



**HAL**  
open science

## Guaranteed Delivery in $k$ -Anycast Routing in Multi-Sink Wireless Networks

Nathalie Mitton, David Simplot-Ryl, Jun Zheng

► **To cite this version:**

Nathalie Mitton, David Simplot-Ryl, Jun Zheng. Guaranteed Delivery in  $k$ -Anycast Routing in Multi-Sink Wireless Networks. EAI Endorsed Transactions on Mobile Communications and Applications, 2013. hal-00782853

**HAL Id: hal-00782853**

**<https://inria.hal.science/hal-00782853>**

Submitted on 2 Dec 2013

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Guaranteed Delivery in $k$ -Anycast Routing in Multi-Sink Wireless Networks

Nathalie Mitton<sup>1</sup>, David Simplot-Ryl<sup>1</sup>, Jun Zheng<sup>2</sup>

<sup>1</sup>Inria Lille - Nord Europe - firstname.lastname@inria.fr

<sup>2</sup>National Mobile Communications Research Lab - Southeast University, China

**Abstract:** In  $k$ -anycasting, a sensor wants to report event information to any  $k$  sinks in the network. 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 at least  $k$  sinks. A node  $s$  running KanGuRou first computes a tree including  $k$  sinks 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. We present two variants of KanGuRou, each of them being more efficient than the other depending of application settings. 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 several sinks are available, each offering same services. Then 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. Although many anycasting protocols have been deployed in wired networks [14], developing an efficient anycast routing protocol for wireless networks is challenging. Energy consumption

and scalability are two challenging issues in designing protocols for sensor networks since they operate on limited capacity batteries while the number of deployed sensors could be very large. To the best of our knowledge, only few protocols have been designed for anycasting (when  $k = 1$ ) in wireless networks. Most of them are based on an adaptation of an anycast routing for wired networks [1] and need flooding techniques that do not scale. Other ones [3, 7, 8] need a costly tree structure that is not robust and does not scale well in dynamic networks. Position based anycasting algorithm [8] is greedy and localized but optimizes neither hop count nor power consumption. Only algorithms proposed in [9] are geographic localized anycast routing protocols that guaranteed delivery (therefore loop-less), are memory-less, and scalable. In case of localized position based anycasting problem considered in this article, sensor nodes are merely aware of their positions, positions of their neighbors, and positions of all actors/sinks. But to the best of our knowledge, so far, there is no efficient position-based  $k$ -anycasting. To date, there is no so much work in the literature. Most of works are adaptation of wired solutions [13] and are thus centralized. Others use flooding [15] 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 [16] but requires a regular update of routing tables which also have to maintain paths towards every sink.

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 [16], 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 [9] and the splitting techniques of MSTEAM [5], 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. When this latter is less than 30% while the second one (KanGuRou-kPRIM) offers better energy saving when the percentage of sinks to reach is greater than 30%.

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 work

$k$ -Anycast was first introduced in [13] for wired networks. Propositions in wireless networks firstly appeared in [12] proposing centralized solutions and thus does not really meet wireless networks requirements. [15] 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 [16]. 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.

Anycasting for wireless networks has first been modeled in [1]. Although many anycast protocols have been deployed in wired networks [14], there are very few for anycasting in wireless networks in the literature and only one of them [8] is geographical. Most of existing solutions are based on anycast for wired networks and need to build some structures. For example, in [7] a shortest path anycast tree rooted at each source is constructed for each event source. Sinks are the only leaves of the tree, and can dynamically join/leave the tree, which is updated accordingly. Data is delivered to the nearest sink on the tree. The algorithm thus simultaneously maintains paths to all sinks, and requires memorization of routing steps. Building a tree requires a lot of message exchanges. Tree-based protocols are not scalable, since the maintenance is costly when network has dynamic changes or when actors are moving.

