Performance Evaluation and Analysis of Delay Tolerant Networking

Earl Oliver, Hossein Falaki
David R. Cheriton School of Computer Science
University of Waterloo
{eaoliver, mhfalaki}@uwaterloo.ca

ABSTRACT

Wireless opportunistic connections are the primary mechanism for transferring data between disconnected nodes, such as vehicles, remote sensors, or village kiosks [8, 4, 13] in a delay tolerant network (DTN). Opportunistic connections may last from seconds, as in the case of a rapid drive-by, to several minutes. During this short connection window, it is important to maximize data transfer between DTN nodes. We use microbenchmarks to study the wireless transfer performance of the DTN reference implementation, which is the most widely used DTN implementation today [9]. Existing DTN deployments utilize low-cost, low-power devices that tend to have slow CPUs [13]. Based on these characteristics, we hypothesize about the effect of control parameters on opportunistic data transfer. We test our hypotheses through a series of experiments and show that the principal performance bottleneck is the CPU. We also found that the choice of DTN bundle size affects performance by a factor of up to 60.

Categories and Subject Descriptors: C.4 [Performance of Systems]: Measurement techniques

General Terms: Measurement, Performance, Experimentation

Keywords: Delay Tolerant Networking, Performance, Evaluation

1. INTRODUCTION

In recent years delay tolerant networking (DTN) has emerged as a means to reliably transport data to and from disconnected regions of the Internet [10]. Although DTN is designed to support heterogeneous networks and interfaces, most projects based upon DTN, such as DieselNet [8], SeNDT [4], and KioskNet [13], are based upon IP and a combination of wireless and wired links. At the edges of the network, each project utilizes opportunistic wireless connections to transfer data *bundles* [12] between mobile vehicle and stationary throwbox, sensor, or kiosk respectively. Opportunistic connections may last from seconds, in the case of a rapid drive-by, to several minutes. For large scale deployments, such as in KioskNet, where potentially gigabytes of data can be transferred between buses and rural kiosks, maximizing data transfer during opportunistic connections is important.

In this paper we examine the behaviour of the DTN reference implementation (DRI) by the IRTF DTNRG [1] on low-power devices. Although several other DTN implementations have been developed, we chose to examine the reference implementation because, to the best of our knowledge, it is the most widely used. We hypothesize how various parameters and resources affect the performance of this implementation. Our hypotheses are then tested with several experiments that capture operating system state as well as the DRI data transfer statistics. From our evaluation, we validate or invalidate elements of our hypotheses and suggest possible improvements to the DRI. Throughout this analysis, we are primarily interested in the throughput between DRI nodes during opportunistic connections. For the remainder of this paper we define this quantity as a measure of the DRI *performance*.

Existing evaluations of the DRI have focused on routing and the DRI's superiority over existing store-and-forward architectures such as SMTP [9]. To the best of our knowledge, this is the first paper focusing on the low-level performance of the DRI. The main contributions of this paper are:

- Analysis of the most widely used reference implementation for disconnection tolerant network architectures
- Discovery of the bottlenecks to DRI performance on low-power devices
- Demonstration of the significant effect of bundle size on DRI performance
- Development of a methodology for evaluating the performance of other mobile systems

This paper begins with a brief overview of the DRI architecture. We then outline the details of our test platform. Section 4 presents our experimental hypotheses. Section 5 outlines our method to evaluate our hypotheses and describes our results. This section also highlights areas we believe performance improvements can be made. In section 6 we conclude and outline future work.

2. DRI ARCHITECTURE

In [9], Demmer describes the major components of the DRI. *Convergence layers* are the interfaces between the DRI

MobiEval'07, June 11, 2007, San Juan, Puerto Rico, USA.

and different network protocol suites. *Persistent storage* holds bundles during store-and-forward operations. Berkeley DB is currently the most widely used application for persistent storage. The *router module* is responsible for making routing decisions, which are performed by the *forwarder module*.

