2016

Effective C2 Comms to the Tactical Edge in Challenged, Disrupted, and Denied Environments

Rohrer, Justin P.; Monahan, Michael K.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/57740

MONTEREY, CALIFORNIA

EFFECTIVE C2 COMMS TO THE TACTICAL EDGE IN CHALLENGED, DISRUPTED, AND DENIED ENVIRONMENTS

Report Type: Final Report
Period of Performance: 09/01/2015-01/31/2017

Project PI: Dr. Justin P. Rohrer, Assistant Professor of Computer Science, GSOIS
Additional Author/Authors:
Student Participation: Michael K. Monahan, ITACS, Computer Science

Prepared for:
Topic Sponsor: Maj Jeffrey Sykes, USMC, HQMC Aviation
Research Sponsor Organization (if different): HQMC Aviation
Research POC Name: Maj Scott Cuomo
Research POC Contact Information: scott.cuomo@usmc.mil, 703-309-3320

Title: Effective C2 Comms to the Tactical Edge in Challenged Disrupted, and Denied Environments

Report Date: 17/03/2017 Project Number (IREF ID): NPS-FY16-M343-C
Naval Postgraduate School / School: GSOID/CS

EXECUTIVE SUMMARY

Project Summary
Native IP networks are ill-equipped to handle the communication challenges found in wireless comms environments, resulting in communications outages that degrade C2 data flow and subject the user to disconnection, timeouts, and repeated login requests. We counter these limitations by integrating DTN (Disruption-Tolerant Networking) technology into the IP network using software+hardware or software-only solutions as appropriate. This work evaluates the tradeoffs between the currently available DTN software implementations and seeks to identify the one with the highest technical readiness level, as well as any barriers to adoption that may be present. We find that no current implementation is fully ready, and that each have particular pros and cons to adoption.

Keywords: disruption-tolerant networks, DTN, RFC5050, bundle protocol, BPA

Background

Native IP networks are ill-equipped to handle the communication challenges found in wireless comms environments, resulting in communications outages that degrade C2 data flow and subject the user to disconnection, timeouts, and repeated login requests. We counter these limitations by integrating DTN (Disruption-Tolerant Networking) technology into the IP network using software+hardware or software-only solutions as appropriate. For greatest effectiveness, this solution will be customized to support specific C2 applications and prevent application timeouts.

Disruption-tolerant networking (DTN) is an approach to computer networking that seeks to address the technical issues in heterogeneous networks that may lack continuous network connectivity. DTNs accomplish this by providing a means for some intermediary device or series of devices (e.g. gateways or routers) to intercept and place packets into persistent storage (this could be a hard drive, solid state flash, RAM, etc), during times when a link or a set of links experience disruptions or severe latency. When network conditions once again become normalized, the intermediary device will again forward packets to its next hop.

Some previous DTN-related research at the Naval Postgraduate School involved evaluating and testing the operation of the SPINDLE DTN software. SPINDLE is developed by Raytheon BBN Technologies for the DARPA Disruption Tolerant

Networking program for exclusive use by the DOD. One of the questions that arose during this research was how does SPINDLE compare to some of the other open-source DTN software implementations. A previous DTN comparison study did just this, and compared the performance characteristics of the RFC 5050 DTN Reference Implementation, NASA/Jet Propulsion Laboratory's ION DTN software and University of Braunschweig's IBRDTN software.

There is no known study that compares SPINDLE with these other open-source DTN implementations. A SPINDLE software installation can use a large amount of memory resources, so a key motivation was to see how SPINDLE performance and the overall functional operation of the SPINDLE software agent, compares with the ION and IBR DTN software implementations, of which both have significantly lower memory footprints. Many mobile devices, such as tactical handheld units, wireless sensor motes, and other embedded systems have minimal amount of memory and CPU processing power, so it may be of benefit to use an alternative DTN implementation.

