4C38 Presentation: "Multiple Data Mules"

T.Schooley, M.Le, D.Nguyen, M.Patel

February 6, 2006

Source:

D. Jea, A. Somasundara, and M. Srivastava, "Multiple Controlled Mobile Elements (Data Mules) for Data Collection in Sensor Networks", *DoEE, UCLA*

Mules ?

$Def^{\underline{n}}$: Mule

The sterile hybrid offspring of a male donkey and a female horse, characterized by long ears and a short mane.

Outline

- Context / Motivation
- Single Data Mules
- Multiple Data Mules
- Load Balancing Algorithms
- Analysis / Conclusion

Context

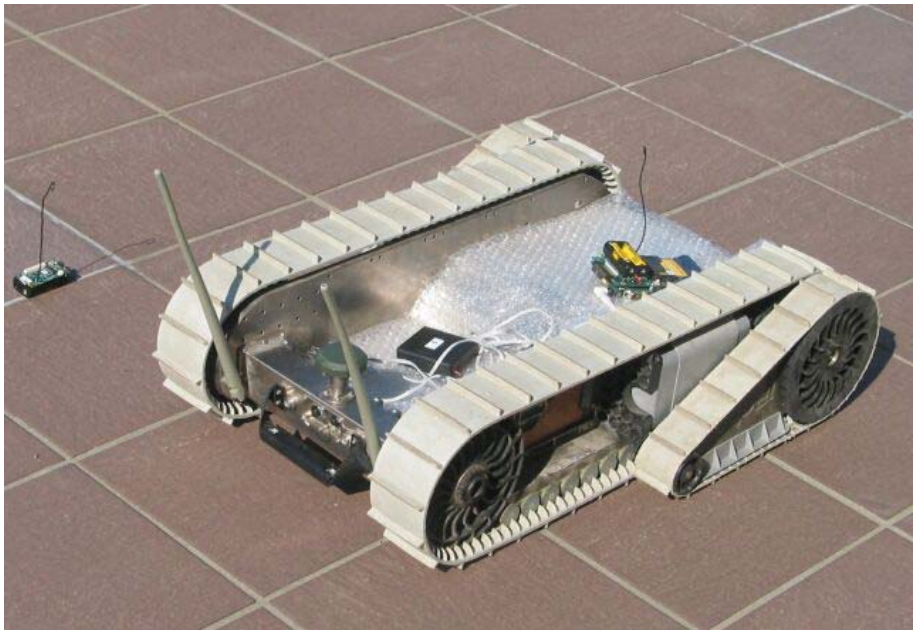
What are *Data* Mules ?

- Base stations that traverse the sensornet
- Tradeoff between network lifetime and data latency
- Benefits:
  - Avoids complex multihop routing
  - Spreads the resource load better (no hotspots)
  - Increases capacity (no bottlenecks)
- Disadvantages:
  - Mules use very expensive technology
  - Add large delays to sensornet queries
  - Mules need to recharge (extra delays)

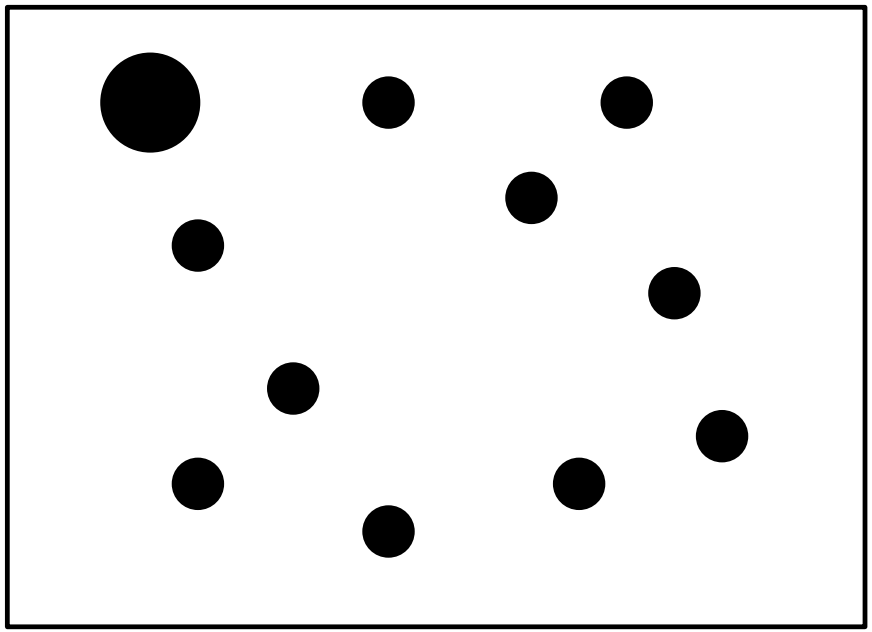
Prior Work

Here's one they made earlier...