3. DEVICE CHARACTERISTICS

In this section we introduce the devices used for our experiments. This configuration is currently used in [13]. We expect other deployments to use similar low-power devices to save cost and power. We also investigate disk I/O and network limitations of our test bed through a series of microbenchmarks.

3.1 Test Bed Hardware and Operating System

Our test bed consists of four low-power, low-cost computers from Soekris Engineering (net4801). These computers have a single 266 MHz processor and 256 MB of integrated SDRAM. The test bed setup is illustrated in figure 1. We installed Atheros 802.11abg wireless cards on *beta* and *gamma* nodes. These cards were set to use 802.11a at a frequency of 5.825 GHz (channel 165) in ad-hoc mode. During our experiments no other 802.11a devices were within range of our lab on this channel.

We installed Stable Debian on all the test bed nodes and upgraded the kernel to Linux beta 2.6.8-3-386. We used the DRI CVS head as of February 22, 2007. GCC version 3.3.5-3 was used on these machines to compile the DRI and other benchmarking tools.

3.2 Microbenchmarks

We conducted a series of microbenchmarks to determine the performance limitations of the network and disk subsystems. Parameters such as CPU clock rate and RAM read/write delays are fixed and known, but the actual disk I/O throughput and network interface throughput depend on several other parameters. Disk I/O latency may vary for different operating system, file system, and virtual memory system combinations. Wireless throughput and delays are also affected by parameters such as environmental noise. The results of the microbenchmarks allow us to generalize our evaluation to other hardware platforms.

3.2.1 Disk I/O Microbenchmarks

Our DRI nodes use the MK4032GAX Toshiba 40 GB 2.5" notebook hard drives with ATA-6 interfaces. The rotational speed of these disks is 5,400 RPM. They have an 8 MB buffer and their transfer rate is stated to be 100 MB/s [7].

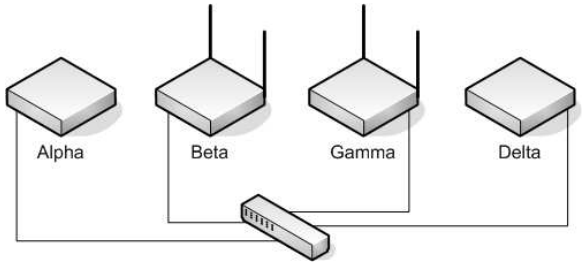

Figure 1: Test bed configuration

	read	**read** (cache hit)	**write**
Scattered	0.6 MB/s	13.0 MB/s	4.0 MB/s
Contiguous	12.0 MB/s	-	10.3 MB/s

Table 1: Average values of I/O microbenchmark results for hard disk

	read	**read** (cache hit)	**write**
Scattered	1.6 MB/s	13.0 MB/s	2.8 MB/s
Contiguous	4.0 MB/s	-	2.8 MB/s

Table 2: Average values of I/O microbenchmark results for Compact Flash

We measured disk I/O throughput in two cases: scattered data and contiguous data. For the scattered data measurements we wrote a benchmark program that reads/writes varying amount of random data from/to 1000 to 4000 files and measures the effective throughput. Contiguous data throughput was measured by using the same program but reading/writing 1 GB of data from/to a single file. We also measured the disk I/O throughput for contiguous read/write without file system intervention by using the standard Unix *dd* command. We repeated these experiments using 4 GB Lexar Professional Compact Flash as an alternative storage device. These devices are stated to sustain a minimum write speed of 19.4 MB/s [3].

The disk I/O microbenchmark results are presented in Table 1. As a side note, we found in the scattered disk I/O microbenchmarks that if the total data size is less than 1.5 MB, the effective throughput is dominated by the number of files and is not affected by the size of files.

Compact Flash I/O microbenchmark results are presented in Table 2. Compact Flash I/O delays were observed to be highly variable.

