Latency and Lifetime Optimization for k-Anycast Routing Algorithm in Wireless Sensor Networks

Lucas Augusto de Araujo Marques Leao, Violeta Felea

Latency and Lifetime Optimization for k-anycast Routing Algorithm in Wireless Sensor Networks

Lucas Leão ✉ and Violeta Felea

FEMTO-ST institute, Univ. Bourgogne Franche-Comté, CNRS
DISC, 16 route de Gray, 25030 Besançon, France
{firstname.lastname}@femto-st.fr

Abstract. Wireless Sensor Network (WSN) applications frequently require different objectives, such as reliability, timely communication and longevity. To cope with that, a WSN can be conceived with multiple sinks and routing protocols designed over different communication schemes. In this paper, we address the communication latency and the network lifetime issues in Multi-Sink WSN deployment. We propose GeoK, a k-anycast geographic routing protocol that considers a linear combination of network metrics during the decision process of the next hop. Packets are forwarded over exactly k sinks, with targets and duplications being defined on the fly. Simulation results show a better performance for GeoK, with improvements of approximately 13% for the average latency, and 30% for the maximum energy consumption, compared to existing work.

Keywords: Wireless Sensor Networks · Routing · k-anycast · Latency · Network Lifetime · Geographic Routing

1 Introduction

The Multi-Sink Wireless Sensor Network (MS-WSN) is a particular case of the WSN containing multiple sinks. Compared to Single Sink solutions, the use of several sinks normally leads to better network performance. On top of that, it also improves the network manageability, providing more flexibility and continuity [8]. By increasing the number of sinks, the number of hops a packet has to travel before reaching any of the sinks is decreased. It has a direct impact on the performance of metrics such as energy consumption and latency.

There are different communication schemes in MS-WSN implementations. **Unicast:** a path towards one of the sinks is defined and reused for future communications. **Anycast:** data is addressed to a group of sinks. It can be 1-anycast, when data is forwarded to anyone of the sinks, or k-anycast when data must reach any k sinks. **Multicast:** information is routed towards all sinks.

The k-anycast is a flexible communication scheme. A k-anycast routing protocol may behave as 1-anycast routing solution when $k = 1$ or even as a multicast routing protocol when k equals the number of all sinks. However, the complexity of the algorithm increases, specially when the target sinks and duplications are

decided on the fly, during the packet forwarding. In those cases, the packet is not addressed to one specific sink, but to a group of sinks, and the final destination is decided on the way based on the network metrics at each hop.

In this paper we propose GeoK, a k-anycast geographic routing algorithm focused on network lifetime and latency optimizations, capable of assuring the packet routing to exactly k different sinks. Our algorithm differs from the literature by combining several techniques to reduce both the latency and the maximum energy consumption. The next hop decision is taken using a linear combination of network metrics, along with dynamic weights. The metrics weights vary according to the ratio of k and the available sinks. Because of that, the paths are constantly changed during the lifetime of the network, which balances the energy consumption. Both duplication and target sinks are decided on the fly. We also combine and adapt two different void handling techniques, making use of the face routing and the passive participation [3]. Finally, when packets must be duplicated, we benefit from MAC-level information to create an ordered sending list based on the duty-cycle schedule of the neighbor nodes participating in the forwarding task.

The remainder of this paper is organized as follows. Section 2 presents a brief discussion on the existing works in MS-WSN adressing k-anycast routing. Section 3 traces our assumptions and the system model. The description of our solution is detailed in section 4, along with the testing scenarios and the discussion of the performance evaluation in section 5. We conclude the paper in section 6 with the future perspectives.

2 Related Works

The number of works dedicated to k-anycast routing is limited. Most of the solutions are designed as 1-anycast routing protocols. Nevertheless, an 1-anycast routing solution could be used as k-anycast if we assume that packets are duplicated at source and forwarded in sequence. The problem with this strategy is that there is no guarantee that k different sinks will be reached. Depending on the forwarding criteria, chances are that the k packets will be forwarded to the exactly same sink. In that sense, k-anycast solutions are essentially different from 1-anycast protocols, with added complexity and special objectives.

