Disruption Tolerant Shell*

Martin Lukac
UCLA CENS
3563 Boelter Hall
Los Angeles, CA 90095
mlukac@cs.ucla.edu

Lewis Girod
MIT CSAIL
32 Vassar St.
Cambridge, MA 02139
girod@nms.csail.mit.edu

Deborah Estrin
UCLA CENS
3563 Boelter Hall
Los Angeles, CA 90095
destrin@cs.ucla.edu

ABSTRACT

Wireless network technology is being applied to a wide range of scientific and engineering problems and across a wide dynamic range of spatial scales. When node placement is constrained by the application (e.g, coupled to sensor placement needs), and can not rely on pre-existing infrastructure (e.g., cellular infrastructure or power-lines), such systems may experience erratic link qualities and intermittent node disconnection. These characteristics, combined with unpredictable environmental conditions, make it difficult to rely upon traditional end to end connections for regular high bandwidth data acquisition and for system management and configuration. We have implemented and deployed such a "challenged network" system of 50 nodes for use by seismologists along a part of the Mesoamerican Subduction Experiment (MASE) broadband seismic array, stretching 500 KM from Acapulco to Tampico through Mexico city. In addition to supporting Delay Tolerant data transfer of relatively high bandwidth seismic data, our system includes a reliable asynchronous remote shell interface (referred to as Disruption Tolerant Shell, DTS) to accomplish the management on these types of system. We present the implementation of this solution and its evaluation on a 13 node portion of the MASE network.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management; C.2.2 [**Network Protocols**]: Applications

General Terms

Design, Management, Performance

Keywords

Delay tolerant networking, ad-hoc networks, wireless sensor networks, system management

*This material is based upon work supported by the National Science Foundation under Grant No. CCF-0120778.

SIGCOMM'06 Workshops September 11-15, 2006, Pisa, Italy.

1. INTRODUCTION

As existing wireless technology is being applied to a wider range of scientific and engineering problems, it is becoming more difficult to rely upon traditional end to end connections for regular high bandwidth data acquisition and for system management and configuration. Sensor placement is necessarily determined by the application, with secondary consideration to connectivity. For deployments located far from pre-existing infrastructure such as cellular systems, powerlines, or wired network access, the deployed system must create its own network infrastructure. In our case, scale and placement requirements imposed by the application substantially reduce the feasibility of provisioning a high availability *end to end* network.

Creating end to end connectivity is more than just an issue of hardware costs. Each station requires permission, installation, adequate solar power, and protection. Therefore nodes are frequently placed at a distance that push the capabilities of the wireless links. These "stretched" links are particularly sensitive to environmental transients and therefore the resulting network is "challenged" in that we can not count on high availability end to end paths. For example on a path A-B-C-D-E, each link may be available 95% of the time, but because the disruptions are not necessarily correlated, the end to end path availability is 81%. Our application requires every bit of data to be delivered and all management tasks to be reliably completed within the time it takes to establish and maintain a reliable end to end connection.

Patterns of poor links, disconnections, and disruptions can make it difficult to obtain an end to end connection a sufficient percent of the time to achieve necessary bandwidth and latency needs. With such variability in the network, the existing Delay Tolerant Networking (DTN) techniques [6] work well for data delivery, but existing system management methods and tools fail. While these conditions may not be the common case and end-to-end solutions work much of the time, they may fail at the times when configuration and management is most necessary. These tools are what we describe as "online" applications: they expect reliable end to end links with low latencies. Adapting these tools to work in challenged network environments requires changing the way the tool fundamentally operates, changing the underlying network services model, or both. We have designed and deployed a system to achieve the required application performance of delivering sensor data and managing nodes over such a network that experiences erratic link qualities and intermittent node disconnections.

The Mesoamerican Subduction Experiment (MASE) broadband seismic array [3] is a challenged network. MASE consists of 100 seismic stations stretching 500 KM from Acapulco to Tampico via Mexico city. Of these 100 stations, 50 are stand-alone data-logger systems, while 50 are part of an experimental networked sensing system. The networked nodes are based on the Stargate [10] platform and are networked to peers over 5-10 Km distances using hi-power 802.11B cards and directional antennae. In some cases, the best network topology reflects the physical topology, and the result is a tree like configuration. In other cases, the network topology is more complex, particularly when trade offs were made for a good sensor location. In all cases, relay nodes were required.

Because of poor links and other disruptions, end-to-end performance in this network falls off rapidly as the number of hops increases. This has led us to use DTN techniques for data transfer rather than multiple parallel end to end connections. The sensor data is buffered, stored into bundles, and transferred hop by hop until it reaches a sink node. Our implementation of this technique only changes the way a data delivery tool operates and not the underlying network services: we use TCP to transfer the data bundles between hops. In addition to delivering the data, we add meta-data to the bundles as they are transferred between links to track the movement of the data and to collect information about the individual links.