3.2.2 Network Microbenchmarks

The goal of the network microbenchmarks is to measure maximum wireless throughput between nodes *beta* and *gamma*. We conduct two microbenchmarks with custom tools. The first measures throughput between a TCP client on *beta* sending data to a TCP server on *gamma*. In the second

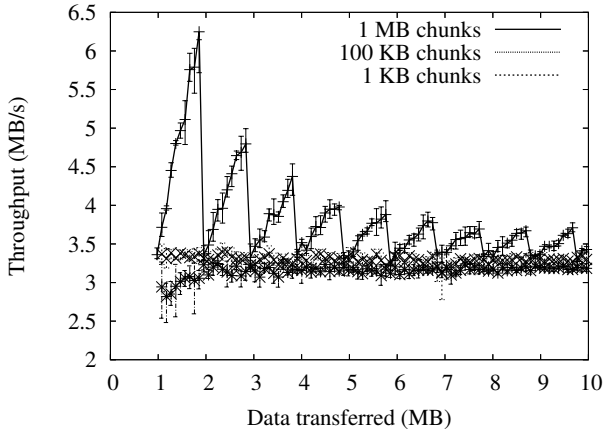

Figure 2: Network throughput vs. data size when disk is not involved.

microbenchmark, a TCP client on *beta* reads data from a file and sends them to *gamma* where the TCP server writes the received data to another file. Each test is conducted four times with different data block sizes (1 KB, 10 KB, 100 KB, 1 MB).

Figure 2 illustrates the throughput vs. transferred data size in the first network microbenchmark. The throughput increases as the data block size increases. The maximum average throughput (with data block size of 1 MB) is 4.0 MB/s and the minimum average throughput (with data blocks of 1 KB) is 3.2 MB/s. The saw-tooth pattern formed when using 1 MB chunks is due to high fixed system call costs relative to linear "data touching" costs [11]. Using 1 MB chunks to transfer i MB of data causes $\lceil \frac{i}{1MB} \rceil$ socket write calls into the kernel. For all data points in each tooth, the number of socket write calls is constant while the amount of data transferred increases. At the peak of each tooth, the cost of an additional system call is significantly higher than the throughput gained by transferring an additional 0.1 MB. Figure 3 illustrates the throughput vs. transferred data size in the second network microbenchmark. In this experiment throughput is highly variable. The maximum average throughput (with data block size of 1 MB) is 3.2 MB/s and the minimum average throughput (with data blocks of 1 KB) is 1.9 MB/s. This figure also shows the saw-tooth pattern. During both network microbenchmarks the CPU was saturated.

4. HYPOTHESES

Based on our microbenchmarks, we hypothesize about the effect of system resources and parameters on the performance of the DRI.

4.1 Resources

We begin by observing the relationship between disk I/O throughput and network throughput. In the best case, disk I/O (13 MB/s for cached read) exceeds maximum network I/O throughput (6.15 MB/s) by 111%. For larger amounts of data, the gap between disk I/O (cached read) and average network throughput (3.4 MB/s) increases to approximately 282%. When performing contiguous disk I/O or reading cached data the disk significantly outperforms the network.

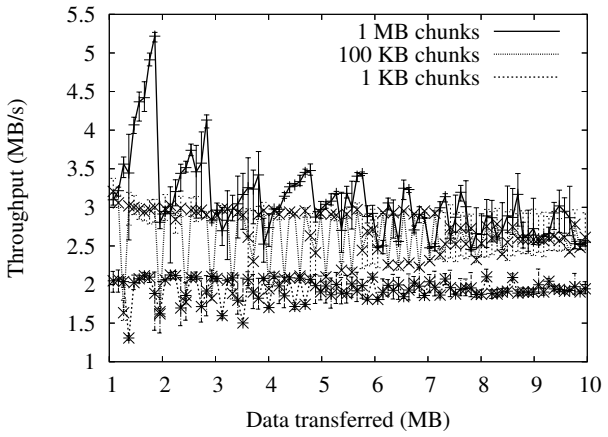

Figure 3: Network throughput vs. data size with disk involvement.

