Application of DHT-Inspired Routing for Object Tracking

Pengfei Di and M. Yaser Houri
System Architecture Group,
Universität Karlsruhe (TH), Germany
{di|houri}@ira.uka.de

Qing Wei and Jörg Widmer
DoCoMo Euro-Labs,
Munich, Germany
{wei|widmer}@docomolab-euro.com

Thomas Fuhrmann
Computer Science Department,
TU Munich, Germany
fuhrmann@net.in.tum.de

Abstract

A major problem in tracking objects in sensor networks is trading off update traffic and timeliness of the data that is available to a monitoring site. Typically, either all objects regularly update some central registry with their location information, or the monitoring instance floods the network with a request when it needs information for a particular object. More sophisticated approaches use a P2P-like distributed storage structure on top of geographic routing. The applicability of the latter is limited to certain topologies, and having separate storage and routing algorithms reduces efficiency.

In this paper, we present a different solution which is based on the scalable source routing (SSR) protocol. SSR is a network layer routing protocol that has been inspired by distributed hash tables (DHT). It provides key-based routing in large networks of resource-limited devices such as sensor networks. We argue that this approach is more suitable for object tracking in sensor networks because it evenly spreads the updates over the whole network without being limited to a particular network topology. We support our argument with extensive simulations.

1. Introduction

With the technological improvements of the recent years, large wireless sensor networks (WSN) are about to be widely used. Major applications will be remote monitoring, event detection, and object tracking. In the latter case, typically a centralized instance such as a monitoring console needs timely information about the location of one or multiple objects that move through the WSN.

For the purpose of this paper we distinguish between sensor nodes and tracked objects, assuming that the tracked objects are even more resource-limited than the sensor nodes. One may think of the tracked objects as being tagged with radio frequency identification (RFID) tags or small W-LAN tags. We call the sensor node that needs the information about the tracked object the *inquirer* or *sink*, and the sensor node which contains the requested location information about the tracked object the *source*.

1-4244-1455-5/07/$25.00

For example, a researcher may want to track the wild deers living in a large national park. She might want to do so in real-time and be able to locate any individual at any instant in time. But she will typically only request the location of a few deers at once. The deers might have preferred places and typical times to move between these places. They might move individually or in herds. Many other such applications are conceivable, including commercially relevant applications such as asset tracking, and applications to national security.

Furthermore, we assume the WSN to be potentially supported by some additional links based on technologies such as Wireless LAN, Ethernet, or WiMax. For example, RFID detection points could be placed along the trails in the park and connected via W-LAN and Ethernet if applicable. Wire-line shortcuts can circumvent the general capacity problems in WSNs and enable the researcher to monitor a large area. In commercial applications this scenario maps to urban mesh networks and large company networks.

Unlike other WSN applications where static interest patterns can be established in the network, object tracking has to cope with the uncertainties of mobility. Depending on the object mobility scenario and the request patterns this can lead to severe scalability issues:

Let us for example consider the straightforward pull approach where the sink floods the network with a request message whenever it needs the location data of a particular object. A sensor node that receives such a request will answer with the location data for the requested object. Clearly, such an approach does not scale to frequent location requests in large networks.

An alternative straightforward approach is pushing the location information to a central location registry. The inquirer can then retrieve the data it needs directly from the registry without flooding the network. Clearly, the push approach does not scale either. In particular, it is not suited for large networks with many highly mobile objects: Whenever an object moves from the vicinity of one sensor node to that of another sensor node these nodes would have to send an update. Even worse, many of the updates would be overwritten in

the registry without having been read by the monitoring application. But in our assumed scenario the publishing nodes cannot know which data might be requested. So they have to publish permanently all the data. Increasing the intervals with which the sensor nodes update the registry is no help because this would reduce the timeliness of the data that happens to be requested.

Several variations of the basic push and pull approach have been studied (cf. sec. 2). Most notably, geographic hash tables (GHT) avoid the problems of traffic concentration at some centralized node. There, a sensor node that detects an object entering or leaving its vicinity sends an update to another sensor node whose address is determined by hashing the detected object's unique ID to the geographic area covered by the WSN. Thereby all the update events in the network are uniformly distributed over the network. Geographic routing such as greedy perimeter stateless routing (GPSR) then efficiently sends the update information to the node that is closest to the position obtained from hashing.

However, GHTs are not suitable for our scenario: A GHT with GPSR typically assumes a WSN where all nodes are evenly spread over a 2-dimensional area of fixed size. They cannot easily cope with irregular network areas, largely varying node densities, or an extension of the covered area (at a later stage during the operation of the network). For example, a GHT using a hash function that uniformly distributes data over the network area may overload nodes in very sparsely populated areas. Due to the particular properties of GPSR they are ignorant of potential shortcuts in the network topology, too.

Therefore, we propose to use a DHT-inspired routing protocol such as SSR which provides key based routing. Like with GHT, a sensor node that detects an object entering or leaving its vicinity sends an update to another sensor node. Unlike a GHT this sensor node is determined by hashing the object's unique ID to the address space of the sensor network. Thereby all the update events in the network are uniformly distributed in the whole network regardless of the area's shape, actual size, and potential variations in the node density. Furthermore, this approach can benefit from shortcuts through the network, for example, wired W-LAN access points.

