Starburst SSD: An Efficient Protocol for Selective Dissemination

Tahir Azim
Stanford University
Stanford, California 94305
Email:tazim@cs.stanford.edu

Qasim Mansoor
National University of Sciences and Technology
Rawalpindi, Pakistan
Email:47qasim@niit.edu.pk

Philip Levis
Stanford University
Stanford, California 94305
Email:pal@cs.stanford.edu

***Abstract*—We present Starburst, a routing-based protocol designed to efficiently disseminate data items to small subsets within a sensor network. Starburst constructs a routing hierarchy to enable fast, efficient and reliable dissemination to nodes in a sensor network that satisfy data-specific predicates. The protocol is based on the idea that when only a few nodes need an update, it is more efficient and much faster to route to those nodes directly. When every node needs an update, algorithms such as Trickle are more efficient. Starburst therefore dynamically determines what portion of nodes need an update and locally adapts its delivery policy accordingly. We also present dynamic beacon selection algorithms which enable scalability and fault tolerance in Starburst. We have implemented and evaluated Starburst on top of both the BVR and S4 routing protocols with promising results. Our simulations show that Starburst reduces both the transmission cost and latency of existing dissemination protocols by at least 50% for small subsets of nodes, and performs no worse than them for larger sets. Finally, tests on the Motelab testbed validate our simulation results.**

I. Introduction

Dissemination involves sending data items, such as commands, configuration values, and small programs, to nodes within a sensor network. This enables the administrator of the network to maintain the network and to reconfigure it to perform a different function.

Our work focuses on the problem of selective dissemination, which is the efficient dissemination of data to a subset of nodes in a sensor network. This subset is defined as a set of nodes that satisfies a certain predicate. For example, we may want to disseminate a new command to all nodes whose most recent temperature sensor reading was less than 40 degrees. The command could be used to request those nodes to turn on the heating system for their surrounding area, or to increase their sampling frequency. It is possible, of course, that only one or two nodes satisfy this temperature predicate. Unlike multicast, however, where nodes control their own membership in a multicast group, in selective dissemination, the transmitter controls who will receive updates.

Current approaches deliver data to every node in a network, even if only a small subset of nodes need it. Trickle [11] relies on nodes periodically broadcasting the current status of their updates and listening for new updates to disseminate data; the SNMS management system [15] uses the Drip dissemination layer to deliver commands to every node in a network even if the command is for a single node; the Fetch bulk transfer protocol uses broadcasts across the entire network to send a command to a single node [16]; and, Melete [18] limits the scope of Trickle to reduce packet transmissions to nearby nodes but spreads to the entire network if the subset is far from the originator of the update. As networks scale, these approaches are highly inefficient.

We present a protocol named Starburst which is designed to carry out efficient selective dissemination without transmitting updates to all nodes in the network. It combines the speed of fast, coordinate-based, unicast routing protocols with the wide reach of a rate-controlled, broadcast protocol to achieve this goal. We successfully implemented and tested Starburst on top of both the beacon vector routing (BVR) [4] and S4 [12] routing protocols. The protocol design is described in detail in Section III.

The rest of the paper is organized as follows. In Section IV, we propose two algorithms for dynamically selecting routing beacons for coordinate-based routing protocols using traffic patterns generated in the network. This enables us to add more beacons into the network as it grows, while ensuring that they are uniformly distributed across the network. In Section V, we describe further optimizations we implemented to make the protocol work efficiently in practice. We describe our evaluations and results in detail in Sections VI and VII.

Overall, this paper makes the following contributions: (i) Defines Starburst, a protocol for efficient dissemination to small, dynamic sets in a network. (ii) Proposes a mechanism for adapting between routing and Trickle to achieve this, giving an improvement of at least 50% in both transmission cost and latency for small subsets while approaching the cost of Trickle for larger subsets. (iii) Proposes and evaluates algorithms for selecting beacons in the underlying network hierarchy, showing that our algorithms improve on a random beacon selection approach by at least 16%.

II. Existing Approaches

In this section, we provide a brief review of existing dissemination and routing protocols, with greater emphasis on protocols that we used in our work.

