A Demonstration of the MeshTest Wireless Testbed for Delay-Tolerant Network Research

Brenton D. Walker Ian D. Vo Matthew Beecher Matthew Seligman
Laboratory for Telecommunications Sciences
College Park, MD, USA
brenton@ltsnet.net idv2101@columbia.edu beechema@iastate.edu seligman@ltsnet.net

ABSTRACT

MeshTest is a laboratory-based multi-hop wireless testbed that can subject 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 the testbed nodes, and and have been using it to run experiments. Our current experimental scenarios are similar to the well known Data MULE and Message Ferry models, but a large variety of other experimental scenarios are possible.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Measurement

1. PURPOSE OF THE DEMO

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 often use 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 nodes and real DTN implementations.

Our goal is to demonstrate the MeshTest testbed in operation, and show that it is an ideal platform for DTN experimentation, development and debugging. We have installed the DTN2 Reference Implementation on the testbed nodes and have run a variety of experiments. A wide variety of other experiments are possible, however. It is also possible to place other wireless devices, including most laptops, inside the shielded enclosures to test different types of DTN hardware and software. We hope that other workshop participants have DTN implementations that they would be willing to try on the testbed.

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

2. TESTBED DESCRIPTION

MeshTest consists of a rack of 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.

CHANTS'08, September 15, 2008, San Francisco, California, USA.
ACM 978-1-60558-186-6/08/09.

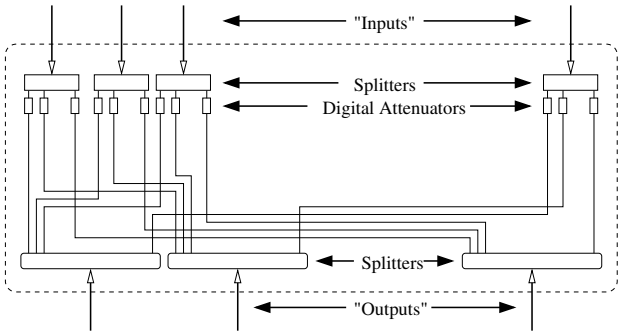

Figure 2: RF matrix switch diagram, showing n upper I/O ports, b lower I/O ports, and nb Ethernet-controlled digital attenuators with ranges 0-127 dB

The enclosures provide roughly 80dB of isolation, preventing inadvertent cross-talk.

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 software from the ORBIT testbed [3] have been acquired.

The matrix switch allows us to control the attenuation experienced between the devices. Figure 2 shows the logical construction of an $n \times b$ switch. It has n inputs, each of which is split b ways and fed through a digital attenuator attenuators to one of b buses. In our current switch there are 16 inputs and 4 busses, for a total of 64 attenuators. Each bus has a direct, unattenuated, external connection which we do not currently use, except for testing and calibration.

Note that the RF switch only simulates inter-node channel loss, and not propagation times. A practical evaluation of the testbed's RF environment can be found in [5].

3. TESTBED CONTROL SOFTWARE

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.

The RF matrix switch can be accessed via a TCP socket. Further, attenuation levels can only be set one at a time. Entering these settings by hand becomes unwieldy when dynamic simulations involving multiple nodes need to be conducted in real time.

The issues the control software needs to address are:

- Keep track of the locations and movements of each individual node
- Calculate the effective attenuation between two nodes based on the physical scenario
- Generate and apply switch settings in real time

To solve these issues, we developed a Java Swing application to provide a portable, graphical interface in which users can place nodes onto a "map", represented by a default grid of 1 by 1 kilometer sections. To facilitate a variety of scenarios, the scale of the map can be changed, with new grids overlain to provide perspective of the changing scale. Additionally, each node can be given a trajectory, designated by a series of target destinations. Nodes can travel at different speeds (0-100km/h) between different pairs of destinations, rest at a destination for a specified time, and toggle their status (active/inactive).

As the simulation is run, users can view the wireless nodes traveling along their trajectories in real time. Every 3 seconds, a new inter-node distance matrix is generated based on the current node arrangement, and a new path loss matrix, L is computed by applying the free-space path loss equation After the desired path loss matrix, L, is obtained, we compute an approximate decomposition of the form $A^T A$, where A is a lower rank matrix containing the actual attenuator settings to be applied to the switch. More mathematical details can be found in [1].

Decomposition is achieved through simulated annealing. The algorithm uses a probabilistic, state-based strategy that allows for a quick approximation of a global minimum. Starting with a base state, each step of the algorithm permutes the state to a random "neighbor" state, with probabilistic preference given to states representing better approximations. As the annealing progresses the range of each step is decreased, eventually converging when the step range reaches zero. By allowing the occasional move to a less optimal state, the algorithm can avoid getting trapped in local minima.

4. DTN2 DESCRIPTION

All nodes are running version 2.5.1 of the DTN2 reference implementation, available from the DTNRG website [2]. The default configuration settings are used, except for routing and neighbor discovery. Static routing was used in these experiments since each node had a predetermined role as a source, sink, or intermediate router. For example, in the Data MULE scenario, each source node, also referred to as a sensor node, contained routes to the sink through each Data MULE. For the Message Ferry scenario, routes are added to allow sensors to send and receive from one another via the message ferry.

