Performance Characteristics of the Interplanetary Overlay Network in 10 Gbps Networks

A thesis presented to

the faculty of

the Russ College of Engineering and Technology of Ohio University

In partial fulfillment

of the requirements for the degree

Master of Science

John D. Huff

April 2021

This thesis titled

Performance Characteristics of the Interplanetary Overlay Network
in 10 Gbps Networks

by

JOHN D. HUFF

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

Shawn Ostermann
Associate Professor of Engineering and Technology

Mei Wei
Dean, Russ College of Engineering and Technology

Abstract

HUFF, JOHN D., M.S., April 2021, Computer Science

Performance Characteristics of the Interplanetary Overlay Network in 10 Gbps Networks (90 pp.)

Director of Thesis: Shawn Ostermann

The Interplanetary Internet (IPN) is an architecture for standardized communication between nodes located on or around different celestial bodies. The key concept of the IPN is to use standard Internet protocols within local high-bandwidth, low-latency networks and to interconnect these networks using an “interplanetary backbone” comprised of satellites and ground stations communicating using specialized protocols designed for use in low-bandwidth, high-latency networks. This thesis focuses on the performance within local networks constructed for use in an IPN setting. Delay Tolerant Networking (DTN) is a protocol designed to solve the challenges of IPN. This thesis studies the performance characteristics of the Interplanetary Overlay Network (ION), an implementation of the DTN protocol. A hardware test bench was constructed using two high-performance computers directly connected via a 10 Gbps link. A software tool was devised to test the throughput over this link under various configurations of ION. Through this testing, improvements to ION and configuration recommendations were found to increase the performance of ION in 10 Gbps networks. The main increases in performance were the result of locking the threads of ION to the same CPU core and increasing the shared memory allocation to convergence layer processes. Performance of ION was also studied on a test bench utilizing an ARM A53 processor which uses the same ARMv8 architecture used in the High Performance Spaceflight Computing architecture.

Acknowledgments

Thank you Dr. Shawn Ostermann for your guidance and support throughout this process. Your insight and patience have been invaluable. I could not have asked for a better advisor.

Thank you Mom and Dad for your unrelenting love and support. You kept me going and this would not have been possible without you.

Thank you Cary Roberts Frith for always believing in me.

Thank you Mehmet Adalier for your continued guidance. Our many discussions of ION surely improved this work and I look forward to many more to come.

Thank you Antara Teknik LLC and NASA for providing the hardware for our test benches.

Table of Contents

List of Tables

List of Figures

List of Acronyms

BP Bundle Protocol
CCSDS The Consultative Committee for Space Data Systems
CFDP CCSDS File Delivery Protocol
CRC Cyclical Redundancy Check
DTN Delay Tolerant Networking
HPSC High Performance Spaceflight Computing
ION Interplanetary Overlay Network
IPN Interplanetary Internet
JPL Jet Propulsion Laboratory
JSON JavaScript Object Notation
LTP Licklider Transmission Protocol
NASA National Aeronautics and Space Administration
PSM Personal Space Management
SCPS-TP Space Communications Protocol — Transport Protocol
SDR Simple Data Recorder
TCP Transmission Control Protocol
TCPCLA TCP Convergence Layer Adapter
UDP User Datagram Protocol
ZCO Zero-Copy Objects

1 Introduction

With the increased interest in space exploration in both government and commercial sectors, it is vital to develop internationally agreed upon standards for data and communications systems. These standards allow for interoperability among organizations and reduce the development and operating costs of missions. Delay tolerant networking (DTN)[1] research has gained much interest and has been set as a requirement for future missions and architectures such as the future Lunar Communications Architecture [2]. DTN, as described in RFC 4838 [1], seeks to address the issues of communicating in environments with long delays and intermittent disruptions. Utilizing DTN, the Lunar Communications Architecture will feature the Lunar space internet, an internetwork similar to the terrestrial Internet which will embody the Lunar relay network, the Lunar surface network, and the Earth network [2]. Having DTN as a requirement will simplify the process of expanding our space networks, with the eventual goal of creating a general purpose Interplanetary Internet.

The widespread adoption of DTN and its accompanying message protocol, Bundle Protocol (BP)[3], has lead to the creation of many DTN implementations [4]–[10]. This thesis will focus only on NASA's Interplanetary Overlay Network (ION), an implementation of DTN developed by the Jet Propulsion Laboratory (JPL) [4]. ION was designed from the ground up for spaceflight, with many measures taken to ensure reliability and security.

1.1 Motivation

The Laser Communications Relay Demonstration (LCRD) will showcase NASA's latest advancements in free-space optical (FSO) communication [11]. The LCRD payload will be carried by the U.S. Department of Defense's Space Test

Program Satellite 6 which is expected to launch some time in 2021. LCRD will have a downlink bandwidth of 1.2 Gbps, nearly double the bandwidth of NASA's previous FSO demonstration, the Lunar Laser Communications Demonstration (LLCD). LLCD demonstrated a downlink bandwidth of 622 Mbps from Lunar orbit to three Earth ground stations [12]. As one of the experiments performed by LLCD, DTN protocols were used to transfer files between the Earth ground stations and the LLCD Lunar orbiter [13]. This experiment was a success, with DTN helping to mitigate the disruptive effects of cloud coverage which can be detrimental to optical communication. This result paves the way for future experiments using DTN for high bandwidth space communication.

As high bandwidth space communications and high performance spaceflight systems become common, the development of fast and efficient communication software is vital. It is therefore of interest to study the performance characteristics of the ION in real systems capable of 10 Gbps throughput. Through rigorous and fine-grained benchmarking we are able to make conclusions about the aspects of ION that most affect throughput. With this insight we are able to make changes to the ION software and make more informed decisions concerning both software and hardware configurations to yield optimal performance.

1.2 Approach

We first created a hardware testbench with two node, three node, and five node configurations. We then created a benchmarking suite in order to automate the process of running thousands of throughput tests. Our benchmarking tool reads a test described in JSON format and performs throughput tests between nodes in the testbench. The tool runs through the permutations of test configurations, ION node configurations, and ION build configurations as described in the JSON formatted test file. Running all the permutations described by a test often took the tool several days. The high quality of data used to generate the figures in this thesis reveal subtle performance aspects of the ION software.

1.3 Document Structure

Chapter 2 gives background information on Delay Tolerant Networking, Bundle Protocol, and Interplanetary Overlay Network. Chapter 3 describes our experimental setup. This includes descriptions of our hardware testbenches, the ION configurations used, and the testing procedures. Chapter 4 shows the results and analyses for experiments. Chapter 5 summarizes our results and discusses possible improvements to ION.

2 Background and Literature Review

2.1 Delay Tolerant Networking

Delay Tolerant Networking (DTN) is an area of computer networking that addresses issues related to networks prone to long delays and disrupted communication between nodes [14]. With the development of standardized wireless communication protocols in the 1970s, the study mobile ad hoc networks (MANETs) became increasingly relevant. This is often viewed as a precursor to DTN research, which rose with the need for more sophisticated space networking solutions. DTN is especially important in space communication, where interplanetary distances cause latencies measured in minutes rather than milliseconds and end-to-end communication may be impossible due to celestial and orbital mechanics. Standard terrestrial Internet protocols such as TCP/IP are not designed for these types of networks, as delays and disruptions in terrestrial networks are typically seen as anomalies rather than integral aspects of the network [15]. Thus, DTN specific architectures and protocols are a necessity.

There are several examples of networking environments for which DTN architectures are ideal, including the following:

- Deep Space Communications — Deep space communication links are characterized by long delays, with round trip times often in the range of minutes or hours rather than milliseconds. These links are also prone to disruptions caused by orbits. A device on the surface of the far side of the moon may not have an end-to-end connection with Earth, instead it may need to use store-and-forward communication to first pass a message to a Lunar satellite.

- DTN of Things [14] — The Internet of Things (IoT) is a recently popular concept of creating an Internet-like network that connects objects. Typically, IoT networks are assumed to have end-to-end connectively. DTN of Things is the concept of networking objects using DTN to allow communication through multi-hop store-and-forward methods. An early example of a DTN of Things type of application was ZebraNet, which used a wireless sensor network and DTN to track the movement of wildlife in Kenya [16].

Standard Internet protocols such as TCP/IP [17][18] are not suited for delay/disruption prone networks. There are three main areas where terrestrial protocols fail in these networks: speed, resource usage, and routing. TCP performs poorly in environments characteristic of deep-space links [19][20][21]. In space communications, links are prone to bit-errors even with forward error correction. When TCP encounters a bit error in a packet, the entire packet is lost. Whenever TCP encounters a packet loss, it incorrectly assumes the loss is due to congestion rather than a bit error and reduces its congestion window. Handling packet loss due to corruption as loss due to congestion results in unnecessary reduction in bandwidth utilization [21].

There are a variety of reasons why standard Internet protocols perform poorly in delay/disruption prone networks. For example, TCP uses a three-way handshake to establish a connection. This wastes valuable time and lowers the overall bandwidth, as no data is being sent during this period. To achieve reliable packet delivery, TCP uses automated packet retransmission. Because TCP can only receive data in transmission order, a retransmission request may stop transmission for an entire round trip. Since TCP transmission is end-to-end, this could potentially mean several minutes of wasted time. Over lossy deep-space links this would be detrimental to bandwidth. TCP transmission being end-to-end also means that the sender will use massive of amounts of memory storing packets while waiting for acknowledgments.

Routing in disruption prone networks also proves to be a challenge for terrestrial protocols such as the Border Gateway Protocol (BGP) [22]. In a network with intermittent connectivity between nodes, there may be no instantaneous path between a sender and receiver. However, delivery may still be possible. For example, if a rover is on the surface of Mars facing away from Earth, it can still send a packet to a station on Earth by first sending it to a satellite. Then the satellite can store the packet until it orbits to the other side of Mars where it can make a connection to Earth. Standard routing protocols cannot take these temporal dependencies into account when making a route selection.

There have been several proposals to address these issues. The Consultative Committee for Space Data Systems (CCSDS) developed the Space Communications Protocol Specification (SCPS)—Transport Protocol (SCPS-TP), a modification of TCP with extensions to meet the environmental requirements of deep-space communication [20]. It has been shown that the pure rate control variation of SCPS-TP achieves over three times the throughput of TCP in a high-loss, cislunar-like environment [23]. The CCSCS also defines the CCSDS file delivery protocol (CFDP) standard[24]. CFDP adds store-and-forward capability to communication, allowing data to be relayed between spacecraft in disruption prone networks. SCPS-TP and CFDP work well for the specific links over which they are designed to operate, but are not ideal for use over every connection in an Interplanetary Internet which should use a variety of protocols to achieve maximum efficiency.

The Interplanetary Internet (IPN) is a proposed architecture for standardized communication between nodes located on or around different celestial bodies.

Though the original IPN research project was aimed at designing a communication network between Earth and Mars, the name “Interplanetary Internet” was coined “to suggest a far-future integration of space and terrestrial communications infrastructure to support the migration of human intelligence throughout the Solar System” [25]. The concept of the IPN is to use standard Internet protocols within local, high-bandwidth, low-latency networks and connect these networks to each other using an “interplanetary backbone” comprised of satellites and ground stations communicating using specialized protocols such as SCPS-TP and CFDP [25]. The goal of the IPN project is to standardize interplanetary communication protocols to facilitate widespread adoption. Vinton Cerf suggests that if all space agencies adopt these protocols, then after the completion of a spacecraft’s mission it can become repurposed as a node in this interplanetary backbone [26].

Delay/disruption tolerant networking (DTN) is one approach to IPN [27]. The development of a standard DTN architecture began in 1998 when Vinton Cerf met with the digital communications team at JPL in 1998 to propose the development of an Interplanetary Internet after he raised the possibility of IPN in a speech in Geneva the preceding year. Cerf secured funding from DARPA for a team at JPL led by Adrian Hooke to develop the IPN architecture and DTN protocols [28][29]. A standardized architecture for DTN was first defined in RFC 4838 in 2007 [1] and the DTN messaging protocol called Bundle Protocol (BP) was defined in RFC 5050 shortly thereafter [3]. This standardization has led to the widespread use of DTN and development of many implementations which include but are not limited to the following list:

- Interplanetary Overlay Network (ION): NASA/JPL’s implementation designed for spaceflight [4]

- IBR-DTN: An efficient implementation for embedded systems [5]
- DTN2: DTN reference implementation [6]
- DTN7: An open-Source disruption-tolerant networking implementation of Bundle Protocol version 7 [7]
- Terra: A lightweight and modular DTN library [8]
- Postellation: an enhanced delay-tolerant network implementation with video streaming and automated network attachment [9]
- µD3TN (formerly µPCN): A bundle protocol implementation for microcontrollers [10]

DTN acts as an "Internet-independent middleware". It uses an overlay protocol called Bundle Protocol (BP)[3] to interface with applications while using the most effective transport protocol when communicating between nodes. In the network stack, BP sits between the application layer and a transport layer such TCP. BP is designed to avoid the typical query/response communication between client and server. Instead, BP bundles messages together, sending all required data that the server may need at the same time. These bundled messages are referred to as bundles, and are the basic units of communication between DTN nodes [27]. BP is designed around the following principles:

- Regionality: BP uses the optimal transport protocol for a given region of communication [27]. For example, nodes operating over Internet-like networks use TCP while long range satellites may use LTP. This improves throughput, routing efficiency, and congestion control.

- Terseness: BP bundles use a very small header compared to TCP packets[27]. This reduction in overhead is especially important for spaceflight communication, as bandwidth is limited.

- Store and forward: BP uses store and forward communication rather than end-to-end [27]. This means that when a bundle is sent from node A to C by routing through node B, custody of that bundle is given to B and a copy of the bundle is stored on B. A custody signal is sent back to A allowing it to release that bundle from memory. This both reduces memory usage and potentially increases bandwidth. In a end-to-end protocol, if a packet is dropped between nodes A and C it has to be retransmitted all the way from A. With store and forward, the packet only has to be retransmitted from B. If the delays between nodes are long, this greatly increases bandwidth.

2.2 Interplanetary Overlay Network

Interplanetary Overlay Network is an implementation of the Bundle Protocol that is designed from the ground up for spaceflight [4]. There are a few key elements of ION that make it particularly suited for spaceflight:

- Pre-allocated memory for dynamic memory management — For safety reasons, spaceflight software should not make calls to `malloc()` or `free()`. A memory leak or incorrect usage of `free()` could be disastrous. To avoid this, ION statically manages its own memory which it allocates on startup.

- Fault tolerance — ION is able to recover from a complete shutdown or a shutdown of any one of its components. It does this using its Simple Data Recorder (SDR). The SDR is a shared memory database/memory manager that can use filesystem memory for persistence. It implements atomic transactions (meaning that all actions in the transactions take place or none of them do) which ensures that the database never enters an invalid state. The SDR is the core component of ION that provides its reliability, but it may also be preventing ION from reaching higher throughput. All of ION's threads share the same SDR, and only one thread can make a transaction at once. This often causes ION to become effectively single-threaded.

- Lightweight — ION is written in C and has a fairly small codebase, making it both fast and memory efficient.
- Contact Graph Routing — ION implements Contact Graph Routing, a method of calculating the best route for a packet in a network with intermittent connectivity. This is important for spaceflight because end-to-end communication may not be possible due to obstructions from celestial objects.