This paper is structured as follows: In section 2, we give a brief overview of relevant data dissemination and object tracking protocols. section 3 details our proposal. section 4 demonstrates the performance improvement that is gained by adoption of the SSR protocol. Finally, section 5 concludes this paper with an outlook to future work.

2. Related Work

The main purpose of sensor networks is collection or dissemination of sensed information. To this end, a large number of sensor network protocols have been proposed in recent years. Most of these protocols use flooding techniques to disseminate the data itself, or to build up the required control information for data dissemination (similar to route discovery). One of the first data dissemination protocols is SPIN [1]. It focuses on efficient dissemination that delivers an individual sensor's observation to all the sensors in the network. SPIN uses meta-data to avoid the transmission of redundant data. Directed Diffusion [2], TAG [3], and COUGAR [4] use a data-centric naming approach to enable intelligent in-network data aggregation in order to enhance energy efficiency and scalability. GRAB [5] uses mesh forwarding to ensure a high data delivery rate.

All the above mentioned protocols target efficient data dissemination to stationary sinks. Protocols such as TTDD [6], EDDA [7] and SAFE [8] address the case of mobile sinks. TTDD constructs grid networks for each data source. Sources disseminate data through grid nodes to the mobile sinks. Disseminating the data along the grid structure can increase path length by a factor of $\sqrt{2}$. Moreover, maintaining a grid structure per source causes considerable overhead. EDDA is an enhanced TTDD protocol. In EDDA, sources that have the same data type share a single grid structure to disseminate data. Even though EDDA uses unicast instead of flooding, which is used by TTDD, the construction and maintenance of a grid structure in large scale wireless sensor networks still waste a considerable amount of network resources. SAFE uses geographically limited flooding to forward queries to the source. Using flooding in large scale sensor networks causes high traffic load and in turn leads to high power consumption.

The work mentioned above focuses on data dissemination in a certain destination region, but is not necessarily well suited for object tracking within the whole sensor domain.

While an optimized flooding algorithm could be used in the sensor network, e.g., using the MPRs in OLSR [9] to flood, the improvement compared to normal flooding is limited. Especially for a large sensor network, any form of flooding is a huge waste of resource.

To avoid flooding of queries in sensor networks, Data-Centric Storage (DCS) was introduced in [10]. In this approach, observations (sensed data) are stored in selected nodes of the WSN, called data-centric nodes. These nodes are responsible for storing and retrieving sensed data. All sensor nodes as well as all sink nodes are aware of the information of data-centric nodes. Therefore, flooding is not necessary for sending or

querying data.

One of the first approaches that uses a Data-Centric Storage mechanism is the Geographic Hash Table (GHT) [11]. In a GHT, observations are mapped to the locations of data centric nodes in the monitoring area. Sensors will use the GHT to get the location of the nearest data centric node, where the sensed information will be sent to. A sink node interested in some specific data uses the GHT to get the location of the corresponding data centric node that has the required information, and then the sink node will send a query directly to that node to retrieve the data.

GHTs combine *geographic routing* with ideas from distributed hash tables (DHT) in a manner similar to content addressable networks (CAN) [12]. One of the key advantages of geographic routing is that the nodes do not have to maintain routes but only need to know their physical locations and the physical locations of their neighbors. Packets are then greedily forwarded in the direction of the destination's physical location. An example of geographic routing protocols is *Greedy Perimeter Stateless Routing* (GPSR) [13].

For practical purposes, geographic routing requires a location service for mapping the location-independent address of a node to its current geographic coordinate (e.g., *Grid Location Service* [14]). Furthermore, greedy geographic routing may require planarization of the underlying network graph for efficient recovery from dead-ends. Location service and planarization not only create overhead, but are also unsuitable for our scenario where potentially additional wired links provide high-speed shortcuts.

For these reasons, as well as the problem of GHTs to cope with uneven node distribution, we propose to modify the GHT approach by using DHT-inspired routing at the network layer. This allows to tightly integrate the data storage and retrieval with the underlying routing protocol.

First ideas about using DHTs (or more correctly structured peer-to-peer protocols) for routing in MANETs have been proposed independently by several groups. DPSR [15] is a cross-layer approach for using the structured peer-to-peer overlay Pastry [16] as a network-layer routing protocol. It integrates Pastry with dynamic source routing (DSR). DPSR addresses nodes by fixed identifiers. The Ekta protocol [17] enhances DPSR with an indirect routing primitive. Both protocols use flooding to discover routes.

Scalable source routing (SSR) [18] [19] combines source routing with the Chord overlay [20]. It equips nodes with a small, limited size cache that holds source routes which can then be used in a Chord-like manner. SSR uses an iterative mechanism to bootstrap the caches and thereby avoids flooding. Recently, a similar approach has been proposed: Virtual ring routing (VRR) [21]. Contrary to SSR it does not use source routes but builds state along the paths. Thereby VRR trades off header size and state in the nodes along the paths.