In this paper, we introduce KanGuRou which is a position-based  $k$ -anycasting protocol. KanGuRou is an extension of the anycasting protocol proposed in [9] to the  $k$  anycasting. The only known position based anycasting algorithm is proposed in [8], where energy consumption needed to communicate at distance  $d$  is proportional to  $u(d) = d^\alpha + c$ . In the startup phase, each sensor node selects its next hop as follows. Let  $Q$  be a sensor,  $N$  be one of its neighboring nodes, and  $A$  be one of actors. Sensor  $Q$  selects neighbor  $N$  for which  $u(|QN|) + u(|NA|)$  is minimized, over all neighbors and over all actors. This localized anycasting algorithm does not really optimize the power consumption (despite the claim), because it makes decision in the neighbor selection process based on long edges  $|NA|$  which are not power optimal (an analytical proof of this fact was given in [5]). Further, it does not guarantee delivery in the presence of void areas. Initial routes are then used in a centralized data collection algorithm [8] as follows. All sensors within a region, when events occur, are reporting, each one to its actor selected by initial anycasting step. Authors [8] formulate integer linear program to construct data aggregation tree to minimize overall energy spent for reporting. In [9], authors describe EEGPA the first localized anycasting algorithms that guarantee delivery for connected multi-sink sensor networks based on a GFG approach. Three geographic localized anycast routing protocols are described loop-less, memory-less and scalable. They are generalizations of the EtE [4] protocol itself based on the well-known greedy-face-greedy (GFG) [2] unicast routing protocol to anycasting. 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. EEGDA variant is to forward to the first neighbor on the shortest weighted path toward  $v$ . 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 [5]. 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 spans 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. To date, MSTEAM is the best known multicast algorithm.

### 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  $u$  ( $N(u) = \{v \mid uv \in E\}$ ) and  $V(G)$  the set  $V$  of vertices in  $G$ .  $S = \{s_i\}_{i=0,1,\dots,M}$  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  $s$  ( $CT_S(s) = \{v \mid |sv| = \min_{w \in S} |sw|\}$ ). 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  $T$ .  $st(T, a)$  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 [11], *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;  $\alpha$  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  $x_0 x_1 \dots x_i x_{i+1} \dots x_n$ , be the node IDs on the ESP from  $s = x_0$  to  $d = x_n$ . We define the ESP cost as

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

## 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 in  $S$ . 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 get closer to  $S_4$  which may be reached also. 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  $T(s)$  ( $a \in succ_{T(s)}$ ), 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  $S_5$ .