A. Dissemination Protocols

Trickle [11] is a highly efficient, polite gossip algorithm for disseminating small data items to all nodes in a sensor network. It is widely used to maintain consistency across all nodes of a sensor network. In Trickle, each node uses randomized broadcasts at periodic intervals to inform its neighbors of the version numbers of its data items. The neighbors listen to these version numbers and update themselves on hearing a newer version. Trickle uses packet suppression to control the number of broadcast packets, resulting in a finite rate at which updates are disseminated.

Melete [18] extends Trickle to support selective dissemination by bounding the dissemination range. This allows data item delivery to be constrained to a "forwarding region" that covers the desired destinations in the network. However, if the desired destinations are located in a completely different part of the network, the forwarding region grows to encompass the entire network, reducing Melete to Trickle.

Firecracker [9] achieves faster performance by routing to some random destinations, and then using Trickle for dissemination.

B. Routing Protocols

The most efficient point-to-point routing protocols in sensor networks are based on logical coordinates, where a few nodes are selected as reference points or beacons. All other nodes compute their coordinates based on their distance in hops from each of these beacons. All nodes also keep a next hop table to reach each of the beacons. These coordinates are then used to route packets to any destination coordinates.

Beacon Vector Routing (BVR) [4] is one of the pioneers of using logical coordinate routing. Besides having the above features, nodes in BVR also maintain the coordinates of their neighbors. In order to route to some destination coordinates, BVR tries to minimize the distance between the sender and the destination by routing greedily through the neighbor that is closest to the destination. If none of its neighbors is closer to the destination, the sender routes to the beacon closest to the destination. The beacon then attempts to route greedily to the destination, failing which it uses a scoped flood to reach the destination.

Small State and Small Stretch Routing (S4) [12] is a more recent coordinate-based routing protocol, which creates clusters around each node. Besides maintaining a next hop to every beacon, each node also maintains a next hop for each node in its cluster. With this feature, S4 no longer uses the inefficient scoped floods of BVR. Also, S4's packet header, only 8 bytes long and comparable to Trickle's 6 byte header, is much smaller than BVR, which requires destination coordinates in each packet header. Finally, S4 is much less sensitive to inaccurate destination coordinates than BVR. For this reason, we also implemented our protocol using S4 and achieved promising results.

Virtual Ring Routing (VRR) [3] is a coordinate-based routing protocol which uses a technique inspired by distributed hash tables (DHTs) to improve routing performance. GEM [13] routes using a virtual polar coordinate space embedded on to the network. However, our protocol does not map directly to any of the implementations of VRR or GEM.

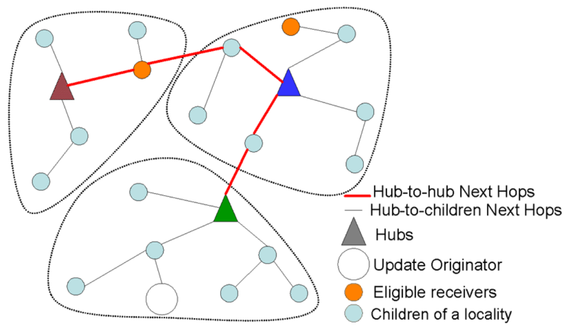

Fig. 1. System topology. Members of a locality send their state to their nearest hubs. Updates are sent to the hub nearest the originator, from where they get propagated to all the hubs, who finally send them to eligible receivers that satisfy the dissemination predicate.

Our system topology is similar to topologies created for hierarchical routing, experimental evaluations of which have been recently published [8]. Our beacon selection algorithms resemble clustering methods for sensor networks [2], except that our algorithms are specifically meant to facilitate selective dissemination.

III. The Starburst Protocol

Selective dissemination involves disseminating updates to a subset of nodes in the network. In our design, this subset is defined as the set of nodes that satisfy certain predicates, called dissemination predicates. The predicates are essentially logical tests over the values of certain variables known as dissemination variables. For example, a selective dissemination command can look like "Disseminate this piece of code to nodes that have sent over 50 packets". In this example, the dissemination variable is the number of packets sent.