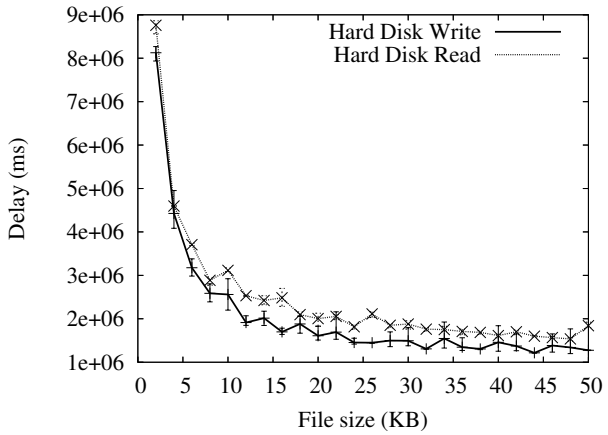

Figure 4: Delay writing 15 MB of data using different file sizes.

We believe this is because the **CPU** is unable to handle interrupts from the wireless NIC. Under these I/O conditions the CPU is the primary bottleneck.

To be highly tolerant to power and hardware failures, the DRI writes received bundles to disk and reads bundles from disk when sending data to other DRI nodes. Frequent reading and writing of bundles within the DRI causes scattered disk I/O. In the worst case, when reading scattered non-cached data (0.6 MB/s), the average network throughput exceeds the disk throughput (3.4 MB/s) by 466%. Scattered I/O reduces the load on the CPU and causes the disk to become the primary bottleneck.

We hypothesize that the DRI's dependency on the disk and frequent scattered **disk access** make the disk the primary DRI performance bottleneck. Due to the poor write performance of Compact Flash, we expect better performance using hard disk than using solid state storage.

Modern OSs have virtual memory systems that use disk storage in conjunction with physical memory to provide virtually unlimited memory to user applications. The drawback to virtual memory is slower memory access when a page fault occurs. Page faults happen more often when a program tries to access dispersed parts of its address space. Poor disk I/O would magnify the cost of swapping data in and out of memory; however, the level of disk I/O due to virtual memory is small compared to bundle read/write disk I/O. We believe that page faults due to inadequate **memory** do not limit the performance of the DRI.

4.2 Parameters

Our network microbenchmarks prove that applications, including the DRI, cannot take advantage of the wireless data rates supported by the Atheros card. In both network microbenchmarks, maximum average throughput was approximately 3 MB/s when transferring large amounts of data. We hypothesize that the performance of the DRI will peak at a **wireless data rate** of at most 24 Mb/s.

In Figure 4 we observe that disk I/O delay is minimized when using larger files. In the DRI, bundle size is an application-defined parameter. Using larger bundles reduces aggregate bundle overhead when transferring a fixed amount of data. We believe that DRI performance improves with increasing **bundle size**.

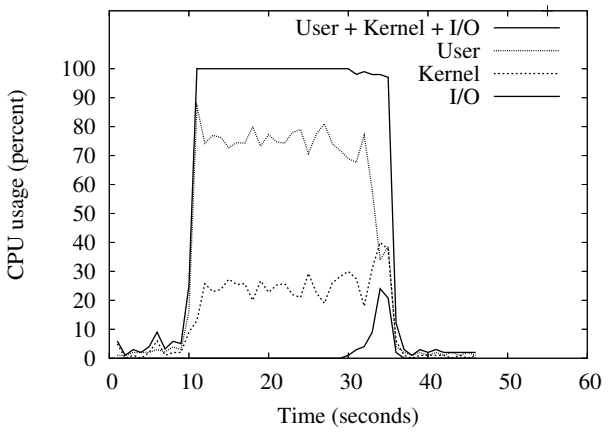

Figure 5: CPU utilization during an opportunistic connection.

