Data-Centric Routing using Bloom Filters in Wireless Sensor Networks

[#] Peter Hebden, Adrian R. Pearce
NICTA Victoria Research Laboratory
Department of Computer Science and Software Engineering, University of Melbourne
Melbourne, Australia, {phebden,adrian}@csse.unimelb.edu.au

Abstract

This paper presents a paradigm for reducing communication costs in wireless sensor networks. The first component is our Distributed Asynchronous Clustering protocol (DAC), which self-organises the network into an infrastructure that supports in-network processing, routing, and deployment. The second component, and the focus of this paper, is a data-centric routing protocol where cluster heads build and maintain sets of Bloom filters to inform routing decsions and filter out unproductive messages. While other data-centric protocols use a flat topology and rely to some extent on flooding, our protocol exploits a two tier hierarchy to provide an adaptable, scalable, and intelligent routing service that is expected to reduce the number of transmissions and extend network lifetime.

1. Introduction

Large scale wireless sensor networks (WSNs) of autonomous, resource constrained nodes may be required to operate for long periods of time without maintenance. In such networks, information of interest should be delivered to the application in a timely and energy efficient manner according to protocols that minimise wasteful radio transmissions.

In general, different applications may require different communication protocols or strategies to maximise network lifetime. Here we consider a data gathering application and its *data-driven* network of wireless sensors. In a data-driven network, a query instructs each node to sense its environment at a certain rate, for a period of time, and transmit matching data back to the sink. Network traffic is composed of packets that flow periodically between many sensors and one or more sinks. While sensor data of interest must be guided to sinks, queries should be disseminated to appropriate nodes without incurring undue communication costs. This becomes more difficult where there are numerous sinks injecting a variety of queries into the network, and only a small fraction of sensors generate data of interest for each query.

To reduce communication costs in large data gathering sensor networks, we propose a combination of techniques from clustering, *data-centric* routing, and Bloom filters. We use the Distributed Asynchronous Clustering (DAC) protocol [1] to generate a near optimal number of well separated cluster heads. As is generally the case, sensor nodes are partitioned into a set of clusters where each sensor may only transmit data to its cluster head. Cluster heads process member data and/or reports from other cluster heads, make routing decisions, and forward the result to another cluster head or the sink. While DAC self-organises the network into hierarchical clusters to support in-network processing, its well separated cluster heads provide an infrastructure for intelligent routing.

Directed diffusion [2], [3], [4], [5] is a well know data-centric routing protocol that has been recognised as the right solution [6]. However, it does not take into account wireless link quality, and it does not scale well in networks where there are many sinks transmitting many different queries. Essentially, each node's routing table grows too large and a more space efficient data structure is needed.

Flooding the network with a message increases the probability that all reachable nodes will receive it. However, an ideal energy efficient protocol would only transmit a message when necessary. If cluster heads are enabled with an adaptive and scalable routing capability, then large systems can move away from flooding and toward optimal messaging. In this paper distributed data structures based on Bloom filters [7] are used by cluster heads to make better routing decisions [8].

Section 2 introduces Bloom filters, and Section 3 discusses WSNs and some design considerations. Section 4 provides some background information on routing, and in Section 5 we present our Bloom Gradient Routing (BGR) protocol. And finally, our conclusions.

2. Bloom Filters

Bloom filters were introduced in 1970 when computer memory was extremely scarce [7]. Early applications include hyphenation, spelling dictionaries, and joining tables in databases. With the advent of the Internet, their usefulness was rediscovered. Essentially, if an application can tolerate a low rate of false positives and memory is scarce, then Bloom filters may provide a practical optimisation technique [9]. Since the WSN domain is far more resource constrained than the Internet, we expect to see many new applications of Bloom filters. Next we provide some details on this data structure and an exemplary Internet application.

A. The Data Structure

A Bloom filter is a space efficient, randomised data structure for representing a set and supporting set membership queries [7]. It is a *filter* in the sense that it can be used to *filter out* an element that does not belong to the set. The space efficiency benefit is gained at the cost of a loss of information and a low rate of false positives.