The Starburst protocol, illustrated in Figure 1, operates as follows:

- A subset of nodes dynamically select themselves to be hubs, based on the network topology and dynamics. The hubs divide the network into a set of localities, with one hub associated with each locality. These hubs also act as the routing beacons needed by the underlying coordinate-based routing protocol. The algorithms used to select the hubs are described in Section IV-B.
- Non-hub nodes join the locality of the closest available hub node, and keep that hub aware of changes in their dissemination variables through "join packets".
- When an update packet is first sent out, a set of fields in the packet identifies the dissemination predicate, namely: $<VariableId, Operator, Operand1, Operand2>$ where $VariableId$ identifies the dissemination variable that will be compared against, $Operator$ identifies the comparison operator (equal to, less than, greater than, less than or equal to, greater than or equal to, between), and $Operand1$ and $Operand2$ are the operands of the operator. $Operand2$ is only used in a 'between' comparison.

- An update is routed to the nearest hub of the originator, from where it is forwarded to all hubs in the network. Each hub then decides which nodes in its locality satisfy the dissemination predicate, and whether to route the update directly to the eligible nodes or to Trickle it to the entire locality. Given N nodes and n hubs in the network, if the sum of hop-counts from the hub of a locality to its eligible nodes exceeds $\frac{N}{n}$, the hub resorts to Trickle. Otherwise, it routes individually to each eligible node. This ensures that the number of packet transmissions in a locality remains within a fixed upper bound, even when the number of eligible nodes is large.
- The protocol ensures that updates are disseminated reliably by using exponential retransmit timers and end-to-end acknowledgments for update packets between hubs as well as between hubs and nodes.

IV. Dynamic Hub Selection

Due to limited storage space on the hubs, it is important to increase the number of hubs in the network as the number of nodes increases. Moreover, hubs should be selected such that they are relatively uniformly spread out across the network. With these goals in mind, we needed to handle two main design issues:

A. Hub Consistency

To enable nodes to agree on a consistent set of hubs, when a node receives a packet from some hub (say B) claiming that B is now the i'th hub, the node computes a hash of B's node ID. If this is less than the hash of the ID of the previously known i'th hub, the node switches to using B as the i'th hub. If the hash is the same for both node IDs, the lower node ID is used to break the tie. This guarantees that, for each hub number, all nodes converge to the same node ID. This scheme works better than simply converging to the smaller node ID, because node IDs often relate to physical location. Our hashing scheme prevents hubs from converging to the same physical region of a network.

B. Hub Selection Algorithms

When an existing hub runs out of storage for nodes in its locality, a new hub needs to be added to the network. The existing hub then runs an algorithm locally to select a new hub. Since each hub only knows about its own locality, the decision is made based only on the information available at the hub. It then sends a packet to that node to make it a new hub. The method outlined above ensures that knowledge of the i'th hub is consistent across the network.

We evaluated three hub selection algorithms to evaluate the quality of hubs selected with each algorithm. These are described as follows:

Farthest hub selection: In this scheme, the next hub selected is the node that is at the maximum distance from all the known hubs in the network. The goal is to spread the hubs as far apart in the network as possible. Suppose C_i = $<x_1, x_2, ..., x_n>$ are the coordinates of the i'th node from hubs 1, 2,..., n. Then, all nodes with x_j=1, for some j ($0<j<n+1$), are eliminated in the first step. Then, in the second step, the node with the maximum distance given by $\sum_{j=1}^{n}(|x_j|)$, is selected as a new hub.

Localized hub selection: This scheme attempts to select a node that is at minimum distance from all the known, non-hub nodes in the network. This is based on the idea that if there is a set of packets all coming from the same part of the network, then that part of the network probably needs its own hub. We implement this using the insight that nodes belonging to the same part of the network are likely to have similar coordinates and hence small inter-node distances. Therefore, our algorithm selects a node that has the smallest sum of inter-node distances from other nodes.

So if C_i = $<x_1, x_2, ..., x_n>$ are the coordinates of the i'th node from hubs 1, 2,..., n and C_j = $<y_1, y_2, ..., y_n>$ are the coordinates of the j'th node from hubs 1, 2,..., n, then we choose the node i such that Σ(distance(C_i, C_j)) [over all j, $0 \leq j \leq$ MAXIMUM_NODE_ID] is minimal, where distance(C_i, C_j) = $\sum_{k=1}^{n}(|C_i.x_k - C_j.y_k|)$.