System management beyond the first few hops into the network becomes difficult as end to end connections become extremely high latency and unreliable. The goal with system management is to perform a management task on all the nodes in the network or to query system information from all the nodes in the network without disrupting the data movement. To accomplish this, we adapted an existing management tool, the remote shell, by changing the way it fundamentally operates. We pair this new type of shell with a new underlying network service call StateSync. StateSync is a reliable and efficient publish-subscribe mechanism that provides a low latency transport for state dissemination similar to DTN. The result of the combination is the Disruption Tolerant Shell (DTS).

DTS uses StateSync to reliably disseminate shell commands and scripts and to return their results. DTS specifically addresses the situations where end to end connections fail at critical times, are intermittent, or are just not possible. DTS provides a tractable management environment: it enables the user to issue commands once and be certain that all nodes will execute them, whenever and however they manage to get connected. The majority of the time, DTS will have lower latency than an end to end management system, including the cases where the end to end systems fail to establish and sustain connections. The remainder of the time DTS will have comparable latency to and end to end system.

DTS is currently deployed on a 13 node network that begins in Cuernavaca, as shown in Figure 1. Section 2 covers related work. Section 3 provides some analysis of the deployed network. Section 4 describes the implementation of DTS. Section 5 discusses our evaluation of DTS.

2. RELATED WORK

DTN ideas and techniques used in our system are drawn from Interplanetary Internet [4] research and other DTN architectures [6]. In particular the idea of buffering data into bundles as well as transmitting the with deliberate store and forwards along each hop towards the destination. However, we do not make any changes to underlying network services for sensor data delivery, but instead use TCP for one hop transfers.

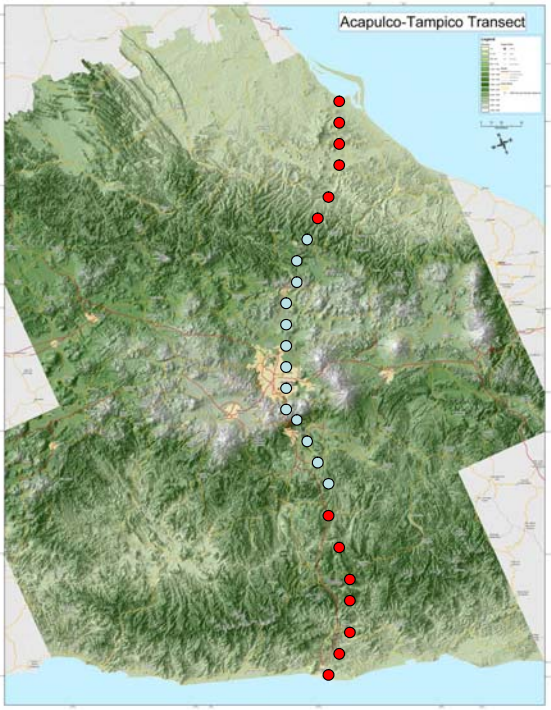

Figure 1: The approximate path of the entire MASE seismic network. The lighter dots centered on Mexico City represent the 50 networked stations of the transect.

DTS depends on StateSync [8] to reliably disseminate management commands and responses. StateSync was designed to extend the ideas of previous abstractions such as Hood [12] and protocols such as Trickle [11] to support of a specific class of applications. Where Hood and Trickle were designed to run on very lightweight sensor platforms such as Motes [9], StateSync is designed to take advantage of the more plentiful resources available on the Stargate platform.

Many distributed system management tools exist which allow remote configuration and command execution. In particular, tools from the distributed processing fields have capabilities similar to DTS. Additionally there are many tools to monitor and collect status information, such as Nagios [2], from large networks. The difference between DTS and these tools is that they all require reliable end to end connections.

3. NETWORK CHARACTERISTICS

The goal of the seismic network is to deliver data from accelerometers at each station. The data produced is packaged every hour to create approximately 1.5MB bundles. The bundles are transported hop by hop on a path towards the sink using a simple TCP transfer component which sup-

Node	Lat	Long	ID	Site Name
A	18°59.974'	99°14.422'		Cuernavaca Norte (repeater)
B	18°59.974'	99°14.422'	CUNO	Cuernavaca Norte
C	18°52.567'	99°12.211'		Palmida (repeater)
D	18°49.720'	99°14.633'	TEMI	Temixco
E	18°52.314'	99°11.851'	JIUT	Jiutepec
F	18°55.740'	99°13.322'		Cuernavaca Centro (repeater)
G	18°55.740'	99°13.322'	CUCE	Cuernavaca Centro
H	18°47.017'	99°12.960'	XOCH	Municipal Xochitepec
I	18°42.050'	99°14.961'	APOT	Apotla
J	18°39.132'	99°15.657'	SJVH	San Jos Vista Hermosa
K	19°03.604'	99°13.006'	PTCU	Cerro Tres Cumbres
L	19°01.890'	99°16.241'	VLAD	Huitzilac
M	18°44.779'	99°14.989'	ATLA	Atlacholoaya