All the latter approaches provide indirect routing. As GHTs have demonstrated, this can serve as a basis for efficient object tracking systems. We describe indirect routing and in particular SSR in more detail in the next section.

3. Scalable Source Routing

3.1. Indirect Routing

Indirect routing decouples packet addresses from the network nodes. Nodes send packets to abstract destinations that the routing protocol maps to a concrete node. This level of indirection enables data-centric communication where packet addresses identify data objects instead of nodes.

Structured P2P networks like Chord enable indirect routing through an overlay at the application layer. The protocols assume a network-layer routing protocol to provide connectivity among the overlay nodes.

Chord creates a virtual address space that is cooperatively managed by all nodes participating in the overlay network. A node A manages all data objects whose addresses fall between A's address and the address of the node B whose address is the smallest of all nodes with addresses larger than A's. Node B is called A's *successor* and consequently, node A is node B's *predecessor*.

For correct routing in the overlay it is both a sufficient and a necessary condition that each node knows its correct successor. Thus, the structure of the overlay is a ring that is formed by the nodes and their pointers to the respective successors. Nodes route packets by forwarding them in increasing direction of the address space until the distance between the address of the current node and the address of the data object cannot be minimized any further. If the network is inconsistent—i.e., if some nodes do not know their correct successors—requests for data objects may get delivered to nodes that have no knowledge of the requested objects.

Besides the successor, Chord nodes store the addresses of $O(\log n)$ additional nodes at exponentially spaced distances to reduce the average request path length from $O(n)$ to $O(\log n)$, where n is the size of the network. By choosing physically close nodes for the additional state, the total *physical* lookup path length is a constant factor longer than the shortest possible length [22].

3.2. DHT-Inspired Network Routing

Scalable source routing (SSR) provides indirect routing at the network layer. It integrates source routing with Chord-like routing in an address space that has the structure of a virtual ring. Nodes have unique addresses such as MAC addresses. Preferably these addresses are uniformly distributed over the address space. In case of MAC addresses this can be achieved by hashing the MAC address into the SSR address space.

In contrast to well-known link-state and distance-vector routing protocols, SSR does not maintain state information for all destinations in the network. A node's forwarding information base (FIB) must only contain the node's physical neighbors and source routes to the node's virtual neighbors in the address space. Physical neighbors are discovered by a link layer mechanism or hello beacons. Source routes to virtual neighbors are constructed by an iterative mechanism (see below). All remaining space in the FIB can be used for opportunistically cached information. We call the FIB *route cache* because it stores source routes using a least-recently-used (LRU) policy. One can think of the route cache to be organized as a tree of nodes so that the respective node (the root node) can retrieve a source route to any of the cached nodes. Typically a small route cache containing only 256 entries suffices even for very large networks of several 100 000s of nodes.

SSR uses two distance metrics for its forwarding decisions: The physical distance of two nodes A and B in hops, that is the length of a source route between A and B; and the virtual distance of two nodes, that is the absolute value of the numerical difference between the nodes' addresses. Other physical metrics such as latency, throughput, or monetary cost are also possible, but not discussed in this brief overview here.

SSR packets contain a source address, a destination address, and a source route. The included source route does not have to span the entire path from the source to the packet's destination. It could end at some intermediate destination. In that case, this intermediate node (i.e., the last node contained in the source route) tries to append a source route from its local cache. It will append the source route to the packet's final destination, if it has cached it. Otherwise it appends a source route to another intermediate node.

Assume an intermediate node I_k or the packet's source $S = I_0$ has to choose the next intermediate node I_{k+1}. It does this so that

- I_{k+1} is virtually closer to the packet's final destination D than I_k, and
- I_{k+1} is physically closest to I_k among all its cached nodes satisfying the first condition, and
- I_{k+1} is virtually closest to D if the selection has not yet yielded a unique I_{k+1}.

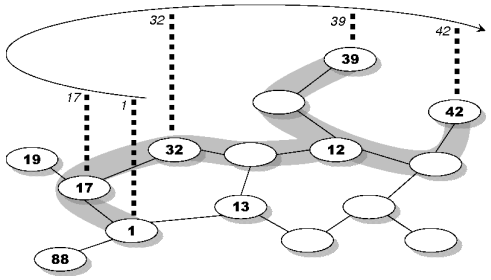


Fig. 1. Illustration of the routing process

This process of forwarding along the source route and appending a new source route is continued until the packet has reached its final destination. Note that by construction this process is guaranteed to succeed at the node that is virtually closest to the destination address because in a consistent network nodes will have a source route to their virtual neighbors.

Figure 1 illustrates this with an example. Assume node 1 wants to send a packet to node 42. The packet is first forwarded to node 17 because that node is physically closest to node 1. Node 17 is preferred over node 13 since 17 is virtually closer to 42 than 13 is. For the same reasons, node 17 forwards the packet to node 32. Node 32 forwards the packet to its successor, node 39, which in turn forwards the packet its successor that coincides with the destination, node 42.