Random hub selection: In this simple scheme, the next hub is selected randomly from among the known, non-hub nodes of the network.

The evaluation results are described in Section VI.

V. Optimizations

In order to further reduce the transmission cost of Starburst, we implemented several optimizations, the most important of which are as follows:

A. Next-Hop Tables

We found that the scoped flooding of BVR was a significant energy drain because it caused increased packet transmissions. To reduce the probability of resorting to scoped floods, nodes in our system keep a next hop to each of the nearby nodes that reports its current state. Thus, when a node sends a "join packet" to its nearest hub, every node in the path to the hub (including the hub itself) stores a next hop to the origin of the packet. This enables a hub to maintain a primary routing path to each node in its locality. If the primary routing path fails to deliver the packet, BVR can still use scoped flooding to reach the destination. Our evaluations show that in the vast majority of cases, nodes in the routing path need to use only the next hops and greedy routing to reach the destination.

To effectively use limited storage space, our policy is to store at each node only next hops with the best bidirectional quality links. This helps to avoid forwarding failures due to poor quality links.

B. Caching updates

When a hub sends an update to a node in its locality, intermediate nodes along the path also cache the update locally. This enables them to peek at join packets going through them, and respond with updates directly if the dissemination predicate is satisfied.

VI. Evaluation in Simulation

We implemented Starburst in nesC for TinyOS 2.x [1]. The implementation can be directly tested using the TOSSIM simulation environment [10] and can also be run on TelosB motes directly. Our original implementation used BVR for routing, but we have also implemented Starburst using the newer S4 routing protocol to overcome the limitations of BVR (such as its large packet header). Both BVR and S4 had to be ported from TinyOS 1.x [7] to TinyOS 2.x for this purpose.

A. Methodology

We collected the following results on a 20 x 20 node grid topology, with nodes separated by a distance of 15 feet (4.7 m) from each other. The topology was generated using the LinkLayerModel tool provided by TinyOS. The code was written in nesC [5] and executed on TOSSIM using its default signal-strength based radio model. One node, in case of BVR, and four nodes, in case of S4, were randomly chosen initially as hubs. All other hubs were then dynamically selected using the "Localized Hubs" hub selection scheme. One node at one corner of the grid was selected as the originator of an update. Each data point was obtained by taking the arithmetic mean of ten experimental runs. A subset of nodes of size n was randomly selected from a uniform distribution as ones that satisfy the dissemination criteria. To measure the transmission cost of dissemination using Trickle, we counted packet transmissions until nodes increased their Trickle interval to 64s. In cases where Starburst used Trickle for dissemination, we used the same rule to count packet transmissions.

B. Efficiency and Latency

Figure 2 and Figure 3 show how Starburst's performance and behavior changes as the number of eligible receivers increases. The figures compare the performance of Trickle with Starburst when it is implemented on top of both the BVR and S4 routing protocols.

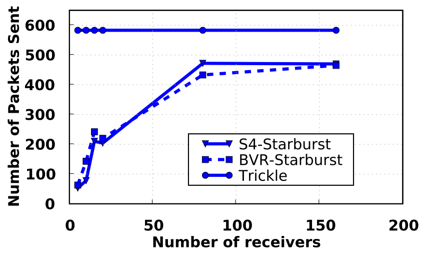

Fig. 2. Comparison of the transmission cost of Trickle with Starburst using BVR and S4 for routing as a function of increasing number of randomly selected receivers on a 20x20 TOSSIM grid .