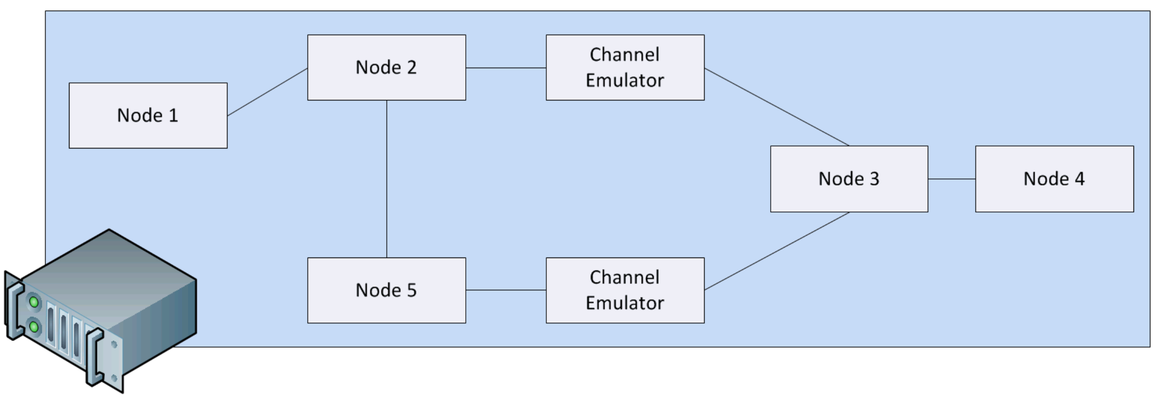
The best-known reference for DTN architecture is Kevin Fall's 2003 paper [5], which generalized from earlier work specific to handling long-delay links. Our work is in the context of applying such an architecture to challenges specific to USMC tactical networks. This research effort concerns implementations of the Bundle Protocol Specification, therefore RFC5050 [4] is our primary reference for correct operation of this software. Vint Cerf and Scott Burleigh also provided context for the protocol described in RFC5050 in [3]. We also refer to and extend the performance evaluation methodology of Wolf et al [1]. The benefit to this is not reinventing the wheel on basic methodology for quantitative evaluation, however they did not address a number of qualitative factors that are key discriminators in the technical readiness of the software, and did not have access to SPINDLE. Addressing these factors will significantly differentiate this work from theirs. While our effort in this case is not specifically concerned with DTN-specific application software, we will be using such software to generate traffic load in our testbed, and as such refer to the Artemois et al [2] survey of such applications.

Our previous work in this area has included developing gateway technology for integrating DTN protocols into IP networks, and these efforts have been published in [6,7,8]. It was this work that highlighted deficiencies in some DTN implementations and necessitated the study currently proposed.

Experimental Results

Test Environment:

Testbed: The DTN testbed is comprised of one physical server and uses computer hardware virtualization to emulate the individual DTN nodes and corresponding network topology. The server is a Dell PowerEdge 2950 Server, with an Intel Xeon 4-core 3.0Ghz processor, 16 GB RAM, 255GB of disk storage, running Ubuntu 12.04 LTS linux. Oracle VirtualBox (v4.3.14) is the virtualization software that was used to create and run the virtual machine instances in the testbed. The DTN guest virtual machines were allocated 2GB of RAM and 8 GB of hard disk storage. The DTN router virtual machines were allocated 512MB of RAM and 2GB of disk storage. Intel VT-x hardware acceleration was enabled in the host machine's BIOS and in each of the VM's settings in VirtualBox. VT-x is a set of processor enhancements to improve virtualization performance that allows for near-native speed in a virtual machine.



Host Nodes: Node 1 and Node 4 represent the end points of the emulated network topology. (see figure 1) Node1 could represent a mobile device in the field, such as a hand-held device carried by a soldier, or a radio unit affixed to a moving vehicle such as a humvee. Node 4 could represent a fixed node connected via a satellite link, such as a command center (where a units in the field can report to). Each host node has two network interfaces configured: one interface for communication with the DTN network and one interface is used for network management.

Router Nodes: The router nodes (node1, node2 and node5) represent the intermediate nodes in the emulated network topology. These nodes can represent fixed nodes (such as a forward operating base) or other mobile nodes, such another vehicle acting as a relay. Node2 and Node3 represent the direct gateways for the host nodes. They have four network interfaces configured: one interface connected to the host node, two links to other router nodes and one interface for network management.

Channel Emulator Nodes: The Channel Emulator VMs represent the radio links between the router nodes. These virtual machines use a linux kernel networking feature called NetEm (short for network emulator) NetEm is an enhancement of the Linux traffic control facilities that allow to add delay, packet loss, duplication and more other characteristics to packets outgoing from a selected network interface. NetEm is built using the existing Quality Of Service (QOS)and Differentiated Services (diffserv) facilities in the Linux kernel. With the ability to orchestrate delay with random variations and random packet loss using NetEm, a wireless networked environment can effectively be emulated.

Testing Methodology:

IPerf was used to take baseline throughput measurements between the two DTN host nodes. Iperf is a commonly used CLI based network testing tool that can be used to generate TCP or UDP traffic between two end points to measure raw network throughput (tests were run using UDP mode which has an option that allows for a data rate to be specified). This provided a frame of reference when comparing network performance between non-DTN and DTN-based network environments in the virtual testbed.

We used the DTNPerf utility to measure bundle transfer throughput for the ION and IBRDTN DTN implementations. DTNPerf is an open source tool, developed by researchers at the University of Bologna that was designed to provide IPerf-like functionality when testing network performance in DTN environments. We used the *dtnsource* and *dtnsink* utilities (which includes a benchmarking option) included with the SPINDLE distribution to test throughput between SPINDLE nodes.

