Delay-Tolerant Network Experiments on the MeshTest Wireless Testbed

Matthew R. Seligman Brenton D. Walker T. Charles Clancy
Laboratory for Telecommunications Sciences
College Park, MD, USA
seligman@ltsnet.net brenton@ltsnet.net clancy@ltsnet.net

ABSTRACT

Delay Tolerant Networks (DTNs) are a class of networks in which a contemporaneous end-to-end path from source to destination generally does not exist. Such networks use on a store-carry-forward communication model which relies on the mobility of nodes to transfer data between geographically separated nodes. DTN researchers have relied heavily on simulation for evaluation, due to the difficulty and expense of running live experiments with real devices running real DTN implementations.

MeshTest is a laboratory-based multi-hop wireless testbed that subjects real wireless nodes running real DTN implementations to reproducible mobile scenarios. It uses shielded enclosures and an RF matrix switch to dynamically control the attenuation experienced between pairs of nodes. The testbed is an ideal platform for DTN testing, offering convenient experimental control and data management.

We have installed the DTN2 Reference Implementation on wireless nodes within the testbed, and in this paper, we report on a series of experiments based on the well-known Data MULE model. Specifically, we investigate the effects of buffer limitations on the data MULEs and sensors node, velocity of the data MULEs, and bundle generation size and rate. We report results on message delivery rate and latency for varying experimental parameters. We found that an encounter between nodes does not guarantee a successful data transfer. In our experience, the quality and duration of the link, contention, and load on the nodes all influence its performance.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication; C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Algorithms, Performance, Experimentation

CHANTS'08, September 15, 2008, San Francisco, California, USA.

1. INTRODUCTION

Delay Tolerant Networks (DTNs) are a class of networks in which a contemporaneous end-to-end path from source to destination generally does not exist. Such networks use a store-carry-forward communication model which relies on the mobility of nodes to transfer data between geographically separated nodes. DTN applications include sensor networks, wildlife tracking networks, interplanetary networks, and user-to-user communication networks in harsh conditions with no communications infrastructure.

DTN researchers have relied heavily on simulation for evaluation, due to the difficulty and expense of running live experiments with real devices running real DTN implementations. Laboratory-based experiments do not reflect the dynamic link conditions nodes experience in the field or the mobility on the scale necessary to test a DTN because it is difficult to achieve indoors. On the other hand, field tests require a large number of subjects and coordination over large geographic areas to generate realistic experiments. Even with careful choreography, such experiments are only marginally reproducible, making testing and debugging difficult. Data management and experimental control are also issues with live tests. An out-of-band communication channel is necessary to avoid the interference between experimental data and control information.

MeshTest is a laboratory-based multi-hop wireless testbed that offers a hybrid between field testing and simulation [2]. The testbed features wireless nodes placed in shielded enclosures with their RF wired into a matrix switch of programmable attenuators. By dynamically adjusting the attenuations between the nodes, the nodes subjected to the effects of arbitrary physical arrangements and mobility scenarios. The testbed allows diverse variety of experiments to be run with real hardware and real implementations, while being able to closely monitor and control the nodes.

Since MeshTest is an ideal platform for DTN experimentation, we installed the DTN2 Reference Implementation on the testbed nodes and ran a variety of experiments similar to the well known Data MULE scenario [19]. This scenario features mobile data MULEs which provide store-and-carry forwarding from stationary data-generating **sensors** to an **access point (AP)** where all data is offloaded. We have generated an 8-hour mobility scenario and ran several experiments, varying parameters under these identical conditions. We report a variety of statistical results to better understand the effects of these experimental variations, including message completion rate, latency, and buffer size.

2. PRIOR WORK

In this section, we briefly describe some DTN applications and highlight prior work done in testing DTN algorithms on various platforms, such as simulations, emulations, and testbeds.

2.1 DTN Applications

Remote location Internet service Remote Internet service projects aim to provide asynchronous Internet access to users without access to infrastructure-based Internet service. In some cases, infrastructure-based Internet service is cost-prohibitive in remote or third-world regions, such as in DakNet [12], TIER [22], KioskNet [9, 18], and the Wizzy Digital Courier [24]. In other cases, traditional Internet service is not available due to the limitations of the technology used, such as the Sami Network [16], where there is no satellite coverage.

Wildlife sensor networks These sensor networks are intended to monitor animals in their natural habitat, be it land or sea, and migrate the data to access points and eventually back to researchers. Unique challenges exist in these networks, including the method of deployment and maintenance. Some examples are ZebraNet [7] and SWIM, a sensor network for whales [20, 21].