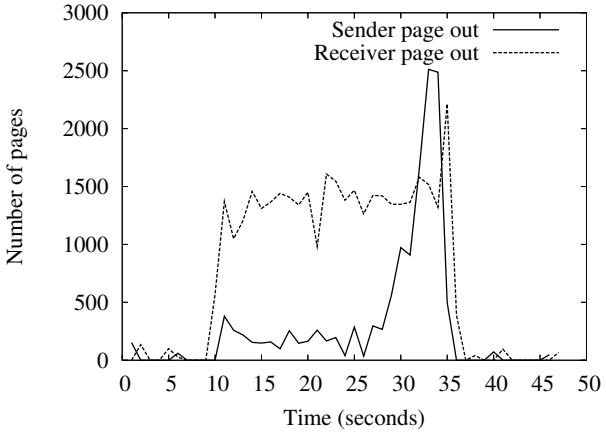

Figure 6: Virtual memory activity during an opportunistic connection.

Parallelism is often used to increase performance of disk bound applications. When one thread is blocked on disk I/O, other threads may utilize the CPU. Given our belief that the DRI is disk bound, we hypothesize that performance can be improved by further **parallelism**.

5. EVALUATION AND ANALYSIS

In this section we outline the method for verifying our parameter and resource hypotheses and present our results. The test scripts used to perform these benchmarks and our data can be downloaded from [6].

Throughout the following experiments we measure system statistics using the *System Activity Reporter* (SAR) [5]. We are particularly interested in: detailed CPU usage (i.e. user, system and I/O waiting time) and virtual memory activity. A combination of this information and the experiment results help us verify our hypotheses in the following subsections.

5.1 CPU

Figure 5 illustrates user, kernel, I/O, and total CPU usage of a DRI node during a 20 second opportunistic connection using a 50 KB bundle size. During the connection window the CPU is used at capacity (75% user and 25% kernel). When the contact is lost I/O activity increases up to 24%. The final increase in CPU activity due to I/O corresponds to a page-out storm between 30 and 35 second into the test. This I/O activity is illustrated in Figure 6. We observed similar CPU behaviour when evaluating different bundle sizes.

We conclude that the CPU, contrary to our expectations, is the first order performance bottleneck for the DRI on the chosen hardware. The throughput of the network microbenchmark, which involves the disk (3.0 MB/s), is more than three times better than the best performance of the DRI (0.87 MB/s).

5.2 Bundle Size

To evaluate the effect of bundle size on DRI performance, we measure data transfer during a simulated opportunistic wireless connection between nodes *beta* and *gamma*. We perform this test using bundle sizes between 2 to 50 KB[1] in

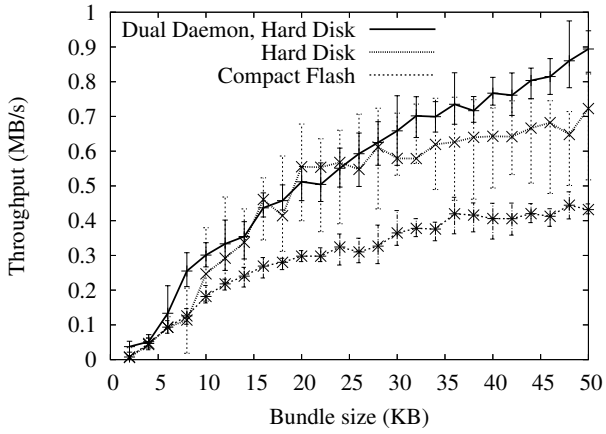

Figure 7: DRI throughput during an opportunistic connection vs. bundle size using the hard disk, Compact Flash, and two DRI daemons.

