Gradient-Ascending Routing via Footprints in Wireless Sensor Networks *

Jai-Jin Lim and Kang G. Shin
Real-Time Computing Laboratory, EECS Department
The University of Michigan, Ann Arbor, MI 48109-2122, USA
{jjinlim,kgshin}@eecs.umich.edu

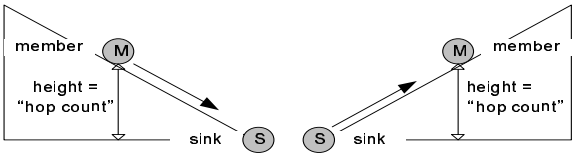
Abstract

*A novel gradient-ascending stateless routing protocol, called GRASP (*GRadient Ascending Stateless Protocol*), is proposed for stationary wireless sensor networks. GRASP is built with a novel packet forwarding method called FBF (*Footprint-Based Forwarding*), in which each node maintains a Bloom filter that holds a dejavu image for the nodes whose packets were relayed through the filter. Forwarding a packet through networked sensors is then just running a membership test of the packet's destination against the networked Bloom filters, either dropping the packet before it reaches the destination or eventually delivering it to the destination.*

An extensive simulation study with various routing scenarios has shown the proposed method to achieve about 99% packet delivery while incurring only a small amount of overhead. It also achieves more than 86% packet delivery even for a lossy channel with up to 30% loss rate. Furthermore, our comparative evaluation against AODV or simple flooding shows that GRASP achieves a comparable, or even better, performance in terms of the packet delivery ratio and overhead under the simulation settings investigated. GRASP is shown to reduce the delivery overhead by a factor of 2.37 over AODV or 3.11 over the simple flooding, while providing more robustness to packet losses—this is a significant improvement for low-power sensors in a lossy wireless environment. Manipulating Bloom filters incurs a small space overhead and does not create any protocol conflict with the underlying gradient-descending protocols, thanks to its stateless routing.

1. Introduction

What distinguishes emerging sensor networks from the Internet is not only their severe resource constraints and unattended operational environment, but also their very different set of goals and principles. As claimed in [1], unlike the Internet that is a collection of independent end-points that share a common routing infrastructure, sensor networks tend to be formed with homogeneous nodes deployed for an application-specific and collaborative purpose.



(a) Gradient-descending (b) Gradient-ascending

Figure 1. Routing and interaction patterns in a data-centric network. Typical tree-based or diffusion-based routing algorithms fall into gradient-descending protocols.

Further investigations into many real-world applications developed on TinyOS [2] based sensors corroborate this observation and show that general arbitrary point-to-point routing is less common in the applications of embedded networks [1]. Instead, interactions tend to be localized with a small number of data sinks (or leaders). This might reflect the data-centric paradigm of sensor network applications, where sinks create and maintain a data-and-events-forwarding network for collecting information from other members/nodes.

However, communications from members to sinks alone are not enough; the opposite direction (i.e., from sinks to members) communications must also be supported. Using so-called *gradient-ascending*[1] interactions, sinks can perform several actions on members, including (i) query allo-

*The work reported in this paper was supported in part by NSF under Grant CNS-0435023 and by ARO under Grant W911NF-05-1-0421.

[1]For ease of exposition, here we coin the two terms that best describe both interactions: "gradient-descending" (from members to sinks) and "gradient-ascending" (from sinks to members). In [3–5], the term "gradient" is often used to mean either the difference of hop-counts to the root of a tree (or a sink) between neighbors, or the hop count to the sink itself. We follow the latter convention. Note that the "gradient" in [6] is more than hop-count. Figure 1 simplifies routing patterns in a data-centric sensor network based on this classification.

cation, change, and migration; (ii) reconfiguration, diagnosis, and summarization; and (iii) other decision-involving actions in a signal-and-response fashion.

In this paper, we address the problem often encountered by sinks: *how to design an energy-efficient, stateless, gradient-ascending routing protocol that delivers packets from a sink to a specific node in a network?*

Either using an expensive *flood-based dissemination* protocol or running a separate *stateful on-demand routing* protocol such as AODV [7] or DSR [8] could be possible solutions.[2] The main drawbacks of either solution to the problem are as follows. First, flood-based dissemination is very costly (hence not energy-efficient) for individual interactions. Usually, it is used for holistic operations on a network, such as network programming via air medium [9,10].

Second, running a separate stateful routing protocol alongside a stateful gradient-descending routing protocol is a daunting task. Note that typical many-to-one gradient-descending routing protocols are a class of *reactive* stateful routing protocols, where routes from many nodes to sinks are created and maintained as needed. Therefore, running another stateful routing protocol introduces several difficulties not only in logically reconciling any protocol conflicts in response to a common network problem such as a link breakdown, but also in physically scaling them down to resource-constrained sensors.

The main contribution of this paper to the above-mentioned problem is the design of a novel gradient-ascending stateless routing protocol, called GRASP. It delivers packets in the gradient-ascending direction by exploiting the (reverse) forwarding histories accumulated in the gradient-descending direction during the data-and-events-forwarding phase to the sinks. GRASP has the following salient properties.

- It is stateless and complements gradient-descending protocols without creating any protocol conflicts. Together with a gradient-descending protocol, it completes a data-and-events collecting sensor network with a control loop.
- It is lightweight in the sense that the forwarding histories are accumulated in a space-efficient data structure called the *Bloom filter* [11], incurring only a small (on the order of few hundred bits) space overhead.
- It is energy-efficient, thanks to a novel packet forwarding method using membership-based broadcast, which incurs significantly less overhead than simple flood-based dissemination.
- It is robust to not only packet losses over a lossy wireless channel, but also gradient field perturbations resulting from changes in (reverse) forwarding links to the sinks.

A location-based stateless routing algorithm such as GPSR [12] could be used, but it requires a location-acquisition component that GRASP does not need. Therefore, GRASP becomes robust to the absence or failure of a location-acquisition component. All that GRASP requires is the accumulation of packet forwarding histories in the gradient-descending direction, which crop up naturally in any data-and-events collecting sensor networks.