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  $\|k\text{-MST}(u, S_a, k_a)\|$ . 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 [9], 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:  $T(s) \leftarrow k\text{-MST}(s, S, k)$  or  $k\text{-Prim}(s, S, k)$  { $k$ -MST of  $S \cup \{s\}$  rooted in  $s$ }
8:  $T'(s) \leftarrow \text{AllocateMST}(s, S, T(s))$  {Allocate remaining targets to  $T(s)$ }
9: for all  $a \in \text{succ}_{T(s)}(s)$  do
10:   $S_a \leftarrow V(st(T', a))$  {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:   $A \leftarrow \{v \in N(s) \mid \|k\text{-MST}(v, S_a, k_a)\| < W\}$ 
15:  if  $A = \emptyset$  then
16:    RECOVERY( $s, k_a, S_a, W$ )
17:  else
18:     $v \leftarrow u \in A$  which minimizes  $\frac{\text{cost}_{ESP}(s, u, a)}{W - \|k\text{-MST}(u, S_a, k_a)\|}$ 
19:    KanGuRou( $v, k_a, S_a$ )
20:  end if
21: end for

```

---

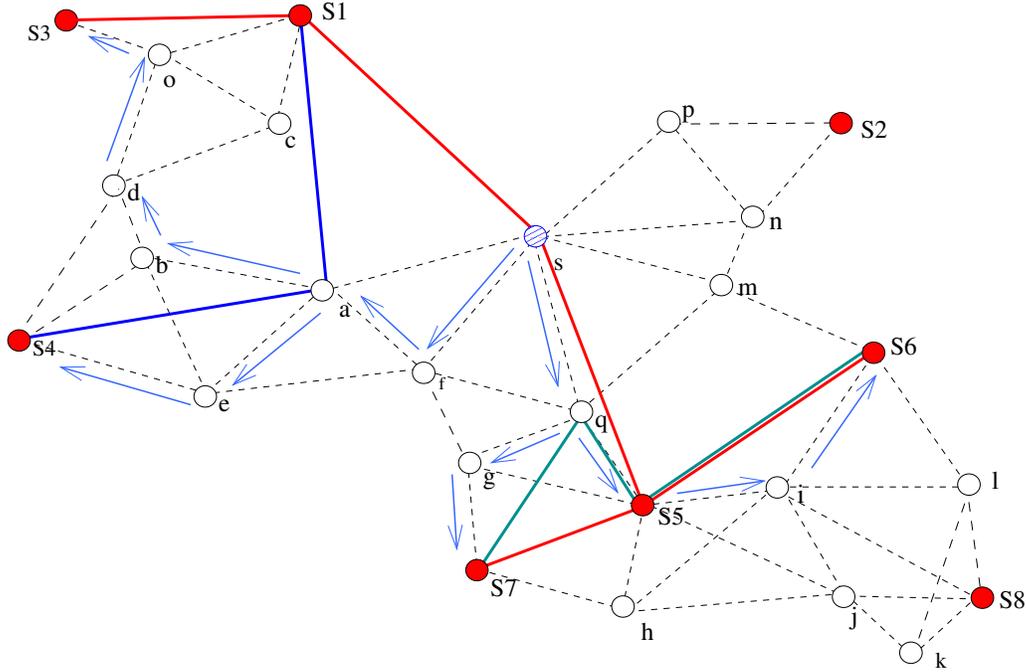
To sum up, let us assume that node  $s$  on Fig. 1 runs KanGuRou toward  $k = 5$  sinks. First,  $s$  computes a 5-MST,  $T(s)$  (red tree).  $T(s)$  has two branches, so  $s$  duplicates the message. First message is sent toward branch of  $S_1$  and has to reach 2 sinks among  $S_1, S_3$  and  $S_4$ .  $s$  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  $boS_3$ . 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 [9]. 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  $W - \|k\text{-MST}(u, S_a, k_a)\|$ . 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  $W > \|k\text{-MST}(u, S_a, k_a)\|$ , node  $u$  which minimizes  $\frac{\text{cost}_{ESP}(s, u, a)}{W - \|k\text{-MST}(u, S_a, k_a)\|}$  is selected. Note that when computing  $k\text{-MST}(u, S_a, k_a)$ , all potential sinks are considered, not only the ones in  $sT(T(s), a)$ . Therefore,  $k\text{-MST}(u, S_a, k_a)$  can include different sinks than  $sT(T(s), a)$ . 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  $S_3$ ).

## 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



**Figure 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.

sink or a node  $v$  for which  $\|k\text{-MST}(v, k, S)\|$  is smaller than  $W$  (Line 4 in Algo. 2). 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. Yet, the weight of the tree issued in  $t$  may have a highest weight than the tree issued on the node which have launched the recovery step.