Urban Internet service Urban Internet DTNs aim to provide Internet access through the use of public mobile resources. UMassDieselNet [5] uses public buses in the city of Amherst, MA to provide asynchronous Internet service to users on buses and along bus routes.

Military communications The DARPA sponsored Disruption Tolerant Networking program [3] is developing DTN-based military tactical radio for situations where peer-to-peer communications are required and a communications infrastructure is not available.

2.2 DTN Research Approaches

We have found that the most common approaches to research and evaluating the performance of new DTN algorithms in challenged networking applications are:

- Simulation and emulation
- Fixed infrastructure testbeds
- Choreographed mobile tests

DTN simulations provide a method for quickly prototyping new algorithms and testing the performance of algorithms under a variety of conditions at relatively low cost. Many facets of a mobile, wireless network are abstracted, such as mobility, interference, fading, and real world issues encountered with real implementations, such as processing limitations on the nodes and implementation bugs. In addition, there are a growing number of different DTN simulators, and no particular simulator has become the defacto tool of choice. It is difficult to compare results from different papers due to the lack of simulator conformity (unlike the fairly standard use of ns-2 for TCP/IP research). The high level of abstraction in simulations leads to results that are of questionable realism, and the lack of conformity limits the usefulness of those results generated.

In [4] Emulab was used to test the DTN2 Reference Implementation under a variety of scenarios. The DTN2 implementation was compared to Sendmail and FTP in its ability to transfer data. Emulab provides a flexible platform to conduct repeatable experiments. The drawback of the emulation approach is that all aspects of the MAC, PHY, and RF environment are still simulated for mobile, wireless applications.

Figure 1: Assembly and connection of the shielded enclosures and RF switch.

Fixed infrastructure testbeds such as DieselNet and [11] provide realism by using an actual DTN implementation and real devices. The DTN2 Reference Implementation, freely available through the DTNRG [6], provides a common platform for experimentation. Its adoption by many researchers makes comparing results possible. Although generated results are real, they tend to be limited to one type of test, and once systems are deployed, they are difficult to monitor, modify and update.

Lastly, choreographed mobile tests provide real mobile and wireless network conditions while allowing for the use of the DTN2 reference implementation and real devices. Conversely, choreographed mobile field tests cannot be re-run arbitrarily and are impossible to precisely replicate. Some wireless testbeds have been considered for DTN experimentation such as the Roomba MADNeT testbed [10]. Although this platform is flexible, the physical test space is limited in size and the external wireless environment cannot be fully controlled.

Given the three approaches described above, there is a clear need for a testbed that can provide realistic mobile wireless network conditions, repeatability, and experimental control, while using the DTN2 reference implementation on real wireless devices. MeshTest addresses all of these needs to some extent, making it an ideal platform for testing mobile, wireless DTNs.

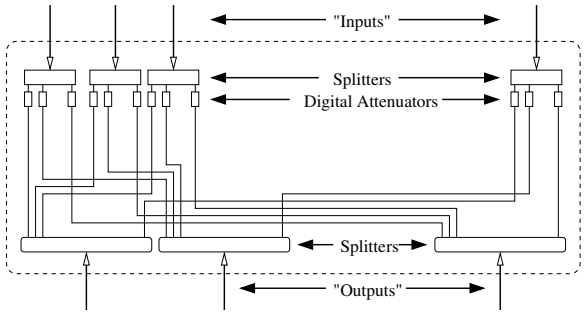

Figure 2: RF matrix switch diagram, showing n upper I/O ports, b lower I/O ports, and nb Ethernet-controlled digital attenuators with ranges 0-127 dB

3. MESHTEST TESTBED

MeshTest consists of a rack of 12 computers in shielded enclosures, an RF matrix switch, and a server that provides experiment control, as depicted in Figure 1. The RF from each computer's WiFi card is cabled through the enclosures and into the matrix switch of programmable attenuators. The enclosures prevent inadvertent cross-talk, and the matrix switch allows us to arbitrarily control the attenuation between the devices. A practical evaluation of the testbed's RF environment and mobility management can be found in [23].

Figure 2 shows the logical construction of an $n \times b$ switch. It has n inputs that connect through nb digital attenuators to b buses. Each bus has a direct, unattenuated, external connection. Note that the RF switch only simulates inter-node channel loss, and not propagation times.

While any device, from cellular telephones to software-defined radios, may be placed into the enclosures, the default configuration involves 802.11-based computers. Through a partnership with Rutgers, nodes and simulation management software from their ORBIT testbed [13] have been acquired.