In GRASP, forwarding a packet through networked sensors is just performing a membership test against networked Bloom filters that keep a dejavu image for other nodes whose packets were relayed. While going through a series of networked Bloom filters, packets may be dropped or eventually forwarded to their destinations. One or more trails can lead to a destination, being continually constructed and reinforced during the data-and-events forwarding phase in the gradient-descending direction.

The rest of this paper is organized as follows. Section 2 describes the work related to the design of GRASP and FBF. Section 3 describes FBF underlying GRASP, which is detailed in Section 4. GRASP is evaluated extensively and comparatively in Section 5. Finally, we conclude the paper in Section 6.

2. Related Work

Gradient-descending and on-demand routing protocols

The authors of [6] proposed a data-centric routing algorithm where a data sink advertises its interest with a list of attribute name and value pairs, and nodes matching the interest subscribe to it.

Schurgers *et.al* [4] proposed a gradient-based routing (GBR), as an extension to the directed diffusion [6], in such a way that the interest message records the number of hops taken during the propagation.

Poor *et.al* [3] proposed an idea similar to GBR. Its difference lies in that instead of identifying which node to forward a packet to, a node broadcasts a packet with the "cost-to-destination" information, which was obtained opportunistically when the destination sends out its message.

Similarly, Ye *et.al* [5] proposed a robust data-delivery protocol, called GRAB, that uses the notion of "credit" for robust packet delivery over a gradient-descending forwarding mesh.

The TinyOS version of DSDV [13] was written by Intel to provide many-to-one routing for one destination at a time, i.e., gradient-descending protocol.[3] Besides many-to-one gradient-descending packet routing, it also uses a flooding

[2] In this paper, a routing algorithm that creates and maintains a route between two end-points is said to be *stateful*. Otherwise, it is stateless.

[3] It is available at /opt/tinyos-1.x/contrib/hsn in a typical TinyOS package release.

mechanism to send packets to every node or a specific node in a network (i.e., gradient-ascending protocol).

For general on-demand routing protocols, DSR and AODV are only considered here since they are representative of, and basic to, all other extensions and variations. First of all, DSR, as source routing, is difficult to use in a sensor with a limited payload. Despite the evolutionary relaxation that could allow for a larger payload, creating and maintaining source routes in DSR is too expensive to be feasible.

Unlike DSR, AODV creates and maintains routes to a destination by performing a flood-based extended ring search as needed. Despite potential reductions in the flooding overhead with the query localization [14], the stateful routing property of AODV makes it hard for AODV to operate together with other stateful gradient-descending protocols.

Bloom filters A Bloom filter is one of the important data structures often used in various areas of computer science to support a membership query with a small amount of storage space. We only examine key extensions to the original one.

Li *et.al* [15] proposed *counting Bloom filters* that provide a mechanism to build a summary of the directory of cached documents at each Web proxy. A counting Bloom filter maintains a counter associated with each bit of the filter. Mitzenmacher [16] proposed *compressed Bloom filters* to reduce the number of broadcasts, the false-positive probability, and/or the amount of computation per lookup.

Rhea *et.al* [17] proposed *attenuated Bloom filters*, an array of Bloom filters. Associated with each d^{th}-row Bloom filter is the summarization of replicas on the neighbor at d hops away. They use the proposed Bloom filters to enhance the performance of existing peer-to-peer location mechanisms when a replica for the queried data item exists closer to the query source.

3. Footprint-Based Forwarding

3.1. Overview

Figure 2 illustrates various applications of FBF in a sensor network for collecting data and events, although use of FBF is not necessarily limited to them. FBF uses the prior forwarding history cached in a Bloom filter (or bitmap). This forwarding history is called a *footprint* of the node that sent packets. The forwarding histories in the gradient-descending direction are usually accumulated during the phase of forwarding data and events to sinks.[4]

The footprint of a packet is a bit vector of the same length as the bitmap, generated after applying independent hash functions to the packet's origin address. As a result of this hashing, some positions in the bit vector are set to 1 and others to 0. Several bit-wise operators are used to cache and access the generated footprint in the bitmap.

In the example of Figure 2(a), a packet originating from node M leaves its footprints on nodes along the trail M→W→A→C→S. Suppose, upon reception of a signal from M, sink S responds to M by returning some information to it. Then, S just broadcasts a packet with the feedback information and its destination[5] as node M.

When the packet is received by S's neighbors, each neighbor performs a membership test of the packet's destination against the bitmap it has. For example, node C will first hash the packet's destination M and get "01010." The hashed bit string is then compared with the bitmap it has, a boxed "01011." If the packet's destination turns out to be in the bitmap, it is rebroadcast, esle it is dropped. This is how the *membership-based broadcast* works.

There may be multiple paths to a given destination, as shown in Figure 2(b). Note that the false positives associated with a Bloom filter may create a non-path appearing to be a real one. Unfortunately, FBF has no way of telling the difference between the two. A packet under FBF is just broadcast as long as it passes the membership test.

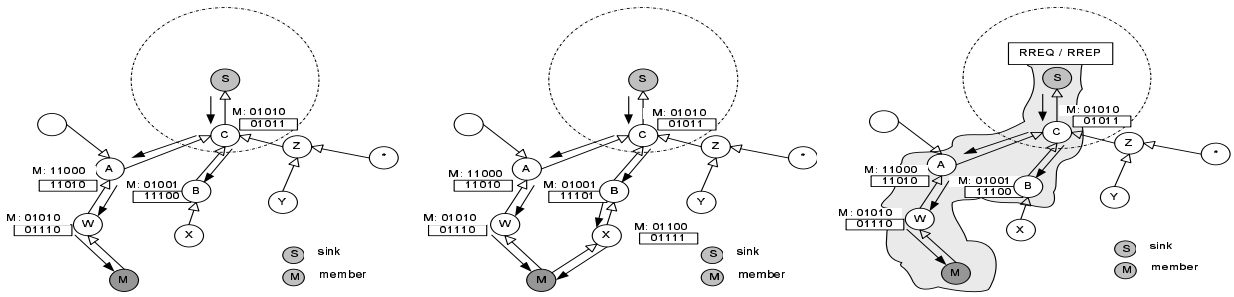
Figure 2(c) illustrates how FBF can be coupled with on-demand routing protocols, such as AODV, to localize a route-discovery query. AODV over FBF could be another solution to the problem, i.e., an energy-efficient stateful gradient-ascending routing protocol. We will not explore this case any further, but it is not difficult to see how FBF will improve the route-discovery overhead.