The figures show that for a small number of eligible receivers, Starburst reduces the cost of transmitting packets by up to 85% as compared to Trickle. For small number of receivers, the dissemination latency is also much less than that of Trickle. In addition, the number of packet transmissions and latency needed to reach the selected receivers asymptotically

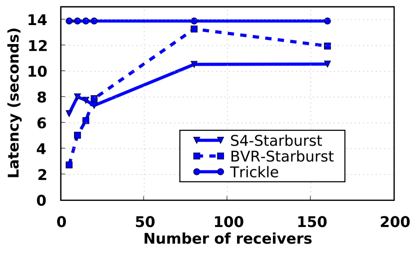

Fig. 3. Comparison of the latency of Trickle with Starburst using BVR and S4 for routing as a function of increasing number of randomly selected receivers on a 20x20 TOSSIM grid.

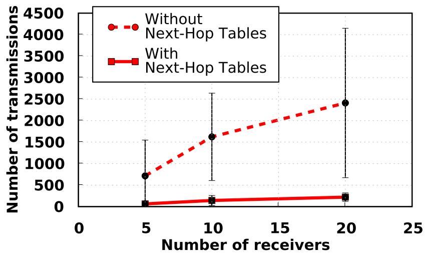

Fig. 4. Transmission Cost needed to disseminate to increasing numbers of randomly selected receivers with and without the next-hop tables optimization on a 20x20 TOSSIM grid. Error bars indicate a standard deviation of ±1 for each data point.

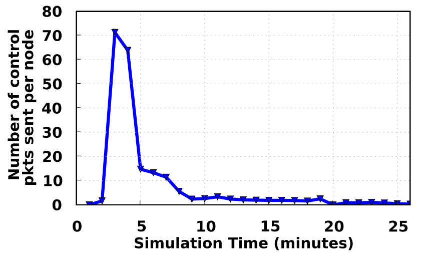

Fig. 5. Control traffic generated per node over time during a typical simulation run.

approaches a fixed limit with increasing number of receivers. This is due to the use of Trickle to disseminate to localities with large number of eligible receivers. Thus, in all cases, Starburst is able to perform at least as efficiently as Trickle in terms of both packet transmissions and latency.

In Figure 4, we show the difference in packet transmissions when next-hop tables are used and when they are not used. In the latter case, scoped flooding is much more of a problem, and therefore, performance is significantly hampered. Finally, we quantify the control traffic generated in Figure 5. Initially, there is a spike in traffic because in the presence of only one BVR routing beacon, routing is very inefficient. Once more beacons have been selected and coordinates become stable, the number of packets sent per node falls almost to zero.

C. Evaluating Hub Selection Algorithms

In this section, we observe the quality of hubs selected by our hub selection algorithms. We evaluate each of the three hub selection algorithms described in Section IV-B using two metrics: (i) the average distance from each hub to each node handled by that hub, and (ii) the number of nodes handled by each hub in its locality. The experiments were carried out on two set-ups composed of grids with 225 and 400 nodes respectively. The maximum number of hubs was set to eight in each case.

The first metric we use for evaluation is the average estimated distance in hops between hubs and their children (i.e. nodes in their locality). The intuition behind this metric is that the lower the average distance between hubs and their children, the lower the average number of transmissions needed by a hub to reach its children. The results for this metric are shown in Figure 6 which shows that for the 400 node grid, the lowest average hub-to-children distance (in hops) is achieved using the localized hub selection scheme, whereas for the 225 node grid, it is achieved using the farthest hub selection scheme.

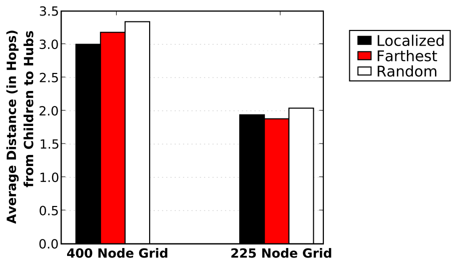

Fig. 6. Quality of Hubs selected by various algorithms in terms of average distance of the hubs from their children.

We further validate this by comparing the way each algorithm distributes children among the hubs with an ideal distribution of children. We define the ideal distribution in our relatively uniform grid topology as one where all the hubs handled exactly the same number of nodes. In an ideal distribution, we would only require adding more hubs if all the hubs were completely filled. Otherwise, some hubs might continue to have few children, while hubs with more children continue to create new hubs. As a result, the distribution of children across hubs can become even more skewed.