The DTN2's link discovery functionality is used to dynamically add newly available links and deactivate recently disconnected links in its current routing table. The MULEs and ferrys were configured to send discovery announcements every second, and the sensors and APs were configured to listen for announcements.

The DTN2 daemon uses the TCP convergence layer to communicate. In theory, for scenarios with short contact times such as Data MULEs or Message Ferries, UDP may provide a more efficient transport service. An attempt to use the UDP convergence layer was made, but we ran into link discovery issues that we could not quickly resolve and leave this as future work.

The DTN2 reference implementation includes a variety of applications, such as *dtnsend* for sending bundles, and a receive bundle application called *dtnrecv*, both of which we used. Both applications process data asynchronously using the local disk to stage data. For example, if dtnsend sends a bundle, the bundle is placed in the node's send queue and dtnsend returns regardless of whether or not the bundle was actually transmitted. Along with these applications, Perl scripts are used to generate bundles at sensors and process received bundles at sinks.

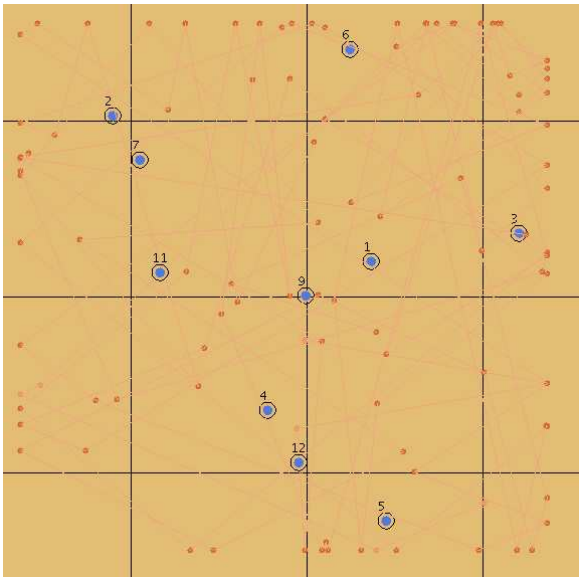

Figure 3: Node arrangement and mobility pattern for the first two hours of the experiment. Nodes 1-7 are stationary sensors, node 9 is the AP, and nodes 11-12 are mobile data MULES. The red dots points where a MULE changed direction.

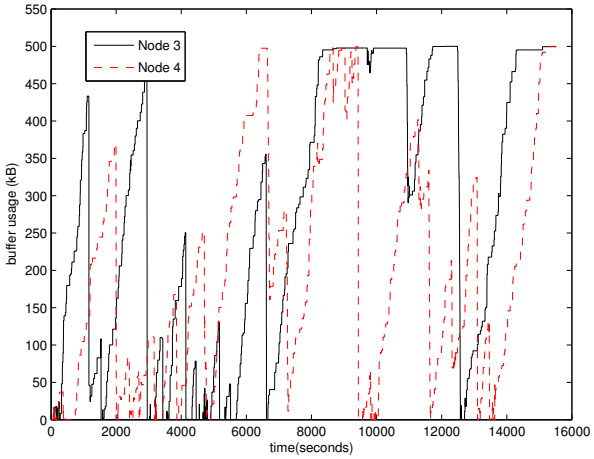

Figure 4: Example of two nodes' storage usage during a Data MULE experiment.

5. EXPERIMENTAL SCENARIOS

5.1 Data MULE Scenario

We set up and ran experiments analogous to the Data MULE model described in [4]. Specifically, we adopt 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 is not coordinated

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.

5.2 Message Ferry Scenario

Experiments based on simplified versions of the Message Ferry scenario [6] were designed and run on the MeshTest testbed. Our experiments use stationary sensor nodes that need to communicate with each other, but lack direct connections. Mobile message ferries provide a store-carry-forward service to facilitate communication. These scenarios contained 8 sensor nodes, each set up in a different zone, where a zone is a 1km x 1km grid square. Sensor nodes were set to listen for discovery announcements produced by the two message ferries. Sensor nodes were not set up to look for or setup links directly to other sensor nodes. When a node detected a ferry it opened an opportunistic link and added the link to its routing table.

Message ferries were initially placed in a random zone and were set to move at a constant speed of 30 km/hr to another randomly selected zone. Once ferries reach their target zone, they immediately select another random zone and begin moving to their new destination. When a link was created by a sensor node, ferries sent bundles destined to the connected node and received bundles generated by the sensor until either all bundles were transferred or the link became unavailable due to message ferry movement. Destinations for each bundle were selected at random.

5.3 Data Analysis

We have written perl scripts to collect and parse experimental output for data analysis, though nothing precludes a user from collecting and analyzing data in any particular way. Figure 4 shows an example of results gathered from a Data MULE experiment comparing two nodes' storage usage. Other statistics we have examined include mean latency and message success rate.

6. REFERENCES

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

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

[3] 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.

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

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

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