The MeshTest testbed uses ORBIT's testbed management software to control the nodes. The additional piece necessary to make MeshTest work is software to map physical arrangements of nodes to the appropriate attenuator settings, and upload those settings to the switch in real-time during an experiment. A GUI-based program was developed to perform these functions. The program allows a user to visually place nodes in a two-dimensional space and draw their desired mobility patterns. The program can also load, display, and process mobility scenarios from XML files generated by other programs.

Programming the digital attenuator settings for the matrix switch is not as obvious as it may seem. In particular, taking an arbitrary physical arrangement of devices one can easily compute a matrix, L, of inter-node attenuations, but adapting these values for the switch's matrix of attenuators can be challenging. In Figure 2, we can see that n nodes connect through nb attenuators to b buses. Let S be an $n \times b$ matrix representing the settings of the nb attenuators. In [1], it is shown that finding appropriate attenuator settings is equivalent to finding S such that

$$\Lambda .* S^T S = L \quad (1)$$

where $.*$ is MATLAB notation for entry-wise multiplication of matrices, rather than standard matrix multiplication, and Λ is the insertion loss of the switch.

In [1] it is shown that one can use simulated annealing [15] to compute an approximate decomposition, S, for a variety of scenarios, including static topologies, mobile topologies, and situations that involve multi-path fading.

4. EXPERIMENTS

We setup and ran experiments analogous to the Data MULE model described in [19]. Specifically, we adopted the three-tier architecture, including the assumptions made about the functionality of each tier. This includes stationary access points (AP) and sensors, and mobile data MULEs. The key assumptions are:

- Each sensor is resource-constrained
- Date MULEs do not exchange data
- Mobility of Data MULEs cannot be controlled

4.1 Software

All nodes within the testbed are running version 2.5.0 of the DTN2 reference implementation available from the DTNRG website[6]. The DTN2 reference implementation includes an application called *dtnsend* for sending bundles and a receive bundle application called *dtnrecv*. When *dtnsend* is invoked, the generated bundle is stored by dtnd on the sensor until a data MULE comes within range. When a MULE comes into contact with the AP, as many bundles as possible are transferred to the AP and stored locally by dtnd. The *dtnrecv* application processes them asynchronously which may cause the application to fall behind the actual data transfer. This functionality is required for the applications to be delay-tolerant.

4.2 Routing

We used the DTN2 *static routing* module with the following two forwarding rules:

- Sensors only forward data to MULEs
- MULEs only forward data to the AP

Each data MULE periodically sends out discovery announcements, and all nodes listen for these announcements. When a node detects a MULE, it opens an *opportunistic* link. All opportunistic links remain in the routing table until dtnd is restarted, though they are marked as "Unavailable" once the MULE is out of range. This discovery scheme was used for two reasons. First, as [19] states, sensor are assumed to be power constrained and should minimize their transmissions. Second, we have observed that if a sensor discovers the AP, the sensor would never pass any bundles to the MULEs. There is a certain probability that a sensor would pick up a discovery announcement sent by the AP, even if they were outside practical communication range. Therefore only the MULEs are allowed to send out discovery announcements.

4.3 The Base Case Configuration

We ran one experiment that serves as the control, or **base case** experiment, and all other experiments vary in one aspect from the base case. In this section, we describe in detail the base case configuration.

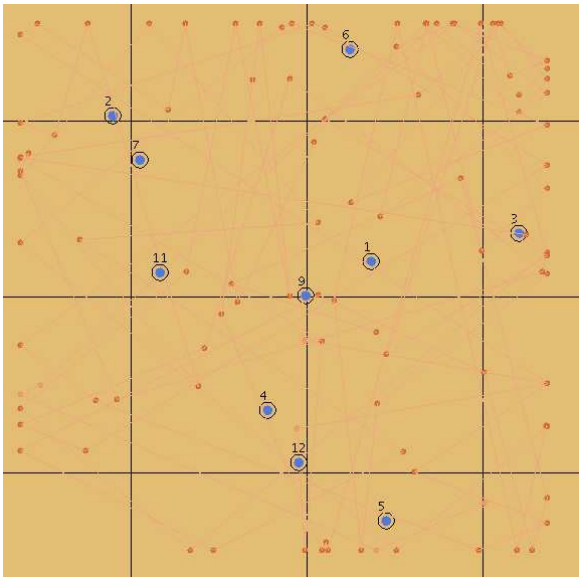

Figure 3: Node arrangement and mobility pattern for the first two hours of the experiment. Nodes 1-7 are stationary sensors, node 9 is the AP, and nodes 11-12 are mobile data MULES. The red dots along the MULE paths are points where the boundary was encountered, or a new random direction was chosen.

