# Experiments with Delay and Disruption Tolerant Networking in AX.25 and D-Star Networks

John Ronan, EI7IG
Telecommunications Software & Systems Group
Waterford Institute of Technology
Cork Road, Waterford, Ireland
Email: jronan@tssg.org

Kristian Walsh
Telecommunications Software & Systems Group
Waterford Institute of Technology
Cork Road, Waterford, Ireland
Email: kwalsh@tssg.org

Darren Long, G0HWW
Bury St. Edmunds
England.
Email: darren.long@mac.com

**Abstract**

This paper examines the performance of DTN bundling layers against native protocols for radio data network with a view to possible deployment for emergency communications purposes. The authors conducted experiments with an existing DTN convergence layer in both AX.25 point-to-point and D-Star multi-hop communications links with hidden transmitters. The experimental results show that the DTN system exhibits marginally better performance than TCP/IP, appears to perform better where there are hidden transmitters, and has a more compact link utilisation pattern than TCP/IP. However, situations where obvious improvements can still be made were also identified.

**Index Terms**

Disruption Tolerant Network, Delay Tolerant Network, DTN, AX.25, D-Star

## I. Introduction

At the 2008 TAPR DCC, [1] briefly introduced Delay/Disruption Tolerant Networking, and cited the work of this paper's authors. This paper provides more detail about the authors' ongoing work with delay- and disruption-tolerant networking.

The experiments made use of the DTN2 reference implementation [2], a flexible software framework for experimentation, extension and real-world deployment of Delay Tolerant Networking (DTN) systems. We have taken this framework and used it to produce a Convergence Layer (CL) for the AX.25 networking protocol. (For D-Star testing, the TCP-CL Reference Implementation was used).

The ultimate goal is to produce a usable, delay-tolerant substitute for existing digital emergency communications systems.

As our AX.25 CL testing was with 1200 baud AX.25 links, it seemed sensible to determine the overhead incurred by the use of DTN, and how a DTN solution compared to existing transfer mechanisms.

The D-Star testing was with Icom ID-1 D-Star radios. Both TCP/IP over D-Star and DTN-CL over D-Star were tested, in both point-to-point and multi-hop configurations.

## II. DTN over AX.25, Point-to-point

### A. DTN Convergence Layer Implementation

The AX.25 Connected Mode Convergence Layer (AX25CM-CL) is a convergence layer implementation for AF_AX25 SOCK_SEQPACKET sockets on the Linux platform.

The AX25CM-CL transports the DTN "bundles" described by RFC-5050 [3] directly over an AX.25 connection that operates solely as a Layer 2 protocol. In this respect, the AX.25 CL is similar to the existing Bluetooth-CL.

Currently, the only major difference between the AX.25 CL and the TCP-CL Protocol is that the AX.25 implementation is extended to include a 32-bit CRC appended to each TCP-CL Protocol segment. This is necessary in order to ensure that any corruption of AX.25 KISS [4] data frames can be detected, as well as providing additional means to detect protocol errors introduced by the implementation.

This CRC was added as, during early testing using two Linux PC's and sound-modems, it was found that occasionally, corrupt frames were received which crashed the receiving DTN daemon. This also happened (though rarely) when testing commenced with hardware TNCs.

The TCP-CL and the AX25CM-CL share commonality provided by the ConnectionConvergenceLayer base class. Supporting classes providing the interface to AF_AX25 SOCK_SEQPACKET sockets are provided as an extension to the DTN2 *oasys* library.

*1) Capabilities and Limitations:* The AX25CM-CL code has been in active development since January 2007, when it was first branched from the TCP-CL and *oasys* support classes. Currently, the AX.25 CL allows for point-to-point links between two peers, and also paths containing a single non-DTN digipeater, for an AX.25 *connected mode* link.

As of yet, no announcement or discovery mechanism has been implemented and thus links must be manually configured and initiated.

## B. Theoretical Performance of AX.25 Links

In assessing the performance of the DTN implementation, it was useful to consider the theoretical performance limits of the underlying data transport. In this case, that transport is AX.25.

*1) Experimental settings:* Table I lists the significant parameters of the experimental AX.25 link.