2.3 High Performance Spaceflight Computing

High Performance Spaceflight Computing (HPSC) is a collaborative project for developing next-generation spaceflight computing hardware [30]. The project is managed by the Jet Propulsion Laboratory with collaborators including NASA's Goddard Space Flight Center (GSFC), NASA's Johnson Space Center (JSC), and the United States Air Force Research Laboratory (AFRL). The technology used in spaceflight hardware generally lags behind state-of-the-art due to the rigorous testing and safety standards required by missions. The proposed design of this new architecture will enable several new possibilities for future missions previously infeasible to limited processing power. In a 2003 study commissioned by the NASA Game Changing Development Program, a series of workshops were held in order to identify use cases of HPSC. Mission designers, scientists, and engineers from JPL, GSFC, JSC, NASA Ames Research Center, and NASA Kennedy Space Center identified nineteen generic applications of HPSC to be used through 2025 [31]. These applications included those for both human and robotic uses, categorized by the Human Exploration Mission Operations Directorate (HEOMD) and the Science Mission Directorate (SMD). A few examples of possible applications of HPSC identified by this study include extreme terrain landing, telerobotic construction, hyperspectral imaging, and telepresence.

3 Experiment Setup

3.1 Hardware

Tests were performed on two distinct testbeds: one to represent ideal conditions and one to represent constrained conditions. While this thesis focuses mainly on ION's performance in ideal conditions, it is useful to explore ION's performance on less powerful hardware. Having a testbench with lower computing requirements also allows for running tests in a network of five machines rather than directly between two machines, as it was less feasible to obtain and connect many high performance machines. In all test cases, only one ION node is being run on each machine. This is done for two reasons. First, running multiple nodes on the same machine may reduce the performance of those nodes due to reaching the limit of the computing resources available. Second, throughput results between nodes running on the same machine are not meaningful as one of the main factors affecting performance is the time to read and write to the network interface. Even if the IP addresses used on each node are assigned to separate interfaces on the same machine with an Ethernet cable connecting the two interfaces, Linux will detect that the addresses are on the same machine and bypass the physical network interface.

3.1.0.1 High Performance Testbench

The high performance testbench (HP-testbench) represents running ION in ideal conditions. This testbench used two nodes running on separate physical machines. These machines communicate with each other via 10 Gbps interfaces directly connected to each other with an Ethernet cable. Figure 3.1 shows the network diagram for the high-speed two node configuration (HP-2). Using the iPerf [32] utility, the measured maximum goodput using both TCP and UDP is 9.41 Gbps. This is less than the rated 10 Gbps capability of the network interfaces

Figure 3.1: HP-Testbench network diagram (HP-2 configuration)

Two high performance machines connected directly via a 10 Gbps link. Both machines use a x540-AT2 Ethernet controller. The maximum goodput using TCP as measured by iPerf is 9.41 Gbps.

because iPerf is measuring goodput rather than throughput. That is to say, it is measuring the rate of useful data transfer, which does not include headers such as the TCP, IP, and Ethernet Frame headers. For each Ethernet frame, 1518 bytes are sent. 18 bytes are for the Ethernet frame header, 24 bytes are for the IP header, and at least 20 bytes are used for the TCP header. This results in at most 1456 useful bytes sent per 1518 bytes, giving a maximum of 95.9% bandwidth utilization, so a measured bandwidth of 9.41 Gbps (94.1% bandwidth utilization) is not unreasonable.

The purpose of the experiments on this testbench is to test the performance of ION in ideal conditions. As such, the computing and network hardware were selected to be capable of high performance. While this scenario is atypical from what is considered a normal use case of ION, it is representative of a possible future trunk in an Interplanetary Internet or any other large network using DTN.

Ideally the two computers in this testbench would be identical, and during the development of the benchmarks that was the case. However, in our final testing the two high performance machines were different, resulting in slightly asymmetrical goodput [1] using ION depending on the transfer direction of the test. This gave the benefit of observing the effect of more factors such as clock speed and cache size.

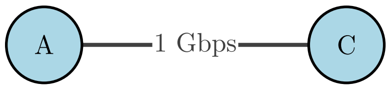
3.1.0.2 Low Performance Testbench

The low performance (LP-testbench) represents running ION in more realistic conditions, specifically running one of the five nodes on a ESPRESSObin v7 which uses a Marvell Armada 3700LP (88F3720) dual-core ARM Cortex A53 which has the same ARMv8 architecture used in the HPSC chiplet architecture [33]. Depending on the configuration, the LP-testbench uses up to five ION nodes with each node running on a separate machine (see appendix A.1 for hardware specifications). The machines are on the same subnet and are connected on the same 1 Gbps Ethernet switch, however the ION configurations are such that the nodes must communicate though a multi-hop path, using the ARM machine as a sort of router (See appendix A.2 for the “ipnadmin” utility “group” commands defining the route table for each node). There are three configurations of this testbench; Figure 3.2a shows the two node configuration (LP-2) network diagram, Figure 3.2b shows the three node configuration (LP-3) network diagram, and Figure 3.2c shows the five node configuration (LP-5) network diagram.

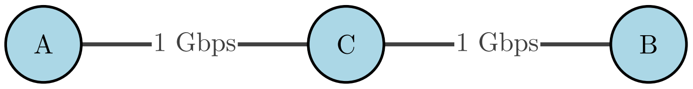
3.1.1 Operating System and Network Stack Considerations

All machines used in our tests are running the CentOS 7 distribution of Linux. For any operating system, there are several TCP/IP parameters that are known to affect network throughput. These parameters include send and receive buffer sizes, TCP window size, and Ethernet frame size. It has been shown that properly tuning these parameters can lead to a significant increase in performance [34]. For our

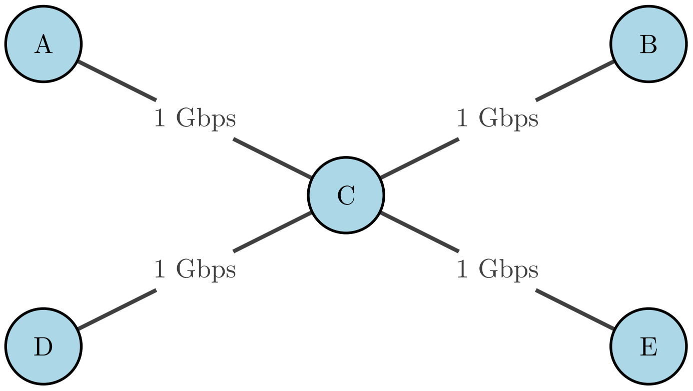
[1] Goodput is the rate of transfer of useful data, whereas throughput includes the transfer of both data and headers. Throughput also includes additional overhead such retransmitted packets.



(a) LP-2 configuration: Two node configuration, connecting an x86 machine and ARM machine



(b) LP-3 configuration: Three node linear configuration, using ARM machine as center node.



(c) LP-5 configuration: Five node configuration, using ARM machine as a sort of router through which all other nodes must communicate.

Figure 3.2: LP-Testbench configurations

All machines are connected to each other over a single 1 Gbps Ethernet switch.

experiments we increased the max receive and send buffer size, enabled TCP selective acknowledgments, and increased the maximum TCP window size by enabling TCP window scaling. These settings are changed by editing the /etc/sysctl.conf file in Linux. The values we used for these settings are shown in appendix A.3. We did not use jumbo Ethernet frames, as the standard 1500 byte frames are more commonly used. The effect of increasing the frame size could be explored in future experiments.

As an additional step to reduce the number of factors affecting performance, the nodes were run on ramdisks. Using the tmpfs utility, each node was run in a filesystem running completely in ram, bypassing the need to read and write to the physical storage device. In some cases, ION is limited by the read and write speed of the storage device. For example, in our CFDP test, throughput is measured by sending a 1 GB file. Storing this file in RAM eliminates the bottleneck of the read speed of the disk.

3.2 Throughput Measurement Method

To measure the throughput of ION, we used a modified version of iPerf3[32] called *tara*BPPerf [2]. *tara*BPPerf works in the same manner as iPerf; A *tara*BPPerf server is started on one machine and a *tara*BPPerf client on different machine connects to that server. The client sends data to the server for a specified amount of time or until a specified amount of data is sent, then the throughput is calculated. In our tests, the throughput was measured by sending data for 10 seconds, and averaging the throughput over the last 5 seconds of that period. This avoids any decrease in average throughput due to TCP slow start and instead measures the

[2] *tara*BPPerf is an internal tool I developed while working for my current employer, Antara Teknik LLC. Antara Teknik is currently evaluating the potential of making *tara*BPPerf open source. Similar goodput results can be achieved by using the “bpdriver” and “bpcounter” utilities in the ION codebase.

average throughput once steady state is reached. After 5 seconds the congestion window is filled and the send receives zero-window packets from the receiver at a constant rate. In this case, 5 seconds is plenty of time to reach steady state as the bandwidth-delay product between the two nodes is very low because they are either directly connected via a Ethernet cable or directly connected to the same Ethernet switch.

3.3 ION configuration

All tests were performed using the same baseline ION configurations with only the variables being tested for being modified for each test as described. This baseline is meant to maximize performance and control for factors outside of those being specifically tested. The full baseline configurations are given in appendix A.2 and follow these guidelines:

- All nodes use the TCP convergence layer adapter. The most common convergence layer adapters for ION are TCP, UDP, and LTP (Licklider Transmission Protocol). The TCPCLA was chosen as it introduces the least overhead.
- All threads for all ION processes are run on the same core. Section 4.3 discusses how this increases performance and produces more consistent throughput results. Without this option it would be up to the operating system's scheduler to determine where to place the threads, leading to significant variability in each test even with all other variables being the same. Note that this setting is not reflected in the configurations shown in appendix A.2 as it is not a standard ION option. Rather, the taskset utility is used to set the CPU affinity of the ION processes on launch.

- Heapmax is set to a value larger than the bundle size. Heapmax is a variable in ION which defines how many bytes of a bundle can be stored in the SDR heap. If a bundle is larger than heapmax, heapmax bytes of the bundle are stored in the SDR and the rest of the bundle is stored as a file in the operating system.
- TCP convergence layer buffer size is set to a value larger than the bundle size. This buffer is used when copying a bundle to the network interface, so setting it larger than the bundle size allows the bundle to be copied to the network interface with a single system call.

4 Experiments, Results and Analysis

The results of a particular goodput test can vary significantly run to run. In order to maintain the statistical significance of our results we repeated each test 20 times. The data in the graphs in this section show the averages of those tests. Additionally (unless otherwise noted), we ran our tests with ION locked to a single core. This gives much more consistent results as the CPU cores the threads run on have a large impact on goodput, and leaving the threads unlocked gives the operating system's scheduler authority over CPU assignment. In section 4.1, TCPCLA buffer size is left to its default of 64 KiB[3]. In all other sections (unless otherwise noted), a TCPCLA buffer size of 512 KB is used.

In each graph, all data are plotted as both a scatter plot and a line representing the mean. Additionally, the grey area behind each line represents a 95% confidence interval calculated using a t-distribution. It should be noted that in some cases this type of t-distribution analysis may not be completely appropriate as the data is not normally distributed. However, showing the confidence interval still does give some visual indication of how the data is distributed and is therefore included in the graphs.

Our experiments cover a wide range of factors. Each line in our graphs represents the result of changing only one of these factors. The factors we tested include are shown in Table 4.1. Note that we did not test all permutations of these factors. Refer to the caption of each graph for a description of the factors being tested. The parameters of our experiments are shown Table 4.2. These are factors that are kept constant throughout all of our experiments.

[3] In this work, we use both "KB" and "KiB" as units. We use KB as meaning 1000 bytes and KiB as meaning 1024 bytes.

Table 4.1: Experimental Design Factors

Factor	Values
Node configurations	HP-2, LP-2, LP-3, LP-5
Bundle size (number of bytes in each bundle payload)	1 KB – 2000 KB
TCPCLA buffer size	64 KiB – 512 KB
All ION threads locked to same core (CPU Affinity)	True, False
Number of data streams	1, 2
Transfer method	BP, CFDP over BP
Segment size (when using CFDP)	1 KB – 64 KB
Checksum algorithm (when using CFDP)	Null, Modular, CRC32, CRC32 (unoptimized), CRC32-slice-16

Table 4.2: Experimental Design Parameters

Parameter	Value
ION Version	3.7
Convergence Layer Adapter	TCPCLA
Heapmax	2100 KB
*tara*BPPerf test time	10 seconds
CFDP test file size	1 GB

4.1 Bundle Size

In many studies of DTN implementations, it is common to test the effect of bundle size on goodput [35][36]. In networks with high bit-error rate, large bundles cause more retransmissions and therefore lower throughput (Lent). However, in networks with low or no bit errors, large bundles tend to increase performance due to lower per-bundle overhead.

To test the effect of bundle size on ION's performance, the goodput between nodes A and B was measured using several bundle sizes. Figure 4.1 shows the results of this experiment for the HP-2 configuration. In this case, the tests were conducted using Node A as the sender and Node B as the receiver, and then the tests were repeated using Node B as the sender and Node A as the receiver. This was done to demonstrate the effect of computing hardware on goodput. On both machines, ION is run with all processes and threads on a single core (section 4.3 analyzes the effect of thread affinity). The results show greater goodput is achieved using Node B as the receiver. The ability of the receiving node to process incoming packets is the limiting factor of goodput, as the receiving node is consistently sending zero window advertisements, meaning that the receive buffer is full and the sender is sending bundles faster than the receiver can process them.

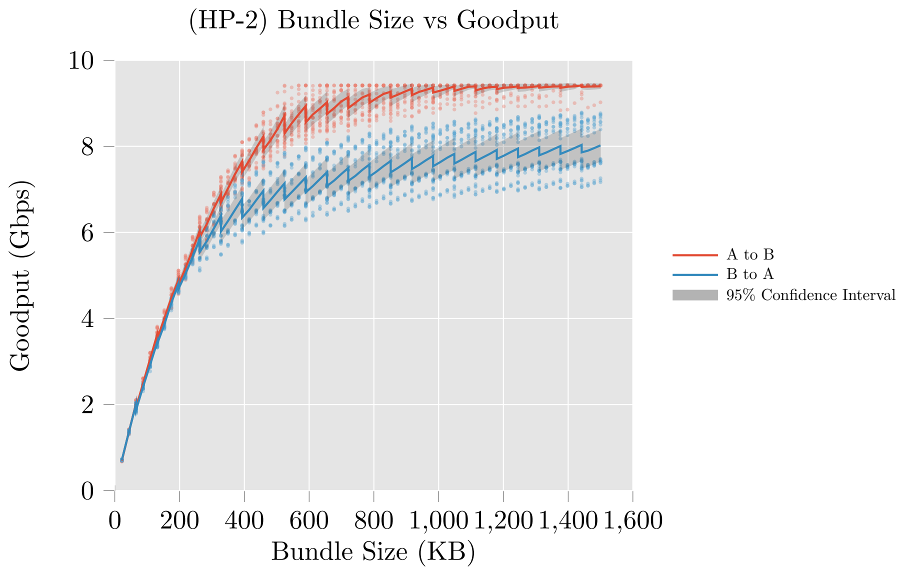


Figure 4.1: (HP-2) Bundle Size vs goodput for transfers from A to B and B to A

Goodput increases with bundle size with diminishing returns. The goodput reaches the maximum throughput of 9.4 Gbps when transferring from node A to node B. However, the transfer of B to A does not not saturate the line. This is likely due to Machine A having a smaller cache.