The authors in [2] propose RelBAS, a data gathering algorithm with a fault recovery scheme specially designed to assure reliability. The algorithm considers the construction of disjoint trees, rooted at each sink. However, the sensor nodes serving as forwarders are also disjoint, meaning that a node serves as forwarder to exactly one tree. In order to improve reliability, the solution considers forwarding the packet to exactly k sinks, which are decided in advance. The nodes are always part of exactly k trees, in order to forward the packet to the k sinks.

In [5] we find RPKAC, a routing protocol for Rechargeable MS-WSN designed to reduce network latency and optimize the energy consumption, but focused on assuring the delivery by forwarding the packets to k sinks. The algorithm builds spanning trees rooted at the source node, with nodes sending route

request messages in order to define the routing path. The neighbor nodes reply with their cost to reach a sink, and the current node decides to join an existing path with the lowest cost. The cost is calculated based on the linear combination of four metrics: hop count to sink, latency, energy cost and energy replenish rate.

The main objective in KanGuRou [7] is to guarantee the packet delivery to exactly k sinks and at the same time reduce the overall energy consumption. The strategy considers a geographic routing towards a set of sinks. The path is constructed in a greedy way. The current node builds a spanning tree to k sinks with minimum cost. The next hop is decided based on the cost of the energy-weighted shortest path (ESP). The solution assumes an adaptable transmission range in order to reduce the energy cost. At each hop, the algorithm calculates the cost over progress and decides to duplicate the packet towards different paths in order to reach k sinks. When a packet is duplicated, it is targeted to a set of specific sinks, in order to assure the reception by k-sinks. The void areas are handled using a recovery mode with face routing.

Routing solutions [2] and [5] are based on the hop-count distance to the sinks. This strategy implies either higher network topology knowledge or an important setup phase, with nodes discovering their hop-count distance to each sink. In [7] the focus is on reducing the energy consumption, favoring transmissions to closer nodes. Since the transmission range is variable, the energy cost to transmit a packet to a closer node is reduced. However, the number of hops is increased and consequently the latency.

3 System Model and Assumptions

We represent a MS-WSN as a graph $G = (V, E)$, where V represents the set of all nodes and E the set of existing wireless links. Each $e \in E$ corresponds to a pair of nodes (i, j) as long as i and j, with $i, j \in V$, are within each other's transmission range (symmetric link). The set of neighbors of i is represented by V_i. We also specify that $V = \{S \cup N\}$ where S represents the set of sink nodes and N the set of sensor nodes, with $S \cap N = \emptyset$. The number of sinks to be reached is denoted by k, with $0 < k \leq |S|$.

We assume that every node is aware of its own geographic position and the geographic position of all sinks. For simplification, the geographic positions are represented by the Cartesian coordinates (x, y), and the distance is always the euclidean distance represented as $|ij|$ with $i, j \in V$. We also assume that packets are generated by sensor nodes only, and that k is given and remains constant for each packet generation. For the energy consumption, we follow the radio model in [6], which considers distance and packet size for the dissipated energy during transmission and reception.

We consider two types of networks: with void areas and without void areas. A network with void areas is defined by the existence of a specific node out of the transmission range of a particular sink and for which there is no other neighbor node that presents a better geographic progress towards that sink than the node itself. In summary, $\exists\{n, s_i\}, n \in V, s_i \in S, (n, s_i) \notin E$ such that $|ns_i| < |v_j s_i|$

for all $v_j \in V_n$. In a MS-WSN it is possible for a node to be in the void area of one or more sinks. The amount of void areas is a fixed number based on a particular node and the set of sinks. However, since k sinks must be reached, the probability of entering a void area increases with a higher k. It is because the selection of a sink is a conditional event that depends on the previous selection.