The original Bloom filter used a simple bit vector, i.e. one bit per cell. To construct a filter, the following procedure is used. Suppose n elements in set S from a large universe U, and we have m bits of memory available. The hash area may be defined as m individual bits with indices 0 through $m-1$. Generate k different indices with k different hash functions applied to each element, and set those bits to 1. Figure 1 shows a filter after one element has been stored. To test an unknown element for set membership, follow the same procedure except just compare each of those k bits to 1. If all are equal to 1, then the element is recognised and probably is a member, else it definitely is not a member.

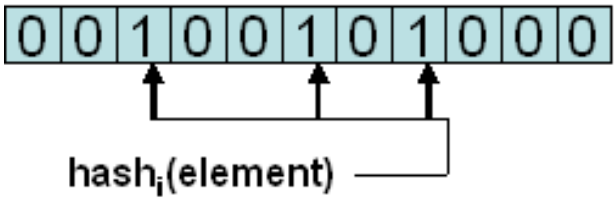

Fig. 1: Storing elements in a Bloom filter: Up to k bits will be set to 1 by k hash functions and one element.

A false positive occurs when we test a new element and it hashes to k bits that have already been set to 1 by one or more elements in S. For example, given a filter where 50% of the bits have been set to 1 and one hash function, the expected false positive rate would be 0.5. However, if two hash functions are being used, the probably that an unseen element will test positive is 0.25.

Usually we want to minimise the rate of false positives while using a limited amount of memory. Given n elements and m bits of memory, what is the optimal number k of hash functions? An optimal k turns on 50% of the bits when the filter is built [9].

$$k = \frac{m}{n} \ln 2 \tag{1}$$

The error rate depends on m, k and n. Let p equal the probability of a bit being zero after the filter has been built. Let f equal the probability of a false positive when the filter is being tested.

$$p = \left(1 - \frac{1}{m}\right)^{kn} \tag{2}$$

$$f = (1-p)^k \tag{3}$$

Notable properties of Bloom filters include: no false negatives, false positive rate is tunable, space requirement does not depend on element size, bitwise operations on Bloom filters are extremely efficient, and they may be implemented with more than one bit per cell to support counting Bloom filters. Consequently, they have been found to be useful in network applications [9].

B. An Exemplary Internet Application

Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol is one of the most cited papers on using Bloom filters in the Internet context [10]. In 1998, when the authors first presented their protocol, web cache sharing was not widely implemented due to the overhead of existing protocols.

Popular web sites may be served by a group of cooperating web servers, known as proxy servers, where each server hosts a subset of the site's pages. Each server hashes its directory to a counting Bloom filter and sends a copy to the group. When a server receives a query, it tests its set of filters for a *hit* to determine which server in the group probably hosts the requested page. When a host deletes a page, it decrements cell values in its filter. Each directory entry was represented with just 8 bits; this reduced storage requirements and communication overhead [10].

Cluster heads may play a role similar to web proxy servers in WSNs. As we will explain in Section 4, they could maintain sets of Bloom filters that represent messages received from data sources and data sinks, and use that information to decide where to forward a query or sensor data.

3. Networking Wireless Sensors

In this paper we have assumed a WSN of small, power constrained sensor nodes. The most essential function of its network layer is to route data from source to destination. However, a protocol that simply minimises the number hops is not well suited to WSNs. The packet reception rate (PRR), opportunity for in-network processing, and distance of each link should be considered.

PRR: Wireless communication is notoriously unpredictable. The quality of each link depends on the environment, frequency spectrum, modulation schemes, and the hardware itself. Link quality can vary suddenly with time and small spatial displacements. Zhao and Govindan systematically evaluate packet in delivery in [11]. Connectivity analysis, neighborhood management, and routing is explored in [12].

In-network processing: Hierarchical clustering creates opportunities for in-network processing. For example, instead of thousands of sensing nodes congesting the network with data transmissions, cluster heads aggregate data by doing some computation (max, min, mean, median, summation, etc.), thereby reducing the quantity of data transmitted.

Distance: The energy required for radio communication rises dramatically with distance. In general, the output power required to transmit over distance d is proportional to d^n where $n >= 2$ and depends on distance and the environment [13]. In such a network, techniques for minimising power consumption

are essential for prolonging network lifetime, and since radio communication is by far the most power intensive task, we designed our clustering and routing protocols to reduce this cost.

Naturally, well separated cluster heads play an important role in reducing energy consumption and network latency [1]. BGR uses cluster heads and sets of Bloom filters to increase expected network liftetime.