TABLE I
CHARACTERISTICS OF EXPERIMENTAL TRANSFER

| parameter | value |
|---|---|
| link speed, bit/s | 1200 |
| $T_{slot}$, s | 0.020 |
| $T_{txdelay}$, s | 0.150 |
| $T_{tail}$, s | 0.020 |
| $p$ | 0.250 |
| data length, bytes | 7182 |

*2) Model Transfer times:* To examine the performance of the AX.25 link, a "best case" model was created by using the timings from the AX.25 specification and inserting the characteristics of the experimental link. Transfer times on AX.25 networks have a small probabalistic component, as a random delay is used for congestion control. To minimise the effect of congestion on the experimental results, a point-to-point link was used. The probabilistic factor, $p$, was set to 0.25[1], which entails an average delay of $0.25T_{slot}$ to each transmission. We endeavoured to calculate *best case* values for transfer times given different window *maxframe* sizes.

$T_{frame}$, the transmission time for one data frame is obtained using the following formula:

$$T_{frame} = \frac{(bytes \times 8.004) + 160}{(bitrate)}$$

where 160 is the number of bits comprising the AX.25 preamble, header, check sequence, and end-of-frame marker.

[1]this was achieved by setting the *persist* value to 64 out of a maximum value of 255

The value 8.004 is used to take into account the additional bits stuffed into payload data octets with the value 01111110. Assuming a uniformly distributed payload byte values, such extra bits will occur on average once in every 256 octets.

Each acknowledgement is a single transmission of a frame with zero payload bits. Allowing for transmission setup and release times, $T_{ack}$, the average time required to send an acknowledgement is:

$$T_{ack} = T_{txdelay} + \frac{160}{bitrate} + T_{txtail} + p \times T_{slot}$$

The number of acknowledgements sent depends on $maxframe$, the acknowledgement window size for the link. Also, where a $maxframe$ of greater than 1 is chosen, the transmitter is allowed to send multiple data frames in one transmission, which eliminates all but one set of $T_{txdelay}$ and $T_{txtail}$ delays.

As each acknowledgement window of data packets is sent in a single transmission, and each such transmission will generate one acknowledgement, the following formula for transmission time of a message containing $frames$ number of data frames can be easily derived:

$$windows = Ceiling(\frac{frames}{maxframe})$$

$$\begin{aligned} T_{message} &= frames \times T_{frame} \\ &\quad + windows \times (T_{txdelay} + T_{txtail} + p \times T_{slot} + T_{ack}) \end{aligned}$$

For a window size of 1, each packet requires an acknowledgement, which will negatively affect the throughput of the link. The half-duplex nature of the physical link used means that acknowledgement frames incur a very high cost as each transmission, be it a single frame, or a group of frames, must also include the initial settling period, $T_{txdelay}$ to allow the transmitter to stabilise before data can be sent.

Using the experimental parameters in Table I, we calculated frame transmission time for a transfer of $n$ 255-byte frames as follows. First, the transfer time for a single frame, without link stabilisation or release delays, $T_{frame}$, is determined, as this is constant regardless of window size:

$$T_{frame} = \frac{(255 \times 8.004) + 160}{1200} \qquad = 1.933971$$

$T_ack$, the time required to send an acknowledgement, is also constant for all window sizes:

$$\begin{aligned} T_{ack} &= T_{txdelay} + \frac{160}{bitrate} + T_{txtail} + p \times T_{slot} \\ &= 0.150 + \frac{160}{1200} + 0.25 \times 0.020 \\ &= 0.308 \end{aligned}$$

Using these values, and formula for $T_{message}$, previously, the values in Table II were obtained. It should be noted that these figures represent a theoretical peak performance of the link, and do not account for collisions, interference[2] or the delay incurred by the transfer of data between the host system and from the TNC devices over RS-232 [5] serial interfaces. Because the model did not take account of these additional overheads or the time required to process higher-level protocol commands, none of the experimental results were expected to reach this level of performance.

[2]great care was taken to monitor the frequency for any interference during the tests

TABLE II
THEORETICAL MINIMUM TRANSFER TIMES, RAW AX.25 TRANSFER

| Window size | timings from model |
|---|---|
| 1 | 67.2 |
| 2 | 60.4 |
| 3 | 58.0 |
| 4 | 57.1 |
| 5 | 56.1 |
| 6* | 55.6 |
| 7* | 55.6 |

* values are the same for window sizes of 6 or 7 as both settings will generate only 5 acknowledgement frames for a 7182-byte transfer

## C. AX.25 Experimental Network

*1) BBS and KISS mode:* File transfer tests were performed using the onboard BBS software on the TNCs, and also by placing the TNC equipment into KISS mode. KISS mode was required for FTP and DTN transfers.

