++, one problem we found with network simulators is a sort of "hidden variables" problem. In the simulation performed for one paper, [13] the OMNET++ configuration file contained 174 separate

[1] http://down.dsg.cs.tcd.ie/sendt/

[2] In the case of our SeNDT hardware, this is an Intel XScale processor.

[3] http://www.arm.linux.org.uk/

[4] http://www.stk.com/

numeric and/or string settings.[5] With so many fairly opaque settings, it is no wonder that simulation results can be hard to replicate. This problem is also discussed by Kurkowski et al. [10]

The second argument against network simulation is that there are difficulties in using the same implementation for a simulation and in a real network stack. As part of our work on LTP, however, one our goals was to achieve interoperability with the University of Ohio Java implementation[6] and another was to be able to use LTP and/or LTP-T in SeNDT sensor nodes, so an implementation that was tied into a network simulator wasn't suitable for that aspect of the work. In other words, we could only validate the basic protocol design via interoperating two different implementations, and that can't really be achieved via simulations.

Lastly, the test setups described below involve not just one, but two or more protocols (sftp, LTP, LTP-T, BP) being tested in parallel. While it might have been possible to implement (or re-implement) LTP-T in a simulator, re-implementing an equivalent to the BP reference implementation would have represented significant effort and would have required yet another round of testing to validate the putative BP simulation against the BP reference code. All-in-all an easier and better plan was required.

One possible easier plan would have been to use implementations running on a Virtual Network User Mode Linux (VNUML[7]). VNUML is essentially a way to simulate a network with a set of user-mode Linux (UML) kernels running on a single host, and is typically used to construct/simulate networks as part of Honeynet projects aimed at determining the behaviour of various forms of malware or bad actors. [14] Although DTN simulation in this context is in itself quite an interesting concept[8], it is perhaps better suited to simulation of disrupted networks, rather than delayed links, since, to our knowledge, there is no obvious support for simulating large delays in such a virtual network.

So our final plan was to emulate the network, in particular based on the Netem module[9] [15] that is part of Linux 2.6 based kernel distributions – so emulation in this case can be done at little cost and with relatively excellent fidelity. While Netem is not specifically intended for emulating DTN scale latencies, in fact, 20 minute latencies similar to those required for Earh/Mars emulations work just fine.

Details of our Netem setup are specified below, but for now we consider why network emulation is a good approach. In the first place, emulations, while not perfect, are inherently better in respect of both of the main problems we saw with simulations above. With emulation we get to use a real protocol implementation so that it can be interoperated and different combinations of protocols can be tested over the same wires. We also have less of a "hidden variables," problem, since the network traffic is, by definition, externally visible and can be recorded and monitored etc. which makes replicating/checking results much easier[10].

III. General Test Setup

We carried out a set of "single hop" and "multi-hop" tests to compare LTP, the BP and LTP-T against one another and a standard TCP application - secure FTP. These tests involve various delays and essentially showed that each of the protocols performed as expected.

Our hardware platform consisted of five Dell Latitude laptops (two model D410, three D300), each running a fresh installation of Ubuntu (version 6.10 server edition). The D400 systems are 1Ghz Pentium IIIs, with 256 MB of RAM. The D410s are Pentium M processors running at 2GHz and have 512MB of RAM. All network traffic is over 100Mbps Ethernet. A single 8-port Ethernet hub (3Com OfficeConnect Dual Speed Switch 8 plus) connects the five hosts. The hub is also connected to the campus network, eventually via a switched port, so there is little extraneous traffic affecting the overall setup.

All of the configuration settings, scripts and other instructions required to reproduce this test setup are present on the SeNDT web server.[11]

A. *Software Versions*

The LTP/LTP-T code used here is, not surprisingly, our own implementation[12] called LTPlib. The specific version used for the tests described below was checked in on May 1st 2007.[13]

In order to carry out the comparative evaluations described below we had to do a little work on the BP implementation. We started from the dtn-2.3.0 release[14] (dated December 2006) which doesn't support bundle fragmentation. In order to be able to test with large bundles (our tests use files of up to 4MB), we had to modify the UDP CL to support fragmentation at the UDP layer[15].

This change also includes a trivially-simple UDP rate control delay after each UDP packet is sent. In the absence of rate control, UDP packets will basically not arrive. However due to the fact that we are running from userland and not the kernel, the minimum operating system kernel timer granularity

---

[5] The source for that simulation is available at: https://down.dsg.cs.tcd.ie/eslab37/

[6] http://irg.cs.ohiou.edu/ocp/ltp.html

[7] http://svn.ehas.org/svn/dtnehas/trunk/dtnvnuml/README

[8] The fact that the simulator has to hide its nature from the application is independently interesting.

[9] http://linux-net.osdl.org/index.php/Netem

[10] In the case of LTP-T, the log files can contain sufficient information to reconstruct the entire set of LTP sessions that form the end-to-end LTP-T session.