The forwarding process corresponds to appending a source route. To this end, there is an appending flag in the SSR packet header. The receiver node 42 knows that node 1 doesn't have a direct route to it via this flag, and sends a route update message to node 1. This causes node 1 to send the packet directly to it the next time, resulting in a 6-hop route, just 1 hop longer than the optimal route.

Clearly, consistent operation requires all nodes to have source routes to their virtual neighbors. These can always be obtained by an iterative mechanism even when all nodes bootstrap simultaneously. We briefly summarize this mechanism [23] here. Unlike many other ad-hoc routing protocols such as ad-hoc on demand vector routing (AODV) or beacon vector routing (BVR) SSR's mechanism does not require any flooding. It is thus also suited for large networks.

Upon bootstrapping, a node N discovers its physical neighbors P_i. If $\{P\}$ is not empty, at least one of these neighbors will bear an address $P_i < N$ (assumed left virtual neighbor), or at least one will bear an address $P_i' > N$ (assumed right virtual neighbor). If $\nexists P_k$ such that $P_i < P_k < N$, P_i is an assumed direct left neighbor.

Assume without loss of generality that P_i is an assumed direct left neighbor of N. Then N sends P_i a message indicating N's presence and the assumed direct

neighborship. If P_i happens to know (a source route to) another node N' so that $P_i < N' < N$ it replies with a message indicating N''s presence to N. This message also contains a corresponding source route so that N can enter N' into its cache.

This procedure is continued until no new nodes are discovered. It can be shown that in static, reliable networks this algorithm is guaranteed to converge into the globally correct state. In dynamic networks or in presence of packet loss (partial) convergence is still (highly) likely when the neighborship indications are always piggy backed onto the payload traffic in the network.

Theoretical arguments indicate that this iterative process converges in $O(\log N)$ in almost every random graph [24]. This is supported by simulation results. Note further that once the network is operational, newly joining nodes can be integrated in $O(1)$ by sending an integration request to the node that has so far been responsible for the joining node's address.

Summarizing this brief introduction into SSR we can conclude that SSR is a suitable indirect routing mechanism to form the basis for our object tracking scenario: It provides key based routing directly in the network layer. Nodes do not need to hold large FIB's, small route caches suffice. SSR does not assume any topology in the network so that unlike geographic hash tables our SSR-based approach can operate in arbitrarily shaped areas with regionally varying node density. If available, it can also benefit from additional network links overlaid over the WSN.

4. Performance Evaluation

In this section, we compare the performance of a location tracking mechanism based on SSR with the performance of a flooding based query scheme and that of sending periodic updates to a central node.

4.1. Scenarios

For our scenario we investigate a sparse regular network. This is the usual scenario for such evaluations because it allows to cover a large sensor network area with few sensing devices and also reduces mutual interference between nodes. We chose the OMNET [25] simulation framework for the performance evaluation because of its good scalability, which allows simulation of large networks. We have further built wrappers for the 802.11 MAC and AODV-UU [26] modules from ns-2 and integrated them into OMNET. For this reason, our simulation results are comparable to simulations with ns-2, the most widely used network simulator.

We use a grid network topology, where each node has 4 physical neighbors in its receive range and 4 additional neighbors in its interference range (except for nodes at the border of the network). Specifically, the distance between two neighbor nodes is 50 meters. Transmit power is set such that the receive range (or transmit range) is approximately 55 meters, and the interference range (or carrier sense range) is 75 meters (i.e., slightly larger than $50\sqrt{2}$). Nodes communicate via an 802.11 MAC protocol.[1]

In all experiments, there is one data sink through which the requests are sent. In each simulation run it is randomly chosen among the nodes. Mobile objects move within the network area and the actual appearance of objects within the sensing range of a sensor node will be detected as an event by that node. Such events are reported either proactively by routing the information to a data collection node (*update case*), or the data remains at the node where the event occurred and tracking an object then requires flooding the network to find the most recent event pertaining to a specific object (*request case*). These simple mechanisms are compared against using SSR to distribute the objects' information to nodes that match the hash of the object's identifier (i.e., the object's virtual SSR address), which is a combination of the request and update case.[2]

The requirements for the timeliness and accuracy of the location information, as well as distribution and rate of requests for objects largely depend on the specific application (as for example in wild-life tracking vs. battlefield scenarios). We therefore investigate various different request rates and analyze the age of the obtained information. A single request packet can be used to query for the location of several objects. Similarly, some application may want to recover the history of the past locations of an object, in which case not the most recent event for that object, but all events for that object (for a given time period) should be sent back. Thus, in the request case, a single flooding can cause multiple replies to be sent back to the sink. The number of reply packets triggered by one request packet is called *reply rate*.

In the update case, updates are sent to the sink in individual packets. Routes are set up using AODV-UU, whose active route timeout is set large enough to avoid unnecessary route discovery. An update packet is generated and sent to the sink as soon as the sensor detects the occurrence of any object detection event. Since the sink receives all the information of objects detected by every

[1] We use the usual simple unit disc graph model, where a packet received from within the receive range does not have bit errors, while a packet from outside the receive range but inside interference range is always corrupted. Packets from nodes farther than the interference range are not detected.