*2) Network:* Figure 1 shows the experimental network used to measure the system performance.

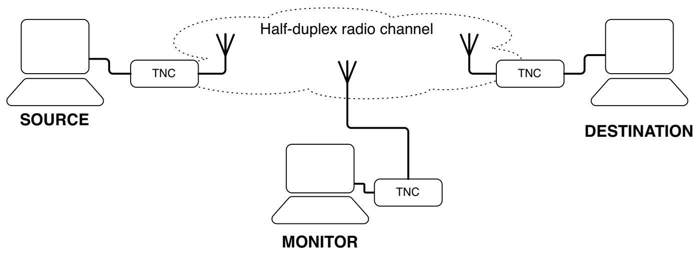

Fig. 1. Experimental setup used to measure AX.25 performance. Source and Destination devices were connected on a single RF data channel (i.e., half-duplex)

As the AX.25 TNCs and transmitters used for Source and Destination nodes were not identical, transfers were performed in both directions in an effort to minimise the effects of using different equipment. One side used a SCS PTCIIex TNC, connected to a Yaseu FT-847. The other used a KPC3+ TNC and a Yaesu FT-1500M transmitter.

A third integrated transceiver and TNC (Kenwood TH-D7 in KISS mode) was used to monitor the radio channel to log all transmitted AX.25 frames and allow the for the measuring of transfer times.

To obtain a valid set of readings, five transfers of the candidate test file were completed for each setting of the AX.25 window parameter ($maxframe$). These readings were then combined using a simple average in order to give an indicative time for the given window setting.

The test file used contained 7182 bytes of ASCII-encoded text data. For BBS mode tests, this data file was pre-loaded into the source TNC's built in Bulletin Board System (BBS) server.

For testing of the linux AX.25 implementation, we used a combination of *ax25d* (part of the Linux AX25 [6] subsystem) to respond to the AX.25 connection request, and *uronode* [7] to deal with the connection itself. The *axmail* [8] program was used to access a local SMTP mailbox to which the test file had been pre-loaded.

When it came to testing using TCP/IP [9] [10], both TNCs were first configured into KISS mode and then the Maximum Transmission Unit (MTU) and window sizes were set according to Table III on the

Linux host before each transfer was begun. Transfer of the file data for TCP/IP tests was performed using FTP.

TABLE III
TCP/IP TEST SETTINGS

| Window Size | MTU | TCP Window |
|---|---|---|
| 1 | 255 | 255 |
| 2 | 255 | 510 |
| 3 | 255 | 765 |
| 4 | 255 | 1020 |
| 5 | 255 | 1275 |
| 6 | 255 | 1530 |
| 7 | 255 | 1785 |

For the DTN test, The *dtncp* utility was used command to send the file from source to destination.

Obviously the application-layer protocols used by the *ftp*, *axmail* and *dtncp* tools all add their own small amount of overhead to the file transfer (above that already added by AX.25). However, it was considered to be valid to include this in the final results, as the amount of additional data is quite small in relation to the file being transferred, and will be representative of "real world" usage.

## III. DTN OVER D-STAR, POINT-TO-POINT AND MULTI-HOP

### A. *Convergence Layers*

For D-Star testing, both the reference TCP/IP Convergence Layer and the NACK-Oriented Reliable Multicast (NORM) based CL [11] were used to investigate DTN performance. NORM was chosen for examination in response to research done by [12] which indicated that NORM would be suited to multi-hop wireless DTN links.

### B. *Experimental Network*

Figure 2 shows the experimental network used to measure the system performance. Each node in the network was operated by an ICOM ID-1 transciever. The devices marked "Source" and "Destination" had no direct visibilty to each other: a deliberate arrangement to investigate the effect of hidden transmitter interference.