- Source: "Intelligent Fluid Infrastructure for Embedded Networks"
- Paper suggested *controllable* mobile elements to increase network lifetime
- Assumed a pre-arranged grid of sensors (motes)
- A single mobile router



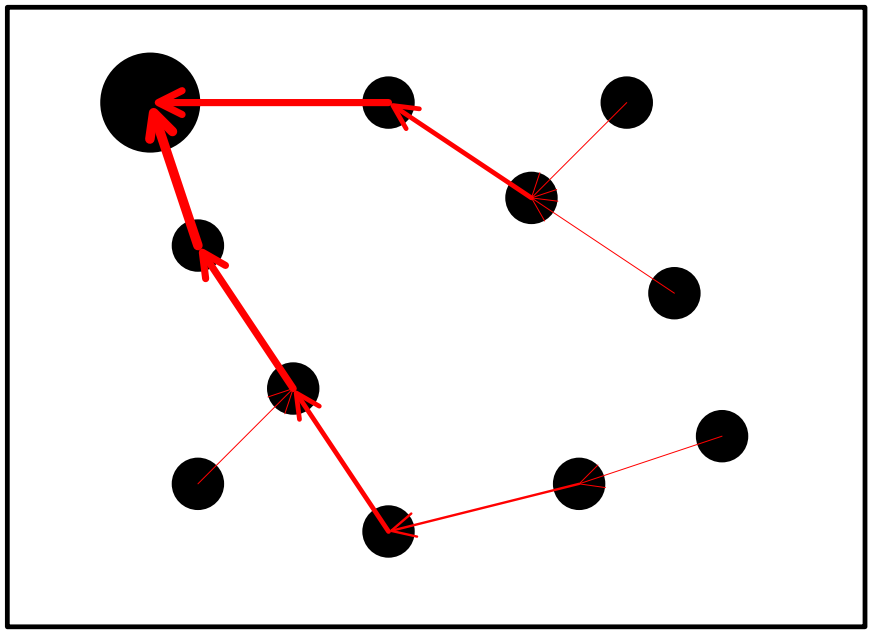
Motivation



Widely Used Data Collection:

- Single base station
- Multihop routing algorithms
- SPOF
- Hotspots near base station
- Bottlenecks
- Unbalanced resource consumption

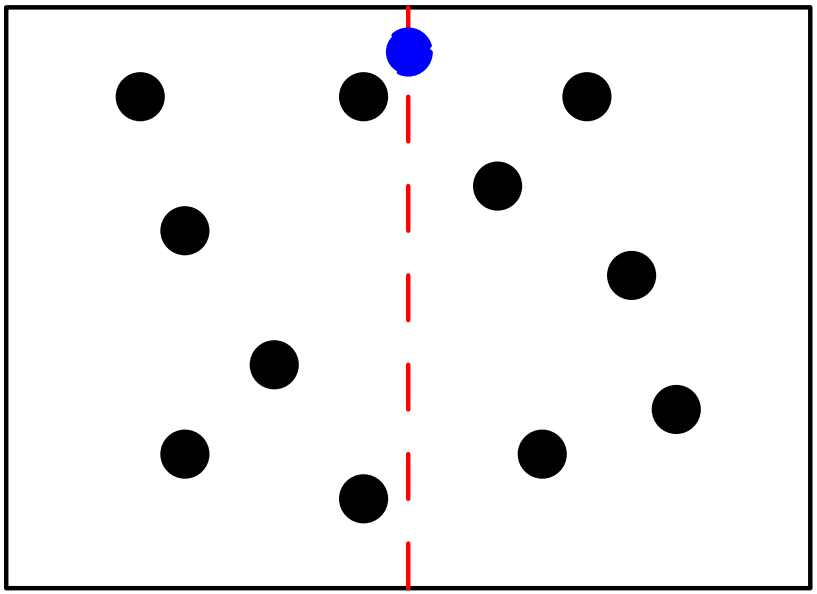
Motivation



Widely Used Data Collection:

- Single base station
- Multihop routing algorithms
- SPOF
- Hotspots near base station
- Bottlenecks
- Unbalanced resource consumption