[11] https://down.dsg.cs.tcd.ie/thesis/test-setup/

[12] https://down.dsg.cs.tcd.ie/ltplib/

[13] https://down.dsg.cs.tcd.ie/thesis/ltplib-20070501.tgz

[14] http://www.dtnrg.org/docs/code/dtn_2.3.0.tgz

[15] The change required is included in the dtn-users mail archive: http://mailman.dtnrg.org/pipermail/dtn-users/2007-March/000553.html

for this delay is of the order of 20ms, so that a 4MB transfer requires almost an entire minute. Though very slow, this is acceptable for our purposes, since the same rate-control code is used in the LTP code, so that gross comparisons remain fair.

The version string reported by the "sftp –v" command is "OpenSSH_4.2p1 Debian-7ubuntu3.1, OpenSSL 0.9.8a 11 Oct 2005".

B. *Emulating Delays*

For these tests we emulate a set of fixed delays (e.g. 10s), which can be easily done with Netem. We use two[16] hosts (e2m and m2e) to act as IP routers, using the Netem module in order to enforce link delays. Note that what happens is that a packet is sent from, e.g., terr to landers via e2m. The packet arrives normally at e2m (thanks to IP routes setup for that purpose) but is then delayed for the relevant period (by Netem) before finally being forwarded onto the landers host. Traffic in reverse is also appropriately delayed.

With this setup, a ping from terr to landers can take for example 40 minutes to complete, but it does complete! This is a nice demonstration that IP is, in fact, delay tolerant, even though TCP is not.

IV. SINGLE-HOP TESTS

In this section we describe the setup for, and results from, a set of single-hop tests, where the DTN hop is in this case, a UDP hop. The goal here is to establish the baseline performance characteristics of the various protocols and also to be able to compare protocols in environments that don't occur in our main test scenario, e.g. an LTT of 1s. The protocols tested here are LTP (all green), LTP (all red), LTP (1st 1KB red), BP/UDP and, representing TCP, SFTP (i.e. FTP/SSH).

Each test run consists of 5 iterations of a set of 13 file transfers, for different file sizes. Test runs were repeated with 7 different latencies. The variations are chosen to span a large range (in exponentially growing steps) but where (most of) the protocols remain operational. The LTTs used are 0ms, 1ms, 10ms, 100ms, 1s, 10s and 100s. Note that the RTT is 2 x LTT. The file-sizes used are 1KB, 2KB, 4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, 1MB, 2MB and, lastly 4MB[17] totaling 40MB. These combinations give us 5 x 13 x 7 = 455 samples per protocol, though since not all protocols work with all latencies we sometimes had few samples in our results, though generally around 400.

At the low latencies, there is some variation in timings, e.g., LTP related timings include the time to load the binary. For higher delays, the timing spreads are small enough that our conclusions about the relative performance of the protocols can be confidently drawn. Note that the plots below are of actually test values and not averages (hence the lack of error bars).

[16] One host would probably have sufficed but there are buffer sizing issues, so using two hosts in order to reduce the influence of such artifacts seems like a better option.

[17] test-setup/sftp/mkdata is a script that generates the test-files required.

File Size	Statistic	Disc-BP	LTP-T
4MB	Avg	36.15	64.00
	Std-dev	0.05	3.09
2MB	Avg	36.51	54.03
	Std-dev	0.34	16.25
1MB	Avg	36.23	45.25
	Std-dev	0.35	22.34
512KB	Avg	36.61	43.13
	Std-dev	0.35	28.57

Table 1 – BP/LTP-T DTN Goodput for large files.
Values are in KBps.

In the tests, files are sent from the landers node to the terr node, with the m2e and e2m routers being interposed (at the IP layer), and running Netem to add the relevant delays

The main noteworthy results from these tests are that SFTP fails between 1s and 10s LTT, and that the BP and LTP options perform almost identically over most of the range of tests.

Initially, TCP outperforms the BP and the LTP options, especially when file sizes get bigger and TCP leaves slow start. This is however partly due to the rate control (one packet/20ms) enforced in the BP and LTP code bases. At 10ms LTT, however TCP's advantage is diminishing, so that it would be effectively unusable at 100ms LTT. By10s LTT, TCP is timing out, in this case due to a timer in the SSH daemon ("`LoginGraceTime`") that has a default setting of 120s. With the TCP handshake, and a few cryptographic and login RTTs, this limit is quickly reached.

Tests with small file sizes and small delays show large discrepancies in terms of the timings of the other protocols, basically due to implementation issues, for example, as we've instrumented the LTP test we start a new process each time we measure, which leads to a noticeable delay for these tests.

Our conclusion from this set of tests is that, as expected, both the BP and LTP outperform TCP at higher latencies, but that there is, so far, little to choose between LTP and the BP in terms of performance.