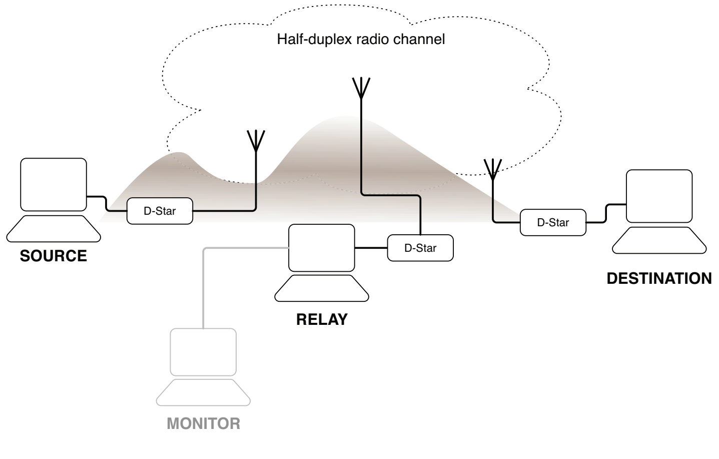

Fig. 2. Experimental setup used to measure D-Star DTN performance.

As for the AX.25 testing, five transfers of the test data were performed and the times obtained were averaged. FTP and *dtncp* were used for plain TCP and DTN transfers respectively.

Dynamic routing was disabled in the test network, to avoid routing broadcasts interfering with transfer times.

## IV. Results

Tables IV and V list the results obtained for transfers between the two TNC devices. Table VI lists the results from the D-Star transfers, for path lengths of one and two hops.

TABLE IV
PTCIIex reading from KPC3+

*average transfer times in seconds for window sizes from 1 to 7*

| Win. size | BBS | KPC3+ KISS | Both KISS | TCP/ IP | DTN | *model* |
|---|---|---|---|---|---|---|
| 1 | 119.2 | 75.0 | 75.6 | 146.0 | 84.6 | 67.2 |
| 2 | 87.2 | 65.0 | 67.0 | 104 | 75.8 | 60.4 |
| 3 | 86.2 | 60.6 | 61.6 | 99.8 | 73.8 | 58.0 |
| 4 | 86.2 | 59.6 | * 63.0 | 98.4 | † 72.0 | 57.1 |
| 5 | 86.2 | 57.6 | 58.4 | 97.2 | † — | 56.1 |
| 6 | 86.2 | 57.0 | * 60.2 | 97.6 | † — | 55.6 |
| 7 | 86.2 | 57.2 | * 61.0 | 95.8 | † — | 55.6 |

* receiver timeout on last frame group ; †window sizes above 3 are not honoured, see section V-A2.

TABLE V
KPC3+ reading from PTCIIex

| Win. size | BBS | PTCIIex KISS | Both KISS | TCP/ IP | DTN | *model* |
|---|---|---|---|---|---|---|
| 1 | 76.6 | 76.2 | 76.2 | 173.2 | 85.8 | 67.2 |
| 2 | 65.0 | 65.0 | 67.0 | 104.2 | 75.6 | 60.4 |
| 3 | 61.4 | 62.0 | 62.0 | 98.8 | 74.8 | 58.0 |
| 4 | 59.6 | 59.8 | * 63.2 | 97.2 | † 71.25 | 57.1 |
| 5 | 58.2 | 58.6 | 59.6 | 96 | † — | 56.1 |
| 6 | 57.8 | 58.0 | * 60.8 | 95.8 | † — | 55.6 |
| 7 | 57.6 | 57.6 | * 61.6 | 95.2 | † — | 55.6 |

* receiver timeout on last frame group ; †window sizes above 3 are not honoured, see sectionV-A2.

TABLE VI
D-Star performance

| Hops | FTP TCP | DTN TCP | DTN NORM |
|---|---|---|---|
| point to point | 193.6 | 190.2 | 184.0 |
| 2 hops | 418.8 * | 412.8 | — † |

* individual results exhibited high deviation from this mean — see section V-B1 ; †NORM measurements not possible for two-hop network.

## V. Discussion

### A. AX.25 point-to-point

*1) Overhead of DTN:* Overall, the performance of the DTN layer was acceptable, and disproved the authors' "gut instinct" which suggested a much greater overhead from using DTN bundling. While a full set of measurements could not be obtained, the CL implementation shows considerable promise, and comfortably out-performs TCP/IP.

*2) Problems with large window sizes on AX.25:* Tables IV and V are missing readings for window sizes (AX.25 $maxframe$ parameter) above 4. This is because of a bug which was discovered during testing. On the fifth consecutive run with $maxframe$ set to 4, the AX.25 implementation in the host computer appeared to enter an unstable condition: instead of obeying the chosen $maxframe$ setting, the source unit flooded the receiver with all 29 frames of the data transfer, causing a breakdown in flow control for both source and destination.