4. Data-Centric Routing

WSN messages must be routed between sink(s) and data sources via resource constrained nodes such that QoS and lifetime guarantees are met. Here we assume that the network has used the DAC protocol [1] to self-organise into a set of clusters with well separated cluster heads that support the flow of information between the sink(s) and many sensors. In a typical data gathering application, the sink sends a query to the network and data of interest flows back to the sink. To reduce unnecessary transmissions, each cluster head needs routing information to forward queries in the direction of relevant nodes and sensor data in the direction of the interested sink.

In Section 5 we will present a scalable, adaptive, and energy efficient protocol called Bloom Gradient Routing (BGR). This protocol uses Bloom filters to store information about query and sensor messages received from proximate nodes and inform routing decisions. Before presenting BGR in more detail, Sections 4-A and 4-B provide some background on routing and related work.

A. Routing Protocols

Routing protocols may be classified as *address-centric* or *data-centric*. Address-centric is better suited to networks with a small number of possible destinations because small explicit routing tables may be maintained at each node. Data-centric is better suited to networks with a very large number of possible destinations and where the application is typically interested in gathering data or information from the network but not from individually addressable nodes, i.e. explicit routing tables are not required.

In address-centric routing, packets are routed from one addressable node to another via the shortest path according to some metric. Data-centric routing is more complex. Queries in the form of *named data* should be routed from the sink to sensor nodes that are likely to have data of interest. Sensor node data should be routed to the sink along paths that facilitate data aggregation. Reverse multicast trees provide paths that naturally facilitate data aggregation. Figure 2 shows an example where the number of transmissions has been reduced from 6 to 4. However, optimal data aggregation requires the formation of a minimum Steiner tree on the network graph and is NP-complete [14], and while approximation algorithms do exist, we assume that data aggregation is done opportunistically.

B. Related Work

Reverse path forwarding is a general approach to data dissemination [15]. Essentially, sensor data flows in the reverse direction of query propagation to reach the sink.

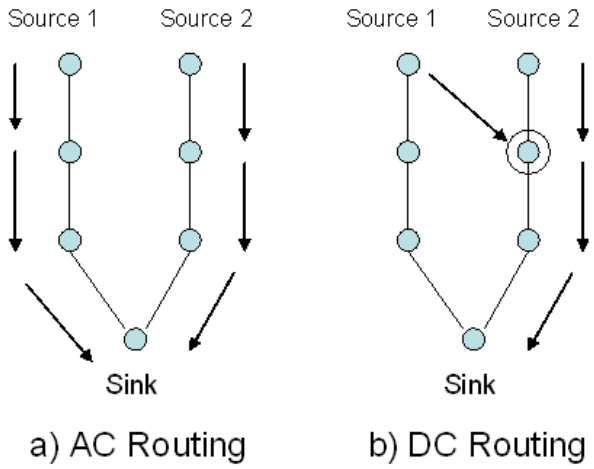

Fig. 2: The network on the left is *address-centric*: each packet is routed by the shortest path to the sink. The network on the right is *data-centric* in the sense that packets are routed to facilitate data aggregation.

1) Directed Diffusion: This [2], [3], [4], [5] is an important data-centric routing protocol for flat sensor network topologies. It provides mechanisms for routing queries and sensor data. Each sensor uses an attribute-based naming scheme that names the data it generates according to one or more attributes. Initially the sink has no prior knowledge of which nodes probably have data of interest. So, it floods the network with the query, i.e. the query is propagated throughout the network by adjacent nodes.

The interaction of disseminated queries and matching sensor data establish *gradients* for data to flow toward the sink that expressed that interest. For example, a sink may flood the network with an interest in the form:

$type : temperature, op : GT, value : 20$

Sensors with matching data generate attribute-value data tuples in the form:

$type : temperature, id : 12, value : 21$

The data is routed along reverse paths to the query's source. After the sink receives this data, it reinforces some paths to the data sources. Based on the strength of the gradient, intermediate nodes forward sensor data and queries for data along efficient paths. This particular version of Directed Diffusion is known as two-phase pull [16].

2) Gradient Based Routing: A number of other researchers have also found the gradient concept useful [17], [18], [19]. Messages flow through a multi-hop network according to the gradient stored in each node. In some implementations, gradient is based on the "height" of a node, which is proportional to its distance from the sink [17]. In that case, data flows downhill toward a sink, and queries flow uphill toward sensor nodes. However, regardless of implementation, gradient indicates direction, i.e. the next node on a multi-hop path.