4 Geographic K-Anycast Routing

We propose GeoK, a geographic routing protocol for MS-WSN that assures the packet routing to exactly k sinks using a k-anycast communication scheme, focused on reducing the overall latency and maximizing the network lifetime. The next hop is selected by the current node based on the calculation of weighted metrics. The selection of the targeted sinks is done based on the ordered list of energy cost to reach each of the available sinks. The algorithm can be divided into three steps with a preprocessing at the beginning of the algorithm, and the actual routing at the end. The preprocessing is responsible for retrieving the packet information and triggering the GeoK processing. We denote the current value of k as k_{curr} and the set of available sinks as S_a. The first processing step is responsible for filtering the neighbor nodes, in order to create a list of candidate forwarders. The filtering takes place by eliminating the neighbor nodes with negative progress and neighbors in void areas. If no neighbor is found, the recovery mode is triggered and a neighbor is selected using a void handling technique. The second processing step is dedicated to effectively selecting the forwarders. The candidates are evaluated based on the weighted metrics, and a forwarder is selected for each sink, respecting the size of k_{curr}. The third processing step is triggered only if $k_{curr} < |S_a|$, and it is responsible for distributing the remaining sinks throughout the selected forwarders. Finally, the actual routing is responsible for the packet transmission and an eventual duplication.

Preprocessing. The solution starts at the current node with the execution of the pre-processing whenever a new packet is generated or received and has to be forwarded. The first step is to check if the current node is a sink itself and if it is in the list of target sinks. If the current node is one of the target sinks, the k_{curr} must be decremented, and the current node must be removed from the list of available sinks. If k_{curr} is still greater than zero, the multi-hop process continues with the selection of the new forwarder with the algorithm 1.

First Step: Candidates Filtering. The candidate filtering step is responsible for creating a list L of candidate forwarders. The list can be composed entirely of candidates with positive progress, candidates issued from the recovery mode, or a mixed list, with candidates presenting positive progress and candidates issued from the recovery mode. Since the algorithm has to find a suitable forwarder for each packet, it is possible that for networks with void areas, only part of the k can be covered by candidates with positive progress. Because we need to assure the delivery to exactly k sinks, it is necessary to complete the candidate list with the neighbor nodes issued from the recovery mode. The filtering process is described in algorithm 1.

The filtering starts by checking if the node itself offers a positive progress in relation to the previous hop (line 3). If the current node n progress to the set of available sinks (S_a) is smaller than the value from the previous hop, it means that the packet was in recovery mode and no positive progress was yet found. In this case, the packet must keep the recovery mode status and the next hop is selected using the face routing [3]. The recovery algorithm follows the same principles of the work in [7]. If the current node offers a positive progress in relation to the previous hop, the normal candidate selection is started. The algorithm selects only the neighbor nodes with positive progress to at least one

Algorithm 1 $[F, p_{new}] = GeoK(n, k_{curr}, S_a, V_n, H, p_{prev})$

Input: n: current node, k_{curr}: current number of sinks to be reached, S_a: set of available sinks, V_n: set of neighbors of n, H: set of neighbor nodes in void areas for a set of sinks, p_{prev}: previous value for the packet progress towards S_a

Output: F: forwarders with the sinks and k, p_{curr}: current progress towards S_a

1: $F \leftarrow \emptyset$; $L \leftarrow \emptyset$ /* Set of pairs [candidate, sink] */
2: $p_{new} \leftarrow ProgressTowards(n, S_a)$
3: **if** $p_{new} > p_{prev}$ **then**
4: $S' \leftarrow \emptyset$ /* Set of found sinks */
5: **for all** $v_j \in V_n$ **do**
6: **for all** $s_i \in S_a$ **do**
7: **if** $dist(n, s_i) > dist(v_j, s_i)$ **and** $\{[v_j, s_i]\} \notin H$ **then**
8: $L \leftarrow L \cup \{[v_j, s_i]\}$
9: **if** $S' \cap \{s_i\} = \emptyset$ **then**
10: $S' \leftarrow S' \cup \{s_i\}$
11: **end if**
12: **end if**
13: **end for**
14: **end for**
15: $k' \leftarrow |S'|$
16: **if** $k' < k_{curr}$ **then**
17: /* Lack of candidates, void handling is triggered */
18: **if** $k' > 0$ **then**
19: $F \leftarrow Fwd(n, L, S', k')$ /* Select the forwarders among the candidates */
20: $k_{curr} \leftarrow k_{curr} - k'$
21: **end if**
22: $S' \leftarrow S_a \backslash S'$
23: $SendVoidNotification(n, V_n, S')$
24: $L = Recovery(V_n, S', k_{curr})$ /* Recovery mode using face routing */
25: **end if**
26: **else**
27: /* Current node does not offer a positive progress */
28: $L = Recovery(V_n, S_a, k_{curr})$ /* Recovery mode using face routing */
29: $p_{new} \leftarrow p_{prev}$
30: **end if**
31: $F \leftarrow F \cup Fwd(n, L, S_a, k_{curr})$
32: **return** $[F, p_{new}]$