The fact that, once manifested, this behaviour persisted across all subsequent runs of the DTN test would suggest that a the Linux kernel AX.25 implementation itself is partially responsible for the problem.

*3) Over-Acknowledgement:* Currently, the AX25CM-CL produces a flurry of (TCP-CL Protocol) segment ACK messages (one for each segment/frame) and sends these as distinct frames. (Figure 3) A more conservative approach to ACK generation would be prudent, by aggregating more than one ACK into a frame and/or adopting a selective ACK mechanism. The first approach (aggregating multiple ACKs into a single frame) should in theory already happen, but doesn't in practice: further investigation may required to discover the root cause of this.

This problem could be minimised by deferring the first AX.25 acknowledgement in such a manner that it is sent in the frame containing the DTN (or TCP) acknowledgement message.

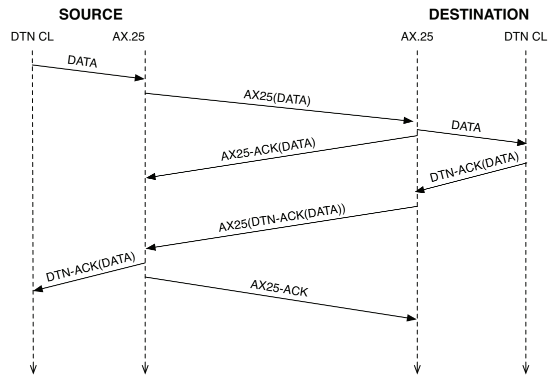

Fig. 3. Multiple acknowledgements produced by the DTN CL

*4) AX.25 Acknowledgement timeouts:* The "Both KISS" entries in Tables IV and IV contain some anomalous timings, marked with an asterisk. Following investigation of the logs, it was discovered that these are due to the message transfer ending with a window containing only one or two frames. In these cases, the receiver does not send an acknowledgement immediately, but instead waits to see if any more frames arrive that would fill the window. As no such frames follow, the receiver times out (AX.25 T2 or *resptime* timer), and acknowledges what it has received.

*5) Firmware bottleneck in KPC3+:* The "BBS" column in Table IV shows for window sizes above 3 frames, the TNC's own firmware was the bottleneck to data transfer. Using KISS mode on this device shows a marked improvement. The testing was done with a KPC3+ and version 8.3 (KPC3P-7ADC9125-8.3) of the firmware. A brief loan of a KPC3+ with version 9.1 of the firmware was obtained. It performed much better in this test averaging 72 seconds with *maxframe* set to 7, but it still is much slower than either the PTCIIex or KISS mode.

## *B. DTN/D-Star*

*1) Variability of FTP performance:* From the average figures shown in table VI, it would appear that the DTN transfer is faster than using FTP, albeit only marginally. However, the raw measurement data taken for FTP multi-hop transfers exhibits a higher standard deviation than that for the DTN (FTP: $\sigma$ = 51 seconds, versus $\sigma$ = 21 seconds for DTN). Examination of the packet logs for the FTP tests shows that the most likely cause for this was indeed hidden transmitter interference, as destination and source nodes attempted to simultaneously send to the relay node.

Deviation in DTN results was much less random, and would suggest a that there is a fixed-interval polling mechanism at work within the DTN stack: some transfers were relayed almost immediately, others incurred a delay of up to 20 seconds.

*2) NORM on multi-hop networks:* Unfortunately, the authors encountered difficulties in getting the NORM software to operate for more than one network hop. While this is probably due to a configuration issue, time pressures prevented more detailed investigation. The results for NORM on point-to-point transfers show that it is significantly faster than either TCP or the existing DTN over TCP implementation.

*3) Link Utilisation:* One interesting aspect of the DTN implmentations was their link utilisation pattern. Figure 4 compares the traffic observed for DTN and FTP. Because of bundling, the DTN transfer does not make use of the link between hop 1 and 2 until all of the data has been transferred from source to hop 1. In contrast, the FTP transfer produces traffic on both segments for the duration of the transfer.