To determine what neighbor to reach, it applies an EtE-like Face routing [4]. EtE-like Face routing differs from the traditional Face [2] 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 1-2). This allows considering longer edges. Face algorithm is applied to determine next hop  $v$  to reach over the faces on the CDS (Line 5).  $v$  is then reached by following an ESP (Line 7) and not necessarily by following the direct edge.

#### 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  $\zeta = (S, E_\zeta)$

---

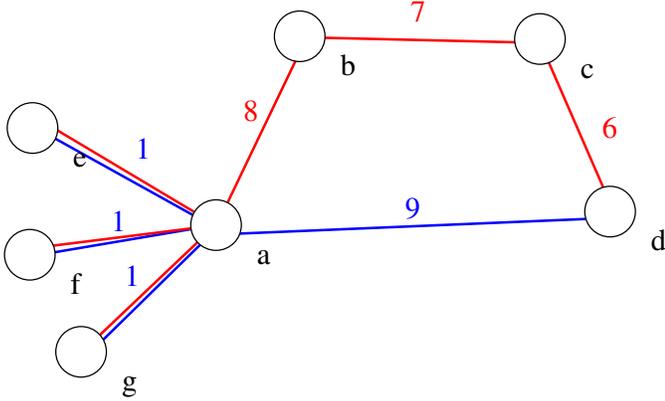
#### Algorithm 2 RECOVERY( $u, k, S, W$ ) - Run at node $u$ .

---

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

with  $E_\zeta = \{uv \mid u, v \in S^2\}$ . 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 1), 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  $i$  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



**Figure 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.

---

**Algorithm 3**  $k\text{-MST}(u, S, k)$  – 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:   for  $v \in A$  do
5:      $w \leftarrow x \in T$  which minimizes  $|xv|$ 
6:      $P(v, 1) \leftarrow w$  {Path from  $v$  to  $T$  in 1 hop with
       minimum cost.}
7:      $l(v, 1) \leftarrow |vw|$  {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  $v \in A$  do
11:       $y \leftarrow x \in T$  which minimizes  $|vx|$ 
12:       $\forall w \in A$   $z \leftarrow x \in T$  which minimizes  $|wx|$ 
13:      Select  $w \in A$  such that  $|wz| < |vy|$  which
        minimizes  $(l(w, i-1) + |vw|)/i$ 
14:       $p(v, i) \leftarrow p(w, i-1).w$  {Path from  $v$  to  $T$  in  $i$ 
        hops with minimum cost.}
15:       $l(v, i) \leftarrow l(w, i-1) + |vw|$  {Weight of  $p(v, i)$ .}
16:    end for
17:  end for
18:  select  $v \in A$  and  $j \in [1 \dots k]$  which minimizes
     $l(v, j)/j$ 
19:  while  $p(v, j) \neq \emptyset$  do
20:     $(w, x) \leftarrow$  first edge in  $p(v, j)$  { $w$  is supposed to
      be in  $T$  while  $x$  is not in  $T$ }
21:     $T \leftarrow T \cup (\{x\}, \{(w, x)\})$ ;  $A \leftarrow A \setminus \{x\}$ ;  $k \leftarrow k - 1$ 
22:     $p(v, j) \leftarrow p(v, j) \setminus \{(w, x)\}$ 
23:  end while
24: end while
25: Return  $T$ .