2 KB intervals. The test begins by switching the wireless frequency on *beta* to 5.18 GHz (channel 36) to disable the link to *gamma*. We then enqueue 15 MB worth of data at *alpha* for delivery to *delta*. Bundles enqueued at *alpha* are transferred rapidly across the wired link and are enqueued at *beta*. We chose to transfer data between *alpha* and *delta* to separate application level overhead from our system evaluation. A delivery receipt was requested by *alpha* for each bundle delivered to *delta*. Although small (54 byte) bundles add extra transfer overhead and reduce overall throughput, they are common in DTN deployments to acknowledge bundle delivery.

After enqueueing bundles, we initiate an opportunistic connection by switching the wireless frequency on *beta* back to 5.825 GHz. Each DRI node is configured to retry its connection to a lost peer every second. After switching *beta* back to a common channel, its link with *gamma* is quickly restored and bundles transfer between nodes. After a fixed connection window of 20 seconds we switch *beta*'s wireless frequency back to 5.18 GHz to break the connec-

[1]50 KB is the maximum in memory bundle size supported by the DRI API.

	Compact Flash	Hard Disk	Ratio
Small bundles	0.09 MB/s	0.10 MB/s	90%
Large bundles	0.42 MB/s	0.67 MB/s	62%

Table 3: Comparison of DRI performance using hard disk and Compact Flash

tion to *gamma*. A 20 second connection window is chosen so that data would remain in *beta*'s bundle queue when the test is over. We then observe the goodput during the opportunistic connection to be the number of bundles removed from the queue times the bundle size divided by 20 seconds.

In our standard test configuration, (one DRI daemon process, using hard disk and 256 MB of memory), workloads consisting of 50 KB bundles transferred on average 23.8 times more data than workloads consisting of 2 KB bundles. Figure 7 illustrates these results. We saw an identical effect of bundle size when running two DRI daemons in parallel. Figure 7 also includes the effect of solid state storage which is discussed in 5.4.

Given that the CPU is fully utilized in each case, we attribute performance gains using large bundles to reduced per-byte bundle overhead and minimized I/O delays as observed in our microbenchmarks.

These results are limited due to evaluating only bundle sizes supported by the DRI in-memory API. In future work we will explore the complex trade-off between DRI performance and power efficiency. Given that larger bundles cannot use the DRI in-memory API, they incur the cost of additional disk reads/writes. Finding the optimal bundle size will be a point of interest.

5.3 Memory

To verify that the CPU is the primary bottleneck on devices with limited memory we used GRUB to reconfigure the Linux kernel on the test bed machines to use a fraction of the available physical memory (128 MB and 64 MB). For each amount of memory we re-run the tests in 5.2. We monitor paging activity of the OS during these tests.

Figure 8 illustrates the DRI throughput during a 20 second opportunistic connection vs. bundle size using 256 MB, 128 MB and 64 MB of physical memory. The average throughput of the tests are statistically indistinguishable. The number of page-ins during an opportunistic connection is negligible. These observations prove that CPU is by far the primary bottleneck of DRI performance.

5.4 Disk

To assess the effect of disk access delays during an opportunistic connection we re-run the previous tests using Compact Flash cards instead of hard disks. Based on the microbenchmark results the Compact Flash write throughput for scattered data is 70% and for contiguous data is 27% of hard disk write throughput and its contiguous read throughput is 33% of the hard disk. Table 3 compares the DRI throughput using Compact Flash and hard disk. These results confirm our expectation that the DRI would perform better using hard disk than using solid state storage. With Compact Flash we found that 48 KB bundles yield the highest throughput. Using 48 KB bundles transferred on average 59.2 times more data than our worst result using 2 KB bundles. Although Compact Flash offers poor performance