Gradient Broadcast (GRAB) addresses the problem of robust data forwarding to the sink via unreliable sensor nodes over error-prone wireless channels [18]. Their sink maintains a *cost*

field, and each node maintains an indicator of the cost of forwarding a packet from itself to the sink. Data flows toward lower cost nodes, so the cost field implies a gradient because direction is implicit. However, the cost value at each node is not the same as the gradient (vector) defined in Directed Diffusion. For robust delivery, GRAB uses a mesh of multiple paths from source to sink. The source assigns a *credit* to each report it transmits to control the degree of path redundancy, i.e. more credit translates into a wider mesh of multiple paths.

Gradient-Ascending Stateless Protocol (GRASP) was presented in [19]. In GRASP, a forwarding history is stored at each node on the path from source to sink - each history is represented by a Bloom filter. When a source node transmits a packet, the packet's *origin address* is hashed to a Bloom filter at each node along the multi-hop path to the sink. GRASP was primarily motivated by the theory that membership-based broadcast is more energy efficient than flood-based dissemination. However, because GRASP defines a protocol for routing queries from a sink to specific, addressable sensor nodes, we see it as an *address-centric* protocol. This is in contrast with many research efforts which emphasise the *data-centric* nature of WSNs.

5. BLOOM GRADIENT ROUTING

The design of BGR reflects the costs and benefits of flooding for message dissemination. Flooding the network guarantees delivery, but at the cost of implosion, overlap, and resource blindness [20]. An interesting paper by Braginsky and Estrin presents *Rumor Routing*, which is proposed as a compromise between flooding queries and flooding sensor data [21]. *Gossip Routing* also provides a scheme that reduces flooding - nodes flood by sending the message to random neighbors and redundant connectivity allows most nodes receive it [22]. In contrast, BGR implicitly learns from application, network, and device characteristics. This information is stored and shared in the form of Bloom filters and aggregates. The sink is able to decide, with some degree of confidence, whether or not it is worth sending a query into the network. If so, the query only needs to be sent to the cluster heads - a small percent of nodes.

A. Bloom Filter Gradients

The combination of hierarchical clustering and *Bloom filter gradients* provides a simple technique for reducing transmissions where queries are disseminated to unkown data sources in the network rather than to nodes with specific IDs. Figure 3 shows one cluster head with eight pairs of *port Bloom filters* (small rectangles), one pair for each port, and a *cluster Bloom filter* at the center of cluster head CH. The cluster Bloom filter represents data messages from cluster member sensor nodes. When a cluster head receives a query, it uses its cluster Bloom filter to decide whether or not to query its own members. The port Bloom filters represent received data from specific neighbors, and are used to decide whether or not to forward the query to each neighboring cluster head. In Figure 3, the small white rectangles indicate empty, the light gray indicate received data, and the black indicate received queries. Next we discuss using port filters for routing between sinks and cluster heads in more detail.

Each sink transmits a message to establish the shortest path from each cluster head to each sink. We assume that there are a relatively small number of sinks, and storing next hop path data for each sink does not require a significant amount of memory. However, a simple hop count metric does not necessarily define the best way to route data back to the sink, i.e. a small number of long hops may be less reliable. A better metric might consider several properties of each link such as Euclidean distance, energy levels of transmitter and receiver, and packet reception rate.

Figure 4 shows four cluster heads and their gradients implemented with Bloom filters. Here we assume that each cluster head has eight adjacent cluster heads ("neighbors") and eight ports. Each port maintains two Bloom filters: one for received query messages and one for received data messages. They represent what has been received from adjacent cluster heads. Each filter accumulates a data-centric *residue* when the payload of each received message is hashed to k cells in the port's Bloom filter. In a one bit per cell filter, this creates a representation of what messages have been received from a specific neighbor. After messages have been received by a node and stored in its filters, the node can evaluate its filters to decide which neighbor(s) to forward a query or data message to. As shown in Figure 4, query messages (solid arrows) follow matching data residues, and data messages (dashed arrows) follow matching query residues. However, each cluster head needs information about the probability that a destination of interest is reachable via one neighbor versus another - such as indicated by a received message count.