3.2. Data structure and hash functions

A Bloom filter is used to manipulate the footprint of a node. It is a way of representing a set $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ of n elements (nodes' addresses) to support membership queries. It uses a bit vector $\mathcal{B}$ of length m with k independent hash functions $\mathcal{H} = \{h_1, h_2, \ldots, h_k\}$. For each element $a \in \mathcal{A}$, each hash function $h \in \mathcal{H}$ maps a to an integer in $\{0, 1, \ldots, m-1\}$. Thus, for each element $a \in \mathcal{A}$, the bit positions $\{h_1(a), h_2(a), \ldots, h_k(a)\}$ in $\mathcal{B}$ are set to 1. Conversely, given an element $a \in \mathcal{A}$, a membership test of a against $\mathcal{B}$ is a simple procedure that checks the bit positions $\{h_1(a), h_2(a), \ldots, h_k(a)\}$ in $\mathcal{B}$; if all the relevant bits are set to 1, a is a member; otherwise, it is not.

[4]The forwarding histories for sinks can also be cached during the gradient setup or data dissemination phase from sinks. But such an option is not explored here since the underlying gradient-descending routing protocol is expected to always maintain the (reverse) forwarding links towards the sinks.

[5]Here, we assume that a bit in a packet's header is used to distinguish a broadcast packet from a unicast one. If a broadcast packet requires the destination address like 0xFFFFFFFF, the actual destination address will be carried in the packet's payload.



(a) Stateless routing: single path (b) Stateless routing: multiple paths (c) Route query localization

Figure 2. FBF can be directly coupled with stateless routing or used as a low-level primitive to localize the route-discovery query. The hashed footprint for a packet en route is shown without a box, whereas the Bloom filter at each node is shown with a box. A membership test of "M" against networked Bloom filters is performed when sink "S" sends data to it. Two independent hash functions and a Bloom filter of length 5 bits are associated with each node. Each node selects its hash functions independently of others.

We use a *counting Bloom filter* [15, 16], an extension to the basic Bloom filter such that a counter per bit is defined and used to keep track of the number of times the bit is set to 1. For a c-bit counter, the counter ranges from 0 and $C_{max} = 2^c - 1$. A Bloom filter without any counter can be considered as a counter Bloom filter with a 1-bit counter.

In a counter Bloom filter, when a counter changes from 0 to 1, the corresponding bit is set to 1. Similarly, when a counter changes from 1 to 0, the corresponding bit is set to 0. For a non-zero counter, the corresponding bit is set to 1. At any time a counter c is non-negative and no larger than C_{max}, regardless of the increment or decrement in c.

A hash function $h \in \mathcal{H}$ is defined as a parameterized function as follows:

$$h(a) \triangleq h(a|p,q,r,m) = ((p \times a + q) \times r) \bmod m \quad (1)$$

where p, q, r are prime numbers and $r > m$. To make an independent hash function, prime numbers are to be selected independently of other hash functions and other nodes.

3.3. Footprint and operators

Formally, the footprint $\mathcal{F}(a)$ of a node whose address is a, is a vector of m bits with each of bit positions $\{h_1(a), h_2(a), \ldots, h_k(a)\}$ set to 1 and the other bit positions set to 0.

For simplicity, a counting Bloom filter is denoted by $\mathcal{B}_c$, which consists of a bitmap (or an original Bloom filter) $\mathcal{B}$ and its c-bit counter vector $\mathcal{C}$. By default, $\mathcal{B} = \mathcal{B}_1$. Using common bitwise-AND (&) and bitwise-OR (|) operators, we define the operators to manipulate the footprint of a node over a counting Bloom filter $\mathcal{B}_c$ as follows.

Stamping footprints $\mathbf{S}(\mathcal{B}_c, a)$:

$$\mathcal{C}[i] \leftarrow \mathcal{C}[i] + 1,\ i \in \{h_1(a), h_2(a), \ldots, h_k(a)\}$$
$$\mathcal{B}[j] \leftarrow 1,\ j \in \{\ i\ :\ \mathcal{C}[i] > 0\}. \quad (2)$$

Erasing footprints $\mathbf{E}(\mathcal{B}_c, a)$:

$$\mathcal{C}[i] \leftarrow \mathcal{C}[i] - 1,\ i \in \{h_1(a), h_2(a), \ldots, h_k(a)\}$$
$$\mathcal{B}[j] \leftarrow 0,\ j \in \{\ i\ :\ \mathcal{C}[i] = 0\}. \quad (3)$$

Initializing footprints $\mathbf{I}(\mathcal{B}_c)$:

$$\mathcal{C} \leftarrow 0, \mathcal{B} \leftarrow 0. \quad (4)$$

Losing footprints $\mathbf{L}(\mathcal{B}_c)$:

$$\mathcal{C}[i] \leftarrow \mathcal{C}[i] - 1, i \in \{\text{randomly chosen } k \text{ bit positions}\}$$
$$\mathcal{B}[j] \leftarrow 0,\ j \in \{\ i\ :\ \mathcal{C}[i] = 0\}. \quad (5)$$

Testing footprints $\mathbf{T}(\mathcal{B}, a)$:

$$\prod_{i=1}^{k} \mathcal{B}[h_i(a)] \quad (6)$$

where $\mathcal{B}[h_i(a)]$ is the bit at $h_i(a)$ position of $\mathcal{B}$.