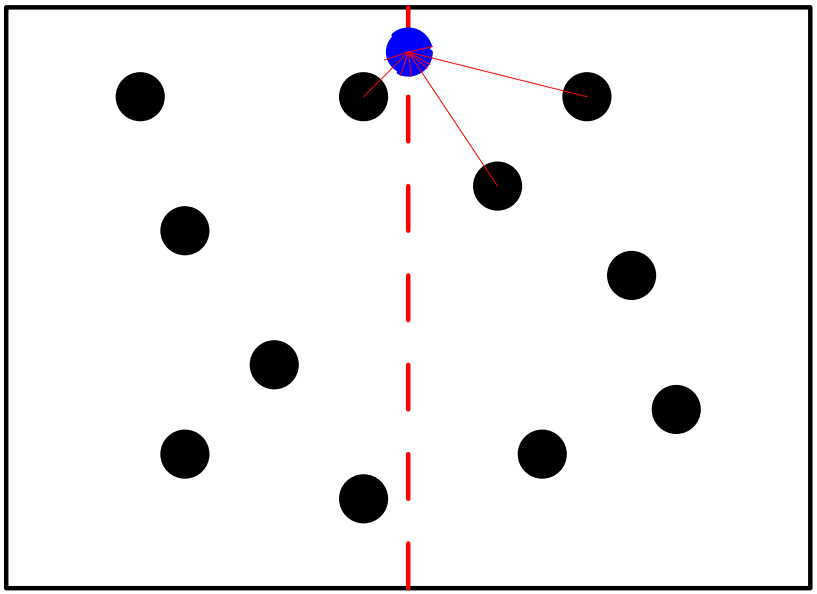
Single Data Mule



Details:

- Given:
  - Single Data Mule
  - Fixed path
- Goal:
  - Schedule of Data Mule to maximize data collection
- Design:
  - Network Algorithms
  - Adaptive Motion Control

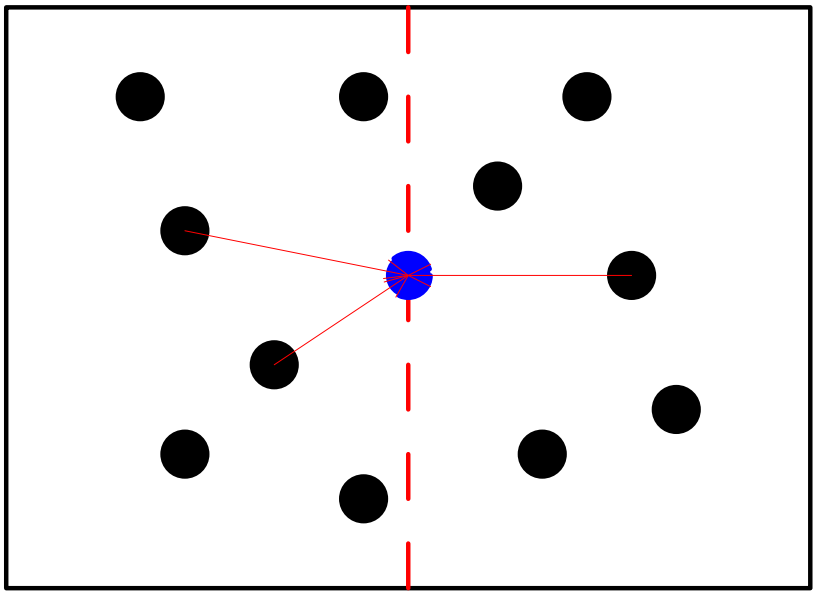
Single Data Mule



Details:

- Given:
  - Single Data Mule
  - Fixed path
- Goal:
  - Schedule of Data Mule to maximize data collection
- Design:
  - Network Algorithms
  - Adaptive Motion Control

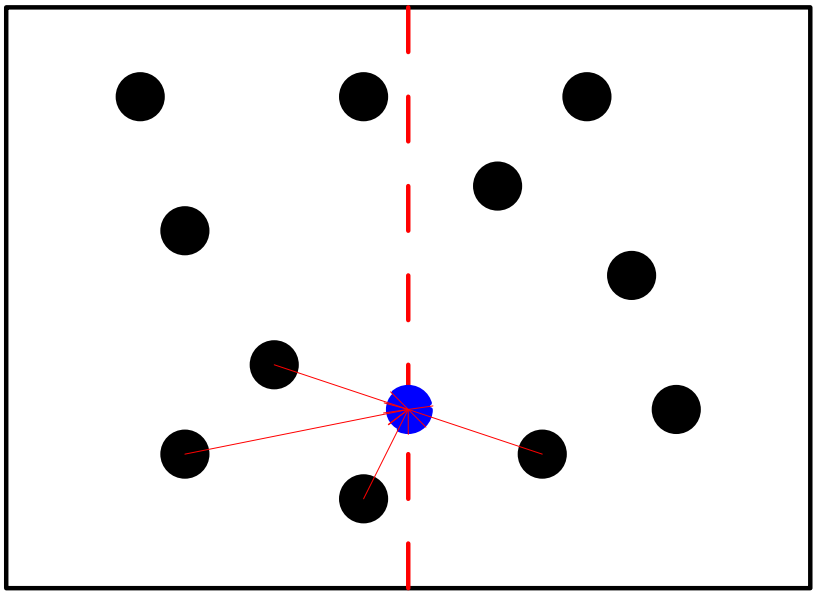
Single Data Mule



Details:

- Given:
  - Single Data Mule
  - Fixed path
- Goal:
  - Schedule of Data Mule to maximize data collection
- Design:
  - Network Algorithms
  - Adaptive Motion Control

Single Data Mule



Details:

- Given:
  - Single Data Mule
  - Fixed path
- Goal:
  - Schedule of Data Mule to maximize data collection
- Design:
  - Network Algorithms
  - Adaptive Motion Control

Problems

1. Some nodes might not be in range of the Mule!
   - Tree structures are formed
   - All nodes pass on messages to the root of their tree
   - The root nodes talk to the Mule directly
2. Mule could walk out of range before transmission starts/completes
   - This is where Adaptive Motion Control comes in
3. Nodes might transmit as soon as signal is found (rather than waiting for a slightly stronger/better signal)
   - Not addressed in the paper
   - Adaptive Motion Control might even contribute to this problem

Network Algorithms

1. Routing Tree Initialization
   - Mule traverses path, broadcasting beacons
   - Nodes rebroadcast the beacon, taking note of the hop count
   - Eventually all nodes know how many hops they are from the Mule
   - With this, they can choose a parent to pass on messages to
2. Local Multihops
   - All nodes send their data to the parent nodes, before the Mule traverses the path again
   - This allows any advantages in wrapping payloads to be used (ex. minimizing packet header overhead)
3. Data Collection
   - The Mule traverses the path, collecting data from all in-range nodes.

Motion Control Algorithms

- The Mule only has control over speed, and wants to *maximize data collection* by scheduling its use of speed optimally.
- Three different approaches were simulated:
  1. Fixed speed, and not stopping for anything (think "Italian drivers")
  2. Fixed speed, but stopping until all data is collected (SCD)
  3. Adaptive speed, travelling twice as fast as above, and calculating stop time based a data collection threshold (ASC)
- Conclusion: SCD worked slightly better than ASC in terms of received data per round trip - but this implies longer delays!

Summary of Single Mules

- A single Mule doesn't scale well
- Adding more Mules isn't straight-forward
- A Mule is also a SPOF (which they wanted to avoid)
- Tree structure introduces routing algorithms (which they wanted to avoid)
- Tree structure creates bottlenecks and SPOFs (which they wanted to avoid)
- But this step was needed before leaping towards a better solution
- This work identified the importance of speed control
- What happens with more Mules ?

Summary of Single Mules

- A single Mule doesn't scale well
- Adding more Mules isn't straight-forward
- A Mule is also a SPOF (which they wanted to avoid)
- Tree structure introduces routing algorithms (which they wanted to avoid)
- Tree structure creates bottlenecks and SPOFs (which they wanted to avoid)
- But this step was needed before leaping towards a better solution
- This work identified the importance of speed control
- What happens with more Mules ?