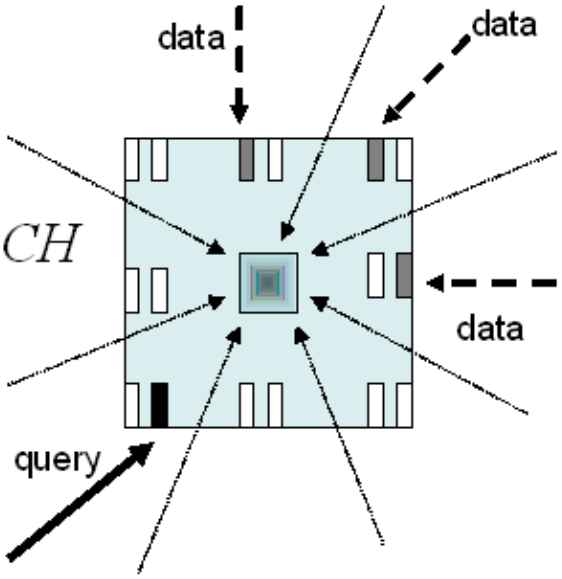

Fig. 3: Cluster Head Enabled with Bloom Filters: one pair at each port for each cluster head neighbor, and one *cluster filter* (center square).

B. Counting Bloom Filters

Counting Bloom filters are a space efficient representation of the number of times an element has been stored. In BGR, messages are hashed to a Bloom filter such that each time a certain attribute-value pair, or *element*, is received, k cells in the filter are incremented. After the received message is processed, the node may forward the message to its neighbor(s). By testing the element against the Bloom filters for each neighbor, the node can determine which neighbor(s) to forward to.

BGR routing decisions may depend on several parameters, but here we assume a simple case where each received message is forwarded to the highest scoring neighbor(s). When a node is ready to forward a message, it tests the filters for its other neighbors. If none test positive, it forwards the message to all other neighbors. If one tests positive, it forwards the message to that neighbor. However, if more than one tests positive, it must choose the best neighbor(s) for the message according to a scoring function. For a given message, the highest scoring filter will be the one with the largest minimum count.

$$score(filter) = min(h_{i=1..k} : filter[h_i(element)]) \quad (4)$$

In other words, for hash functions $h_{1..k}$, a filter is scored by hashing the message of interest to k cells in the filter; the minimum count is the score for that filter. A higher score indicates a stronger gradient from that neighbor, so the packet is forwarded out the port(s) represented by the highest scoring filter(s).

C. Adaptation

As discussed above, a cell's count is incremented each time an element hashes to its location in the Bloom filter. If we assume just 4 bits per cell, then a cell becomes saturated after 15 hits. This leads us to two closely related problems. First, when a cell becomes saturated it is not able to record the successful reception of new packets. Network conditions may change such that good links go bad. In other words, the network will lose its ability to adapt to recent events. Second, the cell counts may reflect stale information.

BGR adapts to change by applying a decay function to some filter cells each time a new packet is received. The message is hashed to k cells. For each cell, the decay function will decrement a cell's function according to a probability distribution. The probability of decay p depends on the cell's count and maximum value: $p = count/max$.

Next each cell's value is incremented to record the current message. This procedure guarantees that counts stay within range and facilitates adaptation without explicitly considering the passage of time. The decay function could also be applied as a function of count and time, where decay rate is an application specific parameter.

D. Link Quality

Link quality, as measured by message delivery performance, may be seen as the most basic aspect of wireless communication. Ideally, a simple mechanism would measure actual performance of links, which may be asymmetrical, and discard links to poorly performing neighbors. It has been estimated that, depending on the load, anywhere between 50% and 80% of communication energy is wasted on repairing lost transmissions [11].

The counting Bloom filters maintained by the cluster heads also serve as a topology control mechanism which favors higher quality links over lower quality links. Successful transmissions increment counters at the receiver and, in effect, "pull" messages along the more reliable paths.

