Energy efficient k-anycast routing in multi-sink wireless networks with guaranteed delivery

Nathalie Mitton, David Simplot-Ryl, Marie-Emilie Voge, Lei Zhang

Energy efficient *k*-anycast routing in multi-sink wireless networks with guaranteed delivery

Nathalie Mitton*, David Simplot-Ryl*, Marie-Emilie Voge†, Lei Zhang*

*INRIA Lille - Nord Europe - firstname.lastname@inria.fr
†Univ. Lille Nord de France - Marie-Emilie.Voge@lifl.fr

**

Abstract. In k-anycasting, a sensor wants to report event information to any k sinks in the network. This is important to gain in reliability and efficiency in wireless sensor and actor networks. In this paper, we describe KanGuRou, the first position-based energy efficient k-anycast routing which guarantees the packet delivery to k sinks as long as the connected component that contains s also contains sufficient number of sinks. A node s running KanGuRou first computes a tree including k sinks among the M available ones, with weight as low as possible. If this tree has $m \geq 1$ edges originated at node s, s duplicates the message m times and runs m times KanGuRou over a subset of defined sinks. Simulation results show that KanGuRou allows up to 62% of energy saving compared to plain anycasting.

1 Introduction

Wireless sensor networks have been receiving a lot of attention in recent years due to their potential applications in various areas such as monitoring and data gathering. Sensor measurements from the environment may be sent to a base station (sink) in order to be analyzed. Other sensors may serve as routers on a path established to deliver the report. In large sensor networks, there may exist a bottleneck (around sink) if a single sink collects reports from all sensors. Scenarios with multiple sinks are then being considered, where each sensor reports to at least one sink, usually the nearest one. In wireless multi-sink sensor networks, anycasting is performed when any of sinks may receive the report from sensors, and meet application demands. However, the cost of anycasting may depend on the distance between the receiving sinks and the reporting sensor. It is therefore desirable that selected algorithm reaches one of sinks close to the event. For reliability, load-balancing and security purposes, it is then useful to ensure that at least k sinks receive the messages (where the overall number of sinks is greater than k) whatever the k sinks. To date, there is no so much work in the literature. Most of works are adaptation of wired solutions [9] and are thus centralized. Others use flooding [10] and not suitable for high dynamic networks (such as wireless sensor networks). A distributed k-anycast routing protocol based on mobile agents is proposed in [11] but requires a regular update of routing tables which also have to maintain paths towards every sink.

** This work was partially supported by CPER Nord-Pas-de-Calais/FEDER Campus Intelligence Ambiante and the ANR BinThatThinks project.

In this paper, we introduce KanGuRou (k-ANycast GUaranteed delivery ROUting protocol), a **position-based**, **energy-efficient** localized k-anycast routing protocol that **guarantees delivery** (therefore loop-less), is **memory-less**, and **scalable**. Unlike [11], it does not maintain any routing table and does not need to add any information neither on nodes nor in the message, which makes it scalable regardless of the number of sinks/nodes. It inspires from energy-efficient anycast EEGDA algorithm [5] and the splitting techniques of MSTEAM [4], proposing a new tree construction to ensure reaching k sinks. At each step, the current node s computes a spanning tree over k sinks with minimal cost. A message replication occurs when the tree spanning s and the set of sinks has multiple edges (later called branches) originated at the current node. Since there may be more sinks than the k to be reached, all of them are not spanned by the tree. The number of sinks k' spanned by each branch determines the number of sinks to be reached by each message. All sinks (not only the ones spanned by the tree) are distributed over every edge. The next hop is chosen in a cost-over-progress (COP) fashion, *i.e.* to the neighbor v which minimizes the ratio between the cost to reach v and the progress provided by v. The cost from s to v is the cost of the energy-weighted shortest path (ESP). The progress is computed as the difference between the weight of the trees computed by s and v resp. If s has no neighbor with positive progress, node s applies a EEGDA-face like routing, which is a face-based recovery mode. We prove that KanGuRou guarantees delivery to exactly k sinks. We present two variants which differ in the way the tree is computed. KanGuRou is evaluated through extensive simulations and results show that both variants of KanGuRou are energy efficient. Results show that KanGuRou allows up to 62% of energy saving and that every variant performs better regarding the percentage of sinks to reach.

The remaining of the paper is organized as follows. Section 2 gives an overview of the literature about k-anycasting and present works on which KanGuRou is based. Section 3 introduces our notations. Section 4 presents KanGuRou. Section 5 presents simulation results. Finally Section 6 concludes the paper.