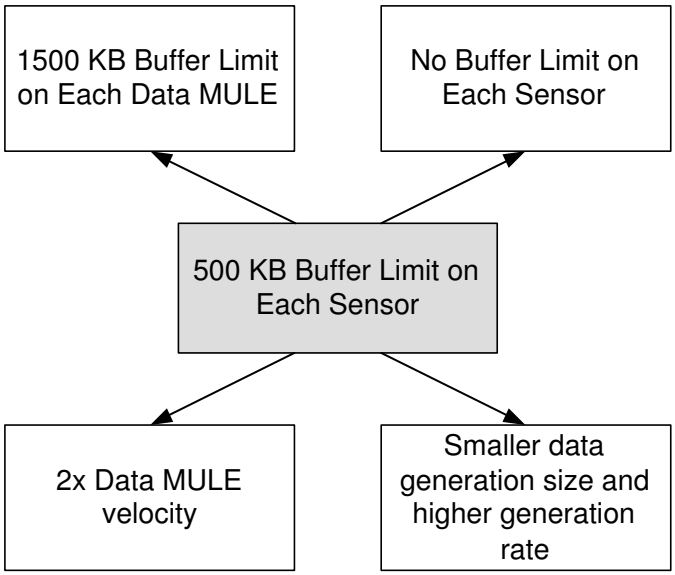

Figure 4: The relationships between our different experiments. The middle gray box denotes the control, or *base case* experiment, and all other experiments vary a single aspect of the base case.

4.3.1 Data Generation

Each sensor in the experiment generates bundles according to a Poisson process with intensity $\lambda = 0.0333/sec$, i.e. on average one bundle every 30 seconds. The size of the bundles is normally distributed with mean $\mu = 10$KB and variance $\sigma^2 = 5$KB. The data generator on each node uses the same random seed for each experiment to ensure repeatability. The random seeds are based on the nodes' IP addresses.

4.3.2 Node Positioning

The position of each of the seven sensors was chosen uniformly randomly in a 3km $\times$ 3km area. The AP is placed in the center of the area. The initial position of each MULEs was also chosen uniformly randomly. A single arrangement of nodes was chosen and used for all experiments. The positions of all nodes during an experiment are shown in figure 3.

4.3.3 Node Storage

Each sensor's bundle storage is restricted to 500KB. In the base case, no limits are placed on the storage space of the MULEs.

4.3.4 Mobility

The data MULEs follow a Random Direction mobility pattern [14]. On a bounded square, this model gives a uniform long-term spatial distribution. That is, the model does not suffer from *edge effects*, which would tend to disadvantage nodes close the edges of the test area. In our implementation, each MULE chooses a direction θ uniformly from $[0, 2\pi)$, a velocity v from [20km/h, 40km/h], and a trip duration from [2min, 10min]. Then, each MULE travels in the chosen direction at the chosen velocity for the chosen amount of time and repeats this process. When a node encounters the boundary it reflects, that is, the component of the velocity perpendicular to the boundary is reversed.

Figure 3 shows the paths followed by the MULEs during the first two hours of the eight hour experiment.

4.4 Experiment Methodology

In this section we describe the rest of our experiments. Figure 4 visualizes the relationships between the different experiments. The following is a list of parameters that we experimented with:

- **Buffer Capacity:** The base case has a 500KB limit on each sensor and no limits on the MULEs or AP. One of the other experiments removes all buffer capacity limits, and another experiment adds 1500KB buffer limits to the MULEs.
- **Data MULE Velocity:** In the base case data MULEs choose their velocity uniformly from [20km/hr, 40km/hr]. We ran one experiment which doubled the speed of the MULEs, so they fall in the interval [40km/hr, 80km/hr].
- **Data Generation:** The data bundles are generated according to a Poisson process, on average once every 30 seconds, with normally distributed sizes. We ran one other experiment in which 341 Byte bundles were generated on average every 1.0 seconds. This yields the same average data generation rate as the base case.

4.5 Metrics

The following metrics are used to evaluate and compare results among the experiments:

- Bundle Completion Rate (BCR)

$$BCR = \frac{\#\text{ Unique Received Bundles}}{\#\text{ Generated Bundles}}$$

- Data Completion Rate (DCR)

$$DCR = \frac{\#\text{ Unique Received Bytes}}{\#\text{ Generated Bytes}}$$

- Mean Latency - Only data received by *dtnrecv* is included in this metric. Therefore, data residing at its destination node not processed by the application is not included.
- Buffer Usage - Amount of storage used on a node at a given time
- Time-Weighted Network Storage [17] - Average amount of buffer usage, weighted by time. In a DTN, data may be stored for a long period of time, so the nominal buffer usage is not always of particular interest. In many cases, we are particularly interested in how long a node's buffer usage is high or at capacity, which is more accurately captured by the time-weighted network storage.
- Time Spent at Capacity (TSC) - Amount of time a node's buffer size is at its capacity

5. RESULTS

Each experiment lasted 8 hours. This means the experiments actually ran for 8 hours since the testbed runs in real-time. The first 26,000 seconds of results were used for our analysis.