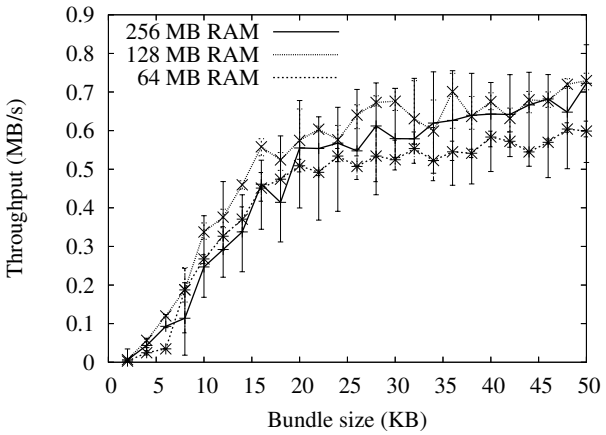

Figure 8: DRI throughput vs. bundle size for different physical memory capacities

relative to hard disk, it may be more practical in power constrained deployments.

We observe three times more disk write activity on the receiving side of the connection. An important performance optimization for the DRI (at the cost of decreasing reliability) would be to delay bundle writes to permanent storage until after an opportunistic connection.

5.5 Wireless Data Rate

We assess the effect of data rate by repeating the bundle size experiment in 5.2 using different wireless data rates. The supported data rates of our Atheros wireless cards are: 6, 9, 12, 18, 24, 36, 48 and 54 Mb/s. For each data rate test, we hold the bundle sized fixed to 50 KB and use the same 15 MB workload over the same duration.

The results of the wireless data rate tests prove our hypothesis that the performance of the DRI would peak using a wireless data rate of at most 24 Mb/s. The maximum throughput of the network microbenchmark using disk was approximately 3 MB/s (see Figure 3). However, the performance of the DRI peaked at a wireless data rate of 12 Mb/s. At 12 Mb/s the DRI transferred approximately 14.8 MB of data from *beta* to *gamma* during a 20 second window. Data transfer for data rates greater than 12 Mb/s deviate from 14.8 MB by at most 0.6%.

As a side note, in environments with power constraints, limiting the data rate to 12 Mb/s could have a significant positive impact on the uptime of the DRI node due to power savings. We plan to explore the performance of DRI with respect to power consumption in future work.

5.6 Parallelism

To study if the degree of thread parallelization is adequate, we run parallel instances of the DRI daemon process. We configure each node on the test bed to run one or two DRI daemons. The DRI configuration system made it easy to separate processes; however, the DRI API needed to be modified to communicate with applications over different TCP ports. We then re-run the previous bundle size test using dual DRI daemons.

The effect of parallelism varies for different bundle sizes. The most significant gain in performance takes place with

bundles greater than 24 KB in size. Despite the fact that the DRI is CPU bound, we gain approximately 16% in performance by using two DRI daemons when bundle sizes are greater than 24 KB. After identifying CPU as the primary bottleneck, we did not expect a performance gain by using two daemons. We used OProfile [2] to trace the execution of the DRI while transferring a 15 MB workload consisting of 50 KB bundles during a 20 second connection window for both single and dual daemons. In both tests, the DRI daemon(s) consumed approximately equal portions of the CPU; however, in the dual daemon case 8.28% less time was spent acquiring and releasing locks. Most of this savings was due to less time wasted on spinlocks. Using dual daemons reduced the amount of CPU time that either daemon could spend trying to acquire the spinlock before CPU preemption.

6. CONCLUSIONS AND FUTURE WORK

To evaluate the performance of the DRI, we studied the physical characteristics of low-cost, low-power hardware used in existing DTN deployments. We presented our hypotheses about DRI performance based on these characteristics and our knowledge of the DRI. We evaluate the hypotheses affecting DRI performance during opportunistic connections using a series of experiments.

Contrary to our expectations, we found that the CPU is the primary bottleneck inhibiting the performance of the DRI; however, performance can be improved by further parallelization. The use of spinlocks is the largest contributor to these contradictory results. We found that an application's choice of bundle size can affect performance by a factor of 24 times, and as high as 60 times when using solid state storage. We also identified the effect of storage delays on DRI performance.