[2] We note that a GHT can also obtain good performance in such a 2D-grid scenario. However, due to the shortcomings of GHTs as stated in sec. 1 and sec. 2, e.g., GHT being infeasible for irregularly shaped areas or variation of the node density, we omit a comparison with a GHT.

sensor, no request packets are required. The number of update packets generated in the network per second is called *update rate*. Similar to the request rate, the update rate may vary heavily in the different scenarios depending on node density as well as the mobility of tracked objects. Given that we assume random object mobility, it is safe to assume that detection events occur uniformly distributed over time and space.

In the SSR case, both request packets and update packets exist. Update packets will be delivered to the responsible node of the detected object. Each object in the network has a responsible node to store the object's information, and to provide this information when requested. In contrast to the request case above, request packets are not flooded but sent directly to the responsible node.

4.2. Metrics

For the performance comparison of the different approaches, the following metrics are used:

- Success rate R (request, reply / update): for the request case and SSR, the success rates corresponds to the percentage of replies that arrive back at the inquirer. This implies, that the request reached at least one node that stored the required information, and that the reply packet was not lost along the return path. For the update case, the success rate corresponds to the percentage of updates that arrive at the central sink node.
- Delay D (request, reply / update): for the request case and SSR, this is the delay between sending the request and receiving the reply (i.e., approximately twice the average packet delay of an end-to-end path). For the update case, this is the delay between the transmission of the update (when the event occurred) and its arrival at the central sink node.

If information that is distributed over the network is requested by a user, two factors are important: 1) the availability of the information, which is represented by the success rate and 2) the time needed to get the answer, which is influenced by the delay. In the following experiments, we will study success rate and delay for each scenario. To facilitate the performance comparison of the protocols, we say that the network is saturated if the delivery ratio is less than 90% and the delivery delay is larger than 300 ms.

4.3. Request Flooding Case

In the request flooding scenario, the inquirer sends the request packets to all of its physical neighbors, who will forward the request further to their neighbors. Here, we vary not only the request rate but also the reply rate (i.e., the number of replies sent back for a single request). From Fig. 2(a), we can observe that the success rate decreases below 90% at a request rate of 8 when the reply rate is below 20. For higher reply rates, the inquiry success rate drops earlier at a request rate of 5.

Fig. 2(d) depicts the reply delay of flooding under different scenarios. When the reply rate is low, the reply delay is below or around 100 ms and thus well below our margin of 300 ms. For a reply rate of 40, the delay satisfies the unsaturated condition only at request rates below 6.

In the simulation of the request flooding case with 400 nodes (not depicted in the graphs), we find that the performance results are very similar to the result in 100 nodes. The difference of the curves is negligible. The traffic "density" per area only depends on the request rate, no matter how large the network is (assuming a constant node density). Thus, in the 400 node case, we mainly see a slight increase in the reply delay due to the increase in path length.