sink, and excludes the neighbors that have already announced being in a void area (line 7). The values in H represent the set of pairs $[v_j, s_i]$ having the neighbor node in a void area for a given sink. The algorithm creates a second sink list, with the sinks for which a neighbor with positive progress was detected (line 10). The size of this list represents the maximum possible k'. If k' is smaller than the k_{curr}, it means that it was not possible to find a positive progress towards all necessary sinks (line 18). In this case, the algorithm tries to create a preliminary list of forwarders for the found sinks and neighbors (line 19), and completes the list with the candidates issued from the recovery mode (line 24). At the same time, and for the set of sinks the algorithm cannot find a suitable neighbor candidate, the void announcement is triggered (line 23). The algorithm creates a list of sinks for which the current node is in a void area, and a broadcast message is sent with the list in order to inform the neighbor nodes. Each node receiving the broadcast message updates its own H set with the information from n.

Second Step: Forwarders Selection. The forwarders selection stage is focused on selecting the most suitable forwarders (the F list) based on the decision

Algorithm 2 $F = Fwd(n, L, S_a, k_{curr})$

Input: n: current node, L: set of candidate forwarders with respective sinks, S_a: set of available sinks, k_{curr}: number of sinks to be reached
Output: F: list with the set of forwarders with the respective sinks and k

1: $F \leftarrow \emptyset$; $S_{used} \leftarrow \emptyset$ /* Set of selected target sinks */
2: $p_k \leftarrow k_{curr}/|S_a|$
3: $S' \leftarrow OrderSinksByEnergyCost(n, S_a)$ /* S': sinks ordered by energy cost */
4: **for all** $s_i \in S'|i \leq k_{curr}$ **do**
5: $S_{used} \leftarrow S_{used} \cup \{s_i\}$; $R \leftarrow \{v_j|[v_j, s_i] \in L\}$
6: $v \leftarrow SelectForwarder(n, R, F, s_i, p_k)$
7: $f \leftarrow find(F, v)$ /* returns the index of the position of v or -1 if nonexistent */
8: **if** $f < 0$ **then**
9: $f \leftarrow |F|$
10: **end if**
11: $F[f].neighbor \leftarrow v$; $F[f].sinks \leftarrow F[f].sinks \cup \{s_i\}$; $F[f].k \leftarrow F[f].k + 1$
12: **end for**
13: $S'' \leftarrow S' \backslash S_{used}$ /* S": set of other possible target sinks */
14: **if** $S'' \neq \emptyset$ **then**
15: /* Distributes the remaining sinks based on the energy cost */
16: $F = DistributeRemainingSinks(F, S'')$
17: **end if**
18: **return** F

metrics (distance, consumed energy and duplication avoidance) and the defined weights for each metric. Each entry in the F list represents a different forwarder for the packet. It means that if $|F| > 1$ a duplication takes place. The F list contains a structure with the neighbor node responsible for the forwarding task, the list of target sinks and the size of the new k (k_{new}). The list of target sinks

may contain exactly k_{new} sinks or more. It depends on the number of available sinks in comparison to k_{curr} and the proximity of the already selected target sink (S_{used}) in relation to the ones not yet selected (S'').