Table 1: The table lists the the thirteen node network one which DTS is deployed. Node A is connected over wired Ethernet to a PC with internet connectivity. Each non relay station is dug into the ground and contains a custom box with a Stargate, a battery, a Kinemetrics Q330, and a Guralp CMG-3T (in a nearby dug out vault). Where convenient, existing radio towers, masts, and rooftops are used otherwise the antennae are mounted to the solar panel mast.

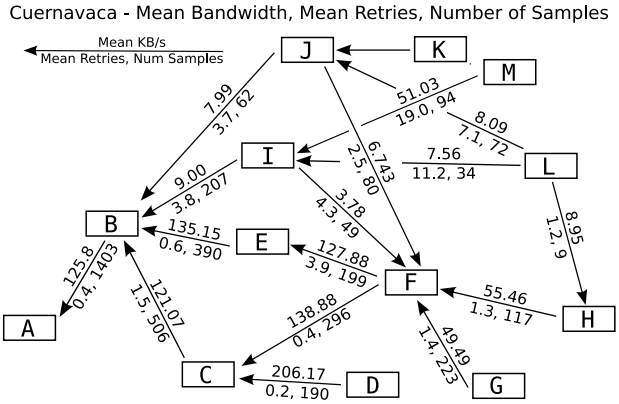

Figure 2: A diagram of the network topology of the Cuernavaca network. The arrows represent the direction of file transfers and are marked with the mean effective bandwidth, the mean number of retries and the number of samples. Node K did not transfer any files over the period of collected files. Not all the transfers for the collected data are shown. Figure 3 shows the range and variability of the bandwidth and retries of characteristic links.

ports resuming a partially transferred file after a timeout[1]. These transfers provide an opportunity to characterize the network by recording information about how long the transfers take and the number of times the TCP connection is lost and reestablished (a retry) for each bundle. As each bundle is transferred over a hop the transfer time and number of retries are added to the bundle. The characterization is important because each node has limited disk space (1-4GB). If the links are not good enough to transfer a nodes local data and the data from upstream nodes, the disk will fill up and eventually data will be lost. When such links are detected, someone must physically visit the stations and attempt to fix the situation. The characterization is also important because it can provide clues as to how well other software might perform on the network.

The collected transfer times between the nodes can be used to determine the estimated bandwidth for each link. The estimated bandwidth for each link is what provides our characterization of the network and can be used to evaluate the effectiveness of the network. To determine the effectiveness we can compare the estimated bandwidth to the required bandwidth for any node along any given path. For instance, consider a node at the edge of the network without any other nodes sending data through it. Given that each node generates 24 files a day with each file approximately 1.5MB, the minimum required sustained bandwidth would be about 416 bytes per second. The minimum required sustained bandwidth will be greater for nodes that are closer to the sink, since they will be along the path to the sink for other nodes.

Each of the sub networks has one station that is connected to the internet, so all the nodes send the bundles to this sink node. To determine the path to the sink, we build a sink tree based on the ETX path metric [5]. To reliably disseminate the path metric information and build the paths we use the StateSync mechanism which is a reliable and efficient publish-subscribe mechanism described in Section 4.5. Each node uses StateSync to publish information over one hop including the sink's ID and the publishing nodes full chosen path and ETX to the sink.

The thirteen-node network beginning in Cuernavaca and heading south contains the stations shown in Table 1. Some of the information collected from the transfer information added to the bundles is represented in Figure 2. Each arrow represents the movement of bundles, with the source pointing towards the destination. Each arrow also shows the mean effective bandwidth, the mean number of retries, and the total number of files transferred along that hop. The effective bandwidth is the bandwidth computed using the file size over the total time it takes to transfer the file including resumes and the time between retries. The data

[1]We initially tried using *scp* but it did not support resumes and when trying to transfer over bad links, it would stall for up to 20 minutes before closing the connection and retrying. If no data is transferred for 45 seconds, our TCP transfer component closes the connection and attempts to resume the file.