5.1 Base Case

As described in Section 4, we put a 500KB buffer limit on each sensor. On average, the data generation rate is 0.33KB/sec. Therefore, a buffer capacity limit of 500KB was selected to ensure that multiple sensors would operate at or near capacity during the experiment.

Figure 5 shows the buffer usage vs. time for Node 3 and Node 4. Node 3 spends the majority of the time from $8000-15000$s at capacity. When a node is at capacity any newly generated data by is dropped due to insufficient storage. On the other hand, node 4 did reach capacity several times, but in all cases, a MULE was nearby to offload some or all of its data. Therefore, the TSC of node 3 is significantly higher than the TSC of node 4. In fact, table 1 shows that the node 3 TSC is more than five times that of node 4. Both nodes experience instances of *partial offloading*, where a MULE was able to take some, but not all of the sensor's data.

In table 1 we observe a correlation between the mean latency and the TSC, and inverse relationship between BCR and mean latency. This is what we would intuitively expect, as sensors that have low TSC must be visited frequently by the MULEs, leading to more frequent bundle deliveries, and the buffering of younger bundles. If the sensors used a different queueing strategy we might see the correlation between TSC and latency disappear.

Based on comparisons with other base case runs, we found that node 2 suffered from several failed connection opportunities in this experiment. This led to poorer than usual results for node 2, and a discrepancy between nodes 2 and 7, despite their physical proximity.

5.2 No Buffer Limit on Sensors

This experiment is identical to the base case, except there was no limit on the sensors' buffer capacity. The buffer usage vs time for three sensors is plotted in figure 6. We observe the steady increase in buffer usage due to data generation, and the occasional sharp decreases corresponding to a data

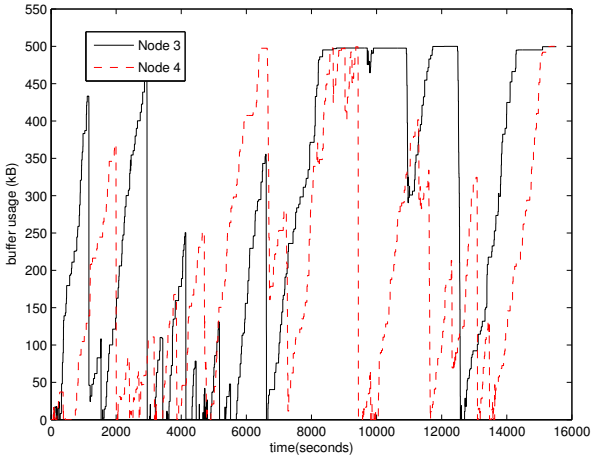

Figure 5: Base case buffer usage for nodes 3 and 4. Both nodes reach capacity at some point, and both experience instances of partial offloading.

$Node$	BCR	DCR	$Latency(s)$	$TSC(s)$
4	0.9250	0.9209	1214.73	1956
1	0.8818	0.8758	1356.83	1708
7	0.8369	0.8211	1775.95	3820
5	0.6118	0.6080	2104.59	9273
2	0.5894	0.5784	2735.99	11747
6	0.5114	0.4948	2797.73	8402
3	0.4889	0.4742	3032.53	11007

Table 1: BCR, DCR, Mean Latency, and TSC for Sensor Nodes with 500 KB Buffer Limit Sorted by BCR

MULE coming within range. Given our particular mobility scenario, node 7 has less frequent contacts with the data MULEs in comparison to the other nodes. This is especially true during the time period, $t = 0-5000$, during which time its buffer usage grows to over 1.5MB.

In comparison to the base case this experiment shows a significant increase in BCR and a decrease in latency, especially for nodes that are further from the AP. This is what we would expect, as the sensor nodes are not forced to drop bundles when their buffers reach capacity. In comparison to the base case results, we do not observe any correlation between mean latency and BCR.

In this experiment nodes 2 and 7 both perform normally and have very similar results. On the other hand node 1 appeared to malfunction in the last hour of the experiment, driving its results down.