The target sinks must be chosen in a way to reduce the energy consumption and the packet latency. To cope with that, the solution creates an ordered list of sinks based on the energy cost (algorithm 2, line 3). The first iteration of the ascending ordering process considers the energy cost from the current node to the closest sinks. The energy cost is calculated and the sink having the smallest value is inserted in the list. Then, the next sink is selected based on the minimum energy cost to either the current node or the already selected sink. This process is repeated iteratively until all the available sinks are inserted in the ordered list.

Once the sink list is created, the algorithm starts the process of searching for the most suitable forwarder. The set of candidates having the sink s_i as target is extracted from L (algorithm 2, line 5), and the selection of the forwarder v is started (algorithm 2, line 6).

The selection of the most suitable forwarder is executed using the decision metrics and their respective weights. The weights vary according to the ratio between the k_{curr} and the number of available sinks $|S_a|$. Initially, the value of the aggregated metric $\omega[v_j, s_i]$ is computed using the process in 1.

$$\alpha \times \frac{D[v_j, s_i] - min(D[*, s_i])}{max(D[*, s_i]) - min(D[*, s_i])} + \beta \times \frac{E[v_j] - min(E)}{max(E) - min(E)} + \delta \times G(v_j) \quad (1)$$

Where D is the list of distances, E is the list consumed energy, G represents a function that returns 0 if the neighbor v_j is already a forwarder and 1 otherwise, α represents the relative weight for the distance metric, β represents the relative weight for the consumed energy metric and δ represents the relative weight for the duplication avoidance metric. Each metric has a different objective. As for instance, the distance metric is focused on selecting the candidate with the highest positive progress towards the target sink. This is important in order to reduce the overall hop count and consequently the latency. The consumed energy metric regards the selection of the node with the smallest consumption, which in time balances the energy consumption, prolonging the network lifetime. Finally, the duplication avoidance accounts for both latency and energy consumption, since the increase of packets in the network not only intensifies the energy consumption, but also multiplies the possibility of congestion and delays. As already mentioned, the weights for each metric are dynamic and adjustable depending on the situation. For a scenario where k is much lower than S, only few duplications may be triggered, and the focus must be on progressing towards the closest sinks, so the weights for the distance metric and energy consumption are higher. On the other hand, if $k = |S|$ the objective changes, and avoiding duplications becomes more important.

The aggregated decision metric ω is calculated using the weighted relative values of the metrics from each candidate. The node having the smallest ω is selected as forwarder. If the selected forwarded v is already part of the forwarders list F, the k value of the existing entry is incremented and the target sink s_i is

added to the list of sinks. Otherwise, a new entry is created in F with the new forwarder, which ultimately triggers a duplication (algorithm 2, lines 7 to 11).

Third Step: Sink Distribution. Since k may be smaller than $|S|$, the destination sink is not fixed, and it may change during the packet forwarding. Even if the forwarders selection makes use of a particular sink as destination to decide on the most suitable forwarder, there are cases where it changes, as for instance when the packet encounters a void area. The change is only possible if $|S_a| > k_{curr}$. In this case, after the forwarders selection, not all sinks are used as destinations $\{S_a \backslash S_{used}\} \neq \emptyset$, and there are remaining sinks to be distributed over the forwarders in F. The distribution process considers the same principle of the sink list ordering, assigning each of the remaining sinks to the forwarder containing the neighbor node or the sinks with the smallest energy cost.

For a remaining sink $s_r \in \{S_a \backslash S_{used}\}$, the algorithm calculates the transmission energy cost E_{tx} from s_r to the forwarder $f_i.neighbor$ where $f_i \in F$ and each of the sinks in $f_i.sinks$. Then, the remaining sink s_r is included in the sink list of the f_i that presents the lowest E_{tx}.

Routing. The actual routing takes place after the forwarders selection. During the actual routing, the packet is duplicated if $|F| > 1$, and it is updated with the forwarder address, the data of the new list of available sinks and the corresponding k. Also as part of the routing process, and for the case a duplication is necessary, the sending order is decided based on the duty cycle of the selected forwarders. The first packet to be sent is the one of the first forwarder to wake-up. The algorithm makes use of the duty-cycle schedule information from the neighbor nodes to determine the encounter moment in order to define the sending order.