We used the Kantorovich-Wasserstein (KW) distance [6], implemented in open source as the Earth Mover's distance [14], to quantify the difference between the ideal distribution and the distributions generated by the three algorithms. The values of the KW distance for each algorithm are given in Table I. The table shows that the best distribution of hubs for the 400 node grid is achieved with the localized hub selection scheme (KW distance 32% less than for random hub selection), whereas for the 225 node grid, it is achieved using the farthest hub selection scheme (KW distance 16% less than for random hub selection). We conclude that our hub selection algorithms result in better performance than simple random hub selection.

	Localized	Farthest	Random
400 nodes	8.08	12.21	11.93
225 nodes	6.2	5.34	6.375

TABLE I
KW DISTANCE OF OPTIMAL DISTRIBUTION FROM DISTRIBUTION GENERATED BY EACH HUB SELECTION SCHEME.

D. Comparison with Existing Dissemination Protocols

Figures 7 and 8 compare the performance of Starburst with that of Trickle for three different cases: (i) n receivers randomly scattered in the grid, (ii) n receivers randomly located in the 10x10 square in the corner diagonally opposite the sender, and (iii) n receivers randomly located in the 10x10 square corner near the sender, using n=5, 10 and 20 in each case. The experiment was done on the same 20x20 topology described previously.

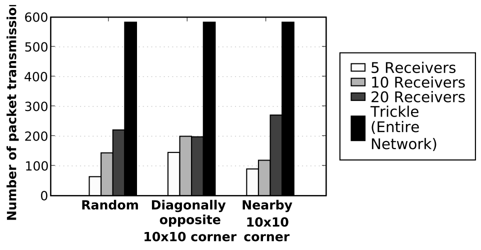

Fig. 7. Comparison of packet transmissions by Starburst with that of Trickle with 5, 10 and 20 receivers on a 20x20 TOSSIM grid in different scenarios.

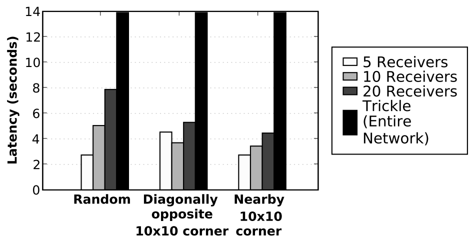

Fig. 8. Comparison of dissemination latency of Starburst with that of Trickle with 5, 10 and 20 receivers on a 20x20 TOSSIM grid in different scenarios.

The figures show that even with a relatively large number of receivers (n=20), Starburst reduces latency by almost 50% while sending 50% fewer packets than Trickle. With a smaller number of eligible receivers, the updates are disseminated with an even lower latency and with as low as 15% of the packet transmission cost. Clearly, for small subsets, Starburst is able to perform better than Trickle in terms of both packet transmissions and latency. We do not compare against Trickle for larger subsets, since in that case Starburst simply reduces to Trickle.

Due to the difficulty of porting Melete to TinyOS 2.x, we were unable to directly compare Starburst's performance with Melete. However, the results for Melete published in [18] indicate that its performance is equivalent to Trickle in scenario (ii) mentioned above, and does not show as much of an improvement as Starburst in scenario (i). It should be noted though that in scenario (iii), Melete is likely to outperform

Starburst, since its forwarding region would be limited to one contiguous corner of the grid.

VII. Testbed Validation

We deployed and validated Starburst on Motelab [17], a sensornet testbed in Harvard University. At the time of the experiments, Motelab consisted of almost 160 active nodes spread over three floors. Nodes communicate with each other over the wireless channel, but can also send logging information to a database through an Ethernet backchannel. S4 was used as the routing layer in all of the testbed experiments. In our experiments, we measured the number of transmissions and latency required to disseminate updates to k=5, 10, and 20 receivers in Motelab. We selected these receivers from two distributions. The first was a uniform distribution: we randomly selected nodes. The second was a stride distribution: we selected a random node i then selected $i + \frac{n}{k}, i + \frac{2n}{k} \dots$, where $n = 160$ is the number of nodes on Motelab. The goal of the stride selection was to select a set of nodes which were likely to be physically distant.