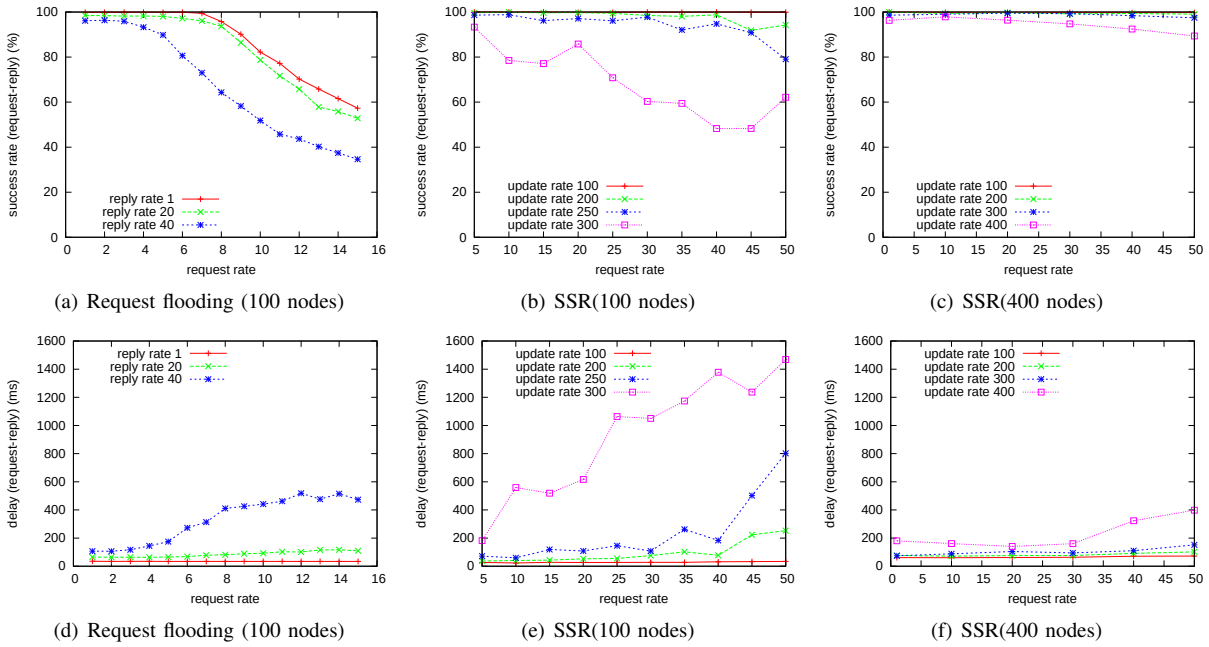
4.4. Update Case

The update case is a very common approach for sensor data collection scenarios. Each node sends an update packet to the sink when a new event is detected, or several events are detected in an interval. Such updates ensure that the information is fresh, and the request rate has basically no impact on the performance, since it only applies to the central sink. In this experiment, we use AODV as routing protocol for the nodes to send updates to the sink.

When events occur very frequently, the paths to the sink may be overloaded by the large amount of update packets. However, many of them may be not even needed. The network capacity around the sink is eventually the bottleneck, where many of the update packets will be dropped.

As shown in Fig. 3(a), when the update rate is above 225 packets/second, the update success rate will drop to 90% in the 100 node scenario. In case of 400 nodes, the update rate is limited to 175 packets/second to achieve an update success rate above 90%. Fig. 3(c) shows that in these two scenarios of 100 nodes and 400 nodes, the delay exceeds 300 ms when the update rate is higher than 225 and 200 packets/second respectively.

From Fig. 3(a), we observe that when the network size increases from 100 nodes to 400 nodes, the permitted update rate decreases. To keep the update success rate above 90%, the permitted update rate is reduced from 225 to 200 packets/second. The reason for this is that the AODV protocol needs to deliver more RREQs (route request messages) to find the route from sensors to the sink. These additional flooding messages exacerbate the traffic concentration at the sink node (which is the only bottleneck in the update case).



(a) Request flooding (100 nodes) (b) SSR(100 nodes) (c) SSR(400 nodes)

(d) Request flooding (100 nodes) (e) SSR(100 nodes) (f) SSR(400 nodes)

Fig. 2. Performance comparison between SSR and flooding with 100 nodes and 400 nodes

4.5. SSR Case

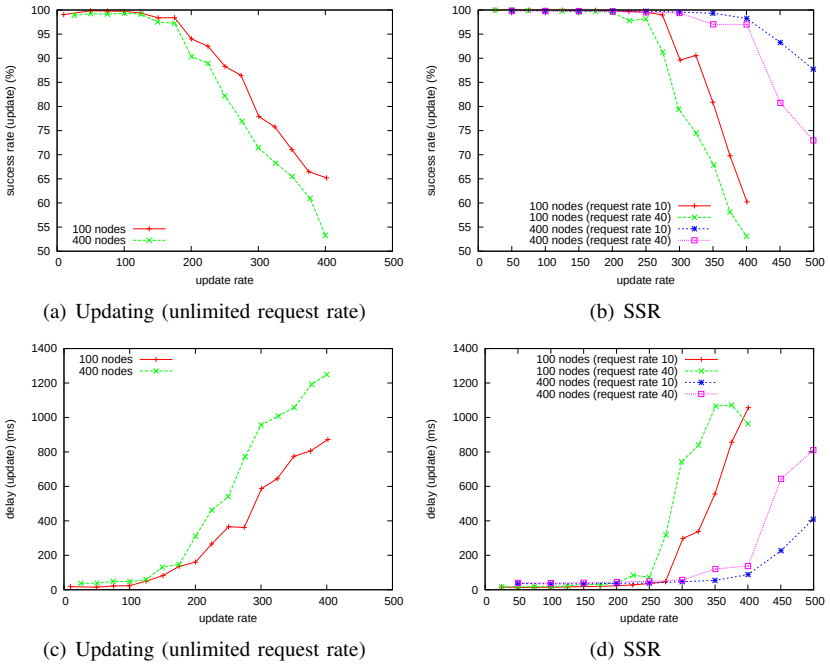
For the SSR case, the update packets are distributed over the whole network, which avoids this traffic concentration around the sink. Information is only delivered to the sink upon a request.

To ensure an even distribution of updates, we give each node a fixed address, and select a hash function that maps all the objects' IDs uniformly to these node addresses. For the simulations we proceed as follows: the node addresses and the object IDs are selected uniformly at random from the same range, and the hash function maps the object ID to the node, whose address is the closest to its ID. However, any hash function that distributes object IDs sufficiently uniformly is suitable. In SSR, nodes can have arbitrary addresses, which makes the selection of a suitable hash function easier than with a GHT, where the hash function might have to account for specific properties of the underlying topology. This provides flexibility to add new sensors to the network, or to remove some defective sensor without having to modify the hash function.