2 Related works

k-Anycast was first introduced in [9] for wired networks. Propositions in wireless networks firstly appeared in [8] proposing centralized solutions and thus does not really meet wireless networks requirements. [10, 2] presents a reactive approach (flooding) and two advanced proactive approaches in which sinks have previously been gathered into components of at most k members and these components are then reached during the routing. To the best of our knowledge, the only distributed k-Anycast routing protocol is based on mobile agents and proposed in [11]. The protocol forms multiple components and each component has at least k members. Each component can be treated as a virtual server, so k-anycast service is distributed to each component. In this protocol, each routing node only needs to exchange routing information with its neighbors, so the protocol saves much communication cost and adapts to high dynamic networks. Nevertheless,

although a first step toward, this algorithm needs to maintain routing tables at each node with as many entries as sinks and is not scalable.

In this paper, we introduce KanGuRou which is a position-based k-anycasting protocol. KanGuRou is an extension of the anycasting protocol proposed in [5] to the k anycasting. In [5], authors describe EEGPA the first localized anycasting algorithms that guarantee delivery for connected multi-sink sensor networks based on a GFG approach. Let $S(x)$ be the closest actor/sink to sensor x, and $|xS(x)|$ be distance between them. In greedy phase, a node s forwards the packet to its neighbor v that minimizes the ratio of cost of sending packet to v through an ESP over the reduction in distance ($|sS(s)| - |vS(v)|$) to the closest sink. If none of neighbors reduces that distance then recovery mode is invoked. It is done by face traversal where edges are replaced by paths optimizing given cost.

KanGuRou also inspires from the multicast routing MSTEAM proposed in [4]. MSTEAM is a localized geographic multicast scheme based on the construction of local minimum spanning trees (MSTs), that requires information only on 1-hop neighbors. A message replication occurs when the MST spanning the current node and the set of destinations has multiple edges originated at the current node. Destinations spanned by these edges are grouped together, and for each of these subsets the best neighbor is selected as the next hop. MSTEAM has been proved to be loop-free and to achieve delivery of the multicast message as long as a path to the destinations exists.

3 Model and notations

Network. We model the network as a graph $G = (V, E)$ where V is the set of sensor nodes and $uv \in E$ iff there exists a wireless link between u and $v \in V$. We suppose that nodes are equipped with a location service hardware such a GPS and are able to tune their range between 0 and R. We note $|uv|$ the Euclidean distance between nodes u and v. We note $N(u)$ the set of physical neighbors of node u, *i.e.* the set of nodes in communication range of node u ($N(u) = \{v \,|\, uv \in E\}$) and $V(G)$ the set V of vertices in G. $S = \{s_i\}_{i=0,1,..M}$ is the set of sinks, with M the number of sinks. Every node is aware of every sink and of its position. We note as $CT_S(s)$ the closest node in S to node s ($CT_S(s) = \{v \,|\, |sv| = \min_{w \in S}|sw|\}$). For a graph $G = (V, E)$ and a set $A \subseteq V$, we denote by $G|_A$ the subgraph of G which contains only nodes of A: $G|_A = (A, E \cap A^2)$.

Tree. Let $T = (V', E')$ be a tree and $a \in V'$ a vertex of T. $\mathrm{st}(T, a)$ is the subtree of T with root a. T is an MST if its weight noted $||T||$ is minimal. The weight of the tree denotes the sum of the weight over all tree edges ($||T|| = \sum_{uv \in E'} |uv|$). In an Euclidean MST, the weight of an edge is equal to its Euclidean length. A tree $T = (V', E') \subset G$ is a k-MST if $|V'| = k$ and that $||T||$ is the tree with minimum weight over all trees of k vertices from G.

Energy. We assume that every node is able to adapt its transmission range. We use the energy model defined in [7], *i.e.* the energy spent to send a message from nodes u to v is such that $cost(|uv|) = |uv|^{\alpha} + c$ if $|uv| \neq 0$. where c is signal processing overhead; α is a real constant (> 1) for signal attenuation. From this energy cost, we introduce the cost of the energy-weighted shortest path ($cost_{ESP}(s, d, t)$) from nodes s to d when aiming at target t. We compute the energy-weighted shortest path (ESP) only over nodes that are in the forwarding direction of the final target to avoid either creating routing loops or embedding the path in the message. Therefore, the shortest path computed from node s to node d is relative to the final target t. Let $x_0x_1...x_ix_{i+1}..x_n$, be the node IDs on the ESP from $s = x_0$ to $d = x_n$. We define the ESP cost as

$$cost_{ESP}(s, d, t) = \sum_{i=0}^{n-1} cost(|x_i x_{i+1}|) \tag{1}$$

4 Contribution

4.1 General Idea

In this section, we present the main idea of KanGuRou which goal is to reach any k sinks among all available sinks S. Nevertheless, given a source node s, the k closest sinks to s in Euclidean distance are not necessarily the k closest sinks in number of hops. Therefore, the routing messages in KanGuRou may change target sinks along the routing path. For instance, on Fig. 1, 5 closest sinks of s are S_1, S_2, S_5, S_6 and S_7. But S_1 is not reachable directly and the path to S_1 will meet S_4 which may be reached instead. In addition, the source cannot determine the k sinks in advance and send k messages, one toward each sink because *(i)* several messages may follow the same path by sections which is useless and costly and *(ii)* since targets may change along the path, this cannot ensure that several messages will not reach the same sink.