To measure end-to-end latency via traversal of BPA agents at each DTN node, we used the “ping”-like utilities that each of the DTN implementations included in their respective distributions. Both SPINDLE and ION’s ping utilities did not include the option to specify bundle payload size (analogous to the “packet size” option found in most standard ping utilities), while IBRDTN did include this option. Since we had access to

SPINDLE and ION source code, we were able to make slight modifications to the code that allowed for us to specify bundle payload size.

All experiments focused on testing throughput using memory-based storage. Disk-based storage testing was not performed during these initial experiments, but should occur in future testing. Latency and throughput tests were performed using the TCP convergence layer adapter (CLA) implementations for ION and IBRDTN testing, while the UDP CLA was used for SPINDLE testing. We were unable to test using Spindle's TCP CLA due to an issue that appears to be related to the routing component not being able to obtain static route information when the TCP CLA is specified the SPINDLE configuration file. At this time, we are unsure if this is due to a misconfiguration in the set-up file or buggy software. In contrast, we were unable to get DTNPerf tests to run successfully using the IBRDTN UDP CLA – so ultimately, the decision was made to test using the CLAs that were known to work properly for each respective DTN implementation.

Both bundle throughput and latency tests were conducted using a bundle payload size range from 1000 to 1e+06 bytes. Fragmentation was disabled for each of the throughput tests. We conducted 10 runs per bundle payload size and restarted the BPA daemon after each run to prevent previous runs from influencing the current test run. When using the DTNPerf utility to run throughput tests, we came across numerous issues that prevented us from using bundle payload sizes < 1000 bytes for ION, and < 100K bytes for IBRDTN. When using the dtnsource utility for SPINDLE throughput testing, we found that the BPA would crash when testing using a 1M bundle payload size. Upon further investigation, it was found that the SPINDLE BPA could only handle a max payload size of 270 Kilobytes before crashing. We also had to set *inter-bundle* send time to 10000 usecs to prevent the SPINDLE BPAs from locking up.

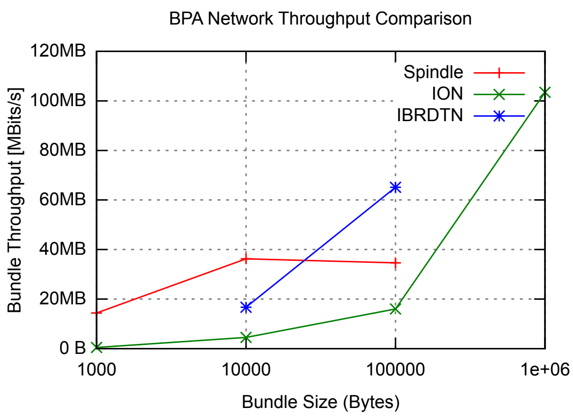


FIGURE 1: BPA THROUGHPUT

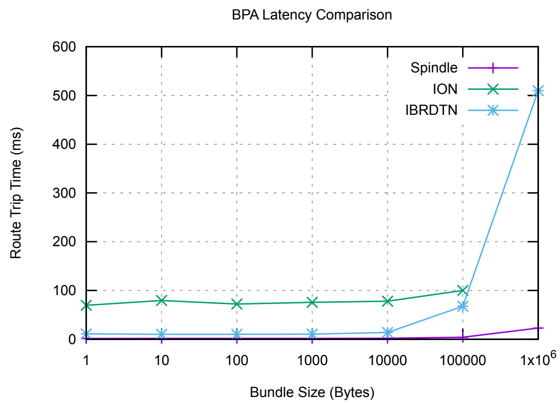


FIGURE 2: BPA LATENCY

Figure 1 illustrates the results of the throughput experiments. SPINDLE performed moderately better when testing using 1Kbyte and 10Kbyte size payloads (this could be due to the fact the UDP CLA was used, and UDP being a connectionless-based protocol, it was not constrained by per-hop TCP acknowledgements imposed by the TCP CLAs). For 100Kbyte tests, IBRDTN performed significantly better than SPINDLE and ION. As previously mentioned, we were unable to test using 1MByte bundle payload sizes for SPINDLE and ION, so to keep in accordance with the prescribed testing intervals (e.g. 1x10^{n} byte intervals) we stopped testing after 100Kbytes. Surprisingly, when testing ION using a 1MByte payload size, it was able to achieve a data rate over 100Mbits per second.