It was unexpected that using node B as the receiver resulted in higher goodput when running nodes on a single CPU. Machine A has a significantly higher core clock speed than Machine B (3.50 Ghz and 2.60 Ghz respectively), and a higher RAM clock speed (3200 Mhz abd 2133 Mhz respectively). However, Machine B has a larger L3 cache than Machine A (30720 KB and 8192 KB). As bundles are received, the data contained in the bundles replaces the cached memory pertaining to the state of IONs processes. With a larger cache this displacement happens less often, meaning fewer cache misses and therefore greater goodput.

The lines in Figure 4.1 appear jagged, sharply dropping every time the bundle size passes a multiple of 65536 bytes (64 KiB). This is due the TCP convergence layer using a buffer of size 64 KiB to read bundle data from the socket. Every time the bundle size passes a multiple of 64 KiB bytes, the convergence layer has to read into this buffer an extra time. This effect is discussed in section 4.2, but generally a larger TCPCLA buffer size results in greater throughput, especially at large bundle sizes.

Figure 4.3 shows similar results in the LP-2 testbench. Transferring from Node C to Node A is faster because Machine A has more computing power, and transfers are bottlenecked by the receiver when both machines are the same. In this case, the bottleneck is coming from the sender, Node C, because we know that Node A is capable of receiving much faster than what was measured in this test. Figure 4.4 shows results for a three node transfer in the LP-3 testbench. This is much slower than the LP-2 and HP-2 testbench because the Node C has to both send and receive.

There are a variety of factors that are responsible for goodput increasing as bundle size increases, all of which are based on the fact that larger bundle sizes mean fewer bundles sent for the same goodput. With fewer bundles there is less computing overhead which results in greater goodput. For each bundle, there is a constant time overhead and linear time overhead. An example of a constant time overhead is locking a semaphore, which takes the same amount of time for any bundle size. An example of a linear time overhead is the time it takes to copy the bundle data into the network interface, which scales approximately linearly with bundle size. Any process that introduces both a constant time overhead and linear time overhead will result in a graph with the type of curve seen in our results. The work rate achieved by this type of process is calculated by the equation:

$$R(w) = \frac{w}{O_c + O_l \cdot w} \tag{4.1}$$

where O_c is the constant time overhead in seconds per load, O_l is the linear time overhead in seconds per work, and w is the work per load (such as bundle size). Without the constant time overhead the rate would always be the same, and without the linear time overhead the rate would scale linearly with the work per load. With both types of overhead, a curve with asymptotic behavior is produced. Figure 4.2 shows this curve plotted with O_c of 2 and O_l of 1.

Sending fewer bundles requires less interaction with SDR. When a bundle is created, it is stored in the SDR as a zero-copy object. Because the SDR has to manage a preallocated area of memory, storing an object can be costly depending on the internal memory allocation algorithm used. For large objects, the SDR uses sequential fit allocation. This algorithm has linear allocation and deallocation time complexity, so for a large number of objects it becomes inefficient. However, in these tests the bundles are only stored for a short period of time, so the number of objects is fairly small. Additionally, any transaction with the SDR, whether it be reading or writing, requires the entire SDR to be locked. This is a major hindrance when running ION threads across multiple cores as it introduces a large source of thread

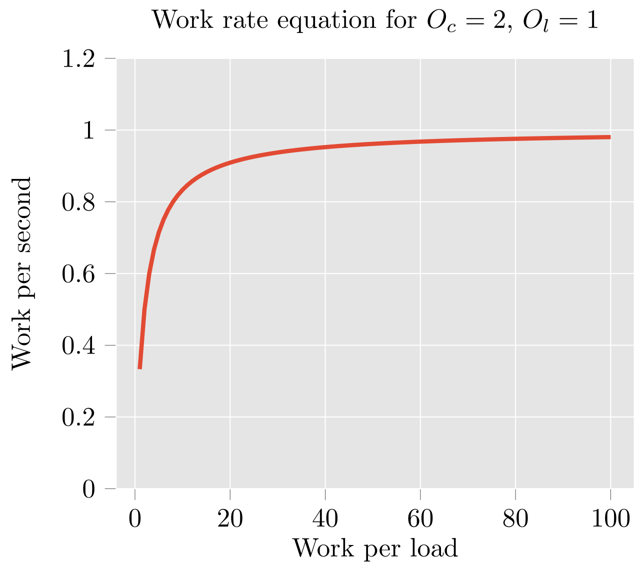


Figure 4.2: Example work rate equation graph

Graphing the work rate equation with $O_c = 2$ and $O_l = 1$. This is a simple model of how bundle size affects goodput. With only a constant time overhead per bundle, goodput would increase linearly with bundle size. With only a linear time overhead per byte sent, the goodput would be constant. Including both of these factors produces a graph which increases in an asymptotic manner. Reducing the constant time overhead would reduce the bundle size to approach the asymptote, and reducing the linear time overhead would increase the asymptote (maximum possible goodput for any sized bundle).

contention. For the tests shown in this section, this is not a factor as the threads are all being run on the same core. Regardless, the process of locking and unlocking is costly as it requires interacting with the kernel.

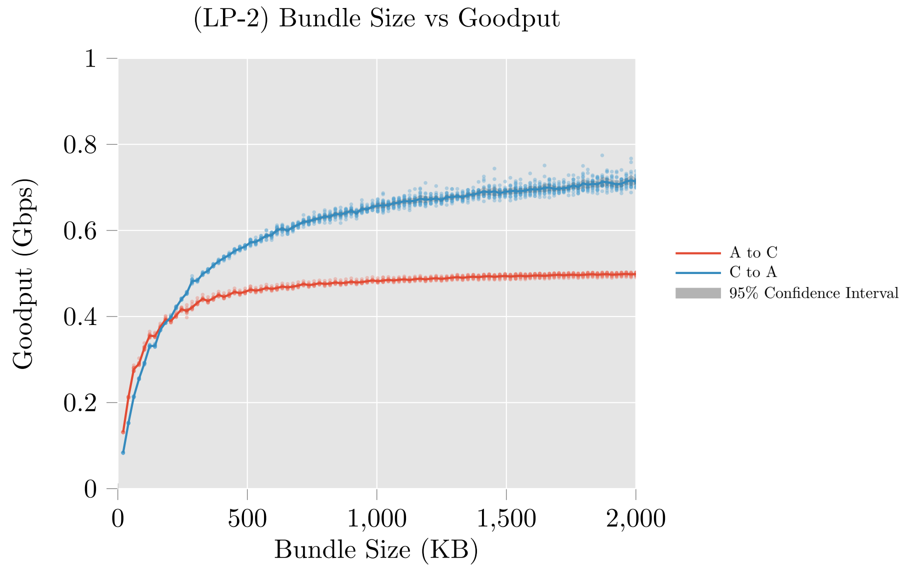


Figure 4.3: (LP-2) Bundle Size vs goodput for transfers from A to C and C to A

Goodput increases with bundle size in the LP-2 testbench. As expected, transfers from C to A are faster than A to C because on the same hardware, receiving a bundle takes longer than sending a bundle. Because we know A is able to receive much faster than the maximum of 716 Mbps C to A transfer shown in this graph, the bottleneck in this case is how fast Node C can send data.

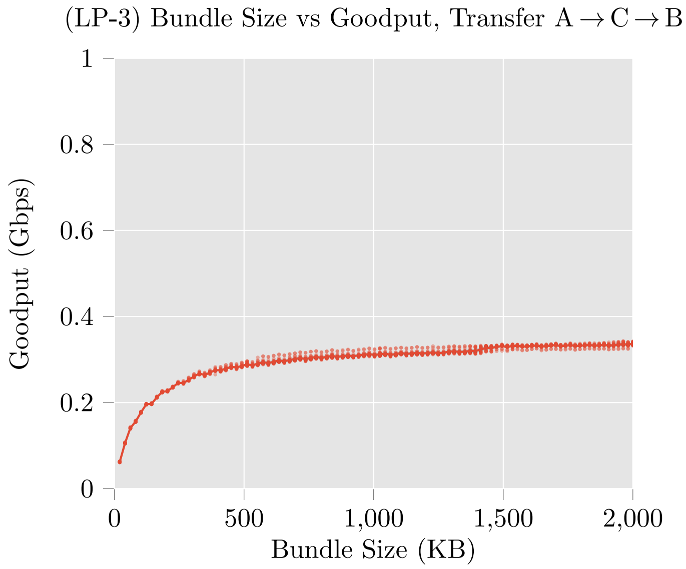


Figure 4.4: (LP-3) Bundle Size vs goodput for transfers from A to B, routing through C

This graph shows the results from a multi-hop test, transferring from Node A to Node C and then to Node B, where Node C is the ARMv8 machine. As expected, this testbench had the slowest goodput with a maximum of 337 Mbps at a bundle size of 2000 KB.

4.2 TCPCLA Buffer Size

A major factor affecting ION's performance is the size of the buffers used in the TCP convergence layer adapter. These buffers are used to read bundles to and from the TCP socket. As the buffer increases in size, the number of system calls decreases. This decreases the time needed to read bundles from the socket and therefore increases goodput. Figure 4.5 shows the effect of buffer size on goodput at several bundle sizes using the HP-testbench, each line representing a different bundle size. Goodput increases as the buffer size is increased, especially when using large bundle sizes. Figure 4.6 shows similar results in the LP-2 testbench.

In both HP-2 and LP-2, goodput stopped increasing once the TCPCLA buffer size reached somewhere between 500 KB and 1000 KB. Figure 4.7a and Figure 4.7b show goodput vs bundle size for a buffer size of 512 KB compared to the default buffer size of 64 KiB. With bundle sizes above 64 KiB, there was a significant increase in goodput using a buffer size of 512 KB.

In modern operating systems, data is pushed to the outbound queue (SendQ) using a system call [37]. This system call typically takes data passed in a buffer and copies it to the SendQ. Similarly, a system call is made to copy data from the inbound queue (RecvQ) into a buffer. The size of the queues and the size of the buffers determine the amount of data that can be copied at one time. It is much more efficient (requires fewer system calls) to copy a single buffer of size n to or from a queue than it is to copy n buffers of size 1. Goodput increases with buffer size until the buffer becomes larger than the queue. In this case, the system reads enough data from the buffer to fill the SendQ and waits for the data to be cleared from the queue before reading more data from the buffer. This process of waiting and reading has almost identical cost to making multiple system calls as a multiple context switches occurs during each read cycle [37].

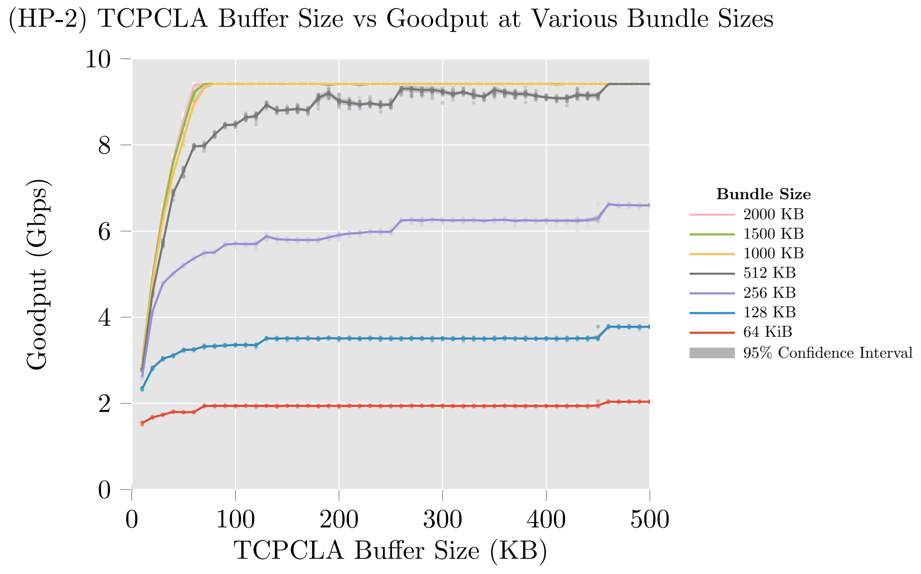


Figure 4.5: (HP-2) TCPCLA buffer size vs goodput for various bundle sizes, A → B

Increasing the TCPCLA buffer increases goodput for all of the seven bundle sizes tested when transferring from Node A to Node B. Once the buffer size is larger than the bundle size, there should no longer be any gain in performance for increasing the buffer size further, as the number of socket reads required has already been reduced to a single call. However, there is an anomalous bump for the 64 KiB, 128 KB and 256 KB bundles at a buffer size of about 450 KB. In order to choose a new default value for the TCPCLA buffer size, the minimum buffer size at which there is no longer an increase in goodput for any bundle size is selected. This graph shows that goodput only increases slightly after about 300 KB.

Increasing these buffer sizes has a larger impact when copying a large amount of data at a time. In ION's case, the maximum amount of data copied at once is determined by the size of the bundle. Therefore, the best performance is achieved when sending and receiving large bundles (>1 MB). It is also important to have the

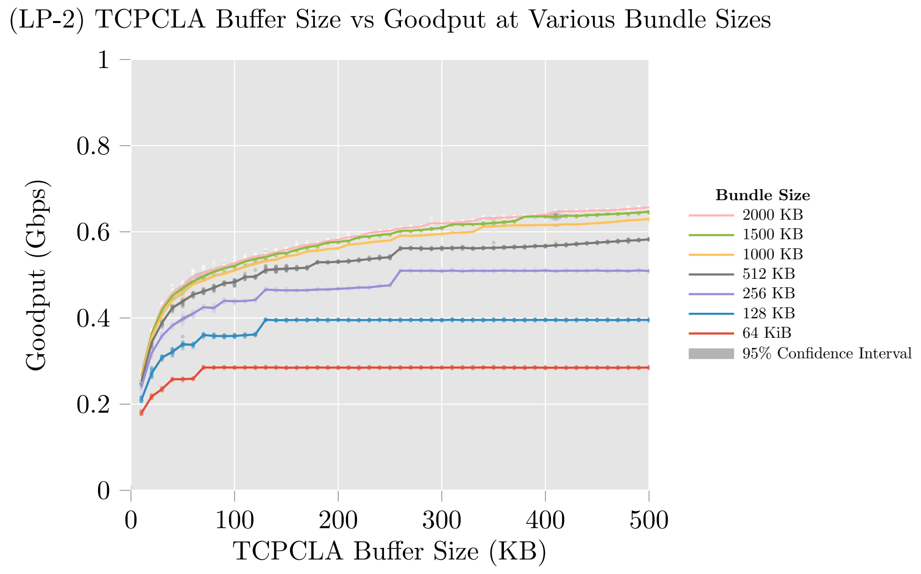
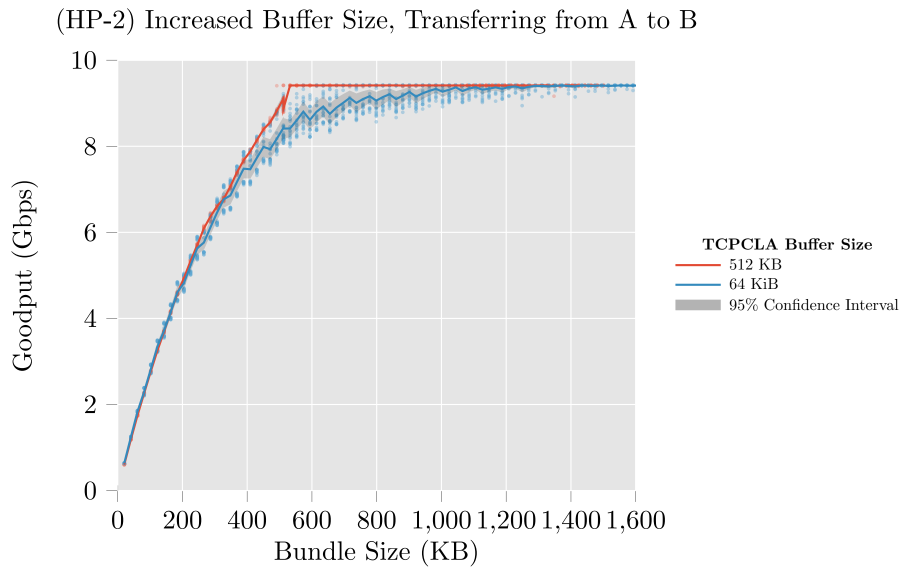