KanGuRou (Algo. 1) proceeds as follows. Fig. 1 illustrates it.
(1). Node s holding the message first checks whether it is a sink. If so, it removes itself from the set of available sinks and decrements the number of sinks k to reach. If $k = 0$, the algorithm stops. (Line 2).
(2). Node s computes a tree $T(s)$ by running Algo. 3 (k-MST(s,S,k)) or Algo. 4 (k-Prim(s,S,k))) detailed later in Section 4.4, depending of the variant of KanGuRou (Line 7). $T(s)$ contains node s and exactly k sinks of S. If there are several edges/branches originated at s, a message duplication occurs. On Fig. 1, $T(s)$ appear in red and contains sinks S_1, S_3, S_5, S_6 and S_7. There are two branches originated at node s: one toward S_1 and one toward S_5.
(3). s distributes the remaining sinks (Line 8), *i.e.* sinks that are not in $T(s)$ (Sinks S_2, S_4 and S_8 on Fig. 1) over every branch. Thus, for every successor a of s in $T(s)$ ($a \in \text{succ}_{T(s)}$), a subset $S_a \subset S$ of the sinks is assigned to a as detailed in Section 4.5. On Fig. 1, branch of S_1 is assigned with Sinks S_1, S_3 and S_4 while Sinks S_2, S_5, S_6, S_7 and S_8 are associated to branch of S_5.

(4). At this step, node s knows: *(i)* its successors $a \in \text{succ}_{T(s)}$ in $T(s)$ (Sinks S_1 and S_5 on Fig. 1), *(ii)* the number of sinks k_a to reach per successor a, *i.e.* the number of sinks in the subtree of a $st(T, a)$ (2 in branch of S_1 and 3 in branch of S_5 on Fig. 1), *(iii)* the set of available sinks to reach per branch, *i.e.* S_a defined at the previous step. Node s then sends as many packets as the number of its successors in $T(s)$. (Loop line 9) Thus, for each branch of $T(s)$, *i.e.* $\forall a \in \text{succ}_{T(s)}$, s selects a next hop based on a Greedy-Face-Greedy approach as follows. For every a, s computes the weight of the k_a-MST for each of its neighbors $u \in N(s)$ over S_a targets $||\text{k-MST}(u, S_a, k_a)||$. On Fig. 1, s will compute 3-MST over Sinks S_2, S_5, S_6, S_7 and S_8 to find the next hop for branch S_5 and 2-MST over Sinks S_1, S_3 and S_4 for branch S_1. If there exists no neighbor u for which the weight of tree over S_a $||k\text{-MST}(u, S_a, k_a)||$ is smaller than $||sT(T, a)|| + |sa|$ (weight of the branch of $T(s)$ dedicated to a), node s switches to recovery mode (line 16) till reaching a node with positive progress towards a. If so, next hop v for branch toward a is determined through the greedy mode in a COP fashion (Line 18). Message is sent to node v with parameters k_a and S_a which will run KanGuRou again (Line 19) and so on till k_a sinks have been reached in this branch. As shown in [5], this ensures the packet delivery as soon as the network is connected.

Algorithm 1 KanGuRou(s, k, S) – Run at node s to reach k targets in S.

1: **if** $s \in S$ **then**
2: $k \leftarrow k - 1$; $S \leftarrow S \setminus \{s\}$
3: **if** $k = 0$ **then**
4: exit {All sinks of this branch have been reached}
5: **end if**
6: **end if**
7: $T(s) \leftarrow$ k-MST(s, S, k) *or* k-Prim(s, S, k) {k-MST of $S \cup \{s\}$ rooted in s}
8: $T'(s) \leftarrow$ AllocateMST$(s, S, T(s))$ {Allocate remaining targets to $T(s)$}
9: **for all** $a \in \text{succ}_{T(s)}(s)$ **do**
10: $S_a \leftarrow V(\text{st}(T', a))$ {Nodes in sub-tree of T' rooted in a}
11: $k_a \leftarrow |T \cap S_a|$ {Number of targets to be reached in S_a.}
12: $v \leftarrow CT_{S_a}(s)$
13: $W \leftarrow ||sT(T, a)|| + |sa|$
14: $A \leftarrow \{v \in N(s) \mid ||\text{k-MST}(v, S_a, k_a)|| < W\}$
15: **if** $A = \emptyset$ **then**
16: RECOVERY(s, k_a, S_a, W)
17: **else**
18: $v \leftarrow u \in A$ which minimizes $\frac{\text{cost}_{ESP}(s,u,a)}{W - ||\text{k-MST}(u, S_a, k_a)||}$
19: KanGuRou(v, k_a, S_a)
20: **end if**
21: **end for**