V. TWO-HOP TESTS

Our next set of tests pit the BP against LTP-T and are more telling since we start to see the effects of routers or relays on goodput. Here we will use the orbiter host as a relay between the landers and the terr hosts.

Since we are no longer much interested in low latencies, the test setup here uses LTTs of 10ms 100ms, 1s, 10s and file sizes as described for the single hop tests. The high latency hop is the terr/landers link, the landers/orbiters link has no artificial delay introduced.

In this case we see that LTP-T outperforms the BP, essentially requiring about half as long to forward larger packets when there is a single high-latency link. This same benefit should accrue at each DTN hop, so that on longer paths, LTP-T should also outperform the BP.

Table 1 captures this numerically, by comparing some statistics for the protocols for larger file sizes, where LTP-T's advantage is more noticeable. The figures in the tables are averages and standard deviations for Delay-Discounted DTN goodput, that is, we first discount the time taken for the transfer by 1 LTT, and then calculate the DTN goodput (filesize/delay).

We believe the higher standard deviation of the LTP-T figures is caused by a combination of factors. First, LTP-T tests require loading the ltpd binary for the sending side and this adds varying amounts of time. Second the LTPlib is less stable code than the BP reference code and the variability is perhaps due to mutexes locking out threads and/or time taken for logging[18] – certainly the LTPlib hasn't been optimised in this respect. However the standard deviation for the BP for 512KB files is also high, so perhaps that code base could also be further optimised. For smaller file sizes however the comparisons are less convincing.

VI. Conclusions

This tests reported above demonstrate the effectiveness of the DTN transport strategy – LTP-T, being "closer" to the layer below, can forward packets on receipt whereas the BP code receives the entire bundle before starting to forward.

However, one should be clear – we are not here saying that LTP-T is "better" than the BP, but just that, as a DTN transport it is easier for LTP-T to outperform the BP in some circumstances. Secondly, as an overlay, it is harder, though possible, for the BP to take advantage of CL specifics so as to perform similarly to a DTN transport.

Our results indicate that LTP-T can outperform the BP/UDP in some cases and that LTP-T could be used as a basis for more complex DTNs. Of course, the BP remains a more flexible protocol since it supports a number of different lower-layer options, while LTP-T is only defined when running over UDP.

References

[1] "Delay and Disruption Tolerant Networking," Stephen Farrell and Vinny Cahill, ISBN 1-59693-063-2, Artech House, 2006.

[2] Cerf. V. et al., "Delay-tolerant Networking Architecture," Internet RFC 4838, April 2007.

[3] Farrell, S. and Cahill, V., "LTP-T: A Generic Delay Tolerant Transport Protocol," TCD Computer Science Technical Report TCD-CS-2005-69, 7 December 2005.

[4] Scott K. and Burleigh, S., "Bundle Protocol specification", Internet-draft, draft-irtf-dtnrg-bundle-spec-10.txt, July 2007, work-in-progress.

[5] http://www.stk.com/

[6] Fall, K. and Varadhan, K,"ns Notes and Documentation," Technical report, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997.

[7] Bruno, D. "The JBDS (Java Based Deep Space) Simulator: A New Approach," Trinity College Dublin Computer Science Technical Report, TCD-CS-2006-63, December 2006. https://www.cs.tcd.ie/publications/tech-reports/reports.06/TCD-CS-2006-63.pdf

[8] Varga, A., "The OMNeT++ discrete event simulation system," In European Simulation Multiconference (ESM'2001), Prague, Czech Republic, June 2001.

[9] Pawlikowski, P., "Do Not Trust All Simulation Studies of Telecommunication Networks," Proc. International Conference on Information Networking (ICOIN) 2003, Jeju Island, Korea, 2003.

[10] Kurkowski, S., Camp, T., Colagrosso, M., "Manet simulation studies: The incredibles," Mobile Computing and Communications Review pp. 50-61, October 2005.

[11] Heidemann, J, et al, "Effect of detail in Wireless Network Simulation," USC/ISI TR-2000-523b, January 2001.

[12] Heidemann, J. Mills, K. Kumar, S., "Expanding confidence in network simulations," IEEE Network, Vol 15, Issue 5, p58-63, Sep/Oct 2001.

[13] Farrell, S., "A Flexible Interplanetary Internet", presented at the 37th ESLAB Symposium, Tools And Technologies For Future Planetary Exploration, ESTEC, Noordwijk, The Netherlands, 2-4 December 2003.

[14] Fermín Galán, David Fernández, "Use of VNUML in Virtual Honeynets Deployment", IX Reunión Española sobre Criptología y Seguridad de la Información (RECSI), Barcelona (Spain), September 2006. ISBN: 84-9788-502-3.

[15] Hemminger, S., "Network Emulation with NetEm," Linux Conference Australia, LCA2005, Canberra, April. 2005.

[18] This is substantial for LTP-T routers at present – in some test runs for the Martian network, log files have reached the 2GB limit!