Figure 4.6: (LP-2) TCPCLA buffer size vs goodput for various bundle sizes, A → C

The TCPCLA buffer size test was repeated on the LP-2 testbench. These results show goodput when transferring from A to C. Unlike the results from the HP-2 testbench, the goodput for large bundle sizes does not saturate the line, and instead an asymptotic increase in goodput is displayed as TCPCLA buffer size is increased. At a buffer size of 500 KB the asymptotic increase is low, but there is still slightly more goodput to be gained than what is shown in this graph

size of the bundles be the same size as the buffers or a multiple of the size of the buffers. If the bundle is larger than the buffer size, the bundle will be copied to the buffer and sent to the queue in multiple chunks, requiring multiple system calls. If the bundle size is larger than the buffer size and not a multiple of the queue size, the last chunk to be copied to the buffer will be smaller than the previous chunks



(a) (HP-2) TCPCLA bundle size vs goodput for TCPCLA buffer size of 64 KiB and 512 KB, A → B

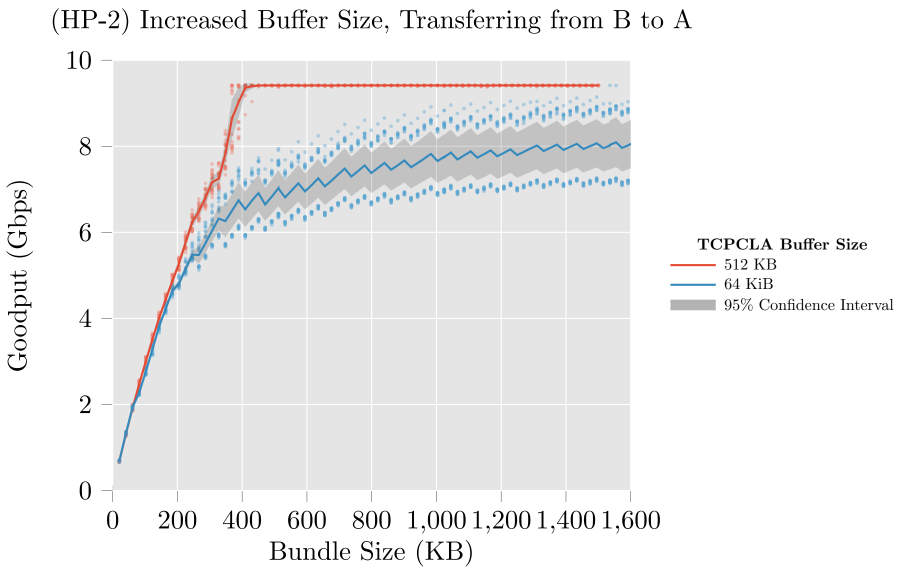
Increasing the TCPCLA buffer size from 64 KiB to 512 KB increases goodput. These results shows transfers from Node A to Node B. In this case, there wasn't much goodput to be gained as the 10 Gbps line was saturated at bundle sizes larger than 1 MB. For discussion on the bimodal distribution of the 64 KiB results see section 4.3.1.

Figure continued on next page

(which are the same size as the buffer). If this final chunk is significantly smaller, performance is decreased. For example, if the bundle size is $S + 1$ where S is the size of the buffer, the process of copying it to the socket would be as follows:

1. Copy a bundle chunk of size S to the buffer.
2. Make a system call to write S bytes of the buffer to the socket.
3. Copy a bundle chunk of size 1 to the buffer.
4. Make a system call to write 1 byte of the buffer to the socket.

Writing this bundle takes two system calls rather than one, while only copying an extra byte compared to if the bundle were size S. While the system call for



(b) (HP-2) TCPCLA bundle size vs goodput for TCPCLA buffer size of 64 KiB and 512 KB, B → A

Increasing the TCPCLA buffer size from 64 KiB to 512 KB increases goodput significantly. These results shows transfers from Node B to Node A. In this case, using a buffer size of 64 KiB did not saturate the 10 Gbps line. After increasing the TCPCLA buffer size to 512 KB, the 10 Gbps line was saturated with bundles of size 400 KB and above.

Figure 4.7: (HP-2) TCPCLA bundle size vs goodput for TCPCLA buffer size of 64 KiB and 512 KB, A → B and B → A

Increasing the TCPCLA buffer size from 64 KiB to 512 KB increases goodput, especially when transferring from node B to node A. The blue line shows the goodput when using a 64 KiB buffer and clearly displays a sawtooth behavior. Goodput drops slightly every time the bundle sizes crosses a multiple of the buffer size. This is because once the bundle size increases past a multiple of the buffer size, and additional call to the socket read function is required. These additional calls are accumulated as bundle sizes increases, which is what causes the overall slowdown.

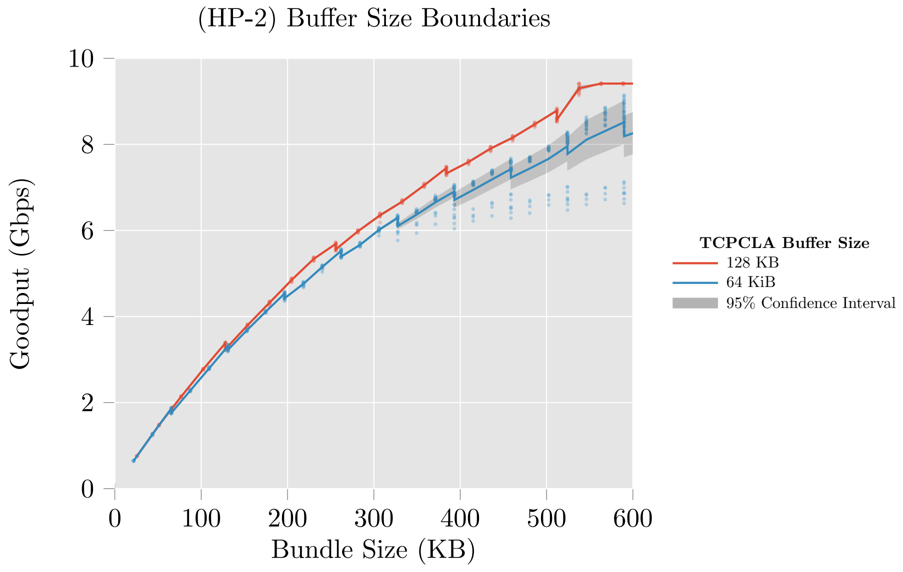


Figure 4.8: (HP-2) Buffer Size Boundaries

Goodput drops every time the bundle size overflows the buffer size. Passing a buffer size boundary causes an extra socket read/write system call to be made and thus adds extra overhead, leading to a slight decrease in Goodput.

copying the single byte takes less time than copying the S size chunk to the socket, the fixed overhead for making the extra system call adds a significant amount of processing time and therefore lowers goodput. Figure 4.8 demonstrates the drop in performance at these buffer size boundaries.

4.3 Processor Affinity

The fundamental principle of ION's architecture design is distributed processing. Rather than running a single complex daemon, ION is broken up into many smaller components running concurrently. These components run tasks that are chained together to execute the full ION pipeline. As outlined in the Design and Operation manual included with the ION software package [38], a few of the benefits of this design choice are the following:

- Maintaining a codebase of logically coherent, small components is more manageable. Resolving conflicts across the codebase is easier, especially considering the multi-organization development environment of the ION project. Additionally, the relative simplicity of the components reduces the introduction of defects.

- Individual components of an ION node can be shut down or introduced during runtime. This allows the complexity and scale of a node to be changed without the need of restarting the node with a new configuration.

- Clear interfaces between components simplifies flow control and reduces the risk of excessive resource consumption.

ION was primarily designed for computers used in spaceflight. These typically use lower power processors with a single core, meaning that the concurrency of ION's components is only logical; tasks are never executed at the same time. It is reasonable to assume that running ION on a multi-core processor would improve performance; tasks could truly run concurrently and therefore the pipeline would process data faster. However, that is not always the case.

When the processes for ION components are launched, the operating system uses a scheduler to control which CPU to run the threads for each process on. Since ION runs many threads due to its distributed processing design, it is almost certain that the threads will run on different CPUs. Schedulers can also change what CPU a thread is being run on during runtime. By manually setting the processor affinity for a thread, a thread is guaranteed to run on a specific CPU or set of CPUs.

To test the effect of CPU affinity, we ran goodput tests with ION's threads being locked to the same core (single-core) and compared the results to when the threads are not locked to any core (multi-core). Appendix A.4 describes how the "taskset" command is used to launch ION nodes in single-core mode. Figure 4.9 shows the results of these tests for the HP-2 testbench. Transfers from A to B were tested as well as transfers from B to A. In both single-core and multi-core tests, transfers from A to B had higher goodput, similar to the results seen in Figure 4.1. For most bundle sizes, single-core performance is significantly higher than multi-core performance. This is because ION is often effectively single-threaded due to only a single thread being able to access the shared database at a time. Therefore, locking the threads to a single core has minimal performance loss due to reduced concurrency, and instead gets a performance boost due to less cache-misses. When the threads are run on the same core, keeping the cache coherent is faster because less data needs to be copied to the L3 cache.

In transfers from Node B to Node A, single-core performance was significantly higher for most bundle sizes. However, using bundle sizes above 1700KB had higher goodput while running on multiple cores. It seems that the benefit of lower cache misses when using single-core become less relevant at larger bundle sizes when receiving on machines with smaller L3 caches. In a similar vein, receiving on the ARMv8 machines performs better on multi-core as seen in Figure 4.10. The ARMv8 machine similarly does not have as robust of a cache as Machine B.

To store and pass data between components, ION uses a shared object database called Simple Data Recorder (SDR) which runs in shared memory. This gives access to the same data to every thread of every process in an ION node. Reading and writing to shared memory has little performance cost while using a single thread. However, accessing the same data in shared memory on multiple threads could have a performance cost due to memory caching. A CPU cache holds data in memory which is most likely to be used by a CPU next. High performance CPUs typically have multiple levels of caching referred to as L1, L2, L3, etc. The L1 cache is the

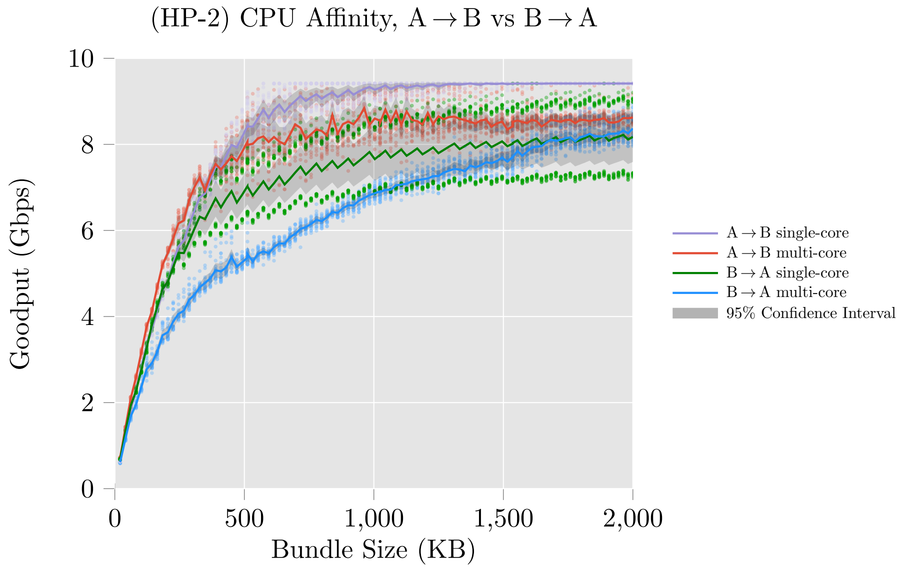


Figure 4.9: (HP-2) Single-core vs multi-core goodput.

Here, the goodput achieved while locking each ION node's processes and threads to a single core is compared to when not setting the affinity of its processes or threads. For most bundles sizes in both the A → B and B → A direction, locking to a single core resulted in a higher goodput. This is because ION often runs effectively serialized due to only one thread being able to access the SDR shared database at a time, so locking to a single core has minimal performance hit due to reducing concurrency, and instead benefits from the locking because cache-misses are reduced. If all threads run on the same CPU core, keeping the cache coherent is faster because less data needs to be copied to the L3 cache. Above bundle sizes of 1700KB, multi-core average goodput becomes higher for transfers from B to A compared to single-core goodput. However, it does not surpass the upper mode of the bimodal distribution displayed by the single-core results. This indicates that single-core has the potential to be faster on average for large bundle sizes if the cause of this bimodal distribution was understood. For more discussion on the bimodal distribution see section 4.3.1.

fastest to access, and stores data to be used by a particular processing core of a CPU. The L2 cache stores data to be used by a particular CPU. And finally, the L3 cache stores data to be used across multiple CPUs [39]. By accessing the same data in the SDR on threads running on separate CPUs, data must be written to lower levels of cache which is much slower than accessing that data on the L1 cache.

To handle objects containing large amounts of data, such as bundles, ION uses zero copy objects (ZCO). ZCOs are essentially pointers to data in the SDR. ZCOs reduce the effect of this caching issue, but do not eliminate it. Bundle data are still processed by several components in the pipeline and thus cache synchronization is still needed.

Compounding the issue of cache synchronization in the method of shared memory access in the SDR. Accessing data in the SDR must take place within critical sections called ”transactions” which serve two purposes:

- Safety: mutual exclusion is used to prevent race condtions and deadlock.
- Atomicity: When performing a sequence of database updates, either all of the updates are applied or none of them are, ensuring database integrity.

This mechanism results in only a single thread having access to data in the SDR at any moment. Considering that some of these transactions take a significant amount of time, and the frequency with which all threads require access to the SDR, an ION node becomes effectively single-threaded. The transaction mechanism also decreases performance due to the high cost of the context switches caused by the mutual exclusion mechanism.

Figure 4.10 shows goodput results in single-core compared to multi-core in the LP-2 testbench. Unlike in the HP-2 results, multi-core had significantly higher goodput. One explanation for this is that because it has less compute power, Node C is being bottlenecked by a different part of ION which can be executed in parallel, therefore giving a benefit to using multi-core.