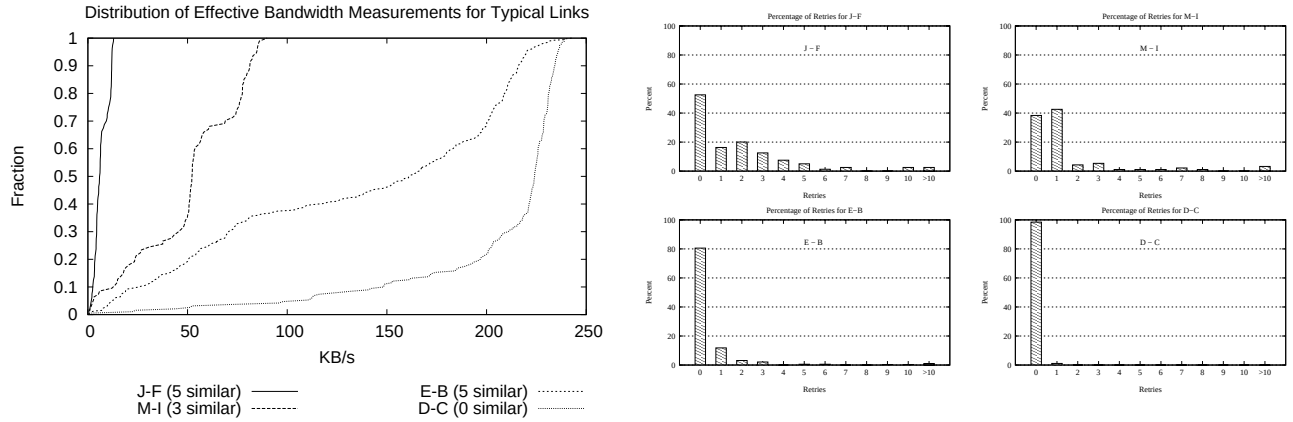

Figure 3: The bandwidth and retry percentages for a selection of four nodes. The chosen links are representative of all the links in the network and the number of similar links is indicated. More than half of the links have a highly variable bandwidth which indicates the instability of the links and that performance varies over time.

shows that the bundles take multiple paths to the sink. This indicates that the links are variable enough to cause the path to the sink to change. The link variability is most likely due to environmental conditions.

Figure 3 shows the distribution of effective bandwidth measurements for four representative nodes and the number of other hops that have similar distributions as well as the percentage of retries for the representative hops. This data supports, as does Figure 2, that over the entire network, the bandwidth varies greatly from hop to hop. The hops that have a larger overall bandwidth have a large variability over that bandwidth. This indicates the instability of the links and that performance varies over time. This is most likely due to the long distances of the links and the environmental conditions.

4. DTS

DTS is a remote management facility designed to manage large numbers of nodes connected by challenged networks. DTS makes this management problem tractable by ensuring exactly one execution of a series of commands, and by providing centralized collection of responses, given a range of disconnected and poorly connected networks. Ensuring that all scripts run on all nodes is the key to providing a tractable management environment, and as our tests in Section 5 show, DTS achieves 100% success rate. This result follows from the design of DTS, which ensures that it will succeed as long as there is *eventually* a connection between a given node and a node that has already received the command. In this section, we describe DTS from the top down: what DTS provides to the user, details on the implementation of DTS, how DTS uses a reliable and efficient publish and subscribe mechanism called StateSync, and how StateSync works.

4.1 The DTS Service Model

DTS provides a centralized management interface in which commands to all nodes are issued from a management station, broadcast to the network, and asynchronous responses from all nodes are collected and reported back to that station. DTS does not assume that all commands issued will be idempotent; thus nodes receiving a command execute it exactly once. This means that DTS cannot protect against failures stemming from failures in the commands themselves that yield indeterminate results, for example a script that causes an unexpected node reboot. However, in these cases DTS does guarantee to report that such a failure potentially occurred. If a user issues a script known to be idempotent and that script fails, the user can repeatedly reissue the command via DTS until success responses have been received from all nodes.

The responses to commands are also broadcast to the entire network. This means that in addition to being visible at the management station, collated responses can also be seen from any node in the network. This feature is quite useful to technicians in the field who want to monitor results and perform maintenance operations with rapid turnaround. Commands may also be issued from the field, and these commands and their responses will also be collated at the central management station.

The latency observed by a user of DTS varies depending on the state of the network. In cases where the network is well–connected, DTS performance is 10s of seconds: slightly slower than parallel end–to–end ssh sessions. However, lengthy disconnections can introduce unbounded latency, especially if a field technician must be dispatched to physically visit a location and repair an antenna. However, the DTS service model ensures that even if some parts of the network are unreachable when commands are issued, they will propagate node to node and be executed on nodes as soon as they become available.

These extreme variations in expected latency make it difficult to devise an algorithm to tell when a particular job has successfully "completed". This is especially true when we consider the wide variety of exceptional conditions that are encountered during maintenance tasks, and the fact that some nodes may remain off-line permanently. Rather than relying on an algorithm, the DTS system relies on the user to resolve ambiguous cases. Since the user knows how many nodes exist in the network and whether they are functional, DTS leaves the determination of whether a particular job has completed on all nodes up to the user.