5.3 Buffer Limit on Sensors and Data MULEs

In this experiment we put a 1500KB limit on the the Data MULE buffers. This is three times the limit on the sensors. We chose this limit expecting that the MULEs would frequently be at capacity, and hoping that this fact would yield contrasting results. Results are shown in table 3. When a MULE is at capacity it will not accept any bundles from the

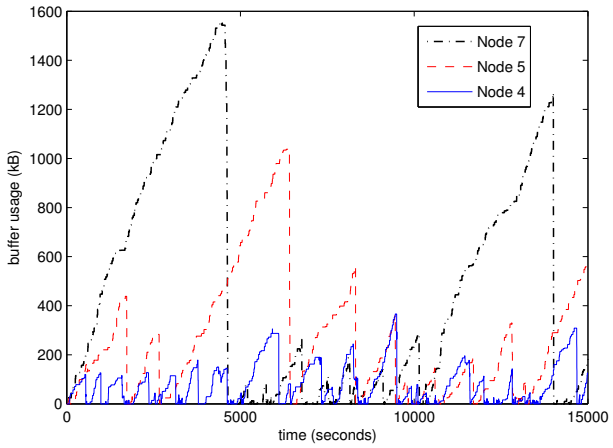

Figure 6: No buffer limit on sensors. We observe that some nodes have more frequent contact with the data MULEs.

Node	*BCR*	*DCR*	*Latency*(*s*)
4	0.9789	0.9787	841.97
7	0.9717	0.9755	1393.56
2	0.9635	0.9625	1723.43
6	0.8788	0.8729	1544.54
5	0.8692	0.8677	1732.58
3	0.8517	0.8638	1676.26
1	0.7996	0.8001	1130.58

Table 2: BCR, DCR, and Mean Latency for Sensor Nodes With No Buffer Limit, Sorted by BCR

sensors. As expected, the BCR and DCR for each sensor node tends to be worse than in the *base case*.

Another effect of the data MULE buffer limitation is starvation. We observe that nodes further from the AP tended to have higher TSC statistics. We hypothesize that data MULEs are often at capacity by the time they reach the more distant nodes. We believe that under such conditions, some mechanism, such as rate-limiting or a more preemptive scheme, such as WRED, needs to implemented to enforce fairness and allow all sensor nodes to offload data. Without this mechanism in place, this scenario always favors sensor nodes that are seen by the data MULE soon after it visits the AP. We leave the research and implementation of a traffic management scheme as future work.

Table 3 includes TSC figures for the MULEs. In our experiments we have observed that MULE node 12 tends to pick up and deliver more bundles than node 11, and this is reflected in the MULEs' TSC values. Node 12 has a full buffer for roughly one third of the experiment, and during this time, it cannot accept any new data.

5.4 Double MULE Velocity

In this section, the velocity of each data MULE is doubled. The mobility pattern used in the other experiments is repeated twice, taking only four hours to complete each time. Results are shown in table 4. One would expect that

Node	*BCR*	*DCR*	*Latency*(*s*)	*TSC*(*s*)
1	0.7286	0.7075	966.47	414
7	0.7146	0.7154	1080.86	787
4	0.7046	0.7043	761.82	0
3	0.6689	0.6530	1835.49	2688
2	0.6376	0.6313	1510.47	3806
6	0.5731	0.5568	1702.47	4082
5	0.4697	0.4693	2071.72	7224
11 (MULE)				5481
12 (MULE)				8516

Table 3: BCR, DCR, Mean Latency, and TSC for Sensor Nodes with 500 KB Buffer Limit and Data MULEs with 1500 KB Buffer Limit Sorted by BCR

by increasing the data MULEs' velocity, there will be more frequent offloading opportunities for the sensor nodes, and more frequent opportunities for the data MULEs to offload data to the access point. We observe that TSC values are trivial for most nodes (nodes 5 and 1 malfunctioned for varying periods of the experiment). The mean latency values are generally less than half what we see in the base case. Note that in previous experiments when a sensor reached capacity the youngest bundles would be dropped, and in this experiment those bundles are more likely to be delivered.

Node	*BCR*	*DCR*	*Latency*(*s*)	*TSC*(*s*)
7	0.9932	0.9921	666.54	0
2	0.9800	0.9776	753.59	201
4	0.9789	0.9810	463.02	0
3	0.9665	0.9674	885.74	19
6	0.9440	0.9443	883.41	0
1	0.8637	0.8572	915.47	3592
5	0.3329	0.3345	42228.13	17443

Table 4: BCR, DCR, Mean Latency, and TSC for Sensor Nodes with Data MULEs Moving at 2x Velocity Sorted by BCR

This experiment revealed that the quality of the contacts can play a role in network performance. For example, the experiment reported above was run with MULEs broadcasting discovery announcements every second. In comparison, figure 7 shows results for a double-speed experiment where MULEs sent discovery announcements every five seconds. In this case the node misses several opportunities to establish a link to the MULE, and actually performs worse than in the base case. The regular-speed experiments were not as sensitive to changing the discovery interval.