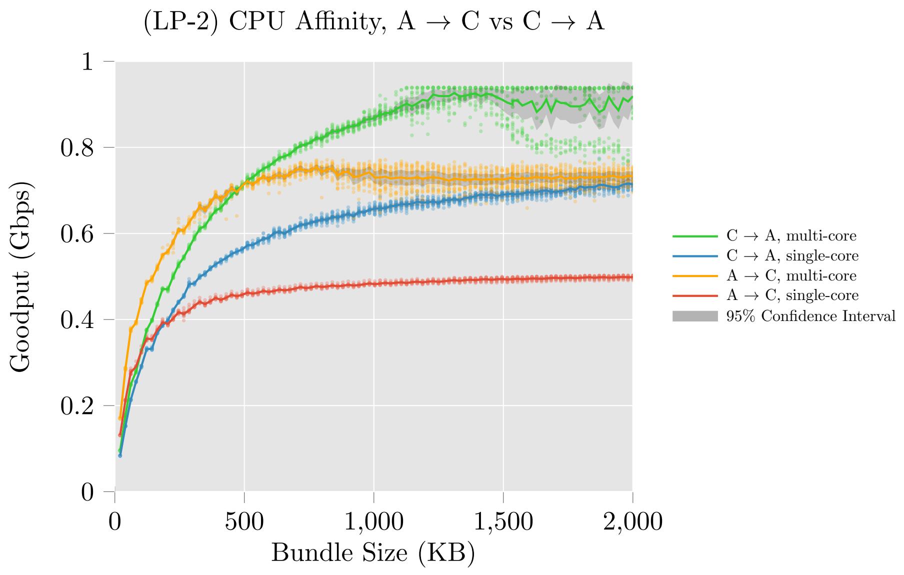


Figure 4.10: (LP-2) Single-core vs multi-core goodput

In the low performance testbench, locking ION's threads to a single core resulted in significantly lower goodput. The goodput while using a bundle size of 2000 KB in the A → C direction was reduced from 733 Mbps to 499 Mbps after locking the threads to a single core. This is different than the results from the HP-2 testbench where goodput is typically increased after locking to a single core. This demonstrates that the effect of thread affinity is dependent on hardware, although the exact cause of such a large difference is unclear. It may be the case that transfers in the LP-2 testbench are being bottlenecked by a different part of the software which is better able to utilize concurrency.

Figure 4.11 shows similar results for a three node transfer in LP-3, and Figure 4.12 shows results for parallel transfers in the LP-5 testbench. As expected, these tests benefit from multi-core as there are more threads that can be executed in parallel.

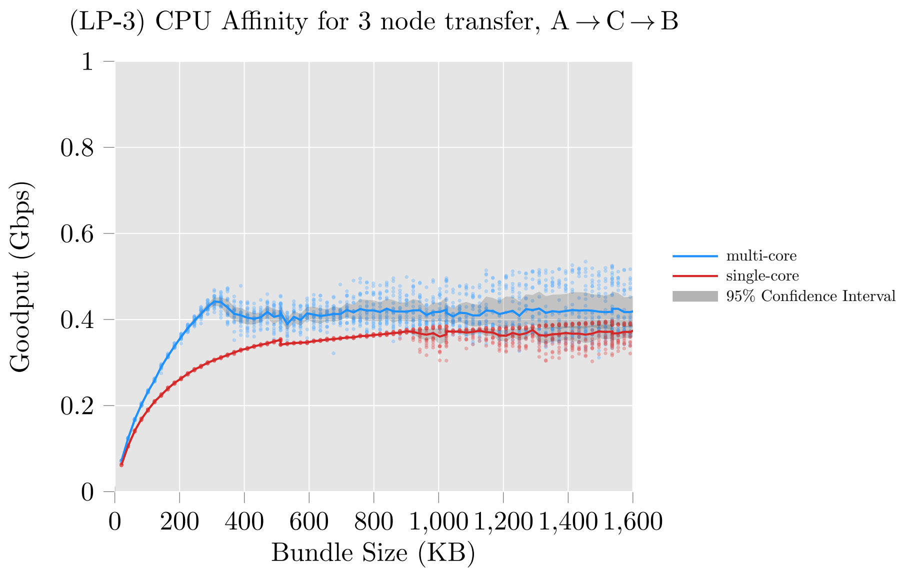


Figure 4.11: (LP-3) Single-core vs multi-core goodput

This graph shows goodput results for transfers from A to B, routing through C, where C is a ESPRESSObin ARM machine. Similar to the results from the LP-2 testbench, goodput in LP-3 is significantly lower when locking ION threads to a single core. When using multiple cores, LP-3 reaches its maximum goodput with a much lower bundle size than LP-2, about 300 KB compared to 600KB.

4.3.1 Multimodal Distributions

In many of our experiments, the results are indicative of a multimodal distribution. That is, the distribution does not center around one peak, but instead has multiple peaks that the results tend to be near. For example, in Figure 4.7b the "64 KiB" line shows two distinct modes that the goodput results tend to be close to. In the case of multi-core results, a multimodal distribution makes sense as there is some probability that the scheduler will run the threads on favorable cores and therefore the goodput will jump significantly higher. However, as evidenced by

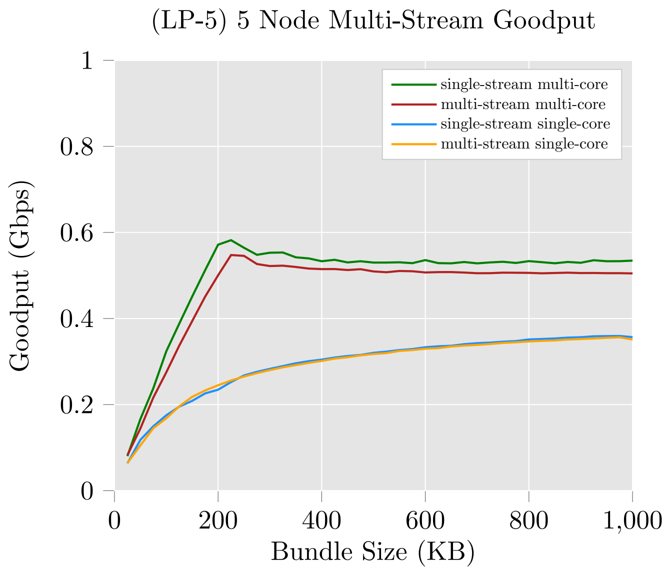


Figure 4.12: (LP-5) Single-stream vs multi-stream goodput

Goodput results from the 5 node testbench. In this case, single-stream refers to one transfer from A to B, routing though C. Multi-stream refers to two simulataneous transfers: A to B,routing through C, and D to E routing through C. Similar to the results from the 3 node test in Figure 4.11, there is a clear benefit from running ION on multiple cores for a multi-hop transfer. In this graph we also see that using a single-stream is faster, as using multiple streams introduces more context switching and thread contention when accessing the SDR.

Figure 4.7b, some single-core experiments also show multimodal distributions. There are a few possible explanations for this:

- During some periods of data collection, the core that ION is being locked to is also being heavily used by some other process. This is unlikely to be the actual cause as for any given ION instance, the goodput results remain in the same mode. That is, the goodput results for a given ION instance are consistently high or consistently low. If there was some process running periodically and slowing ION down, we would expect this to happen fairly randomly within the same instance of an ION node.

- A memory alignment issue within the PSM or SDR could be causing a slowdown. Message buffer alignment is known to have a significant impact on networking performance [40]. When allocating large buffers, the PSM and SDR do account for memory alignment. However, for smaller allocations memory alignment is ignored. Because the PSM and SDR are used by many processes, all the allocation and free calls will not always be in the same order for every instance of an ION node using the same configuration. It is therefore possible that after startup, the PSM or SDR could end up in a state where small memory allocations are improperly aligned and therefore cause lower goodput.

In this thesis, we did not perform an in-depth exploration of this multimodal behavior. This topic is left as future work.

4.4 CFDP

The CCSDS File Delivery Protocol (CFDP) is a file transfer protocol designed for use in space. As spacecraft storage mediums moved from tape recorders to mass storage with random access capabilities, the filesystem paradigm became commonplace for storing and retrieving data. Given the unique requirements for spaceflight operations, the creation of CFDP became necessary in order to facilitate the transfer of files with consideration of the following constraints and environmental factors of spaceflight as outlined in CCSDS Blue Book for CFDP [24]:

- Limited computation and memory capacity of spacecraft

- Communication links characterized by high noise, low bandwidth, asymmetry, disruption, and long delay
- Possible requirement for early access of data regardless of its quality

CFDP relies on the services of underlying Link-layer protocols. In this case, it is relying on ION/BP which sits between CFDP and the Link-layer in the network stack. ION implements an experimental CFDP module and provides a simple utility called bpcp which uses that module to transfer files.

4.4.1 Checksum

In order to assure integrity, every file sent by CFDP is accompanied by a 32-bit checksum. The checksum is computed using the applicable checksum algorithm, which can be any of the first 16 algorithms defined in the Space Assigned Numbers Authority (SANA) Checksum Identifiers registry [41]. This registry contains 4 implemented checksum algorithms, the null checksum, and 11 slots reserved for future use by CFDP. The type of checksum used is conveyed to the receiver as part of an initial metadata message, with Table 4.3 defining which checksum is to be used.

Table 4.3: SANA Checksum Identifiers Registry

ID	Checksum Type
0	Modular Checksum
1	Proximity-1 CRC-32
2	CRC-32C
3	CRC-32
4-14	Reserved
15	Null Checksum

The Null checksum indicates that no checksum should be used. In our testing we use the results from using the Null checksum as an upper bound for the performance of CFDP. The modular checksum is a simple summation checksum in which the data is divided into 4 byte words and computing the 2^{32} modular addition of those words. This type of checksum is not as reliable as the result is independent from the ordering of the words due to the commutative property of addition. In earlier versions of ION this was the default checksum algorithm, but now CRC-32 is used as the default.

The CRC-32 checksums are 32 bit cyclical redundancy checks. CRC-32 calculates a checksum 4 bytes at a time, returning the remainder of a polynomial division of those bytes. This remainder is then used in the CRC-32 calculation of the next 4 bytes, and so on. CRC-32 is position dependent; CRC-32 will detect data that is received in the incorrect order, and is therefore more reliable than the modular checksum. The polynomial used in the calculation affects the error detection ability of CRC-32. The polynomials used in standard Internet protocols achieve a Hamming distance of HD=4 for maximum-length Ethernet messages, whereas Koopman [42] performed an exhaustive search for good polynomials and showed that HD=6 is possible for messages of nearly 16K bits, and HD=4 is possible for messages up to 114K bits. CRC-32 is fast to compute and provides reasonable assurance of data integrity. However, it does not protect against intentional modification of data, in which case digital signatures or message authentication codes must be used to verify the data has not been tampered with.

4.4.2 CFDP Goodput

The goodput achieved while using CFDP over BP is significantly lower than only using BP. This is caused by extra processing overhead involved in reading the file into the SDR, creating the CFDP segment bundles, and managing the CFDP event queue. CFDP uses the same shared memory SDR as BP, so using CFDP introduces more thread contention and context switching. Figure 4.13 shows goodput results on the HP-2 testbench of CFDP over BP compared to just using BP. Note that CFDP bundles will be larger than BP bundles while transmitting the same amount of data due to the CFDP header, but this is negligable at larger bundle sizes. This graph shows that BP is about twice as fast than CFDP, with a maximum of 1.3 Gbps achieved through CFDP. This graph only shows bundle sizes up to 64 KB, as that is the maximum segment size allowed by the CFDP protocol. Bundle sizes larger than 64 KB will be even faster than the maximum goodput of CFDP. This graph also shows that while BP performs better while locked to a single core, CFDP actually benefits from running its threads on multiple cores. The most expensive threads are the thread that reads the file data into a buffer and creates the CFDP header and the thread that communicates between CFDP and BP. These two threads have less thread contention than the most expensive threads used by BP and therefore benefit from multiple cores.

The type of checksum algorithm used by CFDP has a significant effect on goodput. Figure 4.14 shows the goodput on the HP-2 testbench of the checksum algorithms implemented in ION. In this figure, ”CRC32” is a slightly modified implementation of the CRC32 implemented by the version of ION we tested on, where as ”CRC32 (unoptimized)” is the unmodified CRC32 implementation found in ION 3.7. In more recent versions of ION the CRC32 implementation is optimized in a similar fashion. We also added a CRC32-Slice-16 aglorithm which computes the same checksum as CRC32 more efficiently by having a larger lookup table and processing 16 bytes at a time rather than 4. It is an improvement by Bulat Ziganshin on the slice-by-8 algorithm [43]. The CRC32-Slice-16 is almost as fast as

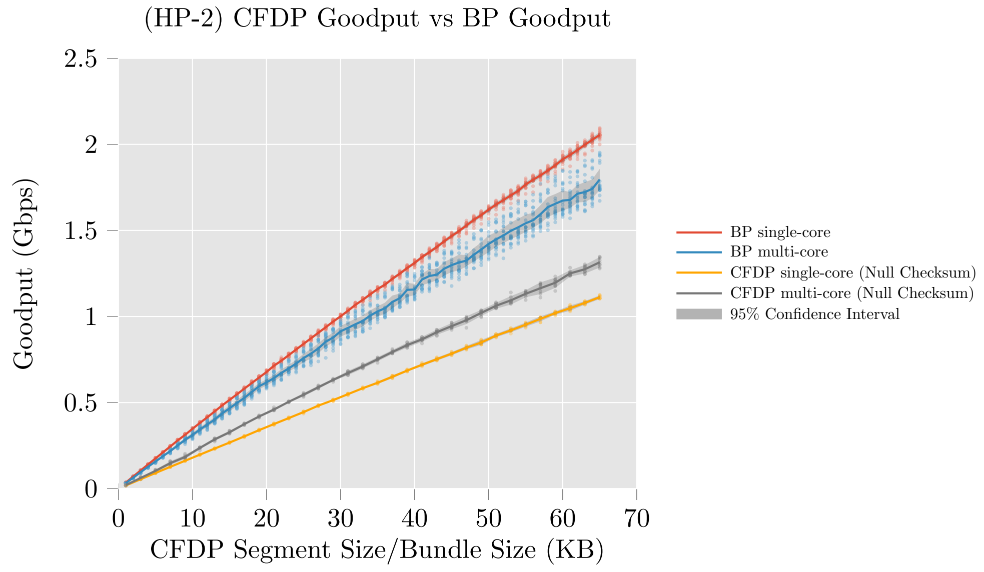


Figure 4.13: (HP-2) CFDP vs BP

We compared goodput while running CFDP over BP to just using BP on the HP-2 testbench. The Null checksum was used in order to compare BP against the maximum possible CFDP goodput. CFDP introduces a significant amount of overhead, with a maximum goodput of 1.32 Gbps at 64 KB segments compared to BP's maximum goodput of 2.06 Gbps at 64 KB bundles. As seen in section 4.3, BP often performs better when running on single-core. However, CFDP has higher goodput when running on multi-core. This indicates that the CFDP and BP processes run concurrently for a significant portion of the time.

not computing a checksum, showing that is possible to almost eliminate the bottleneck caused by checksum in CFDP. Figure 4.15 shows the results from the same tests while locking ION to a single core. Locking to a single core results in lower goodput and less variability in goodput between the checksum algorithms. Figure 4.16 compares single-core and multi-core results for CRC-Slice-16 and CRC32 (unoptimized). At a segment size of about 30 KB the CRC32 unoptimized running on a single core becomes slightly faster than the CRC32 unoptimized on multi-core as the CRC computation begins to bottleneck the system and therefore the single threaded performance begins to become more important for goodput. When using CRC32-Slice-16, our results show goodput is always greater when using multiple cores.