5 Simulation and Results

GeoK protocol was developed using Contiki OS [4] and evaluated through simulations using Cooja [4]. The performance of our solution was compared to an existing approach (KanGuRou) [7], that was adapted to Contiki OS and tested with Cooja under the same configurations. The simulation environment and details are outlined in table 1.

As in equation 2 in table 1, when the number of deployed nodes changes, and if the network area is kept the same, the network deployment density changes. Since we want to keep a similar deployment density over all variations of $|V|$, we make the network area vary with the number of deployed nodes.

The solution performance is evaluated by observing the average latency, defined by the average time a packet takes to be routed from the source to each of the sinks, and the maximum energy consumption, which gives an indication of the network lifetime. As per explanation, we consider a network to be alive as long as all nodes have some energy. Therefore, network lifetime is considered to be the earliest moment at which a node's battery is completely depleted.

The simulation considered two main network scenarios: with void areas and without void areas. The simulation outcome is given likewise considering two

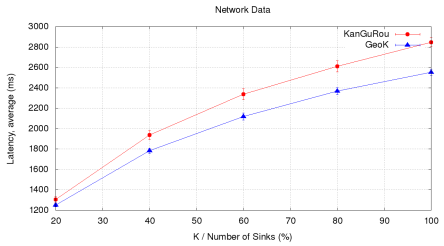
Table 1: Configuration for the simulations

Simulation Settings	
Deployment Density (d)	8 neighbors (on average)
Network Area (variable)	$\frac{\pi \times r^2}{d} \times \lvert V \rvert$ (2)
Comm. Range (r)	50 m
# of Sensors ($\lvert N \rvert$)	50, 100, 150, 200, 250, 300
# of Sinks ($\lvert S \rvert$)	10% of the # of Sensors
k	20% to 100% of the # of Sinks
# of Gen. Networks	300 with voids / 300 without voids
Packet Size	240 bytes
Packet Generation	20% chance at every minute for each sensor
Radio Type	802.15.4
MAC Protocol	CX-MAC, modified version of [1]
Execution Time	120 minutes

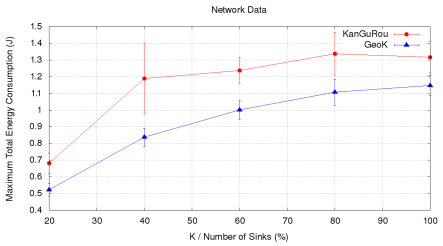
series of results. One with a fixed number of sensors and sinks with a variation of k, in order to evaluate the performance of the solution when k increases, reaching $k = |S|$. And the other where the relative k value is fixed and the number of sensors and sinks varies, in order to evaluate the solution scalability.

5.1 Fixed number of Sensors and Sinks

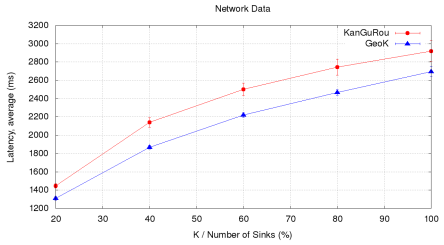
Networks without void areas: for the scenario with no void areas and a fixed number of nodes and sinks, we can see in figure 1a that GeoK shows a better latency performance in relation to KanGuRou, with a growing gain with the increase of k. Although for a small k the options of sinks are better, with nodes being able to choose the closest sinks, the performance of the two solutions are relatively close, with a gain of approximately 4% in favor of GeoK. When k becomes larger, the options of sinks become limited, and the packet must be forwarded to sinks that are much farther. In those scenarios, GeoK shows an improvement of over 10% in relation to KanGuRou. It is explained by the fact that for farther sinks our algorithm is able to find better routes, performing in average 20% fewer hops to reach the more distant sinks. Regarding the maximum energy consumption, we can see in figure 1b that GeoK has also a better performance when compared to KanGuRou. We can notice a maximum gain of almost 30% and a minimum of approximately 13%. That is explained by the nature of our solution that is designed in a way to use different routes during the protocol execution. It means that the energy consumption is distributed, which reduces the chances of nodes depleting their batteries in early stages. Contrarily to the average latency, the performance gain of the maximum energy consumption eases with the increase of k. It is justified by the increase of packets being forwarded. When k is larger, it means that at some point the packet is duplicated. Since there are more packets circulating in the network, the energy consumption raises and the performance decreases. In average, packet duplication is in 30% higher with GeoK when compared to KanGuRou. In order to have a faster progress