The user interface provided by DTS is currently a command–line interface similar to a remote shell with several concur-

rent background jobs and convenient access to the collated responses from completed jobs; a future version will use a database–backed web interface. In addition to executing standard shell commands, DTS includes two more specialized built in features: ongoing status reporting and file transfer.

The first is the ability to create a "status client" on any particular EmStar [7] status device. Anytime the status device updates within a given refractory period, the latest output from the status device is republished. For these types of response, an additional sequence number is included with each response message to distinguish these sequential updates.

This status feature enables existing deployments to be instrumented with unanticipated state reporting "on–the–fly" and while running live. For example, this facility can be used to monitor disk usage or link quality to neighbors without installing additional software on the node. More complex predicates can be implemented by scripts that are pushed out using DTS and then return periodic replies via this status reporting facility. For instance, a script that looked for and reported certain anomalous packets might be instrumental in tracking down a bug that only occurred once the system was deployed into this more challenged and inconvenient environment.

The second feature is the ability to push a file from one node to all the other nodes in the network. This feature uses a single-hop file transfer module (described briefly in Section 3) as an underlying component. For every transfer issued, each node reports a list of known neighbors and the status of the transfer to each neighbor, along with an additional sequence number to indicate the freshness of the transfer status. This feature can be used to upgrade binaries and deploy new scripts.

4.2 How DTS Works

DTS reliably broadcasts commands to the network, executes each command exactly once on each node, and then reliably broadcasts a summary of the output of each command to the network. Commands and responses stored on any given node in the network are reliably synchronized to neighboring nodes as they become available. In this way, data floods through the network via hop–by–hop reliable transfers, independently of any pre-existing routing fabric.

Commands are keyed by the source node (typically the management station) and by a per-source sequence number. Once a node receives commands it will execute them in the order they were issued, and the return value and the output of each command is then broadcast back. Because of the potentially unbounded latencies in the network (especially in the event that links are down permanently), commands and responses persist in the network until the user at the management station chooses to flush them. If any node is unreachable when a command is issued and remains unreachable until after the command has been removed from the network, then the node will never have executed the command.

4.3 Command and Response Dissemination

To disseminate the commands and responses, DTS implements an "epidemic" algorithm that transmits stored command and response data to any neighboring nodes that have not yet received it. This algorithm is implemented above a reliable and efficient single-hop publish–subscribe mechanism called StateSync, and described in Section 4.5. The interface to StateSync allows an application to publish tables of data, and the implementation efficiently propagates changes to those tables.

Using this interface, each node publishes and subscribes to three tables: a commands table, a responses table, and a table containing a per-source starting sequence number. The command and response entries both contain a source identifier, a command sequence number, type fields, and a payload field. The response entries also include a return value field, and depending on the command type, an extra sequence number in the payload to help distinguish newer responses from updating responses. DTS populates these tables based on stored information and submits them to the pub–sub interface to be transmitted to all direct neighbors. As each neighboring node receives new information, it incorporates it into its existing data store and republishes, propagating the data one hop further.

The per-source starting sequence numbers represent the oldest command issued by a node that should be in the system. Any commands and responses with sequence numbers below this value for the particular command source node will be discarded. The per-source starting sequence number is controlled by the user, who may increment it after all nodes have reported responses. This mechanism provides a simple and efficient way for users to clear out old command data after a maintenance operation completes, although it does not provide the ability to remove arbitrary command from the network. For synchronous maintenance operations over the entire running network, this mechanism provides a simple and easily understood interface.

4.4 Execution and Storage

The commands are executed by piping them to a forked instance of Bash[1]. While the order of execution of commands from different source nodes is not guaranteed, commands from any *single* source node are guaranteed to be executed in order according to sequence number. When a command completes, the first 4KB of its output is published in a response entry. If a command has not finished within a given time limit, any data output up to that time is published as a response entry. The command is then allowed to finish and the remaining output is discarded. The time limit was chosen based on our experience and the types of commands we frequently run. There are many ways to handle long running commands and future version of DTS will support allowing the user to choose how timeouts are handled.

The shell also provides the ability to execute a command only on a specific node. All other nodes in the network still respond to this type of command with an acknowledgment, thus assuring the user that the command is propagating through the network. The user may also restrict commands from running on the node they are issued at.

If at any point a node is rebooted, a script fails, or the DTS service crashes, DTS ensures that no command will run more than once. Several mechanisms are implemented to achieve this. First, whenever a new command is received from the network, it is immediately saved to persistent storage on disk. These command tables are read from disk on start-up and populated into the system.