To sum up, let assume that node s on Fig. 1 runs KanGuRou toward $k = 5$ sinks. First, s computes a 5-MST, $T(s)$ (red tree). $T(s)$ has two branches, so s duplicates the message. First message is sent toward branch of S_1 and has to

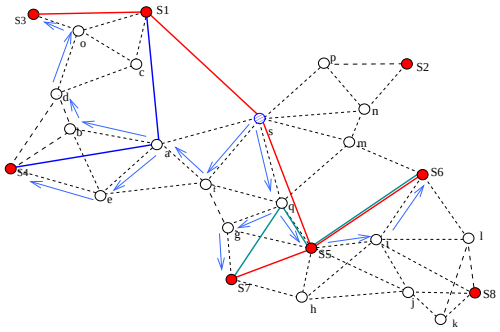


Fig. 1. Sinks appear in red. Red links represent the 5-MST rooted in s, blue links the 2-MST rooted in a over S_1, S_3 and S_4, green links the 3-MST rooted in q over S_2, S_5, S_6, S_7 and S_8. Arrows show the message path.

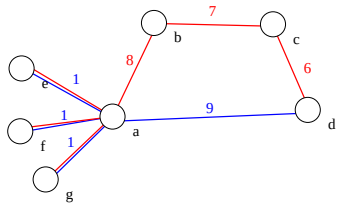


Fig. 2. Illustration of MST and k-MST for $k = 4$. If root is node d, the optimal 4-MST (in blue) includes edges da, ae, af, ag while edge ad will not be included in the MST (in red). So, k-MST is not always included in the MST.

reach 2 sinks among S_1, S_3 and S_4. s computes the COP and selects node a. To reach node a, message is sent to node f since path sfa is less energy consuming than following the direct edge sa. Node a runs KanGuRou and its tree has two branches. So node a duplicates again the message. First copy has to reach one sink among S_1 and S_3 while second copy has to reach S_4. S_4 is reached via path aeS_4 in a greedy way while other copy is sent along path $bdoS_3$. Second message sent by node s has to reach 3 sinks among S_2, S_5, S_6, S_7 and S_8. Greedy algorithm chooses node q. Tree computed on node q has 2 branches originated at q, so q duplicates the message. First copy is sent to node g which forwards it to Sink S_7. Second copy is sent to S_5. S_5 is a sink but the message still has to reach another sink so S_5 forwards it to its neighbor i which directly forwards the message to S_6. At last, 5 sinks have been reached: S_3, S_4, S_5, S_6 and S_7.

4.2 The greedy mode

Greedy mode is similar to the one used in [5]. When node s runs greedy algorithm toward Sink a, it computes the subtree $sT(T(s), a)$ of $T(s)$ rooted in a. The weight W of the subtree issued from s toward a is thus the weight of $||sT(T(s), a)||$ plus the weight of the edge sa to reach it: $W = ||sT(T(s), a)|| + |sa|$. Then, to select the next hop, node s performs a COP approach in which *(i)* the cost considered is the cost of the energy weighted shorted path (Eq. 1) from node u to its neighbor v,*(ii)* the progress is the reduction of the weight of trees $W - ||\text{k-MST}(u, S_a, k_a)||$. Only neighbors providing a positive progress are considered. If no such node exists, the greedy approach fails and s switches to recovery mode. If there exist neighbors u such that $W > ||\text{k-MST}(u, S_a, k_a)||$, node u which minimizes $\frac{\text{cost}_{ESP}(s,v,a)}{W-||\text{k-MST}(u,S_a,k_a)||}$ is selected. Note that when computing k-MST(u, S_a, k_a), all potential sinks are considered, not only the ones in $sT(T(s), a)$. For instance, on Fig. 1, 2-MST computed by node a (blue tree) over S_1, S_3 and S_4 includes S_1 and S_4 (while the one rooted in s includes S_1 and S_3).

Algorithm 2 RECOVERY(u,k,S,W) - Run at node u.

1: $(V', E') \leftarrow$CDS$(V, E) \cup S \cup \{u\}$ {Extract a CDS graph from G}
2: $(V', E'') \leftarrow$GG(V', E') {Build the Gabriel Graph of G'}
3: $u' \leftarrow u$, $T \leftarrow$ k-MST(u', S, k)
4: **while** $||k - MST(v, k, S|| > W$ **do**
5: $v \leftarrow$ FACE(u', T) {Compute the next node on the proper face}
6: **while** $u' \neq v$ **do**
7: $u' \leftarrow$ESP$(u', v, CT_T(u'))$ {Compute the ESP from u' to v}
8: **end while**
9: **end while**
10: KanGuRou(v, k, S)

4.3 The recovery mode

Recovery mode is detailed in Algo. 2. A node u enters the recovery mode while trying to reach k targets among the sinks in S if it has no neighbor which k-MST has a smaller weight than its own weight W toward the considered branch. u runs RECOVERY till reaching a sink or a node v for which $||$k-MST$(v, k, S)||$ is smaller than W (Line 2 in Algo. 2)[1].