3.4. Clocking footprints $\mathbf{C}(\mathcal{B}_c)$

By definition, counters associated with a footprint indicate the number of times the bits are set to 1. Such counters could be translated to time ticks, representing the elapsed time since the last time the bits were set. In general, the interpretation of counters is application-specific. In the next section, we will describe how GRASP makes use of these counters.

3.5. Footprint loss model

The entire footprints can be erased through the **E** operator, or a random footprint can be lost through the **L** operator. Of course, it is possible for footprints to remain forever

once stamped in a bitmap, i.e., no loss model. Usually, the choice of a proper footprint loss model and its periodic or occasional enforcement are up to applications that use FBF. In the next section, we will describe the footprint loss model defined by GRASP.

4. GRASP

4.1. Per-sink Bloom filter

Usually, a gradient-descending routing protocol operates over a directed acyclic graph (DAG) rooted at the sink because the gradient setup and maintenance procedures in a gradient-descending protocol ensure the availability of routes from nodes to individual sinks that nodes are communicating with. Thus, a node can discern which DAG to follow in forwarding packets to a given sink. Also, such a DAG has per-sink properties like the interest and the lifetime.

This defines a per-sink gradient-based network, which is independently created, destroyed, and maintained by an individual sink. Thus, it is natural and convenient to have a per-sink Bloom filter such that the lifetime of a Bloom filter in a node equals the lifetime of the per-sink gradient-based network. By using the per-sink Bloom filter, a node handles multiple Bloom filters independently of other filters and records the history of forwarding packets in the filter associated with a specific sink.

Since each of multiple Bloom filters at a node is handled independently of others, we describe the operation of GRASP for the case of a single sink.

4.2. Footprints accumulation and loss model

GRASP requires to (i) accumulate footprints during the packet forwarding phase to the sink, and (ii) initialize footprints during the gradient setup phase.[6] Accumulating footprints involves the invocation of $\mathbf{S}$ operator to a relevant Bloom filter, whereas initializing footprints corresponds to the invocation of $\mathbf{I}$ operator to a relevant Bloom filter.

Accumulation of footprints without erasing them over the lifetime of a Bloom filter may make the filter useless by having too many bits in the filter set to 1. However, this is unlikely to happen due to the *spatial locality* of a packet forwarding in a gradient-descending protocol. By "spatial locality," we mean that the nodes involved in the packet forwarding from a specific node to the sink are limited within their geographic region in a network. The spatial locality occurs naturally because (i) sensors are stationary, and (ii) a gradient-descending protocol tends to forward packets along minimum-hop paths, reducing the possibility of packets randomly wandering through the network towards a sink. Note that nodes close to the sink still have many bits set due to the traffic concentration, i.e., they form a bottleneck towards the sink.

However, the assumption of stationary sensors does not exclude any local perturbation in the gradient field. For example, a change of the minimum-hop path from a given node to the sink can still occur when the forwarding node to the sink is selected based on other factors, such as nodes' residual energy. Despite such a perturbation, nodes within some region, like an ellipse-shaped region, around both the node and the sink may be frequently used. In fact, GRAB [5] defines such an ellipse-shaped forwarding mesh for reliably delivering packets to the sink, which still shows the spatial locality despite the local perturbation of a gradient field.

It is also possible to define either a periodic or an occasional footprint loss model. For example, invocations of $\mathbf{L}$ operator followed by $\mathbf{S}$ operator may define an *aperiodic random footprint loss model*. This ensures those rarely-used footprints to disappear. Conversely, those frequently-used footprints will reinforce themselves. However, care should be taken to adopt such a loss model due to the pigeon hole principle; a bit in the filter represents the membership relationship with several other nodes, not just with a specific node. Thus, even turning a single bit off may break paths to other nodes. In Section 5, we will investigate the impact of this footprint loss model on the routing performance.

4.3. Forwarding gradient-ascending packets

Assuming the accumulation of footprints during the phase of forwarding data/events to the sink, GRASP forwards any gradient-ascending packet from the sink as in Algorithm 1, which involves interactions with FBF and MAC upon reception of a packet from one of its neighbors, `prev`.

Algorithm 1 GRASP

`procedure recv(Packet p)`

1. assert `p.src` == `sink` and find $\mathcal{B}_c$ for `sink`
2. `rebcast.suppress(p.src,p.seq,p.prev)`
3. if (`p` is not received before and $\mathbf{T}(\mathcal{B}_c, \texttt{p.dst}) == 1$),
   (a) `delay` = $W \times (\mathbf{C}(\mathcal{B}_c, \texttt{p.dst}) + 0.11 \times U(0,1))$
   (b) broadcast `p` after `delay`
   (c) `rebcast.enque(p)`
4. else drop `p`

[6] We distinguish the network-wide gradient setup broadcast such as `ADV` from the localized gradient maintenance broadcast such as 1-hop `HELLO`, both of which may change forwarding nodes to the sink. The gradient setup is a less frequent operation that is usually invoked when a sink switches to another, while the gradient maintenance is usually and often exercised to balance loads or energy consumptions among nodes for a given sink.

Line 1 checks if a packet originates from the right sink for which a Bloom filter $\mathcal{B}_c$ is created. This step is necessary since GRASP assumes a per-sink Bloom filter. If a Bloom filter matching `sink` is found, the packet forwarding procedure is simple, as shown in line 3.

In line 3, a duplicate broadcast identification method is used to eliminate any redundant broadcast, which is a trivial operation in any broadcast-based-dissemination. After passing the membership test, the packet is broadcast again with some delay to avoid MAC-level broadcast collisions. The delay is set to no larger than the maximum forwarding delay, which is $1.11W$, where W is a preset forwarding delay, as seen in line 3-a. The second term in the delay is a tie-breaking random delay when the first terms are same.