Second, before a command is executed, a new marker file

is created to indicate that execution has begun on that command. As soon as that command finishes, the output of that command is stored in a separate result file. If DTS starts up and it finds an the marker file and no result file, it assumes there was a failure and does not execute the command, instead publishing a failure notice.

Third, published "start" sequence numbers are saved persistently upon arrival from the network. This helps to quickly weed out old commands and responses when there are lengthy disconnections and node outages.

4.5 Reliable State Dissemination

StateSync [8] is a reliable and efficient publish–subscribe mechanism. It defines a data model based on tables of key–value pairs (KVP), to which new KVPs may be added or removed. It implements a broadcast dissemination protocol, in which published data is propagated a specified number of hops.

StateSync lends itself to applications which require reliable delivery, have large amounts of data to share, and for which the data elements being shared have an expected lifespan that is long compared to the system latency requirements. DTS falls under the scope of these requirements. DTS requires reliable delivery of its tables in the presence of bad links, and StateSync's log mechanism enables updates to be efficiently propagated in small pieces. DTS response tables can include up to 4KB of data per response, and thus can represent a sizable amount of data. The lifespan of the data in DTS is determined by the user, but data is typically retained for the duration of a maintenance operation, which might be anywhere from 15 minutes to hours. These lifespans are significantly longer than the expected system latency (when well–connected) of 10–20 seconds.

The StateSync implementation supports both single-hop and multi-hop modes. However, in our DTS implementation we use it only in a single-hop mode, and implement our own multi-hop epidemic protocol. Underneath its simple pub–sub interface, StateSync generates a log of publication updates, implements a retransmission protocol, and transparently handles node restarts, late joiners, intermittent joiners, and low quality links. We will describe some more of the details of StateSync in the following subsections.

4.5.1 Data Model and Log Mechanism

The StateSync data model is typed key–value pairs (KVP), with the id of the source node implicitly appended to the key, so that each node has an independent key space. Each KVP has a type that determines its key length and to which table it belongs. Applications define their own types, and thus create a new table by publishing KVPs of that type. Tables are always published and received as complete tables, as in a soft–state mechanism. This method guarantees that the application can never be out of sync with the StateSync system.

When an application publishes a new table, the system computes the difference between that table and the currently published table, and appends any changes to the current log for that node. Thus, as applications modify their tables, the StateSync log for that node grows, appending "ADD" entries when a new KVP is added or when a value for an existing key changes, and appending "DEL" entries when an existing key is deleted. Large entries are fragmented into smaller log entries, so that the protocol can run over UDP without IP fragmentation. This log is transferred reliably to other nodes, where it is executed to construct a set of tables that are presented to applications. This means that for small changes, such as adding an additional command, the log entry that must be transmitted is quite small. When no changes are occurring, the protocol periodically sends a refresh message containing the sequence number of the most recent log entry, so that receivers know whether they are missing data.

Repeated additions and deletions can result in logs that contain redundant and unnecessary information. When a log reaches a certain level of redundancy, the node publishing the log will "checkpoint" it by appending a "TERM" command, and begin a new empty active log. This checkpointing process does not require any part of the existing log to be retransmitted, because when a "TERM" command is received, the receiver will internally copy over all non-redundant entries in the old log to create the new active log. In the event that a node is completely out of sync, either because it is a new node, or because it has been off-line for a long time, or because of some kind of bug, the system will need to entirely replay both the checkpointed log and the active log.

For DTS, the log mechanism means that the system can recover from brief disconnections without excessive overhead, because only the recent log updates will be required to bring the two peers back into sync. The log mechanism also enables low overhead in cases where the state does not change significantly for long periods of time, because the current state can be verified by transmitting a single sequence number.

4.5.2 Retransmission Protocol

Each source node broadcasts new entries in its log as they are created, but these messages may be lost due to poor link quality. The StateSync retransmission protocol is a receiver-driven protocol that requests retransmissions based on gaps in log entry sequence numbers. In the event that the last entry in the log is lost, a refresh message replaying the most recent sequence number is used to inform receivers that they are missing data. This protocol allows nodes to stay synchronized with higher efficiency than simply retransmitting the entire state periodically.

In the event that a sequence gap is detected, receivers schedule NACK requests for specific ranges of sequence numbers to fill in the gaps. The NACKs have a brief initial delay followed by an exponential back-off. The timing parameters governing the emission of refresh messages and NACKs can be tuned to match application requirements.

When any kind of inconsistency is detected, a special message type called NACK_INIT is issued. This message requests that the full 64-bit log entry sequence number be sent (normally this sequence number is abbreviated). Since this sequence number is randomized on start-up and incremented monotonically, this approach significantly reduces the probability that a node's new log is confused with a previously published log after a reboot.