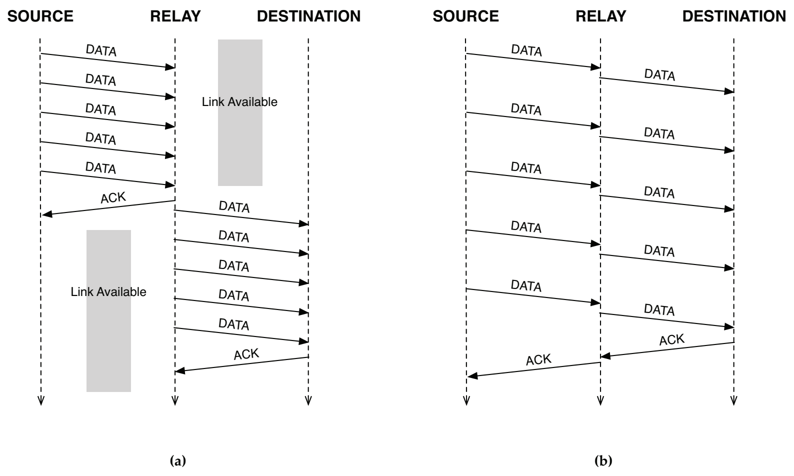

Fig. 4. Link utilisation patterns: (a) DTN, (b) FTP. (TCP flow control has been omitted from both diagrams)

It should be noted that, all other factors being equal, neither DTN nor FTP will use *more* of the available link capacity; however, the DTN implementation uses each link in a more compact fashion.

## *C. Bundle integrity in the AX.25 CL*

In [13] the authors highlight issues of lack of checksum for error detection with bundles, and bundle integrity is a big issue in the current DTN architecture. Without using bundle security mechanisms, there is no built-in error detection (or correction) mechanism available at the bundle layer.

This has two major ramifications.

- Bad bundles can crash the Bundle Protocol daemon (in our case dtnd), or
- Bundles can be delivered to their destination as if they were valid data

As an illustration of the effect of corrupt data on *dtnd*, in early experiments with Prophet router [2], a byte-order issue in the Prophet code was able to bring down an entire *dtnd* network. One of the authors used a PowerPC-equipped computer (big-endian) to connected to a DTN running with Intel and ARM devices (little-endian) machines. As soon as the PowerPC node transmitted Prophet routing data, all other *dtnd* instances in the network crashed. While this isn't an example of the lack of error detection— in this case error detection may not have caught the problem— it does, however, illustrate the problems posed by malformed data being ingested into *dtnd* instances.

The classic end-to-end principle [14] used in the Internet stresses the importance of eliminating in-memory corruption in routers. In a DTN, the storage of data for later retransmission raises the risk of corruption in secondary storage (disks,etc) too, as well as the obvious risk of corruption on error-prone links.

However, the big problem for bundle integrity in a DTN is the possibility of reactive or proactive bundle fragmentation within the network. If either form of fragmentation occurs, determining the integrity of bundle fragments in the network middle is likely to be impossible, and in all likelehood, could only be achieved at the ultimate destination, when de-fragmentation occurs.

The knock-on effects of corrupt bundles in a DTN makes it imperative that CLs adopt a robust error detection scheme in order to ensure that the bundles (or bundle fragments) they pass up to the bundling agent are confirmed to be error free. Robust error checking provides the earliest opportunity to solicit a retransmission of corrupted bundles or bundle fragments in the CL itself. Note that this doesn't solve the actual problem, as in memory or storage corruption may still occur (undetected), and as in the end-to-end argument, ultimate responsibility for integrity checking must reside with the endpoint applications. However, if custody transfer is being used, it would seem to be a major disadvantage if a prospective bundle custodian isn't able to confirm the integrity of the bundle which it is requested to take into its custody.

The issues that were experienced within the earlier implementaions of the AX25 CL resulted in two types of behavior. The first, benign behaviour caused TCP-CL protocol messages to be determined to be invalid, which in turn caused the CL link to be dropped (and subsequently re-established, if viable). The second type of behaviour resulted in the entire *dtnd* service crashing! After adding a CRC32 checksum to each TCP-CL segment, the AX25 CL now detects malformed TCP-CL segments and terminates the connection[3], and assuming that the connection will be restablished and transfer will be attempted again, if connectivity permits. There may still be a residual bug associated with this - which manifests itself as a *stuck socket*, but the authors have made no progress investigating this issue

The issue of bundle integrity is a more difficult one than it appears at first inspection, and it still is not clear how the often conflicting requirements (network transport neutrality versus confidence in bundle integrity) can be easily reconciled. The author's approach is to effectively ignore the entire problem at the bundle protocol level, but implement a robust AX25 CL in order to at least ensure that it didn't exacerbate the problem or take down the node's *dtnd* service.