5.5 Decrease Bundle Size and Increase Bundle Generation Rate

In this experiment we modified the data generation process to maintain the same average data generation rate, but split the data into many small bundles. All bundles in this experiment are 341 bytes, and the generation process is still Poisson, but with an intensity of $\lambda = 1.0/sec$. Since we maintain the same average data generation rate, it is still

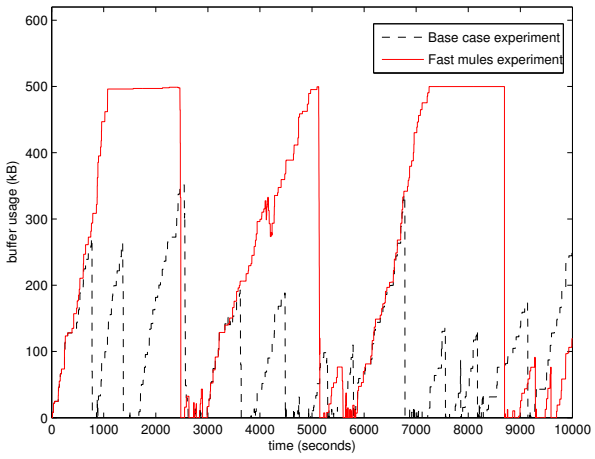

Figure 7: Buffer usage for the 2x velocity MULE experiment with five second discovery interval compared to base case. Despite the more frequent encounters this node had fewer successful uploads to the MULE.

reasonable to make comparisons to the base case. For all nodes, the results in table 5 exhibit lower BCR and higher mean latency compared to the base case. Part of the reason for this seems to be the overhead associated with processing so many bundles. We have observed that when a MULE is near the AP, it will often be running with load average 1.50 or above, and still be unable to move all of its bundles to the AP during the encounter. After about three hours both MULEs seem to be completely overwhelmed and their buffers soar to over 10MB, which corresponds to over 30,000 bundles. Of course our test parameters may seem extreme, but it does show that when dealing with many small bits of data, some sort of data consolidation may be necessary. We note that this sort of behavior would be difficult to accurately model in a simulation.

Node	*BCR*	*DCR*	*Latency(s)*	*TSC(s)*
4	0.5307	0.5307	2472.05	2520
2	0.5199	0.5199	2448.75	1336
7	0.5136	0.5136	2653.64	2846
3	0.4985	0.4985	2588.42	2126
1	0.4530	0.4530	2401.97	1552
6	0.4063	0.4063	3110.20	2314
5	0.1742	0.1742	8573.97	6473

Table 5: BCR, DCR, Mean Latency, and TSC for Sensor Nodes with Bundle Size of 341 bytes and Bundle Generation Rate $\lambda = 1/sec$ Sorted by BCR

6. CONCLUSION

In this paper, we conducted original experiments using the MeshTest testbed and the DTN2 reference implementation. We focused on the Data MULE scenario in which stationary sensors generate data that is aggregated onto mobile data MULEs, and eventually offloaded to an AP. The *base case* experiment, which was derived from the original Data MULE work [19], provided a foundation of results to compare to in terms of completion rate, buffer usage, and time spent at capacity. From the *base case*, we varied several parameters including relaxing the buffer capacity limitation, adding a buffer limitation to the data MULE, and increasing the velocity of the data MULEs.

As expected, removing the buffer limitation on sensors led to better network performance as less data had to be dropped by the sensors. Also doubling the MULE velocity led to more transfer opportunities and better performance, as long as the links were discovered quickly enough to take advantage of them. This demonstrates the need to consider an efficient data exchange algorithm when contact times are small.

When limiting the buffer capacity at the data MULE, we found that some sensor nodes were starved, as the MULEs were providing unfair service to the sensors. This result shows the need for some type of traffic management scheme to provide fair service to all sensor nodes. In addition, the buffer-limited data MULE was biased towards higher latency values in its results, since at capacity, the MULE would always hold older bundles rather than pick up younger ones.

In all experiments, we observed a non-negligible bundle duplication rate. This demonstrates the utility of a duplication detection mechanism at a higher layer than the convergence layer, such as *custody transfer* included in the DTN bundle protocol specification [8].

With real hardware and software comes inevitable malfunctions, and we have observed these in our experiments. Out of several dozen experiments we had very few runs where all nodes performed as expected for the entire experiment. For completeness these results have been included.

Finally, we demonstrated that MeshTest is an ideal evaluation platform for real DTN implementations. It allows us a great deal of realism, as it uses real implementations and real hardware, and also features repeatability and experimental control. We expect that experiments and testing carried out on MeshTest will lead to better, more reliable DTN implementations, and more reality-focused research.