For ease of presentation we only consider a reply rate of 1. Since the information about one object will always be stored on the same sensor node, multiple replies would only be necessary to obtain a longer history of object movements, or to continuously track the object's position for a certain period of time. Of course, queries for different objects have to be sent separately to the corresponding nodes.

We first discuss SSR in comparison to flooding of requests. As shown in Fig. 2(b) and Fig. 2(e), when the update rate is less than 250 packets/second, the inquiry success rate is always larger than 90%, and the delay is mostly less than 300ms, despite the increment of the request rate. If the update rate is larger than 250, the network is considered saturated either because of a low success rate or too high delay. In the 100 node scenario, the delay of SSR can even exceed the delay for flooding in case of extremely high update rates. For 400 nodes, Fig. 2(c) and Fig. 2(f) show that the network is not saturated even with a high update rate of 400 packets/second, w.r.t. both success rate and delay. Again, performance is higher due the better distribution of traffic load. (Note: Although the average path length in the 400-node scenario is doubled, the node number is however quadrupled; therefore, the traffic per node is actually halved and media contention is consequently reduced.)

We now investigate SSR for different update rates and compare it to the AODV-based update case. Fig. 3 shows success rate and delay for this case. As expected, the results for SSR are similar to those observed in the request case: approximately 250 updates/second with 100 nodes and 400 updates/second with 400 nodes can be supported. Again, we can see an improvement in performance for the 400 node scenario. The delay is smaller than that of the 100 node network in case of a high update rate, as shown in Fig. 3(d). The lower concentration of traffic outweighs the increase in path length. In contrast, the update delay in the 400 node network is larger than that in the 100 node network for small update rates. Here, the traffic load does not play



(a) Updating (unlimited request rate)

(b) SSR

(c) Updating (unlimited request rate)

(d) SSR

Fig. 3. Performance comparison between SSR and the update case with 100 and 400 nodes

a significant role and the delay reduction due to the shorter routes in the 100 node network is visible.

Because request packets, reply packets, and update packets in SSR are all unicast packets, and one or both communication end points for these packets are random, the end-to-end delay for these packets is relatively similar. Since request packets originate at the sink and reply packets are delivered only to the sink, the traffic concentration (and packet lose rate) can be slightly higher in that area, but this difference is almost negligible.

4.6. Bootstrapping and Resistance to Churn

We also simulated node failures on a 100 node grid network with SSR. In this scenario, a fraction f of the nodes will fail at a random time instant during the 10 minutes of simulated time. As is shown in Fig. 4, the packet delivery ratio decreases as the node failure rate increases. We observe that even at the high node failure rates (20%), the delivery ratio of the overall network is still satisfactory (86%). Since SSR is a DHT-inspired routing protocol, the packets delivered to the failed node will actually arrive on its virtual neighbor, and if we allow the virtual neighbor to respond these packets, the request delivery ratio is even higher (more than 96%). This property demonstrates the stability of the SSR network.

4.7. Summary of Results

Request flooding: Since in the flooding case, sensors store the information only locally, it supports an unlimited update rate. The request rate should be less than 9 packets/second, under the assumption that the number of reply packets required by the application is not very high (below 20).

Updates: All the information is stored at the sink, supporting an unlimited request rate. Updating works well (according to our criteria of 90% success rate and less than 300 ms delay) when the update rate is less than 200 updates/second with 100 nodes. This limit decreases with an increase in network size.

SSR: The performance of SSR for object tracking is determined by the specific combination of update rate and request rate. It outperforms request flooding for update rates smaller than 250 (or 400) and the update case for all request rates we investigated (max. 40). Furthermore, the maximum supported update rate increases when the network size increases, hence adapting SSR in object tracking is a scalable solution.

5. Conclusions

In this paper, we studied the suitability of indirect routing for object tracking sensor applications, using the example of SSR. It performs much better than flooding of requests for objects or routing of updates to a central entity for reasonable (intermediate) update and request rates. This is possible without some of the disadvantages

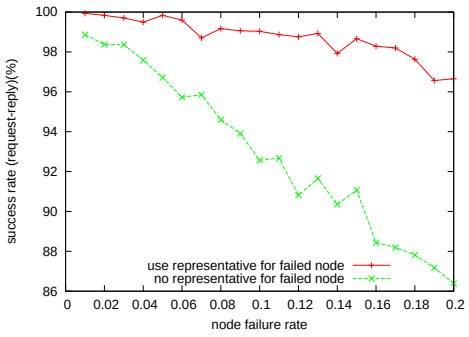


Fig. 4. Effects of node churn on SSR (100 nodes, 60 update/sec, 5 request/sec)

of GHTs, which otherwise have properties similar to our solution. In particular, we are more flexible with regard to the network topologies that are well supported by the protocol.

A particularly intriguing feature of SSR is the fact that it directly supports information storage through key based routing. This is likely to result in a more efficient architecture than having the P2P storage mechanism and wireless routing protocol as separate modules.