5. RESULTS

We evaluated DTS against a comparable end to end management method. Issuing commands using DTS is similar to using ssh as a remote execution tool over end to end connections. We evaluated how well DTS and ssh perform

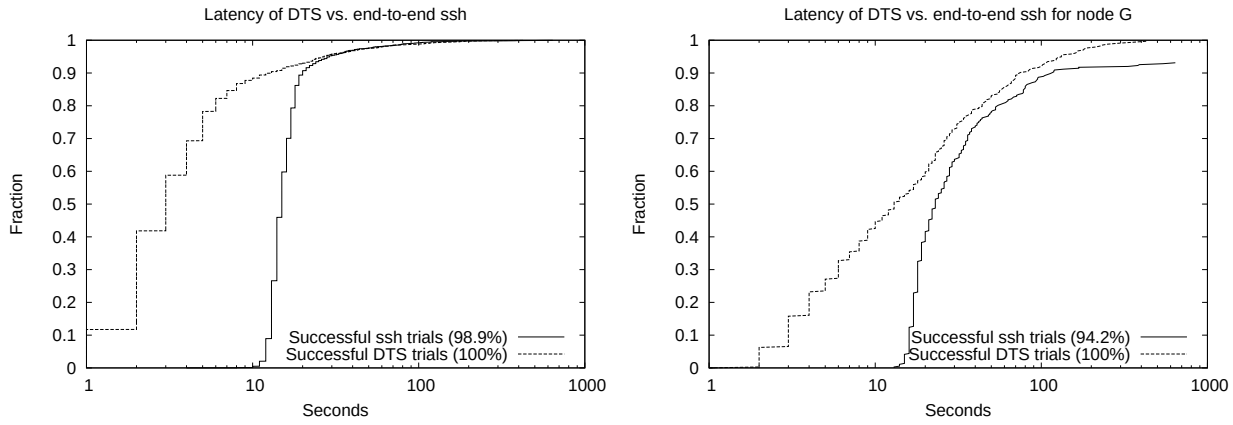

Figure 4: The latency of DTS compared to ssh for all the existing nodes as well as a comparison of DTS and ssh on node G. Node G had the highest latency of all the nodes for both the DTS and the ssh tests. For all the nodes 90% of the time the latency of DTS is lower than the latency of ssh and the remaining 10% of the time the latency is comparable. DTS is successful 100% of the time and in the cases where the end to end connection for ssh fails the latency of DTS has a lower latency.

under relatively typical circumstances for this particular line of 13 nodes. The results show that 90% of the time DTS has lower latency than ssh. The remaining 10% of the time DTS has comparable latency to ssh. As the link quality degrades DTS will out-perform ssh both in latency and in percentage of commands successfully executed because it does not need to create and maintain end to end connections.

5.1 Experimental Setup

We created a statically routed virtual network within the nodes existing ad-hoc network, using standard Linux facilities. The routes were chosen based on the experience and feedback of the students and staff who installed the system and aligned the directional antenna's as well as observations of the output of our link quality estimator. For both the end to end and DTS tests, 11 of the 13 nodes were used (see Table 1). Nodes H and L were inoperable for the duration of the experiment because they were very poorly connected. The poor connectivity would result in an extremely high failure rate for ssh and latency on the order of hours for DTS.

The experiment was driven by a program running on node A. The test program would issue commands to DTS and an ssh script and collect the responses. The same command was issued via ssh and DTS, in an interleaved fashion. This command in each case was "cat" on a small status file (approximately 500 bytes), so that both the request and response should fit in a single packet. The test program would begin by issuing one "cat" command to DTS and wait for responses form all the operating nodes. The test program would record how long it would take for a response from each node to propagate to node A. Once all the responses were propagated to node A, the test program would remove the command from the network and begin an ssh trial. The ssh trial was performed by a script which work fork 11 instances of ssh, one for each active node in the network. The test program would record whether each instance of ssh was able to execute the command or not and how long each instance took to succeed or fail. Once all the instances of ssh had returned, the test program would begin the next DTS trial. There were approximately 360 DTS and ssh trials performed over approximately 12 hours during which the link qualities of the network were relatively stable. The conditions over these 12 hours were relatively typically for this network, but still the entire variability of the network as shown in Section 3 was not explored.

5.2 Performance

Figure 4 shows a CDF of the latency of DTS compared with ssh for all the nodes as well as for node G alone. In the analysis of this data, there are two cases to consider: the case where end to end connections are available and the case where end to end connections are not available. In the cases when were end to end connections were available, in 90% of the experiments DTS had lower latency. In these cases, there were two dominant factors causing ssh to have greater latency. These factors were the launching of 11 processes for each trial and each process creating a cryptographically secure end to end connection. In the remaining 10% of the experiments, the latency of DTS and ssh are comparable. We believe that in these situations there were end to end connections available, but the network speed was slow. The slow network speed had a similar affect on the latency of DTS and ssh.