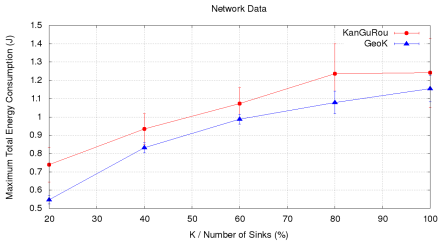
(a) Average Latency (no voids)



(b) Max Consumed Energy (no voids)



(c) Average Latency (with voids)



(d) Max Consumed Energy (with voids)

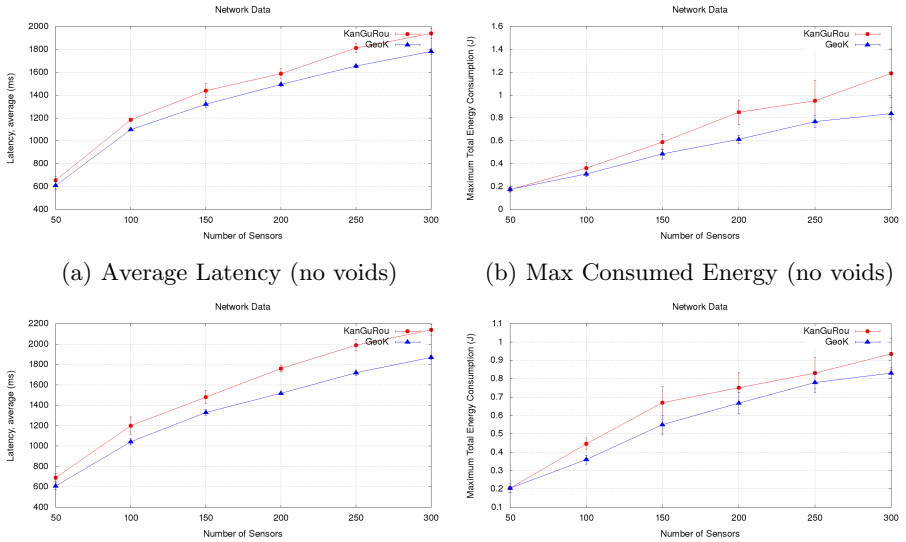
Fig. 1: 300 sensors, 30 sinks and k varying with a step of 20% of the sinks

towards a sink, GeoK is constrained to diverge the path in a earlier stage of the forwarding process, so duplications take place more frequently.

Networks with void areas: when the network with void areas are considered for the same number of sensors and sinks, we can also notice a good performance for GeoK. For the latency, we can see in figure 1c that GeoK has even better results when compared to KanGuRou. That is explained by the void announcement strategy. Since nodes in void areas advertise on their situation, the neighbor nodes may avoid forwarding packets through the problematic area, resulting in better delivery time. However, the performance gain decreases with the increase of k. That happens because with the increment of k the possibility of entering in a void area grows as well. Packets must enter in Recovery Mode more frequently, which increases the hop count. Nevertheless, similarly to the case without void areas, the average hop count for GeoK is smaller than KanGuRou, resulting in lower latencies. GeoK presents a maximum gain of almost 13% and a minimum of approximately 8%.

In terms of maximum energy consumption, GeoK also performed better than KanGuRou, with a maximum gain of almost 26% and a minimum of approximately 7%, as displayed in figure 1d. The same tendency observed in the scenario without void areas is noticed when void areas are present. The gain is reduced with the increase of k. Once again, it can be justified by the fact that more duplications are necessary, increasing the number of packets in the network. When a void area is detected for a specific sink, the node may decide to split the packet in different ways, so the void area is bypassed.