The first term in line 3-a is introduced to favor forwarding packets along a frequently-used trail towards the packet's destination. In doing so, we time how long ago a footprint was created or assess how often it was stamped. It is denoted as $\mathbf{C}(\mathcal{B}_c)$ and has the following definition, which returns the age normalized with C_{max}:

$$(1 - \overline{C}/C_{max}), \text{ where } \overline{C} = \frac{\sum_{i=1}^{k} \mathcal{C}[h_i(a)]}{k} \qquad (7)$$

where $\mathcal{C}[h_i(a)]$ is the current counter associated with $h_i(a)$ bit position of $\mathcal{B}_c$. Then, a node with frequently-used footprints will have this value closer to 0. Note that once a counter reaches C_{max}, it remains there no matter how long ago this limit was reached. Also, $\mathbf{C}(\mathcal{B}_c)$ is not a perfect measure to indicate the freshness of a footprint since a counter is not a physical timestamp. Other refinements on $\mathbf{C}(\mathcal{B}_c)$ can also be explored.

4.4. Reliable membership-based broadcast

Any routing protocol must provide high end-to-end packet delivery performance, which depends largely on retransmissions of in-transit packets. Unlike unicast-based delivery that sends a packet several times to a single next hop, any simple broadcast-based packet delivery transmits the packet once and relies on multiple next-hop nodes for its successful advance towards the destination.

Unlike simple broadcast-based delivery, GRASP that uses the membership-based broadcast may have only a small fraction of multiple next-hop nodes eligible for relaying packets due to the membership test. To get around this, GRASP employs the concept of retransmissions as often used in unicast. In GRASP, a node rebroadcasts a packet several times until either it overhears any of next-hop nodes rebroadcast the packet (i.e., *passive acknowledgement (ACK)*) or it reaches the maximum retry limit.

However, there is a subtle difference between GRASP and a unicast with passive ACKs. In a typical unicast, a node knows which node's transmission to overhear since it knows where a packet is being forwarded to. But, in GRASP no forwarding node is specified. Any neighboring node that has a matched footprint of the received packet will rebroadcast it. Thus, we define a rule for passive ACKs in GRASP. Specifically, a packet `p` sent by a node `f` is said to be passively acknowledged by a packet `q` transmitted by a node `n` if (i) `p.seq == q.seq` and `p.src == q.src`, and (ii) `f.height <= n.height`. Both `seq` and `src` in the clause (i) uniquely identify a packet, as used in a typical duplicate packet identification method. The `height` in the clause (ii) represents a hop-count relative to the current sink of the received packet, which is usually known in the gradient setup phase. Note that a neighboring node's height relative to a given sink is also known automatically as a result of exchanging gradient maintenance messages such as 1-hop `HELLO`.

Improving the packet-delivery performance through retransmissions requires several modifications, some of which are already highlighted in Algorithm 1. First, each node needs to check if there is any outstanding packet (passively un-acknowledged). Line 2 in Algorithm 1 shows this; the `suppress` looks for any packet that can be passively acknowledged by the received packet and suppress any further transmission of packets that are passively acknowledged. The field `p.prev` in line 2 represents the node that has just broadcast the received packet.

Second, a node needs to save the received packet for later retransmissions even after broadcasting it. Line 3-c in Algorithm 1 shows this; the received packet `p` is enqueued into the buffer `rebcast`. Third, a periodic timer needs to be created to check if there is any passively-unacknowledged packet in the buffer and if so, triggers retransmissions within the maximum retry limit. Finally, a destination is supposed to perform 1-hop broadcast once whenever it first receives the packet, as a passive ACK.

5. Evaluation

5.1. Methodology

Basic setting. The *ns-2* [18] is used to evaluate the performance of GRASP for diverse simulation scenarios. Nodes remain stationary after they are randomly deployed in the field of size 1500m $\times$ 300m. The number of nodes deployed in the field is then changed from 60, 80, and up to 100. Each simulation runs 1000 sec. The results over 10 different topologies are averaged and plotted with the confidence level of 0.95. The *ns-2* default AT&T WaveLAN and IEEE 802.11 CSMA MAC are used with the simulation of a lossy wireless channel by letting each node randomly drop packets at the physical layer with probability 0, 0.1, 0.2, and 0.3. The transmission range is homogeneously set to 150m.

Gradient setup, maintenance, and perturbation. A sink enters the field and advertises a gradient setup message, ADV, and each node exchanges a periodic 1-hop HELLO message at an average interval of 10 sec. These messages have a sequence number assigned and incremented by the sink. Based on both the sequence number and the hop count to the sink, each node selects a successor to the sink with the greater sequence number or the smaller hop count if the sequence numbers equal. Both messages are broadcast with a delay proportional to the amount of node's residual energy. The operations described so far enable a gradient field to continually change even in a stationary network — a reasonable and realistic perturbation.

Data report model. Besides gradient setup and maintenance messages, data/events reporting is simulated. Specifically, CBR (Constant Bit Rate) traffic, each packet with a 64-byte payload, is generated at the interval of 4 sec. However, actual transmission is probabilistic to simulate balanced reporting among nodes to some degree.

By adopting a randomized energy-load balanced clusterhead rotation algorithm in [19], each node has the base probability P_b defined as $1/(\ell - i \bmod \ell)$, where $\ell = 1/P_0$ and i is the number of sampling intervals since the last transmission of data. P_0 is set homogeneously to 0.2 in the simulation, but can be changed. Based on this base probability, a hop-count-based probability is developed to prevent nodes close to the sink from using up a wireless channel for their own traffic. The hop-count-based probability model on which actual transmission relies is given as $\min\{\alpha \times H \times P_b, 1\}$, where H is a hop-count from itself to the sink and $\alpha = 0.1$. Note that this probability is updated at every interval as P_b is changed at every interval. Other values for P_0 or α will have less impact on GRASP/FBF since they are more directly related to the traffic intensity than to the operation of GRASP/FBF.