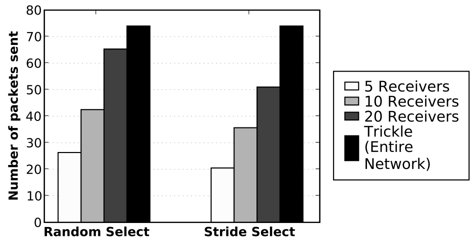

Fig. 9. Comparison of packet transmissions using Starburst and Trickle on Motelab.

The results given in Figures 9-10 show that Starburst performs fairly well on Motelab. It disseminates data items to eligible nodes with much fewer packet transmissions than Trickle. The latency is also smaller than that of Trickle for small eligible subsets, and only slightly exceeds that of Trickle for larger subsets.

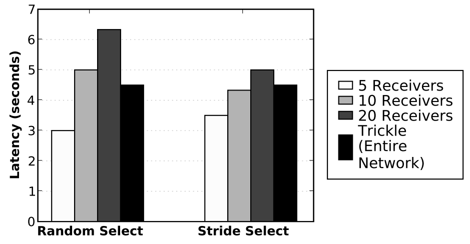

Fig. 10. Comparison of latency with Trickle on Motelab. The latency for Starburst was much less than Trickle for small subsets, and only slightly exceeded that of Trickle for larger ones.

VIII. Conclusion

In this paper, we presented Starburst, a protocol for disseminating data items to small subsets of nodes in a sensor network. Starburst uses a combination of coordinate-based routing and scoped Trickle to yield substantial performance gains over existing dissemination protocols. Our simulations and testbed evaluations confirm that Starburst can enable selective dissemination with fewer packet transmissions and low latency.

References

[1] TinyOS 2.x. http://www.tinyos.net/tinyos-2.x/doc.

[2] A. A. Abbasi and M. Younis. A survey on clustering algorithms for wireless sensor networks. *Computer Communications*, 30(14-15):2826 – 2841, 2007. Network Coverage and Routing Schemes for Wireless Sensor Networks.

[3] M. Caesar, M. Castro, E. B. Nightingale, G. O, and A. Rowstron. Virtual ring routing: Network routing inspired by dhts. In *Proceedings of the ACM SIGCOMM Conference*, 2006.

[4] R. Fonseca, D. Culler, S. Ratnasamy, S. Shenker, and I. Stoica. Beacon vector routing: Scalable point-to-point routing in wireless sensornets. In *Proc. 2nd Symposium on Networked Systems Design and Implementation (NSDI 05)*, 2005.

[5] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *SIGPLAN Conference on Programming Language Design and Implementation (PLDI'03)*, June 2003.

[6] C. Givens and R. Shortt. A class of wasserstein metrics for probability distributions. In *Michigan Math. Journal*, volume 31, pages 231–240, 1984.

[7] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System Architecture Directions for Networked Sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000. TinyOS is available at http://webs.cs.berkeley.edu.

[8] K. Iwanicki and M. V. Steen. On hierarchical routing in wireless sensor networks. In *IPSN '09: Proc. 8th conference on Information processing in sensor networks*, 2009.

[9] P. Levis and D. Culler. The firecracker protocol. In *Proceedings of the 11th ACM SIGOPS European workshop: beyond the PC*, 2004.

[10] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Simulating large wireless sensor networks of tinyos motes. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.

[11] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[12] Y. Mao, F. Wang, L. Qiu, S. S. Lam, and J. M. Smith. S4: Small state and small stretch routing protocol for large wireless sensor networks. In *4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)*, 2007.

[13] J. Newsome and D. Song. Gem: graph embedding for routing and data-centric storage in sensor networks without geographic information. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 76–88. ACM Press, 2003.

[14] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the 1998 IEEE International Conference on Computer Vision*, pages 59–66, 1998.

[15] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN)*, Jan. 2005.

[16] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 381–396. USENIX Association, 2006.

[17] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *Proc. Fourth International Conference on Information Processing in Sensor Networks (IPSN'05), SPOTS Track*, April 2005.

[18] Y. Yu, L. J. Rittle, V. Bhandari, and J. B. LeBrun. Supporting concurrent applications in wireless sensor networks. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, 2006.