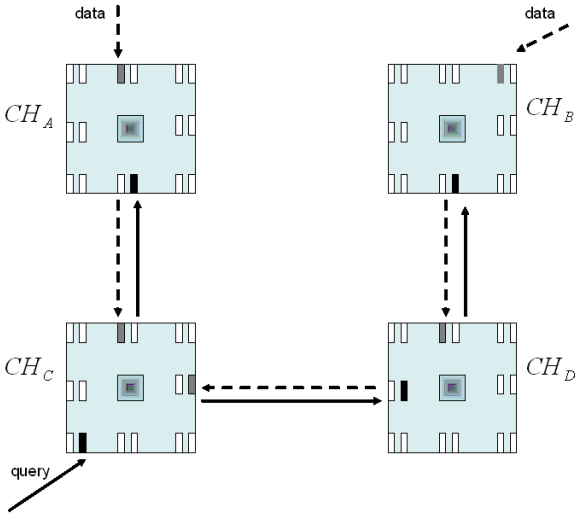

Fig. 4: Four Cluster Heads and their Filters: messages leave a residue as they are routed via cluster heads.

E. Self-Organisation and Data Discovery

BGR reduces unproductive communication by disseminaiting information proactively. During cluster formation each sensor transmits its measurements to its cluster head. Each cluster head stores member data in its cluster Bloom filter and sends a copy to the sink(s). These filters are copied to cluster head port Bloom filters en route up to h hops aways from their origin, where h is a tunable parameter. The subsequent transmission of queries and data, and the residues they leave in port Bloom filters, build multi-hop paths between cluster heads and the sink(s). As a result, a hierarchy of Bloom filters is built that filters out unproductive query and data transmissions. At the top level, the sink can evaluate its set of cluster Bloom filters to decide whether to query the network or not. Each cluster head can examine its port Bloom filters to help decide where to forward a query. It can also evaluate its own cluster Bloom filter to decide whether or not to query its members.

Next we describe part of this filtering process in more detail. As in Section 5-A, each cluster head maintains a set of Bloom filters. One represents data reported directly from cluster members directly, and the others represent query or data sources likely to be found via each neighbor. However, instead of assuming just two Bloom filters for each port as we did in Section 5, here we assume two arrays of Bloom filters for

each port. This compound data structure can be use to represent not only what has been received from each neighbor, but also, from how far away. The first filter in each array represents data received from cluster heads one hop away, the second from two hops, and so on.

Suppose we want data from sensor nodes that measure moisture M, temperature T, and salinity S. Such a tuple's format could be represented by a set of strings: $M = 40, T = 20, S = 10$. Suppose the application is interested in data from sensor nodes where moisture levels M range from 40 to 49, temperatures T from 20 to 29, and salinity S from 10 to 19. First the query must be routed to cluster heads that, according to the port Bloom filters, probably have such data. Here we assume that measurements were binned into deciles before hashing. We can test the elements of our query for set membership by hashing each element's string representation: $hash(M = 40) \wedge hash(T = 20) \wedge hash(S = 10)$ to the cluster Boom filter. The order that elements are hashed does not matter. The query tests negative as soon as an element hashes to a zero in the Bloom filter. If the query tests positive, then such data is probably in the cluster head's subtree. Second, the cluster Bloom filters at each cluster head must be tested for set membership. If the query test positive, then one or more cluster members probably have data of interest.

Although this paper has focussed on data-centric routing, other elements such as sensor IDs could also be stored in Bloom filters and used for routing or other functions where testing for set membership is useful.

6. CONCLUSION

We have presented a scalable paradigm for messaging in a WSN where the DAC protocol was used to self-organise the network into clusters and BGR was used for routing. A hierarchy of Bloom filters were used by the sink, cluster head ports, and each cluster head itself to filter out unpromising transmissions.

Cluster heads used Bloom filters to make data-centric routing decisions based on past events, explicit and implied shortest path information, and the quality of each wireless link. Each cluster head was assumed to have wireless communication links with up to eight neighboring cluster heads. The end points of each link were considered to be a port for transmission or reception. The residue of each message received was stored in the receive port's Bloom filter, so successful message transmissions left a reverse trail for other messages to follow. Data messages followed query message trails back to a sink or sinks, and query messages followed data message trails to sensor nodes. Successful transmissions reinforced the trails, so the strength of each trail was an indication of link quality. In addition, because some nodes may run low on power and network topology may change over time, our routing protocol was designed to be adaptive. A decay function was applied to each Bloom filter to prevent over saturation and to make recent events more likely to influence routing decisions.