To determine what neighbor to reach, it applies an EtE-like Face routing [3]. EtE-like Face routing differs from the traditional Face [1] routing in the way that it does not run over the planar of the whole graph but on the planar of a connected dominated set (CDS) graph only (Lines 2-2). This allows considering longer edges. Face algorithm is applied to determine next hop v to reach over the faces on the CDS (Line 2). v is then reached by following an ESP (Line 2).

4.4 Computing the *k*-MST

Note that computing an exact k-MST is NP-complete. Also note that a k-MST is not necessarily included in the MST as example plotted on Fig. 2 shows. Thus, KanGuRou proposes to use two different tree constructions, both of them being an approximation of the k-MST algorithm. As we will see later, the choice of the variant used in the tree construction will depend on the number of sinks M available in the network and the number k of sinks that need to receive the information. It is important to highlight that this tree is computed on the complete graph of sinks $\varsigma = (S, E_\varsigma)$ with $E_\varsigma = \{uv \,|\, u, v \in S^2\}$. This is independent from the underlying topology.

First variant: The first variant (later called KanGuRou) applies Algo. 3 and builds a tree with exactly $k + 1$ vertices (k sinks and the source) in an iterative way. It starts with a tree which only contains the root (Line 3), node s on Fig. 1. It then has to choose exactly k sinks in S to add in T. To do so, at each step, it computes the shortest path from any vertex to the tree in exactly i hops, for all

[1] Unlike in anycasting, recovery in k-anycasting may reach a sink since the distance considered is not between a node and the closest sink but to the closest k sinks.

Algorithm 3 k-MST(u, S, k) – Return a k-MST of $S \cup \{u\}$ rooted in u.

```
1: T ← ({u}, ∅) {initialize the tree with root u}
2: A ← S {set of nodes to be considered.}
3: while k > 0 do
4:    for all v ∈ A do
5:       w ← x ∈ T which minimizes |xv|
6:       P(v, 1) ← w {Path from v to T in 1 hop with minimum cost.}
7:       l(v, 1) ← |vw| {Weight of the path from v to T in 1 hop with minimum cost.}
8:    end for
9:    for i = 2 to k do
10:      for all v ∈ A do
11:         y ← x ∈ T which minimizes |vx|
12:         ∀w ∈ A z ← x ∈ T which minimizes |wx|
13:         Select w ∈ A such that |wz| < |vy| which minimizes (l(w, i − 1) + |vw|)/i
14:         p(v, i) ← p(w, i − 1).w {Path from v to T in i hops with minimum cost.}
15:         l(v, i) ← l(w, i − 1) + |vw|{Weight of p(v, i).}
16:      end for
17:   end for
18:   select v ∈ A and j ∈ [1 . . . k] which minimizes l(v, l)/j
19:   while p(v, j) ≠ ∅ do
20:      (w, x) ← first edge in p(v, j) {w is supposed to be in T while x is not in T}
21:      T ← T ∪ ({x}, {(w, x)}); A ← A \ {x}; k ← k − 1
22:      p(v, j) ← p(v, j) \ {(w, x)}
23:   end while
24: end while
25: Return T.
```

i from 1 to $k - i$ for all vertices. On Fig. 1, for $i = 1$, s computes the distance from itself to every sink. For $i = 2$, s considers 2-hop paths from itself to every sink and keeps the shorter one as sS_1S_3 to reach S_3. To reduce the complexity of computing a path from a node u to T, it only considers nodes closer than u to T. On Fig. 1, node s will not compute any 2-hop path from s to S_2 since S_2 is the closest sink. Weight of every path is then normalized by the progress it provides, *i.e.* the number of sinks on the path (Line 3) and the path with the lowest weight is then added to the tree. And so on till the final tree includes k sinks. In this way, note that S_2 is not included in path since step 1, path sS_5S_7 (weight 2) is chosen ($\frac{|sS_5|+|S_5S_7|}{2}$ is smaller than all other path ratios as $\frac{|sS_1|+|S_1S_3|}{2}$ or $\frac{sS_2}{1}$). Then at step 2, path sS_1S_3 is added ($\frac{|sS_1|+|S_1S_3|}{2} < \frac{|sS_1|+|S_1S_3|+|S_3S_6|}{3}$, etc) and at last, path S_5S_6 is added.

Second variant: Original Prim algorithm [6] consists in adding iteratively to the current tree (initialized with the root node) the edge with minimum weight which has exactly one extremity vertex in the tree, and so on till every vertex has been added to the tree. KanGuRou-kPrim (Algo. 4) performs similarly but stops when the tree includes and exactly k sinks.

Algorithm 4 k-Prim(u, S, k) – Return a k-MST of $S \cup \{u\}$ rooted in u.