Figure 2 illustrates the results of our latency experiments. SPINDLE was able to consistently sustain the lowest round trip time (rtt), averaging in < 5 milliseconds, with the rtt slightly increasing (~23 ms) when testing using a 1Mbyte bundle payload size (again this could be due to the UDP CLA being for the SPINDLE tests). IBRDTN also performed well for the most part with an average rtt on the order of 10 milliseconds. The rtt appeared to increase significantly when testing using 100Kbyte and 1MByte payload sizes. ION performed rather poorly compared to SPINDLE and IBRDTN with an rtt range between 70 and 99 milliseconds.

Conclusions:

SPINDLE

Advantages

- Consistent data rate using various bundle payload sizes
- In non-disrupted network conditions, very low end-to-end latency through BPA overlay.
- A variety of routing protocols are included in the SPINDLE software distribution including static routing, Prioritized Epidemic (PREP), Anxiety Prone Link State (APLS) and PRoPHET.
- C++ API decently documented.
- Support for both absolute and relative timestamps

Disadvantages

- With the version of SPINDLE that was tested, the BPA will only work with the UDP CLA even though BPA has been configured to use static routing only (SPINDLE documentation dynamic routing must be disabled when using TCP CLA).
- SPINDLE's native file transfer utilites (dtncp/dtncpd and dtnsend/dtnrecv) appear to not be able to send files over 400Kbytes in size using SPINDLE's default configuration. Not apparent if there's a way to modify buffer sizes in config file to allow for a larger data pipe.
- Source code is closed.
- UDP CLA
- Only available for x86 platforms, no apparent support for ARM platform or other mobile-based processors.

ION

Advantages

- Software is open source and free. Updated regularly.
- Lightweight, takes up 750KB of memory
- Optimized for mobile devices, runs on ARM based devices such as the Raspberry Pi.
- Well documented C based API
- Support for both absolute and relative timestamps

Disadvantages

- In non-disrupted network conditions, fairly high end-to-end latency between through BPA overlay
- No neighbor discovery capability

- Only available dynamic routing protocol included in distribution (Contract Graph Routing) is is geared towards only handling scheduled contacts. No support for opportunistic contacts.
- When using dtnperf, tests won't complete when using bundle payload sizes 100 bytes or lower.

IBRDTN

Advantages:

- Although more data is needed (DTNPerf tests were only able to be run using 100Kbyte and 1Mbyte bundle payload sizes), data rates appear to increase as bundle payload size increases. When using a 1MByte bundle payload size, IBRDTN saw a two-fold data-reate increase over SPINDLE.
- A wider variety of routing options have been implemented and included in the IBRDTN software distribution (including static routing, Epidemic, Flooding and PRoPHET).
- Relatively lightweight, takes up ~6MB of memory.
- Software is open source and free. Updated semi-regularly
- Optimized for mobile devices, runs on ARM based devices such as the Raspberry Pi and Beaglebone.

Disadvantages

- Although socket based API well-documented, C++ API is not well-documented.
- Appears that IBRDTN only supports absolute timestamps. (not sure if this should be considered a disadvantage, but it doesn't have to the option to toggle between time modes like the other implementations do).
- When using dtnperf to test, will crash when testing with bundle payload sizes greater than 100K. Tests won't complete when using bundle payload sizes 1000 bytes or lower.

Recommendations for Further Research

We find that IBR-DTN is the most feature-complete, with relatively minor bug fixes needed to be ready for adoption. Therefore, we recommend that future efforts focus on completion of this BPA and integrating it into IP networks, such that IP applications can benefit from the DTN behaviors.

References

[1] Wolf Bastian Pottner et al, "An Empirical Performance Comparison of DTN Bundle Protocol Comparisons", 2011.

[2] Artemios G. Voyiatzis, "A Survey of Delay- and Disruption-Tolerant Networking Applications", June 2012.

[3] V. Cerf and S. Burleigh, "Delay Tolerant Network Architecture", 2007.

[4] K. Scott and S. Burleigh, "Bundle Protocol Specification", RFC5050, Nov. 2007.

[5] K. Fall, "A Delay Tolerant Networking Architecture for Challenged Internets", Proc. SIGCOMM 2003, Aug. 2003

[6] Justin P. Rohrer and Geoffrey G. Xie, "DTN Gateway Architecture for Partially Disconnected Telemetry Environments", In *Proceedings of the International Telemetering Conference (ITC)*, October, 2012.

[7] Justin P. Rohrer and Geoffrey G. Xie, "DTN Hybrid Networks for Vehicular Communications", In *Proceedings of the IEEE 2nd International Conference on Connected Vehicles*, December, 2013.

[8] Lance A. Alt, Justin P. Rohrer, and Geoffrey G. Xie, "Demo: Application-Transparent Deployment of DTN via SmartNet", In *Proceedings of the 9th ACM MobiCom workshop on Challenged Networks*, ACM, September, 2014, pp. 93-96.