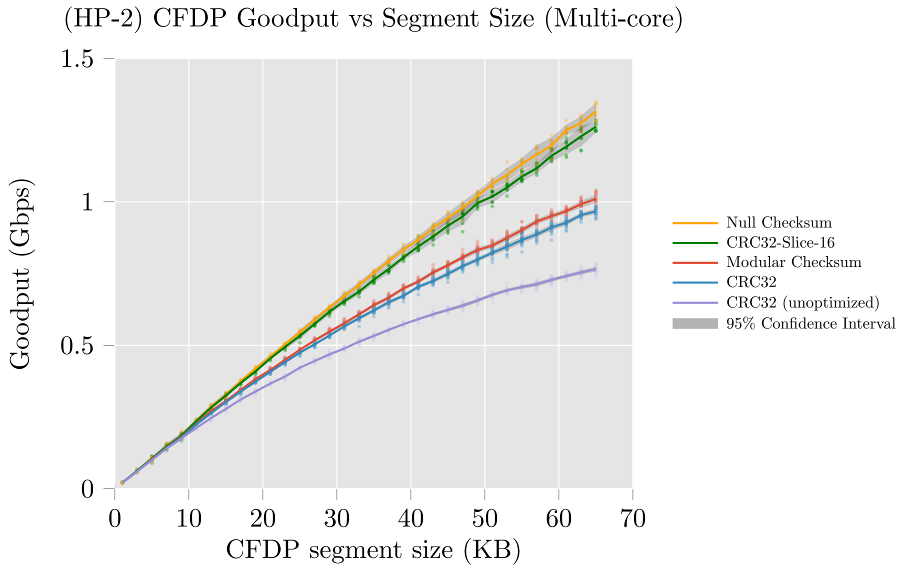


Figure 4.14: (HP-2) CFDP goodput with various checksum algorithms (multi-core)

Comparison between checksum algorithms while running ION on multi-core on the HP-2 testbench. The unoptimized CRC32 found in ION 3.7 performs the worst, with a maximum goodput of 765 Mbps at 64 KB segments. The CRC32-Slice-16 algorithm that we implemented in ION achieves nearly the same goodput as when not calculating a checksum (Null Checksum), with a maximum goodput of 1.26 Gbps at 64 KB segments.

We performed the same checksum throughput tests on the LP-2 testbench. Figure 4.17 compares using different checksum algorithms when locked to a single-core. There is no difference in goodput when using each algorithm. Figure

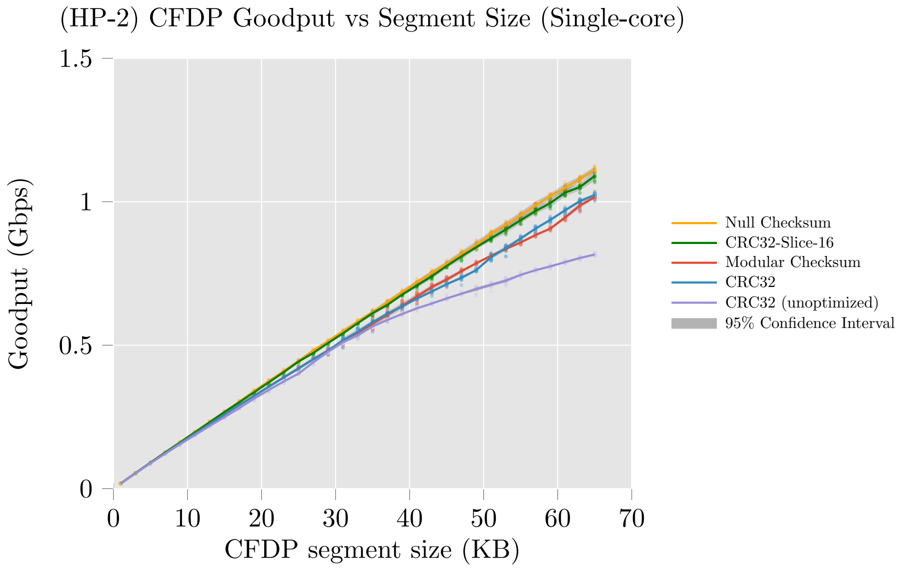


Figure 4.15: (HP-2) CFDP goodput with various checksum algorithms (single-core)

This graph compares checksum algorithms while running ION on single-core on the HP-2 testbench. There is less difference in goodput between the checksum algorithms than when using multi-core.

4.18 compares results from LP-2 one multi-core vs single-core. There is a small but significant increase in goodput when using multi-core.

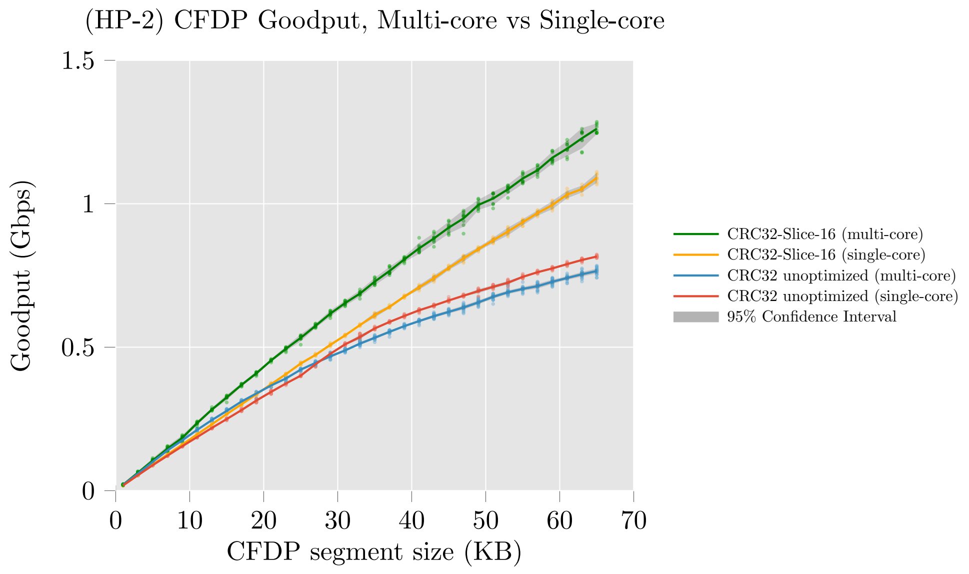


Figure 4.16: (HP-2) CFDP goodput, single-core vs multi-core

Single-core vs multi-core performance for unoptimized CRC32 and CRC32-Slice-16 on the HP-2 testbench. In both single-core and multi-core tests, CRC32-Slice-16 achieves significantly higher goodput than unoptimized CRC32. For CRC32-Slice-16, higher goodput was achieved while running on mult-core, whereas single-core performance was better when using unoptimized CRC32. This is because when calculating the CRC, ION is running effectively as a serial process. Running on a single-core is better when the process is serial because there are fewer cache misses. If ION stays in a serial state for a longer amount of time due to a slower CRC32 algorithm, there is more benefit in running on a single-core.

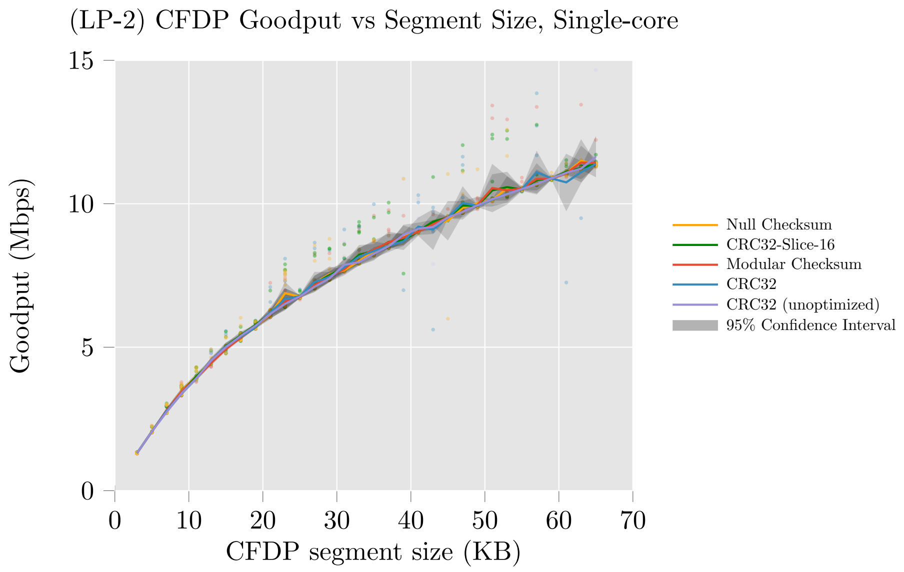


Figure 4.17: (LP-2) CFDP goodput with various checksum algorithms (multi-core)

On the LP-2 low performance testbench, the checksum algorithm used does not have a significant effect on goodput. The maximum goodput achieved at 64 KB segment sizes is about 11 Mbps in all cases. This is nearly 100 times slower than CFDP goodput on the HP-2 testbench. Clearly, CFDP introduces overhead that adversely affects nodes running on low performance hardware.

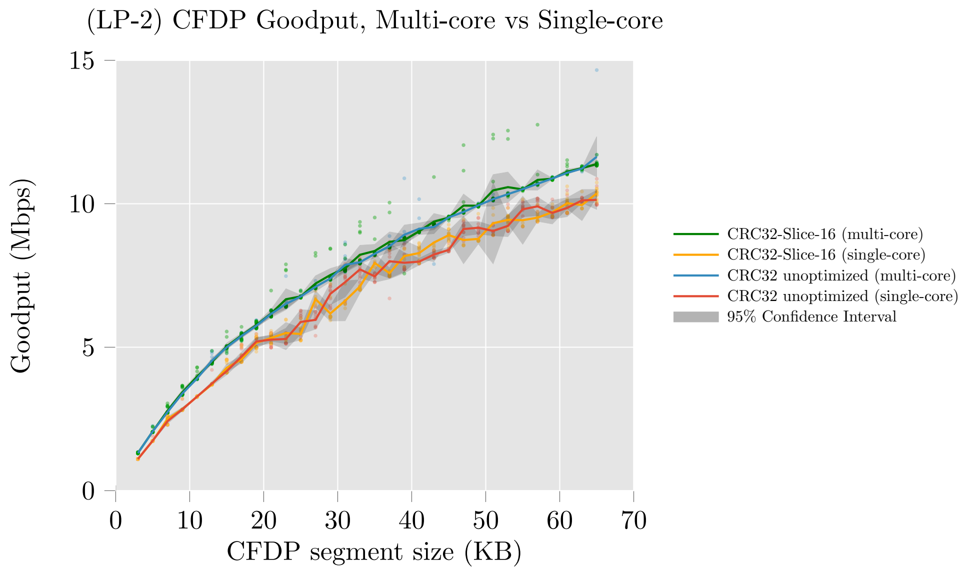


Figure 4.18: (LP-2) CFDP goodput, single-core vs multi-core

Single-core vs multi-core performance for unoptimized CRC32 and CRC32-Slice-16 on the LP-2 testbench. Similar to Figure 4.17, the checksum algorithm used makes no difference in goodput. Running on multi-core gives 11% greater throughput than on single-core at 64 KB segment sizes.

5 Conclusions

5.1 Review

In this work, we have demonstrated that it is possible to saturate a 10 Gbps connection between two ION nodes using the TCP convergence layer adapter. Without any modification to ION, 9.4 Gbps at a bundle size of 1MB was achieved on the high performance testbench when running the receiving node on a machine with a large CPU cache. On the low performance testbench using an A53 ARM machine, a maximum of 716 Mbps was achieved without any modification to ION.

There are several easily adjustable factors that affect goodput. The simplest method to increase goodput is to increase bundle size. This decreases the negative impact of the constant time overhead that is needed for each bundle. In a network with high bit-error-rate, more analysis is needed to determine the optimal bundle size as large bundle sizes will cause excessive retransmission.

Another method to increase goodput is to increase the convergence layer buffer size. This is a buffer used when copying data to or from the network interface. Increasing the size of this buffer decreases the linear time overhead (overhead that scales with bundle size). However, if the buffer size is larger than the bundle size, there is no benefit to increasing its size.

Thread affinity also has a significant impact on goodput. When transferring directly between two ION nodes, locking all the threads in every ION process to a single CPU core increases goodput in most cases we tested. Because ION processes use the same shared memory store, running ION across multiple cores creates overhead from maintaining cache coherency and suffers from thread contention as only one thread can access the shared memory store at once. However, transfers using CFDP benefit from running ION across multiple cores, as well as multi-hop transfers where there is at least one node that is both receiving and sending bundles at the same time. In these cases, there is enough processing that can be done in parallel to overcome the drawbacks of using multiple cores.

Researchers performing DTN benchmarks may refer to the following list of methods used to maximize ION's goodput:

- Adjust kernel networking settings of the machines ION is being tested on. This is especially important when using 10 Gbps networking interfaces as the default settings in most standard releases of Linux are geared toward 1 Gbps interfaces. In particular, increase the default size of TCP socket read and write buffers, enable TCP window scaling, and increase the maximum backlog of packets.

- When using TCPCLA in ION 3.7, use a transmission rate of 0 when defining the contact plans in the ionadmin utility. There is a defect in TCPCLA which causes the incorrect rate limiting to be applied on the receiving node. Setting the contact transmission rate to 0 will prevent rate limiting from being applied, but if rate limiting is still desired it is recommended that ION be rebuilt after making the changes outlined in appendix A.5.1. As a result of our work this defect was reported and patched for the ION 4.0.2 release.

- Use the heapmax option in the bpadmin utility. This sets the maximum allowed number of bytes per bundle to be stored within the SDR. If a bundle is larger than heapmax, then the remaining bytes are stored in the filesystem. If the SDR is configured to run in memory, setting heapmax to the maximum expected bundle size will keep bundles purely in memory which increases goodput significantly.

- Increase the size of the buffer used to copy to and from the socket. In TCPCLA, STCPCLA, the define constants that sets these values are `TCPCL_BUFSZ` and `STCPCL_BUFSZ` . Setting these buffers to be at least as large as the bundle being copied means that only one system call is required for writing to or reading from the socket. In our testing, we found that setting these to at least 512KB or 1 MB was optimal. Appendix 4.2 has information on how to set this value prior to building ION.

To make setting these changes easier, we added a build configuration option to ION called `--enable-high-speed`. Using this flag changes the TCPCLA and STCPCLA buffer sizes to 512 KB, sets the default heapmax value to 512 KB, adds the slice-by-16 CRC32 checksum algorithm, and sets the default checksum algorithm for CFDP to slice-by-16 CRC32. This option will be included in the next release of ION which is expected to launch at the end of May, 2021.

5.2 Future Work

5.2.1 Multimodal Distributions

As discussed in section 4.3.1, many of our single-core goodput results showed multimodal distributions. Further experimentation is needed to fully understand the cause of this. Figure 4.8 shows a clear bimodal distribution when using a buffer size of 64 KB, but not when using a buffer size of 128 KB. One future experiment to run is to find the buffer size at which this bimodal distribution behavior stops. Additionally, the experiments in this work could be repeated while carefully tracking the processes running on the system to confirm the ION processes are staying locked to the same core and there is not a rogue process occasionally consuming a significant number of CPU cycles on the same core as ION.