5.2 Variable number of Sensors and Sinks



(a) Average Latency (no voids) (b) Max Consumed Energy (no voids)

(c) Average Latency (with voids) (d) Max Consumed Energy (with voids)

Fig. 2: 50 to 300 sensors, sinks as 10% of sensors and k as 40% of sinks

Networks without void areas: with the relative k fixed to 40% of the sinks, and the number of sensors and sinks varying, we intend to analyze the behavior of the solution in terms of scalability. For the scenario without void areas, we can notice that GeoK has better performances for both average latency and maximum energy consumption, as in figures 2a and 2b respectively. The increase in network size translates to even better results for GeoK in terms of maximum energy consumption, with a gain of approximately 30%, and a moderate improvement of the Latency of 7% on average. By increasing the network size, the possibilities of different paths is also increased. The packets are able to alternate through different route paths, leading to better energy balance and consequently a smaller maximum energy consumption. However, the gain on the average latency performance becomes stable, even with the increase of the network size. It is because the ratio k/sinks/sensors is kept the same. The k is a ratio of the number of sinks, and the number of sinks is a ratio of the number of sensors.

Networks with void areas: for the case with void areas, we can also notice a performance gain of approximately 12% for the Average Latency, as shown in figure 2c. The better results for GeoK are again due to the void announcements strategy. And the stability of the gain comes from the proportionality of the triplet k/Sinks/Sensors. In terms of Maximum Energy Consumption, we can notice in figure 2d a smaller gain for GeoK, with a maximum of approximately

20%. Because for some cases the packet enters in Recovery Mode, as per the recovery strategy, the used path is always the same. Consequently, the Maximum Energy Consumption increases, since there are fewer variations in routing paths.

6 Conclusion

This paper presented a new k-Anycast Geographic Routing solution for Wireless Sensor Networks with multiple sinks. Our strategy makes use of variable weighted metrics to establish the list of forwarders, as well as the necessity of packet duplication. The main goal was to find a balance between Latency and Network Lifetime optimizations. We tested our solutions through simulations against an existing strategy called KanGuRou. The simulation results indicate that our solution has an overall better performance than the existing protocol, with maximum gains of approximately 13% for Latency and 30% for Maximum Energy Consumption.

As future work, we plan to find ways of reducing the packet duplication rate, without affecting the Latency performance. We also foresee the execution of real-life experiments using a testbed with Contiki OS compatible motes.

Acknowledgments. This work was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq). Computations were performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

References

1. Buettner, M., Yee, G.V., Anderson, E., Han, R.: X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks. In: Proceedings of the 4th Int. Conf. on Embedded Networked Sensor Systems. pp. 307–320. ACM (2006)
2. Chakraborty, S., Chakraborty, S., Nandi, S., Karmakar, S.: Fault resilience in sensor networks: distributed node-disjoint multi-path multi-sink forwarding. Journal of Network and Computer Applications **57**, 85–101 (2015)
3. Chen, D., Varshney, P.K.: A survey of void handling techniques for geographic routing in wireless networks. IEEE Comm. Surveys & Tutorials **9**(1), 50–67 (2007)
4. Contiki OS: The OS for the IoT, http://www.contiki-os.org/, accessed: 2018-03-20
5. Gao, D., Lin, H., Liu, X.: Routing protocol for k-anycast communication in rechargeable wireless sensor networks. Computer Standards & Interfaces **43**, 12–20 (2016)
6. Heinzelman, W.R., Chandrakasan, A., Balakrishnan, H.: Energy-efficient communication protocol for wireless microsensor networks. In: 33rd Annual Hawaii International Conference on System Sciences. pp. 1–10. IEEE (2000)
7. Mitton, N., Simplot-Ryl, D., Voge, M.E., Zhang, L.: Energy efficient k-anycast routing in multi-sink wireless networks with guaranteed delivery. In: International Conference on Ad-Hoc Networks and Wireless. pp. 385–398. Springer (2012)
8. Poe, W.Y., Schmitt, J.B.: Self-organized sink placement in large-scale wireless sensor networks. In: IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems. pp. 1–3. IEEE (2009)