Feedback model. Given what has been described so far, three different feedback models from the sink to members are defined. Each interaction at every 4 sec interval generates 64-byte CBR traffic from the sink to one of members in the network. In fact, it is the actual workload on GRASP. To determine which member to contact, we use three interaction models:

- LRR (Least-Recently Reporting), in which the least-recently reporting node is chosen for the next interaction. For instance, probing sensors or regions that have not been queried for quite a while falls into this model.
- MRR (Most-Recently Reporting), in which the most-recently reporting node is chosen for the next interaction. MRR reflects a signal-and-response interaction.
- RND (RANdom selection), in which a node is randomly chosen for the next interaction. RND reflects random probing of sensors in the network.

Performance measures. The following performance metrics will be evaluated for GRASP.

- Delivery ratio: the ratio of the total number of packets received by members to the total number of packets sent by the sink.
- (Normalized) delivery overhead: the ratio of the total number of packets generated in the network to the total number of packets received by members, normalized with the measured average number of hops between the sink and members under each of the interaction models. Namely, the delivery overhead is the overhead per received packet and per hop.

Two distinct overheads contribute to the packet-generation overhead in the network: *data overhead* and *protocol overhead*. The data overhead is mainly associated with retransmissions, whereas the protocol overhead is associated with a routing protocol itself. For instance, GRASP has only the data overhead due to its rebroadcast scheme. But, AODV has an additional protocol overhead since it is a stateful routing protocol requiring route setup and maintenance.

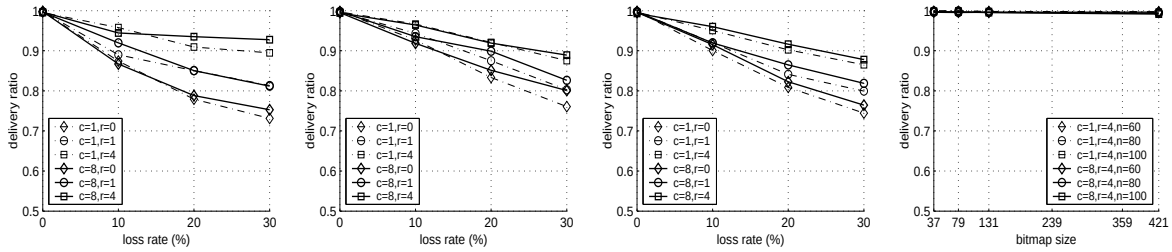
GRASP parameter. Six different prime numbers are used as a bitmap size (m); 37, 79, 131, 239, 359, and 421. We intentionally choose to use them to perform a sensitivity analysis of GRASP with respect to the bitmap size. Sensitivity to other parameters is also analyzed. Four different counters (c) are used: 1, 2, 4, and 8 bit counters. The number of hash functions (k) is changed from 1 to 3. The number of retransmissions (r) is changed from 0 to 4.

5.2. Main evaluation results

We will answer the following questions: (i) what is the delivery ratio GRASP can achieve and the delivery overhead it incurs? (ii) how robust is GRASP to packet losses in a wireless channel? (iii) how sensitive is GRASP to main parameters such as bitmap size, counter size, and retry limit, and interaction model? (iv) how sensitive is GRASP to other parameters such as the corruption of a bitmap, the footprint loss model, the number of hash functions, and the node density/traffic intensity?

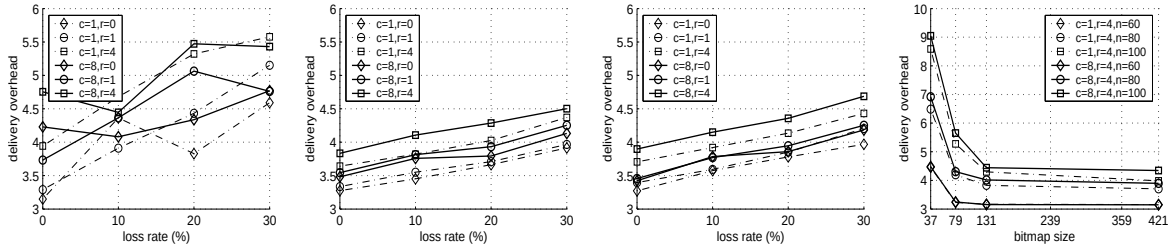
Delivery ratio and overhead of GRASP: GRASP turns out to achieve an about 99.3% or higher packet delivery ratio when the channel is perfect (no random packet loss at the physical layer), regardless of the interaction models — see Figure 3(a) $\sim$ (c) at the loss rate of 0%. Under the same condition, the delivery overhead incurred, depending on the channel loss rate, is 3$\sim$5 transmissions per received packet and per hop, although the variation of the delivery overhead in LRR is high.

At least three causes of delivery overhead were observed even if there is no rebroadcast at all, i.e., GRASP



(a) LRR (b) MRR (c) RND (d) RND

Figure 3. Delivery ratio with ($n = 80$, $m = 421$, $k = 2$) for (a)$\sim$(c) and (n, m) varied for (d).



(a) LRR (b) MRR (c) RND (d) RND

Figure 4. Delivery overhead with ($n = 80, m = 421, k = 2$) for (a)$\sim$(c) and (n, m) varied for (d). The estimated delivery overhead in case of simple flooding is about $\{$10.95 ($n = 60$), 14.60 ($n = 80$), 18.26 ($n = 100$)$\}$.

$c = \{1, 8\}$, $r = 0$ in Figure 4(a) $\sim$ (c) at the loss rate of 0%. One probable cause for this is the existence of multiple redundant paths due to the underlying gradient perturbation described in the simulation setting. Another probable cause could be a false positive that makes a false path appear to be real. An interesting aspect of the delivery overhead in Figure 4(a) $\sim$ (c) is that an increase in the retry limit (r) does not incur a proportional increase in the delivery overhead. For instance, the delivery overhead under GRASP for $r = 4$ incurs an additional overhead of less than that for $r = 0$ in the same situation. This result shows the efficacy of the suppression mechanism in GRASP without deteriorating the delivery performance.