1: $T \leftarrow (\{u\}, \emptyset)$ {initialize the tree with root u}
2: $A \leftarrow S$ {set of nodes to be considered.}
3: **while** $k > 0$ **do**
4: $\quad w \leftarrow x \in A$ which minimizes $|xCT_T(x)|$
5: $\quad T \leftarrow T \cup (\{w\}, \{(w, CT_T(w))\})$
6: $\quad A \leftarrow A \setminus \{w\}$; $k \leftarrow k - 1$
7: **end while**
8: Return T.

To illustrate the difference between both variants, let us consider Fig. 2 and assume a tree construction rooted in node d with $k = 4$. Algo. 4 adds iteratively the edge (and corresponding nodes) with the lowest weight, *i.e.* nodes c, b, a and e (in the order). Resulting tree has a weight of 22. Algo. 3 does not consider edges one by one but multi-hop paths. It thus adds nodes a and e at once ($\frac{|da|+|ae|}{2}$ is the best ratio), then nodes f and g. Resulting tree has a weight of 12.

4.5 Distributing sinks over branches

Once the k-tree rooted in current node has been computed, the set of sinks has to be distributed over each branch. The number of sinks to be reached by branch is given by the number of sinks actually part of the branch. If s is the node in charge of the message, it computes its k-MST $T(s)$. If k_a is the number of sinks to be reached in the branch of $T(s)$ rooted in a, we have $\sum_{a \in \text{succ}_{T(s)}} k_a = k$. The set of potential sinks to reach S_a is sent with the message over each branch a. S_a includes the k_a sinks included in the tree but also part of 'free' ones. S_a sinks have to be selected carefully in order to ensure that exactly k sinks will receive the message. They are such that: *(i)* $\bigcup_{a \in \text{succ}_{T(s)}} S_a = S$ since every sink is candidate and *(ii)* $S_a \cup S_b = \emptyset \, \forall \, a, b \in \text{succ}_{T(s)}$ in order to avoid that a message sent on 2 different branches reaches the same sink in which case, the overall number of sinks receiving the message will be less than k.

In KanGuRou, each sink is assigned to the closest branch regardless of the size of the branches. However, we are aware that this solution is not necessarily the most adequate one since most of remaining sinks may be assigned to the same branch which might be the smallest one. Alternative solutions might be:

- Sinks may be distributed evenly between both branches, based on distance.
- Sinks may be distributed proportionally to the number of sinks to reach per branch.

However, setting in advance the number of sinks to assign to each branch will lead to some other issues. Indeed, issue will appear when sinks are at equal distance of several branches and when a sink p is closer to Branch A, but that Branch A has already been assigned enough sinks, all closer than p. We leave to further work a deeper study on this point.

4.6 Packet delivery to exactly k sinks guaranteed

We show that KanGuRou delivers a message to exactly k sinks as long as the underlying network is connected. Because of page restriction, we only give here the sketch of the proof[2].

Theorem 1. *KanGuRou guarantees the packet delivery to exactly k sinks as long as the network is connected and that the number of sinks in the connected component including s is greater or equal to k.*

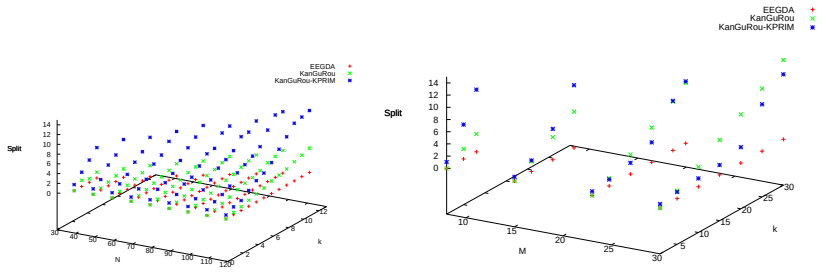
Proof. We apply a mathematical induction.
Initial step. *Theorem 1 is true for $k = 1$.*
When $k = 1$, the 1-MST computed by s running KanGuRou comes to finding $CT_S(s)$, *i.e.* the closest sink to s. KanGuRou comes to EEGDA [5], been proven to guarantee packet delivery as long as the underlying network is connected.
Induction step. *Assuming that Theorem 1 is true for $k = i-1$, $1 < i$, we have to prove that Theorem 1 is true for $k = i$.*
When node s runs KanGuRou, it may either duplicate and forward the message or just forward it once. When s splits the message, s runs several times KanGuRou for $k < i$. If s forwards, itt forwards until finding a sink that will then run KanGuRou for $k = i - 1$ or to a node that split the message and then runs several time KanGuRou for $k < i$, for which Theorem 1 is assumed to be true.

5 Simulation results

In this section, we evaluate the performances of KanGuRou under the WSNet[3] simulator with an IEEE 802.15.4 MAC layer. As there is no comparable algorithm in the literature since KanGuRou is the first position-based algorithm from the literature, we compare the two variants KanGuRou and KanGuRou-kPrim to running k times the plain EEGDA anycast routing protocol [5] to measure the gain provided by KanGuRou. We deploy N nodes (from 35 to 115) at random in a square of $100\text{m} \times 100\text{m}$, every node can adapt its range between 0 and 30m.