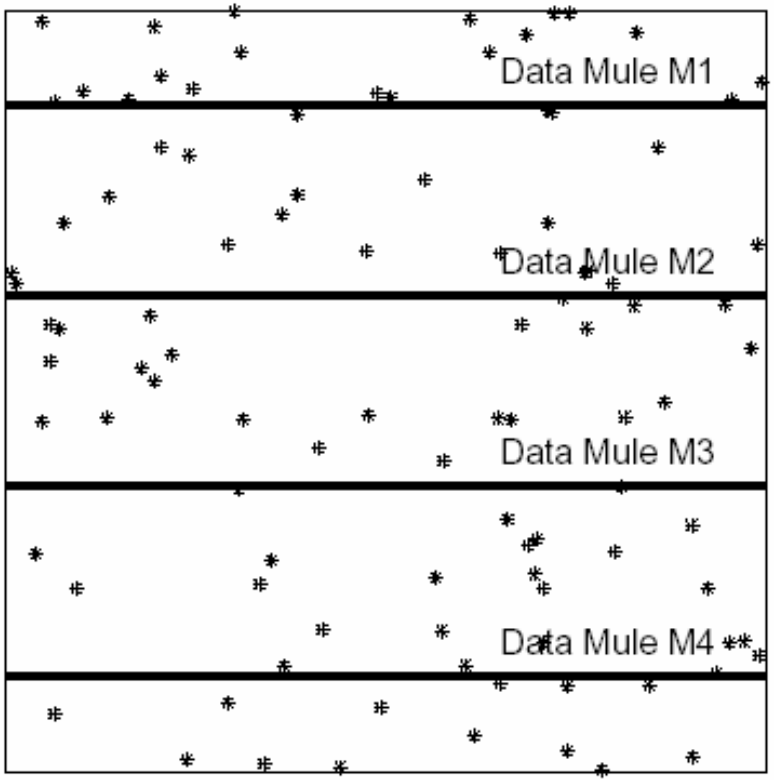
Multiple Mules

Michael Le

Motivation

- Single Mules do not scale well
  - Consider increased density
    - Method: Fixed RTT
      - More nodes means less time to service each node
      - Loss of data
    - Method: Stopping at each node (SCD)
      - Takes longer to service all nodes
      - May not reach node before buffer fills up
      - Loss of data
  - Nodes spread over larger area
    - Mule may run out of battery

Multiple Mules



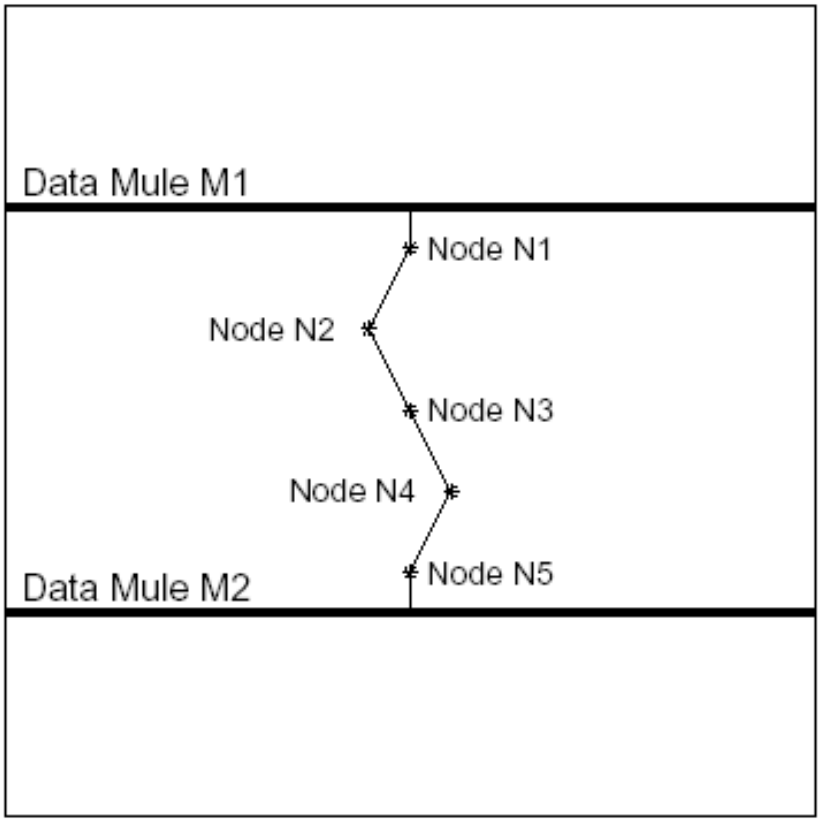
A trivial Solution:

- Assumption: nodes are uniformly distributed
- Divide area into equal parts
  - Mules will service same number of nodes
- Each Mule runs same single Mule algorithm
- Issues:
  - How many Mules ?
  - Handling of nodes shared by 2 Mules ?

How many Mules?

- Function of RTT and time it takes to fill buffer
- If $RTT < buffer\_fill\_time$ then use one Mule
  - Otherwise $\frac{RTT}{buffer\_fill\_time}$ Mules are required
- $RTT = \frac{l}{s} + (num\_nodes \times service\_time) + \frac{l}{s}$
  - $\frac{l}{s}$ = time it takes to traverse one length of path
  - $(num\_nodes \times service\_time)$ = time taken for data collection

Node Sharing



Hop count:

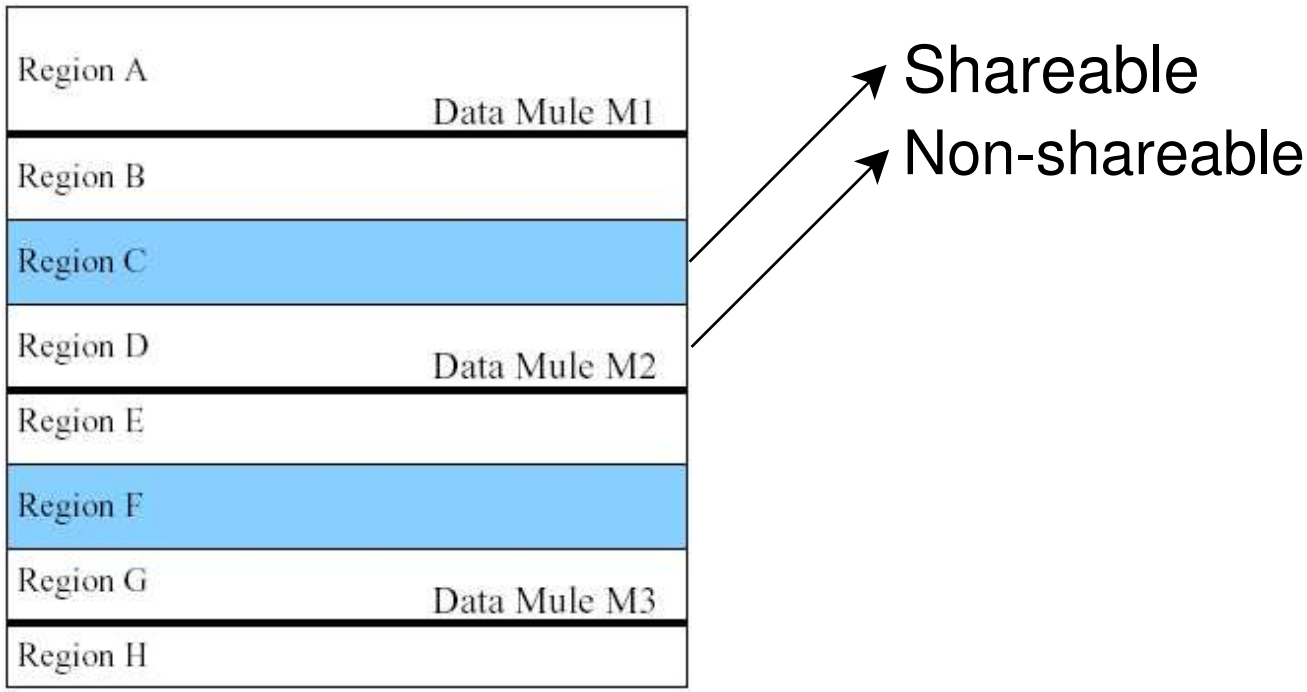
Nodes	Data Mule M1	Data Mule M2
N1	1	5
N2	2	4
N3	3	3
N4	4	2
N5	5	1

*N*3 is shared

- Can randomly decide which Mule will service it

Motivation: Load Balancing

- In real life nodes not uniformly distributed
- Consider following scenario:



- Goal: assign shareable nodes to mules *s.t.* each mule services approximately the same number of nodes

Load Balancing

Multiple Mules with Load Balancing: Approach

1. Initialization
2. Leader Election
3. Load Balancing
4. Assignment
5. Data Collection

1: Initialization

1. Do a broadcast
2. Nodes that hear the signal reply with their IDs
3. Result: list of nodes who are 1 hop away from Mule's path

2: Leader Election

- Assume Mules are equipped with powerful radios
- Mules elect a leader
- Broadcast list of nodes to leader

3: Load Balancing

- Leader classifies nodes into 2 classes: shareable and non-shareable
  - Shareable nodes are either shared with next or previous Mule
- Initially all Mules are the same group with the first Mule called *start_mule* and the last Mule called *end_mule*
- Goal: make load of each Mule equal to average load in group
- Not always possible!

Data Mule	Non_shareable nodes	Shareable nodes
M1	35	10
M2	5	10

- Optimal sharing gives 35 nodes to M1 and 15 nodes to M2

L.B. Algorithm

1. Calculate group average
2. Calculate minimum load that Mule under consideration should take
   - If $minimal\_load > group\_average \Rightarrow$ split & put Mule in first group
3. If split does not happen, try to assign some load it shares with next Mule
4. If maximum load that can be assigned to a Mule $< group\_average \Rightarrow$ split & put Mule in first group
5. Recursively call algorithm for the two groups

4: Assignment

- Load balancing outputs 3 counts for each Mule:
  - No. of nodes to service from set shared with previous node
  - No. of nodes to service from set shared with next node
  - Total no. of nodes to service
- Leader tells each Mule which nodes it must service

5: Data Collection

- Mules traverse path polling for data
- Shareable nodes do not know which Mule they belong to
- Nodes reply when they hear the polling
- But Mule will send ACK only if it is responsible for that node
- Node marks the Mule from which it receives the ACK and ignores the other Mule in the future