We believe that when BGR is implemented on an infrastructure of well separated cluster heads such as generated by DAC, the result is a network that is self-organising, scalable, adaptive, reliable, and more energy efficient than other data-centric protocols that rely on flooding or random forwarding.

REFERENCES

[1] P. Hebden and A.R. Pearce. Distributed Asynchronous Clustering for Self-Organisation of Wireless Sensor Networks, *Fourth International Conference on Intelligent Sensing and Information Processing (ICISIP)*, Bangalore, India, December 2006.

[2] D. Estrin, R. Govindan, J. Heidemann and S. Kumar. Next Century Challenges: Scalable Coordination in Sensor Networks, *In Proceedings of the Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99)*, August 1999, Seattle, Washington.

[3] C. Intanagonwiwat, R. Govindan, D. Estrin. Directed Diffusion: A Scalable and robust Communication Paradigm for Sensor Networks, *Proc. 6th Annual International Conference on Mobile Computing and Networks (MobiCom)*, Boston, MA, 2000.

[4] C. Intanagonwiwat, R. Govindan and D. Estrin. Directed Diffusion for Wireless Sensor Networking, *IEEE/ACM Transactions on Networking*, 11(1): 2-16, Feb 2003.

[5] F. Silva, J. Heidemann, R. Govindan, and D. Estrin. Directed Diffusion, *Technical Report ISI-TR-2004-586, USC/Information Sciences Institute*, January, 2004.

[6] D. Niculescu, Communication Paradigms for Sensor Networks, *IEEE Communications Magazine*, March 2005

[7] B. Bloom. Space/Time Tradeoffs in Hash Coding with Allowable Errors, *CACM*, 13(7):422-426, July 1970.

[8] P. Hebden and A.R. Pearce. Bloom filters for data aggregation and discovery: a hierarchical clustering approach, *Second International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Melbourne, Australia, December 2005.

[9] A. Broder and M. Mitzenmacher, Network Applications of Bloom Filters: A Survey, *Internet Mathematics*, 1(4):485-509, 2004

[10] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary Cache: a scalable wide-area Web cache sharing protocol, *IEEE/AACM Transactions on Networking*, 8(3):281-293, 2000.

[11] J. Zhao and R. Govindan. Understanding Packet Delivery Performance in Dense Wireless Sensor Networks, *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November, 2003.

[12] A. Woo, T. Tong, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks, *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[13] G. J. Pottie and W. J. Kaiser. Wireless Integrated Network Sensors, *Communications of the ACM*, vol. 43(5), May 2000, pp. 51-58.

[14] B. Krishnamachari, D. Estrin, S. Wicker. Modelling Data-Centric Routing in Wireless Sensor Networks. *IEEE Infocom*, 2002.

[15] F. Ye, H. Luo, S. Lu, and L. Zhang. Dissemination Protocols For Large Sensor Networks, *Wireless Sensor Networks*, Raghavendra, Sivalingam, and Znati, Editors. Kluwer Academic Publishers, 2004.

[16] J. Heidemann, F. Silva and D. Estrin. Matching Data Dissemination Algorithms to Application Requirements, In *Proceedings of the ACM SenSys Conference*, Los Angeles, CA, 2003.

[17] C. Schurgers and M. B. Srivastava, Energy efficient routing in wireless sensor networks, *Military Communications Conference*, 2001.

[18] F. Ye, G. Zhong, S. Lu, and L. Zhang. Gradient Broadcast: A robust data delivery protocol for large scale sensor networks, *ACM Wireless Networks (WINET)*, vol. 11, no. 2, March 2005.

[19] Jai-Jin Lim and K.G. Shin. Gradient-Ascending Routing via Footprints in Wireless Sensor Networks, *26th IEEE International Real-Time Systems Symposium (RTSS)*, 5-8 December 2005.

[20] J. Kulik, W. Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks, *Wireless Networks*, Volume 8 Issue 2/3, March 2002

[21] D. Braginsky and D. Estrin, Rumor Routing Algorithm For Sensor Networks, *Workshop on Sensor Networks and Applications (WSNA)*, Atlanta, Georgia, 2002.

[22] Meng-Jang Lin, K. Marzullo, and S. Masini. Gossip Versus Deterministically Constrained Flooding on Small Networks. *14th International Conference on Distributed Computing (DISC)*, Oct. 2000.