5.2.2 Memory Management

A potential method to address the issue of cache thrashing and thread contention is to redesign the memory management system in ION, including the SDR and by extension the PSM. Because flight software prohibits the use of dynamic memory allocation, the PSM is used to allocate memory from fixed blocks of memory created on startup. These blocks of memory may either be private or shared memory. PSM is used for inter-thread communication using shared linked lists, and is also used as the foundation of the SDR which provides inter-process communication.

Similar to the SDR, PSM is not designed with concurrent access in mind; whenever an allocation or free occurs, the entire partition is locked via semaphores. There are many examples of sophisticated allocation systems that could be more suitable. The slab allocator, first introduced in Solaris by Jeff Bonwick [44], is now commonly used in Linux and can have per-cache locking which would allow for concurrent allocations if used carefully. A better system, also building off of the slab allocator, is Jemalloc which is used in BSD. Jemalloc was designed specifically for concurrency. Jemalloc creates four arenas per CPU from which to allocate. Each thread gets assigned to one of its four possible arenas (which are determined by whichever CPU the thread is running on) in a round-robin fashion. When an allocation occurs, only the assigned arena of the calling thread is locked. This greatly reduces the probability of thread contention.

It is not clear whether changing the memory allocation method will result in greater throughput as memory allocation in ION is fairly fast, which might make up for the possibility of thread contention depending on how often ION allocates and frees memory. However, it is almost certain that the locking of the SDR results in a high amount of thread contention. This is a much more complicated problem to solve as the SDR must follow the ACID properties of databases which are the following:

- **Atomicity:** each transaction, although composed of several operations, is treated as a single unit which either completely fails or completely succeeds. This prevents partial updates which could corrupt the database.
- **Consistency:** a transaction can only transform the database from a valid state to another valid state. This also prevents corruption.
- **Isolation:** concurrent transactions result in transforming the database to the same state as if the transactions took place sequentially. The SDR currently solves this problem by only allowing one transaction to take place at a time.
- **Durability:** when a transaction is completed the database will retain its state even in the case of an unexpected shutdown. The SDR implements this by allowing the user to specify that the SDR should use a memory mapped file.

In order to implement the isolation property, ION locks the entire SDR during a transaction. This ensures that transactions do not interfere with each other, but causes a large amount of thread contention which is likely the largest factor affecting the performance of ION. There are many methods to guarantee isolation while allowing concurrency, but most are vastly more complex than simply locking the entire SDR.

A simple method that allows some amount of concurrency is two-phase locking (2PL) [45]. This method uses two types of locks: read-locks and write-locks. Write-locking the SDR blocks other threads from reading or writing to it. Read-locking the SDR only blocks other threads from writing to it, allowing multiple threads to read from the SDR at the same time. This does allow for limited concurrent access to the SDR (multiple reads only). However, Because 2PL requires locks to be acquired prior to a transaction for all objects being accessed, there will still be a large amount thread contention. This is because the entire SDR has to be treated as a single object. When performing a complex transaction such as a `sdr_free()` or `sdr_alloc()`, it is not known ahead of time which part of the SDR is going to be accessed, therefore a lock must be acquired for the entire SDR.

We propose an extension of 2PL called sectioned-2PL. Taking inspiration from allocators such as Jemalloc, the SDR would be broken into multiple arenas. Each arena would manage its own memory, and each component and possibly each data flow in ION would be given its own arena. This level of granularity would be simple to manage while having the benefit of a thread only needing to lock the specific arena(s) it is going to perform a transaction in. This would allow for enough concurrency to likely see a marked improvement in overall performance.

References

[1] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-tolerant networking architecture," Network Working Group, Internet Requests for Comments 4838, Apr. 2007.

[2] "The future lunar communications architecture," Interagency Operations Advisory Group Interagency Operations Advisory Group, Oct. 10, 2019. [Online]. Available: https://www.ioag.org/Public%20Documents/Forms/DispForm.aspx?ID=145 (visited on 03/09/2021).

[3] K. Scott and S. Burleigh, "Bundle protocol specification," Network Working Group, Internet Requests for Comments 5050, Nov. 2007.

[4] S. Burleigh, "Interplanetary overlay network: An implementation of the DTN bundle protocol," in *2007 4th IEEE Consumer Communications and Networking Conference*, ISSN: 2331-9860, Jan. 2007, pp. 222–226. DOI: 10.1109/CCNC.2007.51.

[5] M. Doering, S. Lahde, J. Morgenroth, and L. Wolf, "IBR-DTN: An efficient implementation for embedded systems," in *Proceedings of the third ACM workshop on Challenged networks*, ser. CHANTS '08, New York, NY, USA: Association for Computing Machinery, Sep. 15, 2008, pp. 117–120, ISBN: 978-1-60558-186-6. DOI: 10.1145/1409985.1410008. [Online]. Available: https://doi.org/10.1145/1409985.1410008 (visited on 03/10/2021).

[6] "DTN2: DTN reference implementation," *Delay Tolerant Networking Research Group*, Aug. 2006.

[7] A. Penning, L. Baumgärtner, J. Höchst, A. Sterz, M. Mezini, and B. Freisleben, “DTN7: An open-source disruption-tolerant networking implementation of bundle protocol 7,” presented at the International Conference on Ad-Hoc Networks and Wireless (AdHoc-Now 2019), Springer, 2019, pp. 196–209.

[8] *RightMesh/terra*, original-date: 2018-10-30T07:24:40Z, Nov. 23, 2020. [Online]. Available: https://github.com/RightMesh/Terra (visited on 03/10/2021).

[9] M. Blanchet, “Postellation: An enhanced delay-tolerant network (DTN) implementation with video streaming and automated network attachment,” in *SpaceOps 2012 Conference*, American Institute of Aeronautics and Astronautics, 2012. DOI: 10.2514/6.2012-1279621. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2012-1279621 (visited on 08/14/2020).

[10] M. Feldmann and F. Walter, “µPCN — a bundle protocol implementation for microcontrollers,” in *2015 International Conference on Wireless Communications Signal Processing (WCSP)*, Oct. 2015, pp. 1–5. DOI: 10.1109/WCSP.2015.7341252.

[11] L. Mohon. (Jul. 14, 2015). Laser communications relay demonstration (LCRD), NASA, [Online]. Available: http://www.nasa.gov/mission_pages/tdm/lcrd/index.html (visited on 04/01/2021).

[12] M. Wallace. (Mar. 27, 2018). LLCD: 2013-2014, NASA, [Online]. Available: http://www.nasa.gov/directorates/heo/scan/opticalcommunications/llcd (visited on 04/09/2021).

[13] T. Mai. (Apr. 24, 2015). Disruption tolerant networking experiments with optical comm, NASA, [Online]. Available: http://www.nasa.gov/directorates/

heo/scan/news_DTN_Experiments_with_Optical_Communications.html (visited on 04/09/2021).

[14] A. Silva, *Delay and Disruption Tolerant Networks: Interplanetary and Earth-Bound – Architecture, Protocols, and Applications.* Sep. 4, 2018, ISBN: 978-1-315-27115-6. DOI: 10.1201/9781315271156.

[15] A. Casteigts, P. Flocchini, W. Quattrociocchi, and N. Santoro, "Time-varying graphs and dynamic networks," in *In Proc. 10th Int. Conf. on Ad Hoc Networks and Wireless (ADHOC-NOW*, 2011, pp. 346–359.

[16] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with ZebraNet," in *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS X, New York, NY, USA: Association for Computing Machinery, Oct. 1, 2002, pp. 96–107, ISBN: 978-1-58113-574-9. DOI: 10.1145/605397.605408. [Online]. Available: https://doi.org/10.1145/605397.605408 (visited on 03/15/2021).

[17] "Transmission control protocol," Network Working Group, Internet Requests for Comments 793, Sep. 1981.

[18] "Internet protocol," Network Working Group, Internet Requests for Comments 791, Sep. 1981.

[19] O. Akan, H. Fang, and I. Akyildiz, "Performance of TCP protocols in deep space communication networks," *IEEE Communications Letters*, vol. 6, no. 11, pp. 478–480, Nov. 2002, Conference Name: IEEE Communications Letters, ISSN: 1558-2558. DOI: 10.1109/LCOMM.2002.805549.

[20] R. C. Durst, G. J. Miller, and E. J. Travis, "TCP extensions for space communications," *Wireless Networks*, vol. 3, no. 5, pp. 389–403, Oct. 1, 1997, ISSN: 1572-8196. DOI: 10.1023/A:1019190124953. [Online]. Available: https://doi.org/10.1023/A:1019190124953 (visited on 08/14/2020).

[21] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving TCP/IP performance over wireless networks," in *Proceedings of the 1st annual international conference on Mobile computing and networking*, ser. MobiCom '95, New York, NY, USA: Association for Computing Machinery, Dec. 1, 1995, pp. 2–11, ISBN: 978-0-89791-814-5. DOI: 10.1145/215530.215544. [Online]. Available: https://doi.org/10.1145/215530.215544 (visited on 08/14/2020).

[22] Y. Rekhter, T. Li, and S. Hares, "A border gateway protocol 4 (BGP-4)," Network Working Group, Internet Requests for Comments 4271, Jan. 2006.

[23] R. Wang, N. C. Aryasomayajula, A. Ayyagari, and Q. Zhang, "An experimental performance evaluation of SCPS-TP over cislunar communications links," in *2007 IEEE Wireless Communications and Networking Conference*, ISSN: 1558-2612, Mar. 2007, pp. 2603–2607. DOI: 10.1109/WCNC.2007.484.

[24] "CCSDS file delivery protocol (CFDP)," The Consultative Committee for Space Data Systems, Recommendation for Space Data System Standards CCSDS 727.0-B-5, Jul. 2020, p. 151.

[25] S. Burleigh, V. Cerf, R. Durst, K. Fall, A. Hooke, K. Scott, and H. Weiss, "The interplanetary internet: A communications infrastructure for mars exploration," *Acta Astronautica*, The New Face of Space Selected Proceedings of the 53rd International Astronautical Federation Congress, vol. 53, no. 4, pp. 365–373, Aug. 1, 2003, ISSN: 0094-5765. DOI:

10.1016/S0094-5765(03)00154-1. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0094576503001541 (visited on 08/14/2020).

[26] V. Cerf, *Interplanetary internet*, TEDxMidAtlantic, Dec. 2011.

[27] S. Burleigh, A. Hooke, L. Torgerson, K. Fall, V. Cerf, B. Durst, K. Scott, and H. Weiss, "Delay-tolerant networking: An approach to interplanetary internet," *IEEE Communications Magazine*, vol. 41, no. 6, pp. 128–136, Jun. 2003, Conference Name: IEEE Communications Magazine, ISSN: 1558-1896. DOI: 10.1109/MCOM.2003.1204759.

[28] I. Tzinis. (May 13, 2020). History of DTN, NASA, [Online]. Available: http://www.nasa.gov/directorates/heo/scan/engineering/technology/disruption_tolerant_networking_history (visited on 03/11/2021).

[29] V. Cerf, E-mail, Mar. 11, 2021.

[30] R. Doyle, R. Some, W. Powell, G. Mounce, M. Goforth, S. Horan, and M. Lowry, "High performance spaceflight computing (HPSC) next–generation space processor (NGSP) a joint investment of NASA and AFRL," in *the Workshop on Spacecraft Flight Software*, 2013.

[31] R. Some, R. Doyle, L. Bergman, W. Whitaker, W. Powell, M. Johnson, M. Goforth, and M. Lowry, "Human and robotic space mission use cases for high-performance spaceflight computing," Aug. 19, 2013, Accepted: 2014-03-11T22:53:23Z Publisher: Pasadena, CA : Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2013. [Online]. Available: https://trs.jpl.nasa.gov/handle/2014/44422 (visited on 10/09/2020).

[32] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu, *iPerf3*, 2014. [Online]. Available: https://iperf.fr/.

[33] W. A. Powell, “High-performance spaceflight computing (HPSC) project overview,” Radiation Hardened Electronics Technology (RHET) Conference, Nov. 2018.

[34] B. H. Leitao, “Tuning 10gb network cards on linux,” in *Proceedings of the 2009 Linux Symposium*, 2009.

[35] W.-B. Pöttner, J. Morgenroth, S. Schildt, and L. Wolf, “Performance comparison of DTN bundle protocol implementations,” in *Proceedings of the 6th ACM workshop on Challenged networks*, ser. CHANTS ’11, New York, NY, USA: Association for Computing Machinery, Sep. 23, 2011, pp. 61–64, ISBN: 978-1-4503-0870-0. DOI: 10.1145/2030652.2030670. [Online]. Available: https://doi.org/10.1145/2030652.2030670 (visited on 08/21/2020).

[36] E. Oliver and H. Falaki, “Performance evaluation and analysis of delay tolerant networking,” in *Proceedings of the 1st international workshop on System evaluation for mobile platforms*, ser. MobiEval ’07, New York, NY, USA: Association for Computing Machinery, Jun. 11, 2007, pp. 1–6, ISBN: 978-1-59593-762-9. DOI: 10.1145/1247721.1247722. [Online]. Available: https://doi.org/10.1145/1247721.1247722 (visited on 08/21/2020).

[37] M. J. Donahoo and K. L. Calvert, *TCP/IP sockets in C: practical guide for programmers*, 2nd ed, ser. The Morgan Kaufmann practical guides series. Amsterdam ; Boston: Morgan Kaufmann, 2009, 196 pp., OCLC: ocn305146730, ISBN: 978-0-12-374540-8.

[38] *Interplanetary Overlay Network (ION) Design and Operation.* Documentation included in ION 4.0 distribution, Nov. 29, 2020.

[39] R. v. d. Pas, *Memory Hierarchy in Cache-Based Systems.* 2002.

[40] L. Arber and S. Pakin, "The impact of message-buffer alignment on communication performance," *Parallel Processing Letters*, vol. 15, no. 1, pp. 49–65, Mar. 1, 2005, Publisher: World Scientific Publishing Co., ISSN: 0129-6264. DOI: 10.1142/S0129626405002052. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0129626405002052 (visited on 04/09/2021).

[41] (Jun. 2020). Space assigned numbers authority (SANA), [Online]. Available: https://sanaregistry.org/r/checksum_identifiers/ (visited on 03/15/2021).

[42] P. Koopman, "32-bit cyclic redundancy codes for internet applications," *Proceedings International Conference on Dependable Systems and Networks*, 2002. DOI: 10.1109/DSN.2002.1028931.

[43] S. Brumme. (Feb. 2015). Fast CRC32, [Online]. Available: https://create.stephan-brumme.com/crc32/ (visited on 03/15/2021).

[44] J. Bonwick and S. Microsystems, "The slab allocator: An object-caching kernel memory allocator," in *In USENIX Summer*, 1994, pp. 87–98.

[45] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. USA: Addison-Wesley Longman Publishing Co., Inc., 1986, ISBN: 978-0-201-10715-9.