## VI. Conclusion

The prototype AX.25 DTN CL performs well, considering its early stage of development. However, bugs remain in the implementation that could be addressed to further improve performance.

The measurements taken provide an illustration that, even without considering robustness in the face of disruption, the ubiquitous TCP/IP protocol is not *always* the best choice.

Measurements on D-Star transfers show a slight performance benefit for DTN on point-to-point links, and also on multi-hop transfers. However, the underlying measurements taken for multi-hop transfers show a high standard deviation, due to interference from other transmitters. In light of this variance, we cannot say with confidence that the DTN transfer would always be faster than FTP, but its performance was certainly comparable.

[3]the rationale being that to send malformed packets, the sender must have been placed in an undefined state, one which a complete reset of the connection may resolve

Moving from the TCP convergence layer to the UDP-based NORM convergence layer provided an increase in point-to-point performance over both FTP and the TCP-CL. Unfortunately, difficulties in performing multi-hop tests using NORM prevented the authors from obtaining useful measurements for this configuration.

One interesting outcome of using DTN for multi-hop transfers was that the DTN transfers exibited more compact utilisation of each link. In a multi-channel network, this could yield greater network utilisation. In cases where expensive time-billed point-to-point links are deployed (e.g., satellite phone), the use of DTN rather than straight TCP could give rise to considerable cost savings.

## VII. Future Work

Although only briefly examined for this paper, the NORM convergence layer showed promise, and would be worthy of further investigations. Of particular interest would be investigating the performance of NORM in larger multi-hop networks.

The experimental results obtained for multi-hop DTN over D-Star show a considerable variance due to "hidden transmitter" interference (all three nodes shared the same channel). More testing will be required to determine whether the TCP or DTN transfers are more susceptible to this effect.

The AX.25 CL is still a work in progress, and more work will be required to bring it to mainstream use. Techniques from the NORM convergence layer could also be applied to the AX.25 CL to reduce the number of acknowledgements generated by the AX.25 CL.

Finally, the instability in the face of invalid data highlights the need for more robust error checking, and possibly error correcting, techniques for DTN. Investigation of these techniques is certainly warranted.

## References

[1] F. Brickle, "A brief introduction to delay tolerant networking," in *27th ARRL and TAPR Digital Communications Conference*. 225 Main Street, Newington, CT 06111-1494, USA: ARRL, 2008, pp. 6–8.

[2] "Delay tolerant networking research group - code," http://www.dtnrg.org/wiki/Code.

[3] K. Scott and S. Burleigh, "Bundle Protocol Specification," RFC 5050 (Experimental), Internet Engineering Task Force, Nov. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5050.txt

[4] M. Chepponis and P. Karn, "The KISS TNC: A simple Host-to-TNC communications protocol," in *6th Computer Networking Conference*. 225 Main Street, Newington, CT 06111-1494, USA: ARRL, 1987.

[5] "Eia standard rs-232-c interface between data terminal equipment and data communication equipment employing serial data interchange," August 1969.

[6] "LinuxAX25," accessed on 2009-01-02. [Online]. Available: http://www.linux-ax25.org/wiki/LinuxAX25

[7] "Uronode," accessed on 2009-01-02. [Online]. Available: ftp://ftp.uroweb.net/pub/ax25/

[8] "axMail," accessed on 2009-01-02. [Online]. Available: ftp://ftp.uroweb.net/pub/ax25/

[9] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFCs 1122, 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[10] ——, "Internet Protocol," RFC 791 (Standard), Internet Engineering Task Force, Sep. 1981, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[11] "NACK-Oriented reliable multicast," accessed on 2009-07-20. [Online]. Available: http://cs.itd.nrl.navy.mil/work/norm/

[12] C. Rigano, K. Scott, J. Bush, R. Edell, S. Parikh, R. Wade, and B. Adamson, "Mitigating naval network instabilities with disruption toler," in *Military Communications Conference, 2008. MILCOM 2008. IEEE*, Nov. 2008, pp. 1–7.

[13] L. Wood, W. M. Eddy, and P. Holliday, "A bundle of problems," in *IEEE Aerospace Conference*. IEEE, 2009.

[14] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," 1984.