Results / Conclusion

David Nguyen

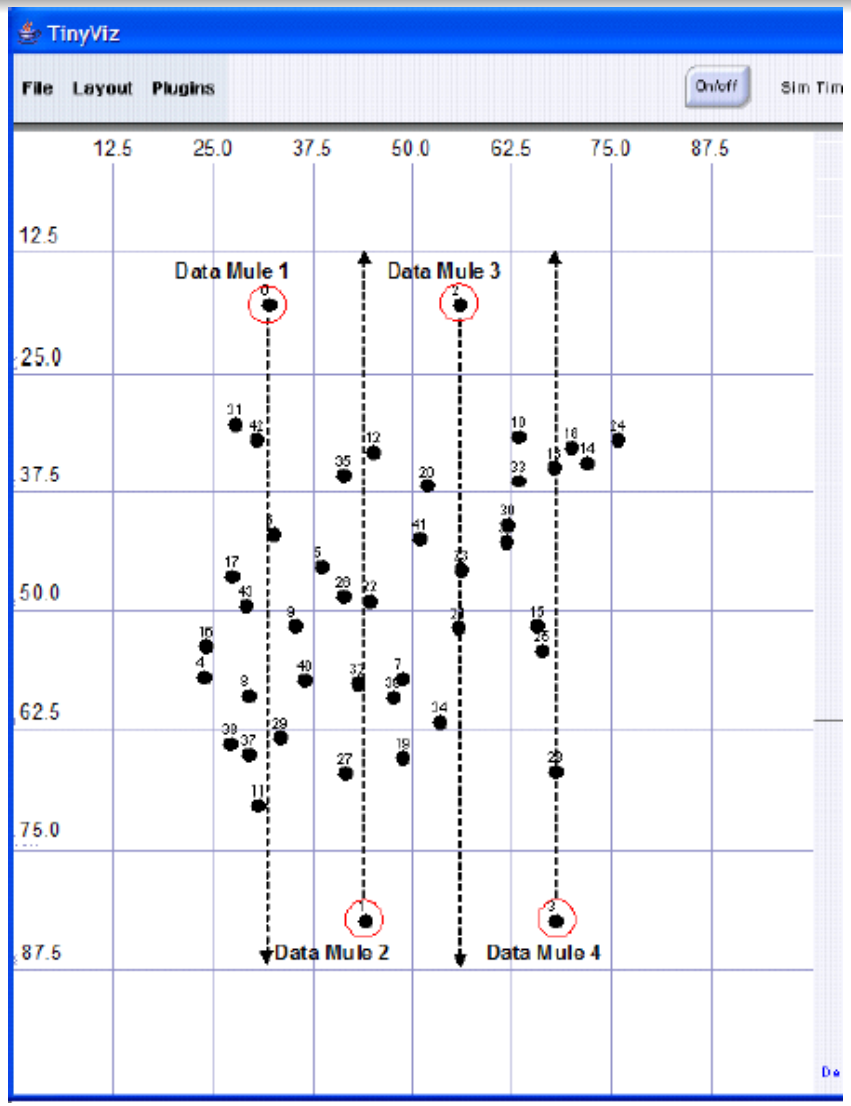
Results of Algorithm

- How to measure up the algorithm ?
- First Come, First Serve
  - Shareable node attaches to first mule it hears
- Equal sharing
  - Shareable nodes are divided in two
- Simulation
  - Implemented in TinyOS
  - TOSSIM was the simulator used

Simulation Variables

Variables

- 40 sensor nodes
- 4 data Mules
- Nodes randomly distributed
- Experiment ran for 5 rounds
- Rounds being RTT of 120 "units"



Simulation Results

- After initialization and leader election:

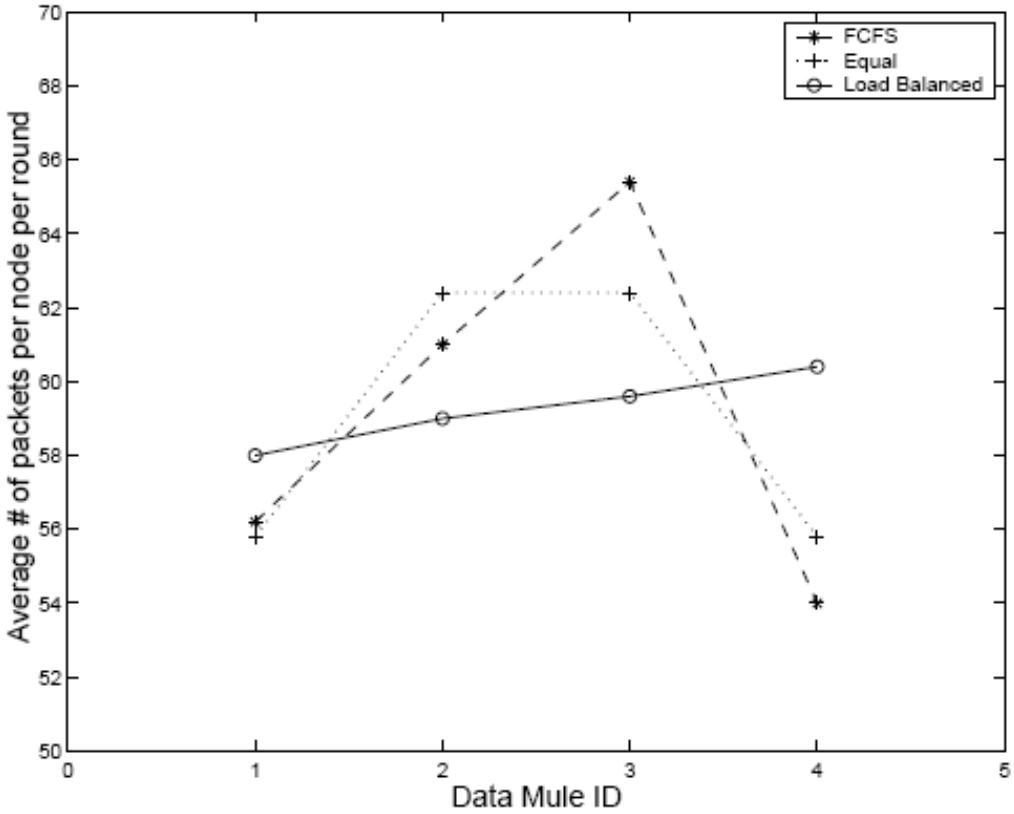
Data Mule	*non_shareable_load*	*shareable_load_neg*	*shareable_load_pos*
$DataMule[1]$	13	0	3
$DataMule[2]$	5	3	3
$DataMule[3]$	5	3	2
$DataMule[4]$	9	2	0

- Use of the load balancing algorithm:

Data Mule	FCFS	Equal sharing	Load Balancing
$DataMule[1]$	16	15	13
$DataMule[2]$	8	8	9
$DataMule[3]$	5	7	9
$DataMule[4]$	11	10	9

Simulation Results II

- From 5 rounds, number of packets received per node were measured
- Average number recorded here for each mule
- "Load balancing leads to more uniformity"



Paper Conclusions

- Addresses data Mule scalability issues
- More mules for more nodes (simple)
- Load balancing is a necessity
- Algorithm "appears" to be sound
- Simulations "justifying" the approach
- Paper itself has scope for expansion

Appraisal

Mitul Patel

Positives

- Controlled mobile element**s** to collect data in wireless sensor networks
- Motivation, challenges and solution are clear and well explained, with examples
- Results suggest approach is feasible and could be utilized for real networks
- Load balancing algorithm is uncomplicated and seems to work well
- Assumptions made are reasonably explained
- Minor variations between 'balanced' mules explained

Problems

- Many assumptions and simplifications
  - Assumption that each node can talk to at least 1 Mule and at most 2 Mules
  - Assumption that all Mules can communicate with one another during leader election phase
  - Consider costs of multihop in load balancing
  - Nodes placed away from region boundary "to avoid 'edge effects'"
  - Mobile element can be added or removed during system runtime
- Limited simulation...

Problems II

- How to position Mules correctly in area ?
- Only one Motion Control Algorithm considered in simulation(!!)
- Only one RTT tested
- Only one test region used
  - What about regions of different node densities?
- Appears as though results were obtained from a single simulation run
  - Thus no error bars or confidence intervals
- One line to sum up results found
  - No evaluation of other two strategies used

Problems III

- Thus simulation is perhaps 'too simple'. Also...
- Load balancing doesn't actually balance all nodes exactly equally, but only where possible
- Paper focuses on network connectivity much more than overall data throughput
- Only practical example is in another paper
- feasibility of multiple Mules ?
  - $\$50,000$, 7 inches tall, 18kg weight
  - Intention of $<$ \$1 per node?

Problems IV

- Paper ultimately feels flat
  - Assumptions, simplifications, lack of tackling more complex issues, few supporting results
- Nothing about how the Mules compare to other forms of data collection
  - Multiple base stations/sinks
  - Single/Multi-hop forwarding
  - Is it really worth it?