Appendix: Hardware and Software Configurations

A.1 Hardware specifications

A.1.1 Node A

Node A	
Architecture	x86-64
CPU Model	Intel® Xeon® CPU E3-1246 v3 @ 3.50GHz
Threads per core	2
Cores per socket	4
Sockets	1
CPU max MHz	3900
L1d cache	32KB
L1i cache	32KB
L2 cache	256KB
l3 cache	8192KB
Memory	DDR3 16 GiB @ 3200Mhz
Network Interface	Ethernet Controller 10-Gigabit X540-AT2

A.1.2 Node B

Node B	
Architecture	x86-64
CPU Model	Intel® Xeon® CPU E5-2690 v3 @ 2.60GHz
Threads per core	1
Cores per socket	12
Sockets	2
CPU max MHz	2600
L1d cache	32KB
L1i cache	32KB
L2 cache	256KB
L3 cache	30720KB
Memory	DDR4 64 GiB @ 2133Mhz
Network Interface	Ethernet Controller 10-Gigabit X540-AT2

A.1.3 Node C

Node C	
Architecture	aarch64
CPU Model	Marvell Armada 3700LP (88F3720) dual-core ARM Cortex A53 processor @ 1GHz
Threads per core	1
Cores per socket	2
Sockets	1
CPU max MHz	1000
L1d cache	32KB
L1i cache	32KB
L2 cache	256KB
L3 cache	None
Memory	DDR4 1 GiB
Network Interface	1-Gigabit Ethernet

A.1.4 Node D

Node-D	
Architecture	x86-64
CPU Model	Intel® Xeon® CPU E3-2224G @ 3.50GHz
Threads per core	1
Cores per socket	4
Sockets	1
CPU max MHz	4700
L1d cache	32KB
L1i cache	32KB
L2 cache	256KB
l3 cache	8192KB
Memory	DDR4 8 GiB @ 2666Mhz
Network Interface	Ethernet Connection (7) I219-LM 1-Gigabit

A.1.5 Node E

Node-E	
Architecture	x86-64
CPU Model	Intel® Xeon® CPU E3-1230 v3 @ 3.30GHz
Threads per core	1
Cores per socket	4
Sockets	1
CPU max MHz	3300
L1d cache	32KB
L1i cache	32KB
L2 cache	256KB
l3 cache	8192KB
Memory	DDR3 8 GiB @ 1600Mhz
Network Interface	Broadcom Inc. NetXtreme BCM5720 Gigabit Ethernet

A.2 ION Configuration Files

All nodes have identical ionconfig files, shown below:

config.ionconfig

```
wmKey 0
sdrName ion
wmSize 10000000
configFlags 1
heapWords 10000000
```

All nodes have identical ionstart scripts, shown below:

ionstart.sh

```
ionadmin config.ionrc
sleep 1
bpadmin config.bprc
sleep 1
ipnadmin config.ipnrc
sleep 1
```

```
## uncomment to enable CFDP
# cfdpadmin config.cfdprc
# sleep 1
```

A.2.1 HP-2 Testbench

A.2.1.1 Node A, HP-2 testbench

config.ionrc (Node A, HP-2)

```
1 1 config.ionconfig
s
a contact +0 +100000000 1 2 2147483647 1
a contact +0 +100000000 2 1 2147483647 1
a range +0 +100000000 1 2 1
a range +0 +100000000 2 1 1
```

config.bprc (Node A, HP-2)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:1.1 q
a endpoint ipn:1.2 q
a endpoint ipn:1.3 q
a endpoint ipn:1.64 q
a endpoint ipn:1.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.101.101:4556 'tcpcli'
a outduct tcp 192.168.101.102:4556 ''
m heapmax 2100000
s
```

config.ipnrc (Node A, HP-2)

```
a plan 2 tcp/192.168.101.102:4556
a group 1 1 2
```

config.cfdprc (Node A, HP-2)

```
1
a entity 2 bp ipn:2.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.1.2 Node B, HP-2 testbench

config.ionrc (Node B, HP-2)

```
1 2 config.ionconfig
s
a contact +0 +100000000 1 2 2147483647 1
a contact +0 +100000000 2 1 2147483647 1
a range +0 +100000000 1 2 1
a range +0 +100000000 2 1 1
```

config.bprc (Node B, HP-2)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:2.1 q
a endpoint ipn:2.2 q
a endpoint ipn:2.3 q
a endpoint ipn:2.64 q
a endpoint ipn:2.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.101.102:4556 'tcpcli'
m heapmax 2100000
s
```

config.ipnrc (Node B, HP-2)

```
a group 2 2 1
```

config.cfdprc (Node B, HP-2)

```
1
a entity 1 bp ipn:1.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.2 LP-2 Testbench

A.2.2.1 Node A, LP-2 testbench

config.ionrc (Node A, LP-2)

```
1 1 config.ionconfig
s
a contact +0 +100000000 3 1 2147483647 1
```

```
a contact +0 +100000000 1 3 2147483647 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
```

config.bprc (Node A, LP-2)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:1.1 q
a endpoint ipn:1.2 q
a endpoint ipn:1.3 q
a endpoint ipn:1.64 q
a endpoint ipn:1.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.40:4556 'tcpcli'
m heapmax 2100000
s
```

config.ipnrc (Node A, LP-2)

```
a group 1 1 3
```

config.cfdprc (Node A, LP-2)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.2.2 Node C, LP-2 testbench

config.ionrc (Node C, LP-2)

```
1 3 config.ionconfig
s
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
```

config.bprc (Node C, LP-2)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:3.1 q
```

```
a endpoint ipn:3.2 q
a endpoint ipn:3.3 q
a endpoint ipn:3.64 q
a endpoint ipn:3.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.101.104:4556 'tcpcli'
a outduct tcp 192.168.100.40:4556 ''
m heapmax 2100000
s
```

config.ipnrc (Node C, LP-2)

```
a plan 1 tcp/192.168.100.40:4556
a group 3 3 1
```

config.cfdprc (Node C, LP-2)

```
1
a entity 1 bp ipn:1.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.3 LP-3 Testbench

A.2.3.1 Node A, LP-3 testbench

config.ionrc (Node A, LP-3)

```
1 1 config.ionconfig
s
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
```

config.bprc (Node A, LP-3)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:1.1 q
a endpoint ipn:1.2 q
a endpoint ipn:1.3 q
```

```
a endpoint ipn:1.64 q
a endpoint ipn:1.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.40:4556 'tcpcli'
m heapmax 2100000
s
```

config.ipnrc (Node A, LP-3)

```
a group 1 1 3
a group 2 2 3
```

config.cfdprc (Node A, LP-3)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.3.2 Node B, LP-3 testbench

config.ionrc (Node B, LP-3)

```
1 2 config.ionconfig
s
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
```

config.bprc (Node B, LP-3)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:2.1 q
a endpoint ipn:2.2 q
a endpoint ipn:2.3 q
a endpoint ipn:2.64 q
a endpoint ipn:2.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.130:4556 'tcpcli'
a outduct tcp 192.168.100.119:4557 ''
```

```
m heapmax 2100000
s
```

config.ipnrc (Node B, LP-3)

```
a plan 3 tcp/192.168.100.119:4557
a group 1 1 3
a group 2 2 3
```

config.cfdprc (Node B, LP-3)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.3.3 Node C, LP-3 testbench

config.ionrc (Node C, LP-3)

```
1 3 config.ionconfig
s
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
```

config.bprc (Node C, LP-3)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:3.1 q
a endpoint ipn:3.2 q
a endpoint ipn:3.3 q
a endpoint ipn:3.64 q
a endpoint ipn:3.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.119:4556 'tcpcli'
a induct tcp 192.168.100.119:4557 'tcpcli'
a outduct tcp 192.168.100.40:4556 ''
m heapmax 2100000
s
```

config.ipnrc (Node C, LP-3)

```
a plan 1 tcp/192.168.100.40:4556
a group 3 3 1
```

config.cfdprc (Node C, LP-3)

```
1
a entity 1 bp ipn:1.0 1 0 0
a entity 2 bp ipn:2.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.4 LP-5 Testbench

A.2.4.1 Node A, LP-5 testbench

config.ionrc (Node A, LP-5)

```
1 1 config.ionconfig
s
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a contact +0 +100000000 3 4 2147483647 1
a contact +0 +100000000 4 3 2147483647 1
a contact +0 +100000000 5 3 2147483647 1
a contact +0 +100000000 3 5 2147483647 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
a range +0 +100000000 3 4 1
a range +0 +100000000 4 3 1
a range +0 +100000000 5 3 1
a range +0 +100000000 3 5 1
```

config.bprc (Node A, LP-5)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:1.1 q
a endpoint ipn:1.2 q
a endpoint ipn:1.3 q
a endpoint ipn:1.64 q
a endpoint ipn:1.65 q
```

```
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.40:4556 'tcpcli'
m heapmax 2100000
s
```

config.ipnrc (Node A, LP-5)

```
a group 1 1 3
a group 2 2 3
a group 4 4 3
a group 5 5 3
```

config.cfdprc (Node A, LP-5)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.4.2 Node B, LP-5 testbench

config.ionrc (Node B, LP-5)

```
1 2 config.ionconfig
s
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 3 4 2147483647 1
a contact +0 +100000000 4 3 2147483647 1
a contact +0 +100000000 5 3 2147483647 1
a contact +0 +100000000 3 5 2147483647 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 3 4 1
a range +0 +100000000 4 3 1
a range +0 +100000000 5 3 1
a range +0 +100000000 3 5 1
```

config.bprc (Node B, LP-5)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
```

```
a endpoint ipn:2.1 q
a endpoint ipn:2.2 q
a endpoint ipn:2.3 q
a endpoint ipn:2.64 q
a endpoint ipn:2.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.130:4556 'tcpcli'
a outduct tcp 192.168.100.119:4557 ''
m heapmax 2100000
s
```

config.ipnrc (Node B, LP-5)

```
a group 1 1 3
a group 2 2 3
a group 4 4 3
a group 5 5 3
```

config.cfdprc (Node B, LP-5)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.4.3 Node C, LP-5 testbench

config.ionrc (Node C, LP-5)

```
1 3 config.ionconfig
s
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a contact +0 +100000000 3 4 2147483647 1
a contact +0 +100000000 4 3 2147483647 1
a contact +0 +100000000 5 3 2147483647 1
a contact +0 +100000000 3 5 2147483647 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
a range +0 +100000000 3 4 1
a range +0 +100000000 4 3 1
a range +0 +100000000 5 3 1
a range +0 +100000000 3 5 1
```

config.bprc (Node C, LP-5)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:3.1 q
a endpoint ipn:3.2 q
a endpoint ipn:3.3 q
a endpoint ipn:3.64 q
a endpoint ipn:3.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.119:4556 'tcpcli'
a induct tcp 192.168.100.119:4557 'tcpcli'
a induct tcp 192.168.100.119:4558 'tcpcli'
a induct tcp 192.168.100.119:4559 'tcpcli'
a outduct tcp 192.168.100.40:4556 ''
a outduct tcp 192.168.100.120:4556 ''
m heapmax 2100000
s
```

config.ipnrc (Node C, LP-5)

```
a plan 1 tcp/192.168.100.40:4556
a plan 4 tcp/192.168.100.120:4556
a group 3 3 1
```

config.cfdprc (Node C, LP-5)

```
1
a entity 1 bp ipn:1.0 1 0 0
a entity 2 bp ipn:2.0 1 0 0
a entity 4 bp ipn:4.0 1 0 0
a entity 5 bp ipn:5.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.4.4 Node D, LP-5 testbench

config.ionrc (Node D, LP-5)

```
1 4 config.ionconfig
s
a contact +0 +100000000 3 4 2147483647 1
a contact +0 +100000000 4 3 2147483647 1
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
```

```
a contact +0 +100000000 5 3 2147483647 1
a contact +0 +100000000 3 5 2147483647 1
a range +0 +100000000 3 4 1
a range +0 +100000000 4 3 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
a range +0 +100000000 5 3 1
a range +0 +100000000 3 5 1
```

config.bprc (Node D, LP-5)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:4.1 q
a endpoint ipn:4.2 q
a endpoint ipn:4.3 q
a endpoint ipn:4.64 q
a endpoint ipn:4.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.120:4556 'tcpcli'
m heapmax 2100000
s
```

config.ipnrc (Node D, LP-5)

```
a group 1 1 3
a group 2 2 3
a group 4 4 3
a group 5 5 3
```

config.cfdprc (Node D, LP-5)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.2.4.5 Node E, LP-5 testbench

config.ionrc (Node E, LP-5)

```
1 5 config.ionconfig
s
a contact +0 +100000000 5 3 2147483647 1
```

```
a contact +0 +100000000 3 5 2147483647 1
a contact +0 +100000000 3 1 2147483647 1
a contact +0 +100000000 1 3 2147483647 1
a contact +0 +100000000 2 3 2147483647 1
a contact +0 +100000000 3 2 2147483647 1
a contact +0 +100000000 3 4 2147483647 1
a contact +0 +100000000 4 3 2147483647 1
a range +0 +100000000 5 3 1
a range +0 +100000000 3 5 1
a range +0 +100000000 3 1 1
a range +0 +100000000 1 3 1
a range +0 +100000000 2 3 1
a range +0 +100000000 3 2 1
a range +0 +100000000 3 4 1
a range +0 +100000000 4 3 1
```

config.bprc (Node E, LP-5)

```
1
a scheme ipn 'ipnfw' 'ipnadminep'
a endpoint ipn:5.1 q
a endpoint ipn:5.2 q
a endpoint ipn:5.3 q
a endpoint ipn:5.64 q
a endpoint ipn:5.65 q
a protocol tcp 1400 100 -1
a induct tcp 192.168.100.121:4556 'tcpcli'
a outduct tcp 192.168.100.119:4559 ''
m heapmax 2100000
s
```

config.ipnrc (Node E, LP-5)

```
a plan 3 tcp/192.168.100.119:4559
a group 1 1 3
a group 2 2 3
a group 4 4 3
a group 5 5 3
```

config.cfdprc (Node E, LP-5)

```
1
a entity 3 bp ipn:3.0 1 0 0
m segsize 64000
s 'bputa'
```

A.3 Linux Network Configuration

/etc/sysctl.conf

```
net.core.wmem_max=12582912
net.core.rmem_max=12582912
net.ipv4.tcp_rmem= 10240 87380 12582912
net.ipv4.tcp_wmem= 10240 87380 12582912
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_sack = 1
net.core.netdev_max_backlog = 5000
```

A.4 CPU Affinity Control

The “taskset” command is used to launch nodes in single-core mode. By using the taskset command to run the ionstart.sh script, all processes launched by the script and the threads of those processes are locked to running on the provided list of CPUs. In this case, the list of CPUs only contains one element, CPU # 0. The command used for launch ION in single-core mode is shown below:

```
taskset -c 0 bash ./ionstart.sh
```

A.5 Software Changes

A.5.1 TCPCLA Rate Limiting Bug Fix

To fix the rate limiting bug in TCPCLA, comment out the following lines in `bp/tcp/tcpcli.c` (`bpv6/tcp/tcpcli.c` or `bpv7/tcp/tcpcli.c` if using ION-3.7.1 or above)

```
1587 receptionRate = rtp->session->neighbor->receptionRate;
1588 if (receptionRate > 0)
1589 {
1590     snoozeInterval = ((float) dataLength / (float) receptionRate)
1591                      * 1000000.0;
1592     microsnooze((int) snoozeInterval);
1593 }
```

A.5.2 TCPCLA Buffer Size

To set the TCPCLA buffer size, change the value on line 15 of `bp/tcp/tcpcli.c` (`bpv6/tcp/tcpcli.c` or `bpv7/tcp/tcpcli.c` if using ION-3.7.1 or above)

```
15  #define TCPCL_BUFSZ         (64 * 1024)
```