SSR was shown to be a highly scalable routing solution [19]. Together with the desirable property of distributing update traffic very evenly in a large network, this allows object tracking even for very large sensor deployments. In the simulations we investigated networks of up to 400 nodes and observed that the larger network even outperformed the smaller one. Moreover, since SSR is a self-organizing protocol, it can recover from node or route failures in a short period of time, which results in a stable overall network.

References

[1] J. Kulik, W. R. Heinzelman, and H. Balakrishnan, "Negotiation-based protocols for disseminating information in wireless sensor networks," *Wireless Networks*, vol. 8, no. 2-3, pp. 169–185, 2002.

[2] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," in *IEEE/ACM Transactions*, vol. 11, Feb. 2003, pp. 2–16.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad hoc sensor networks," in *Proc. of the OSDI Conference*, Dec. 2002.

[4] Y. Yao and J. E. Gehrke, "Query processing in sensor networks," in *In First Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, Asilomar, California, Jan. 2003.

[5] F. Ye, G. Zhong, S. Lu, and L. Zhang, "Gradient broadcast: a robust data delivery protocol for large scale sensor networks," *Wirel. Netw.*, vol. 11, no. 3, pp. 285–298, 2005.

[6] F. Ye, H. Luo, J. Cheng, S. Lu, and L. Zhang, "A two-tier data dissemination model for large-scale wireless sensor networks," in *MobiCom '02: Proceedings of the 8th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM Press, 2002, pp. 148–159.

[7] J. Youn, R. Kalva, and S. Park, "Efficient data dissemination and aggregation in large wireless sensor networks," in *Vehicular Technology Conference*, Sept. 2004.

[8] S. Kim, S. H. Son, J. A. Stankovic, S. Li, and Y. Choi, "SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks," in *Proc. 23rd International Conference on Distributed Computing Systems Workshops*, 2003, pp. 228–234.

[9] T. Clausen, P. Jacquet, and P. Hipercom, "RFC 3626 - Optimized Link State Routing protocol (OLSR)," October 2003.

[10] S. Ratnasamy, D. Estrin, R. Govindan, B. Karp, S. Shenker, L. Yin, and F. Yu, "Data-centric storage in sensornets," in *In Proceedings of the First Workshop on Sensor Networks and Applications (WSNA)*, Atlanta, GA, Sept. 2002.

[11] S. Ratnasamy, B. Karp, S. Shenker, D. Estrin, R. Govindan, L. Yin, and F. Yu, "Data-centric storage in sensornets with ght, a geographic hash table," *Mobile Networks and Applications*, vol. 8, no. 4, 2003.

[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the SIGCOMM 2001 conference*. ACM Press, 2001, pp. 161–172.

[13] B. Karp and H. T. Kung, "GPSR: Greedy perimeter stateless routing for wireless networks," in *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000)*, Boston, MA, August 2000, pp. 243–254.

[14] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris, "A Scalable Location Service for Geographic Ad Hoc Routing," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom '00)*, Boston, MA, USA, 2000, pp. 120–130.

[15] Y. C. Hu, S. M. Das, and H. Pucha, "Exploiting the Synergy between Peer-to-Peer and Mobile Ad Hoc Networks," in *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, Lihue, Kauai, Hawaii, May 2003, pp. 37–42.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware) 2001*, Heidelberg, Germany, Nov. 2001.

[17] H. Pucha, S. M. Das, and Y. C. Hu, "Ekta: An Efficient DHT Substrate for Distributed Applications in Mobile Ad Hoc Networks," in *Proceedings of the 6th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2004)*, English Lake District, UK, Dec. 2004.

[18] T. Fuhrmann, "Scalable routing for networked sensors and actuators," in *Proceedings of the Second Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks*, Sept. 2005.

[19] ——, "A self-organizing routing scheme for random networks," in *Proceedings of the 4th IFIP-TC6 Networking Conference*, Waterloo, Canada, May 2005, pp. 1366–1370.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proceedings of the SIGCOMM 2001 conference*. ACM Press, 2001, pp. 149–160.

[21] M. Caesar, M. Castro, E. B. Nightingale, G. O'Shea, and A. Rowstron, "Virtual Ring Routing: Network Routing Inspired by DHTs," in *Proc. ACM SIGCOMM '06*, Pisa, Italy, Sept. 2006.

[22] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica, "The Impact of DHT Routing Geometry on Resilience and Proximity," in *Proceedings of the SIGCOMM 2003 conference*. ACM Press, 2003, pp. 381–394.

[23] K. Kutzner and T. Fuhrmann, "Using linearization for global consistency in ssr," in *Proc. 4th Int. Workshop on Hot Topics in P2P Systems*, Mar. 2007.

[24] M. Onus, A. Richa, and C. Scheideler, "Linearization: Locally self-stabilizing sorting in graphs," in *Proc. Workshop on Algorithm Engineering and Experiments*, 2007.

[25] A. Varga, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM'2001). June 6-9, 2001. Prague, Czech Republic*, 2001.

[26] H. Lundgren and E. Nordström, AODV Implementation from Uppsala University. [Online]. Available: http://www.docs.uu.se/docs/research/projects/scanet/aodv/aodvuu.shtml