In the absence of high availability end to end connections, DTS will reach 100% of the nodes, whereas ssh fails on some fraction of the nodes. This is evident in the CDF for node G, which only reaches 94.2% since 5.8% of the end to end trials failed. Node G was not the farthest from the sink in terms of hops, but had poor link quality so it had the largest latencies for both DTS and the end to end ssh tests. As the link quality degrades event more, the latency of ssh will become greater quicker than the latency of DTS because the network will be more prone to situations that make end to end connections difficult to create and sustain. DTS does not have this problem because the node bordering a discon-

nection will notice when the link is usable again and will immediately use it to propagate commands and responses.

When an application such as ssh fails, it has to attempt to reestablish and sustain the end to end connections again. This introduces a lot of extra latency as well as the possibility of missing a potentially small window in which the end to end connection is actually available. The extra latency comes from the long delays before ssh finally times out and fails. The length of these delays are not reflected in the CDFs in Figure 4, beacuse the graph only include data from successfull ssh connections. For all the ssh commands which failed, none of the delays were less than 2 minutes with some of the delays in the tens of minutes. One approach to try to improve the latency of ssh and to deal with the failure cases is to shorten the timeout on ssh and retry. This lowers the chances of missing a window in which an end to end connection is available but will use up more resources and network bandwidth. The approach DTS takes is similar, but it is localized to the link with the disconnection instead of the entire end to end path. Finally, there are situations where end to end connection never exist. Ssh will never succeed in these cases. DTS provides a guarantee that it will succeed as long as there is some pattern of connections to every node. This guarantee, coupled with equivalent or better performance in disconnected environments, makes DTS well suited to system management tasks.

6. FUTURE WORK

We plan on deploying DTS on the remaining 3 sections of the MASE network. This will provide more opportunities to evaluate DTS.

Currently the interface to DTS is limited to a test program and the interactive shell. The interface can be expanded to support issuing DTS specific scripts to automate repeated tasks through cron.

DTS does not limit the types of commands that can be issued. Just like any other shell, a user can potentially damage the file system in such a way that it requires re-flashing the node to recover. To help prevent these types of problems, DTS needs to understand the commands being issued and have specific rules preventing certain types of operations and shell commands. For instance, simply requiring full paths for commands and not allowing shell expansion can prevent some common shell scripting mistakes.

The error handling for both individual commands and node state can be improved. As the system scales simply reporting that there was a problem and leaving it up to the user to determine the state of the node is not tractable. There are multiple paths to providing better error handling, but an initial step could be for each node to keep a compressed log to attempt to provide accountability for when there is a problem. This log could be published or retrieved only when queried for.

StateSync currently uses broadcast UDP packets. These broadcast UDP packets do not benefit from link layer retransmissions as all unicast traffic does. Changing StateSync to use unicast UDP packets could potentially benefit the latency and help reduce traffic.

We would also like to implement a centralized control for all four of the lines of seismic stations through a web interface. This will help evaluate the usability and performance of DTS in larger networks.

7. ACKNOWLEDGMENTS

We would like to thank Allen Husker, Igor Stubailo, Paul Davis, and all the students and staff from the UCLA Earth and Space Sciences and UNAM for the countless hours spent building and installing the entire system.

8. REFERENCES

[1] http://www.gnu.org/software/bash/.

[2] http://www.nagios.org.

[3] Middle america subduction experiment (mase), http://www.tectonics.caltech.edu/mase/, http://research.cens.ucla.edu/research. Caltech Tectonics Observatory and UNAM Institutes of Geophysics and Geology and UCLA Center for Embedded Network Sensors and UNAM Center for Geosciences.

[4] 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*, 2003.

[5] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.

[6] K. 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*. ACM Press, 2003.

[7] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *Proceedings of the 2004 USENIX Technical Conference*, Boston, MA, 2004.

[8] L. Girod, M. Lukac, A. Parker, T. Stathopoulos, J. Tseng, H. Wang, D. Estrin, R. Guy, and E. Kohler. A reliable multicast mechanism for sensor network applications. Technical report, CENS, April 25, 2005 2005.

[9] J. Hill and D. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, Nov/Dec 2002.

[10] Intel. Intel stargate, http://platformx.sourceforge.net.

[11] P. Levis, N. Patel, D. E. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of 1st Symposium on Networked Systems Design and Implementation (NSDI 2004), March 29-31, 2004, San Francisco, California*, 2004.

[12] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: a neighborhood abstraction for sensor networks. In *MobiSYS '04: Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pages 99–110, New York, NY, USA, 2004. ACM Press.