Fig. 3 shows the number of times the message is split/duplicated for each algorithm. Obviously, the number of splits performed by EEGDA is equal to 1 whatever the parameters since EEGDA performs independent anycast routings. For both versions of KanGuRou, it is worth noting that when k increases for a given number of available sinks M and of nodes N, the number of splits also increases. This is expected since algorithms need to reach more sinks and respective trees are bigger and thus the message is more likely to be duplicated to reach sinks. Also, for a fixed k, the number of splits increases when the number of nodes (and thus of available sinks) increases. This is due to the fact that more choices are given to the algorithm and thus more ramifications appear (Fig. 3(a)). We can also note (Fig. 3(b)) that the number of duplications is not really impacted by the overall number of available sinks M in the network

[2] Complete proof is available at researchers.lille.inria.fr/~mitton/kangourou.html.
[3] WSNet: http://wsnet.gforge.inria.fr/



(a) $M = 10\% \times N$ (b) $N = 75$

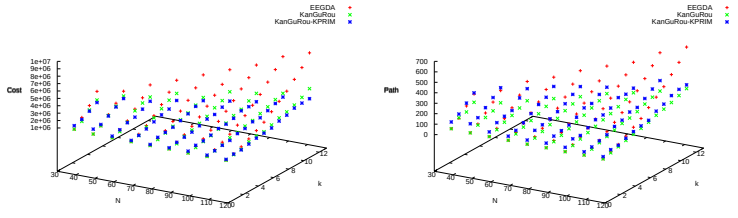
Fig. 3. Number of splits for each algorithm. M = number of sinks.

(number of splits for a given k). At last, we can observe that the number of duplications increases when M increases (in proportion of N) more quickly for KanGuRou than for KanGuRou-kPrim. Yet, for a low value of M, KanGuRou-kPrim produces more duplications than KanGuRou while for high values of M, KanGuRou duplicates more often messages.

First, the number of sinks M is set to be 10% of the total deployed nodes N. Fig.4 shows the energy consumption (computed based on Eq. 1) and the path length in terms of N and k (k varies from 1 to M). Note that for $k = 1$, results are the same for all three algorithms since KanGuRou comes to EEGDA. Simulation results show clearly that KanGuRou, KanGuRou-kPrim result in significant gains on the energy consumption (up to 62.51% (44.33% in average) and up to 74.22% (53.84% in average) respectively) and path length (up to 62.17% (49.07% in average) and up to 56.61% (21.90% in average) respectively) compared to the traditional algorithm EEGDA. An amelioration was indeed expected since in KanGuRou, part of the path is mutualized. Nevertheless, the gain remains important. Globally, we can see that behavior of every algorithm is similar whatever the parameters. Regarding the energy consumption, results show that KanGuRou-kPrim consumes less energy compared to KanGuRou when k is important, and KanGuRou performs better for low k. This is due to the fact that when k increases (for a constant M), k-Prim algorithm gets closer and closer to the optimal k-MST construction. This is also linked to the number of message duplications illustrated by Fig. 3. A high number of splits implies shorter paths.

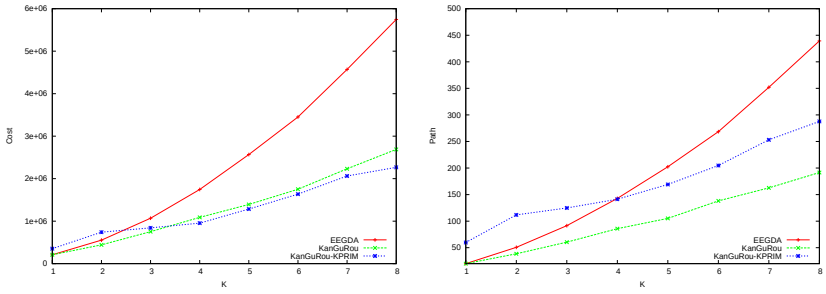
Figure 5 gives a closer look at the energy consumption and the path length in terms of k when the total deployed nodes N is a constant ($N = 75$) and M is set to be 8 sinks. We can see KanGuRou-kPrim performs better regarding energy consumption when k is greater than 3, and KanGuRou always has a gain of the path length compared to the other two algorithms in this case.

In the second scenario (Fig. 6), we fix the number of the total deployed nodes N to 75 and evaluate the performances of the three algorithms (EEGDA, KanGuRou, KanGuRou-kPrim) regarding the overall number of sinks M in the



(a) Energy consumption (b) Path length

Fig. 4. Algorithms performances with regards to N and k for $M = 10\% \times N$.