Finally, we estimate the delivery overhead of simple flooding by assuming that the delivery ratio in such a case is always 100%, regardless of the channel loss rate. As implied from its name, the simple flooding makes every node in the network rebroadcast any non-duplicate packet it received. So, the pre-normalized delivery overhead is just the number of nodes in the network. By using the *measured* average number of hops under each interaction model, the estimated delivery overhead of simple flooding turns out to be about $\{$10.95 ($n = 60$), 14.60 ($n = 80$), 18.26 ($n = 100$)$\}$.

As compared to this simple flooding, GRASP achieves almost the same delivery ratio with a dramatic decrease in the delivery overhead by at least a factor of 3 under $n = 80$ and the RND feedback model: 3.75 from 14.60/3.89 (loss rate 0%) and 3.11 from 14.60/4.68 (loss rate 30%) in Figure 4(c).

Resilience of GRASP to packet losses: GRASP is also resilient to packet losses, achieving more than an 86% packet delivery ratio even in the face of channel loss rate up to 30%, especially under GRASP $c = \{1, 8\}, r = 4$. On the other hand, the delivery overhead tends to increase proportionally to the channel loss rate, but its increase is only marginal.

Obviously, the retry limit plays a major role in both improving the delivery ratio and incurring the delivery overhead under such lossy wireless channel conditions — see Figure 3(a) $\sim$ (c) and Figure 4(a) $\sim$ (c).

Sensitivity of GRASP to main parameters: The bitmap size (i.e., a Bloom filter size) turns out to affect the delivery ratio little and the delivery overhead much – see Figure 3(d) and Figure 4(d). The counter size seems less critical, although it is directly related to the MAC-level broadcast collision — see Figure 3(a) $\sim$ (c). The retry limit, as already discussed, plays a major role in improving the delivery ratio and delivery overhead — see Figure 3(a)$\sim$(c) and Figure 4(a)$\sim$(c). GRASP under LRR shows more variance in the packet delivery ratios and higher delivery overhead than under the other two models. Unlike LRR, the other two interaction models do not seem to affect the performance of GRASP significantly — see Figure 3(a)$\sim$(c) and Figure 4(a)$\sim$(c).

One thing to notice in Figure 4(d) is the delivery overhead when a small-sized bitmap is used. When it is sized too small, a bitmap produces a lot of false positives and thus increases the delivery overhead. However, when it is sized appropriately, the false positive probability can be negligible. For instance, in the figure, a bitmap of size 131 incurs almost the same overhead as a bitmap of size 421.

Sensitivity of GRASP to other parameters: The corruption of a bitmap turns out to increase the delivery overhead significantly as the corruption ratio increases — see Figure 5(b). When the footprint loss model is used, such a high overhead can be avoided at the cost of reduced delivery ratio — see Figure 5(a) and (b). Changes in the number of hash functions alone seem to have less impact on the performances we investigated. However, its use in conjunction with a footprint loss model results in a poor delivery ratio due mainly to the increased false negative probability — see Figure 5(c) and (d). Any increase in the traffic intensity[7] or node density worsens the delivery overhead slightly — see Figure 3(d) and Figure 4(d).

Contrary to a normal bitmap whose bits are cleared with the initialization operator **I** upon reception of each gradient setup message, a *corrupted* bitmap refers to a bitmap that is not clearly initialized. The corruption of a bitmap is realized in the simulation by preloading a bitmap with footprints of randomly-chosen nodes. For example, in Figure 5(a) or (b), a bitmap with an initial load 30% under $n = 80$ means that a bitmap is preloaded with footprints of randomly-chosen 24 nodes. We assume that such corruption can happen for reasons like failure of the initialization operator, the mistakenly or intentionally sharing of a bitmap across multiple sinks, and so on.

5.3. Comparison with AODV

We will answer the following question when AODV is used as a gradient-ascending stateful protocol: *is GRASP as good as, or better than AODV, under what conditions?*

Figure 6 illustrates the main results: (i) GRASP achieves almost the same delivery performance as AODV, while reducing the overhead by a factor of about 2.37 = 7.50(AODV, basic)/3.14(GRASP $c = 8, r = 4$) in Figure 6(b) under $n = 60$; (ii) even in the presence of link failures, GRASP shows performance comparable to or better than AODV, while lowering the overhead by a factor of 2.51 = 9.82(AODV, basic)/3.89(GRASP $c = 8, r = 4$) in Figure 6(d).

For a fair comparison, we set the maximum retry limit equally for both GRASP and AODV.[8] AODV runs over the IEEE 802.11 without any channel reservation (AODV basic) and with RTS/CTS reservation (AODV rts/cts).

As seen in the figure, the main drawback of using AODV as a gradient-ascending routing protocol is energy-inefficiency, i.e., a poor delivery ratio in spite of large overhead as the loss rate gets worse, due primarily to its default extended ring search overhead. Especially, the prohibitive protocol overhead involved in creating and maintaining routes on-demand makes AODV less desirable for a gradient-ascending protocol.

6. Concluding Remarks

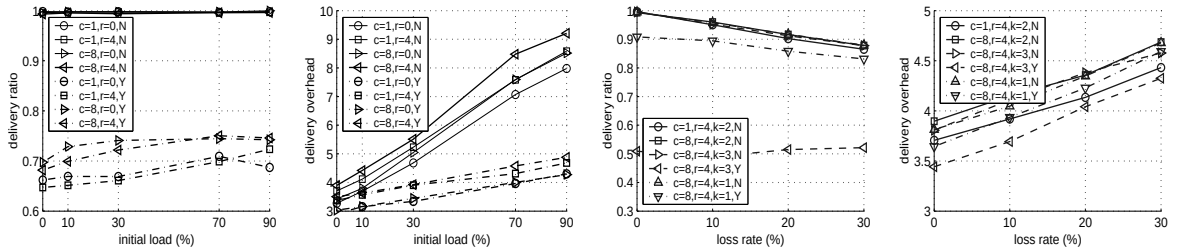
In this paper, we addressed the problem often encountered in sensor network monitoring and control applications. With a control loop in the network, data sinks or coordinating nodes take several types of actions on other nodes. However, existing gradient-descending routing algorithms that set up and maintain only a data-and-events forwarding network from nodes to sinks do not consider the interaction in the opposite direction.