7. FUTURE WORK

Since the purpose of these experiments was to create an initial DTN testing capability on MeshTest, there is a great deal of future work necessary to provide mature, reliable results. First, we need to better understand the functionality and behavior of the DTN2 reference implementation. During our experimentation, we often found it difficult to monitor the real status of the dtnd daemon, especially when tracking which bundles were residing in a particular node's storage. We also observed certain node behaviors that we suspect are bugs. The testbed will be an ideal platform for debugging implementations such as DTN2.

In addition to the experimental setup, there are several items tagged as future work throughout this paper. In particular, we noted that the non-trivial number of duplicate bundles demonstrate the need to study the effects of *custody transfer* mechanisms. Another interesting future task, particularly for the data MULE scenario, is to identify an approach allowing fair service to all sensor nodes in a DTN when data MULE buffer capacity limits exist. For sensor node applications, the data collected would be biased if they

were collected from a biased subset of the sensor nodes. Therefore, research on traffic management schemes (i.e. how much data a sensor node is allowed to offload during a given contact with the MULE) is necessary.

Lastly, the data MULE scenario is representative of a subset of DTN applications, so for future experiments, we intend to study other DTN scenarios as well, such as message ferries [25] and limited or no knowledge DTNs using multi-copy routing algorithms. The work presented in this paper provides an initial demonstration of the utility of this platform for DTN research.

8. REFERENCES

[1] T. Clancy and B. Walker. Meshtest: Laboratory testbed for large wireless topologies. IEEE TridentCOM 2007.

[2] T. C. Clancy and B. D. Walker. Meshtest: Laboratory-based wireless testbed for large topologies. In *IEEE TridentCom 2007*, pages 1–6, 2007.

[3] DARPA Disruption Tolerant Networking Program. `http://www.darpa.mil/sto/strategic/dtn.html`.

[4] M. Demmer, E. Brewer, K. Fall, S. Jain, M. Ho, and R. Patra. Implementing Delay Tolerant Networking. Technical Report IRB-TR-04-020, Intel Research Berkeley, 2004.

[5] UMassDieselNet. `http://prisms.cs.umass.edu/dome/`.

[6] DTN Research Group. `http://www.dtnrg.org/`.

[7] P. Juang, H. Oki, Y. Wang, M. Margaret, P. Li-Shiuan, and R. Daniel. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS-X*, 2002.

[8] K. Scott and S. Burleigh. Bundle Protocol Specification. *IETF RFC 5050*, 2007.

[9] KioskNet. `http://blizzard.cs.uwaterloo.ca/tetherless/index.php/KioskNet/`.

[10] Roomba MADNeT : A Mobile Ad-Hoc Delay Tolerant Network Testbed. `http://dna-pubs.cs.columbia.edu/citation/paperfile/150/reich_MC2R.pdf`.

[11] E. Oliver and H. Falaki. Performance evaluation and analysis of delay tolerant networking. In *MobiEval '07: Proceedings of the 1st international workshop on System evaluation for mobile platforms*, pages 1–6, New York, NY, USA, 2007. ACM.

[12] A. Pentland, R. Fletcher, and A. Hasson. DakNet: Rethinking Connectivity in Developing Nations. In *IEEE Computer*, 2004.

[13] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. IEEE WCNC 2005.

[14] E. M. Royer, P. M. Melliar-Smith, and L. E. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *IEEE ICC 2001*, volume 3, pages 857–861, 2001.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[16] Sami Network. `http://www.snc.sapmi.net/`.

[17] M. Seligman, K. Fall, and P. Mundur. Storage routing for dtn congestion control. *Wireless Communications and Mobile Computing*, 7:1183–1196, 2007.

[18] A. Seth, D. Kroeker, M. Zaharia, S. Guo, , and S. Keshav.

[19] R. Shah, S. Roy, S. Jain, and W. Brunette. Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks. In *IEEE SNPA*, 2003.

[20] T. Small and Z. J. Haas. The shared wireless infostation model: a new ad hoc networking paradigm (or where there is a whale, there is a way). In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 233–244, New York, NY, USA, 2003. ACM.

[21] T. Small and Z. J. Haas. Resource and performance tradeoffs in delay-tolerant wireless networks. In *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 260–267, New York, NY, USA, 2005. ACM.

[22] TIER Project. `http://tier.cs.berkeley.edu/`.

[23] B. Walker and T. C. Clancy. A quantitative evaluation of the meshtest wireless testbed. In *TridentCom 2008*, March 2008.

[24] Wizzy Project. `http://www.wizzy.org.za/`.

[25] W. Zhao, M. Ammar, and E. Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *ACM MobiHoc*, 2004.