We expect to see similar results in other low-powered, limited CPU devices, such as cell phones and smart phones. These results may be different when the DRI is run with a faster CPU(s) and with faster disks. However, we believe that our methodology can be applied to other hardware configurations, and to evaluate the performance of other mobile systems that utilize opportunistic wireless connections.

Our results lead to the following recommendations:

- DRI application developers should use the largest possible bundle size when using the DRI in-memory API.
- The DRI should be restructured to increase parallelism.
- When deploying the DRI, invest more in the CPU than faster storage and increased memory.

Our results suggest several potential improvements to DTN. In practice, sending a large number of bundles with delivery receipts (currently 54 bytes in size) enabled will reduce goodput on subsequent opportunistic connections. In this cases, we believe that performance would be greatly improved by chaining acknowledgement bundles into a single large bundle. We plan to explore this change to the DTN bundle protocol [12] in future work. While our experiments show that a 50 KB bundle size maximizes goodput, we have not explored bundle sizes greater than this size. We plan to modify the DRI in-memory API to support larger bundles. Given that the DRI is able to fragment bundles, we are interested in dynamic methods for finding the optimal bundle size for a given hardware configuration.

We also plan to explore the relationship between DRI performance and power consumption. For example, we observed that an increase of wireless data rate above 12 Mb/s offered no performance gain. We are interested in finding equivalent power optimizations and potential trade-offs between performance and power consumption.

7. ACKNOWLEDGEMENTS

We would like to acknowledge our supervisor Prof. Keshav, Shimin Guo, Aaditeshwar Seth, Prof. Brecht and Nilam Kaushik for their assistance and consultation throughout this project.

This research was supported by grants from the National Science and Engineering Council of Canada, the Canada Research Chair Program, Nortel Networks, Intel Corporation, and Sprint Corporation.

8. REFERENCES

[1] Delay tolerant networking research group. Available from: `http://dtnrg.org`.

[2] Oprofile - a system profiler for linux. Available from: `http://oprofile.sourceforge.net/news/`.

[3] Lexar professional compactflash memory cards, 2007. Available from: `http://www.lexar.com/digfilm/cf_pro.html`.

[4] Sensor networking with delay tolerance, 2007. Available from: `https://down.dsg.cs.tcd.ie/sendt/`.

[5] Sysstat, 2007. Available from: `http://perso.orange.fr/sebastien.godard/`.

[6] Test scripts and data set, 2007. Available from: `http://blizzard.cs.uwaterloo.ca/tetherless/images/5/55/Dtn_perf_eval_oliver_falaki.tar.gz`.

[7] Toshiba hard drives and optical drives, 2007. Available from: `http://sdd.toshiba.com/`.

[8] Umass dieselnet, 2007. Available from: `http://prisms.cs.umass.edu/dome/index.php?page=umassdieselnet`.

[9] Michael Demmer, Eric Brewer, Kevin Fall, Sushant Jain, Melissa Ho, and Robin Patra. Implementing delay-tolerant networking, 2004. Available from: `http://www.dtnrg.org/papers/demmer-irb-tr-04-020.pdf`.

[10] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, New York, NY, USA, 2003. ACM Press.

[11] Jonathan Kay and Joseph Pasquale. The importance of non-data touching processing overheads in tcp/ip. In *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*, pages 259–268, New York, NY, USA, 1993. ACM Press.

[12] K. Scott and S. Burleigh. Bundle protocol specification. *Work in progress (Internet-Draft draft-irtf-dtnrg-bundle-spec)*, December 2006. Available from: `http://www.ietf.org/internet-drafts/draft-irtf-dtnrg-bundle-spec-08.txt`.

[13] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *MobiCom '06: Proceedings of the 12th annual international conference on Mobile computing and networking*, pages 334–345, New York, NY, USA, 2006. ACM Press.