GRASP overcomes this difficulty by exploiting the forwarding histories accumulated naturally during the data-and-events forwarding phase to sinks. GRASP completes the control loop via a lightweight stateless gradient-ascending protocol with a slight storage overhead to manipulate the forwarding histories through a Bloom filter. From an architectural point of view, GRASP with FBF is shown to be an attractive addition to any gradient-descending protocol thanks to its stateless routing property that does not incur any protocol conflict with the underlying gradient-descending protocols. Furthermore, our extensive simulation study has shown that GRASP incurs a very small overhead, and is robust to packet losses. Both its architectural merit and routing performance make GRASP very attractive for practical use in conjunction with any gradient-descending routing protocol.

GRASP needs to be studied further as follows. First, we must study how to support sinks that constantly move around without explicitly updating the entire gradient field. Second, we need to study how to efficiently share a bitmap across multiple sinks or manipulate it in mobile ad-hoc sensor networks by using different footprint loss models other than the aperiodic random footprint loss model evaluated in our simulation. Finally, we need to study the behavior of GRASP/FFP over real deployments that could possibly include unidirectional wireless links.

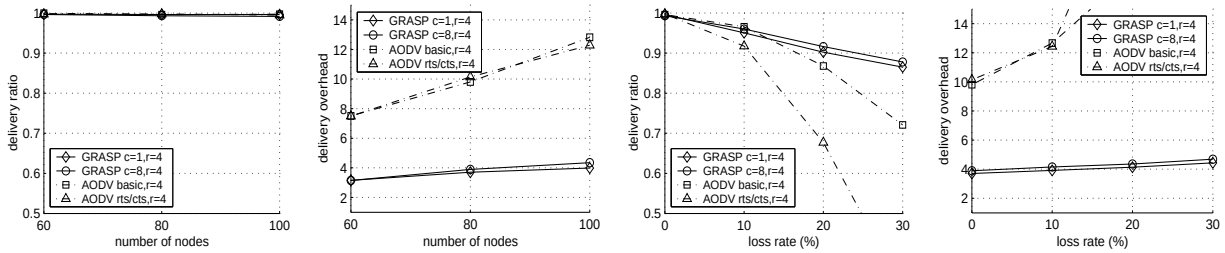
[7] The sensitivity to the traffic intensity is not explicitly evaluated here since it has less impact on the accumulation of footprints in a bitmap, but more impact on MAC-level collisions. However, MAC-level collisions are being evaluated via experimentation that uses a different number of nodes in a network.

[8] Current simulation runs AODV over the IEEE 802.11 MAC which defines two variables for adjusting the retry count, LongRetryLimit and ShortRetryLimit whose default values are 7 and 4, respectively. We set these values to 4.



(a) Delivery ratio (b) Delivery overhead (c) Delivery ratio (d) Delivery overhead

Figure 5. Other sensitivity evaluations ($n = 80$, $m = 421$, RND). $Y(N)$ represents that a footprint loss model is (not) used.



(a) Delivery ratio (0% loss) (b) Delivery overhead (0% loss) (c) Delivery ratio ($n = 80$) (d) Delivery overhead ($n = 80$)

Figure 6. Comparison against AODV (bitmap size = 421, RND).

References

[1] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," in *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[2] UCB, "Tinyos project," http://webs.cs.berkeley.edu/tos/.

[3] R. D. Poor, "Gradient routing in ad hoc networks," in *http://www.media.mit.edu/pia/Research/ESP/texts/poorieee-paper.pdf*.

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

[5] F. Ye, G. Zhong, S. Lu, and L. Zhang, "Gradient broadcast: A robust data delivery protocol for large scale sensor networks," in *ACM Wireless Networks*, 2005.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *ACM/IEEE International Conference on Mobile Computing and Networks (MobiCom '00)*, 2000.

[7] C. E. Perkins and E. M. Royer, "Ad hoc on-demand distance vector routing," in *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[8] D. B. Johnson and D. A. Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*. Kluwer Academic Publishers, 1996, vol. 353.

[9] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 2004, pp. 81–94.

[10] Philip Levis and Neil Patel and David Culler and Scott Shenker, "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," in *First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI)*, 2004.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[12] B. Karp and H. T. Kung, "Gpsr: greedy perimeter stateless routing for wireless networks," in *Proceedings of the 6th annual international conference on Mobile computing and networking*. ACM Press, 2000, pp. 243–254.

[13] M. D. Yarvis, W. S. Conner, L. Krishnamurthy, A. Mainwaring, J. Chhabra, and B. Elliott, "Real-world experiences with an interactive ad hoc sensor network," in *ICPPW '02: Proceedings of the 2002 International Conference on Parallel Processing Workshops*, 2002.

[14] R. Castaeda and S. R. Das, "Query localization techniques for on-demand routing protocols in ad hoc networks," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM Press, 1999, pp. 186–194.

[15] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, 2000.

[16] M. Mitzenmacher, "Compressed bloom filters," *IEEE/ACM Trans. Netw.*, vol. 10, no. 5, pp. 604–612, 2002.

[17] S. Rhea and J. Kubiatowicz, "Probabilistic location and routing," in *INFOCOM (3)*, 2002, pp. 1248–1257.

[18] "The ucb/lbnl/vint network simulator - ns (version 2)," http://www.isi.edu/nsnam/ns/.

[19] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proc. of IEEE Hawaii Int. Conf. on System Sciences*, January 2000, pp. 4–7.