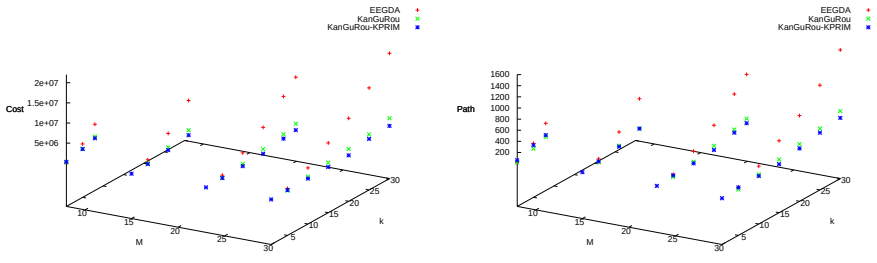


(a) Energy consumption (b) Path length

Fig. 5. Algorithms performance in terms of k over $M = 8$ sinks among $N = 75$ nodes.

network. Obviously, when k increases for a given number of available sinks M, the path and the energy consumption increase since there are more sinks to reach. Similarly, when the number of sinks to reach k is fixed and that the number of available sinks M increases, the path and the energy consumption decrease since algorithms have more choice among sinks and can join closer ones. An important feature is that results show that KanGuRou-kPrim performs better than KanGuRou for high values of M and k. Once again, this is linked to the number of path splitting and that the greater k, the closer to the optimal k-MST, k-Prim algorithm is.

To sum up, the simulation results of different scenarios clearly show that *(i)* KanGuRou variants result in a significant gain of energy consumption and path length compared to the traditional algorithm EEGDA, *(ii)* depending of the percentage of sinks to be reached, one variant of KanGuRou performs better than the other one. When k is small (when $k \leq 30\% \times M$), KanGuRou always consumes less energy than KanGuRou-kPrim, *(iii)* when k is important (when $k > 30\% * M$), KanGuRou-kPrim brings a significant gain compared to KanGuRou especially when M is important. This is highlighted by Fig. 7 which has a closer look at this feature. Figure clearly shows that up to a given number of



(a) Energy consumption (b) Path length

Fig. 6. Algorithms performances with regards to M and k for $N = 75$ nodes.

available sinks, KanGuRou-kPrim performs better than KanGuRou ($M = 23$ on figure).

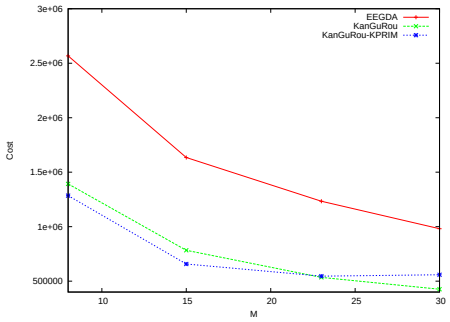


Fig. 7. Algorithms performances for $k = 5$ and $N = 75$ nodes.

6 Conclusion and Future Works

In this paper, we have introduced KanGuRou, the very first position-based k-anycast routing protocol which is energy efficient and guarantees the packet delivery. Two variants are proposed for the construction of the tree. KanGuRou performs well when the number of sinks to reach is lower than 30% of the available sinks in the network while KanGuRou-kPrim performs better for higher values of k. In future work, we intend to claim theoretically how far KanGuRou is from the optimal centralized algorithm and provide some complexity analysis. We also intend evaluate the properties of KanGuRou more deeply (robustness toward mobility, wireless instability, etc).

References

1. P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(8):609–616, 2001.
2. B.Wu and J.Wu. k-anycast routing schemes for mobile ad hoc networks. In *IPDPS*, 2006.
3. E. H. Elhafsi, N. Mitton, and D. Simplot-Ryl. Energy Efficient Geographic Path Discovery With Guaranteed Delivery in Ad hoc and Sensor Networks. In *IEEE PIMRC*, 2008.
4. H. Frey, F. Ingelrest, and D. Simplot-Ryl. Localized mst based multicast routing with energy-efficient guaranteed delivery in sensor networks. In *WOWMOM*, 2008.
5. N. Mitton, D. Simplot-Ryl, and I. Stojmenovic. Guaranteed delivery for geographical anycasting in wireless multi-sink sensor and sensor-actor networks. In *IEEE INFOCOM*, 2009. Short paper.
6. R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
7. V. Rodoplu and T. Meng. Minimizing energy mobile wireless networks. *IEEE JSAC*, 17:1333–1347, 1999.
8. W. Wang, XY Li, and O. Frieder. k-anycast game in selfish networks. In *ICCCN*, 2004.
9. X. Wang. Analysis and design of a k-anycast communication model in ipv6. *Comput. Commun.*, 31:2071–2077, June 2008.
10. B. Wu and J. Wu. k-anycast routing schemes for mobile ad hoc networks. In *IPDPS*, 2006.
11. X. Xu, Y-L Gu, J. Du, and H.-y. Qian. A distributed k-anycast routing protocol based on mobile agents. In *WiCOM*, 2009.