```

---

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 18) 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 [10] 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. KanGuRoukPrim (Algo. 4) performs similarly but stops when the tree includes and exactly  $k$  sinks.

---

**Algorithm 4**  $k\text{-Prim}(u, S, k)$  – 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:    $w \leftarrow x \in A$  which minimizes  $|xCT_T(x)|$ 
5:    $T \leftarrow T \cup (\{w\}, \{(w, CT_T(w))\})$ 
6:    $A \leftarrow A \setminus \{w\}$ ;  $k \leftarrow k - 1$ 
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  $\sum_{a \in \text{succ}_{T(s)}} k_a = k$ . Nevertheless, the  $k_a$  sinks attached to branch of  $a$  are not necessarily the closest

ones in number of hops while other sinks which are not in the tree can be closer (like  $S_1$  on Fig. 1 which is the closest sink to  $s$  but not in hop count). 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)  $S_a \cap S_b = \emptyset \forall a, b \in \text{succ}_{T(s)}$  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  $k$ .

Algo. 5 details how the remaining sinks can be attached to every branch. 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. For instance, on Fig. 3,  $S_A = \{u_1, u_2, u_3, u_4, u_6, u_7\}$  and  $S_B = \{u_5, u_8, u_9, u_{10}, u_{11}, u_{12}\}$ .
- Sinks may be distributed proportionally to the number of sinks to reach per branch. For instance, on Fig. 3, since branch A is supposed to reach 3 sinks over 4, it will be assigned  $\frac{3}{4} = 9$  sinks (sinks  $u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_{10}$ ) while branch B will receive 3 sinks (Sinks  $u_9, u_{11}, u_{12}$ ).

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.

---

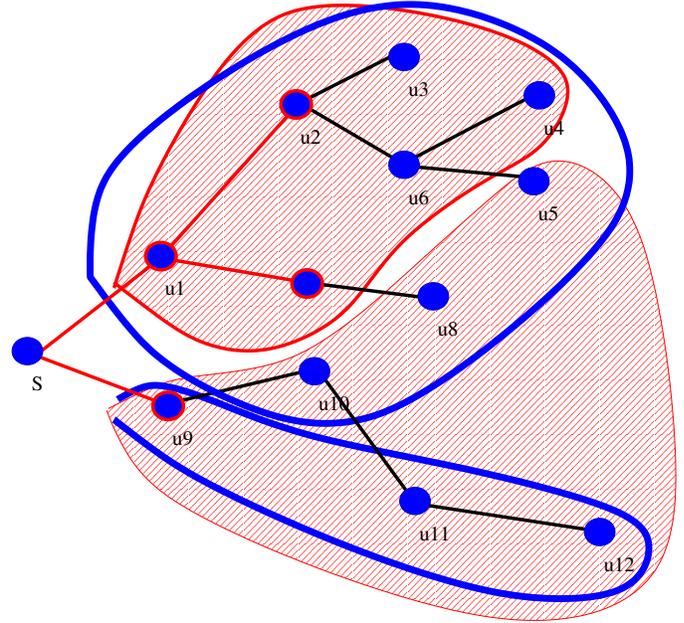
**Algorithm 5** AllocateMST( $u, S, T$ ). Allocate nodes in  $S$  not in  $T$  over  $T$  branches.

---

- 1:  $A \leftarrow S \setminus T$
  - 2: **while**  $A \neq \emptyset$  **do**
  - 3:   select edge  $uv$  in  $V(T) \times A$  which minimizes  $|uv|$
  - 4:    $T \leftarrow (\{u\}, \{(u, v)\})$
  - 5:    $A \leftarrow A \setminus \{v\}$
  - 6: **end while**
  - 7: Return  $T$
- 

#### 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. To do so, we first need to introduce some lemma.



**Figure 3.** Illustration of sink allocation over branches.

**Lemma 1.** Greedy step is loop free.

*Proof.* Greedy step reduces the distance between current node  $s$  and the closest set of  $k$  sinks  $T(s)$ . Loop cannot be created, since it is impossible to go back to a larger distance at a given node by repeated applications of greedy algorithm.  $\square$

**Theorem 2.** 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 demonstrating that Theorem 2 is true. **Initial step.** Theorem 2 is true for  $k = 1$ . When  $k = 1$ , the 1-MST computed by  $s$  running KanGuRou comes to finding  $CTS(s)$ , i.e. the closest sink to  $s$ . The greedy step of KanGuRou thus computes the progress provided by neighbor  $u$  of  $s$  as  $|sCTS(s) - uCTS(u)|$ . Recovery step initiated in node  $v$  will switch back to greedy step as soon as it has reached a node  $w$  such that  $|vCTS(v)| > |wCTS(w)|$ . Yet, KanGuRou comes to EEGDA [9], been proven to guarantee packet delivery as long as the underlying network is connected. Yet, Theorem 2 is true for  $k = 1$ . **Induction step.** Assuming that Theorem 2 is true for  $k = i - 1$ ,  $1 < i$ , we have to prove that Theorem 2 is true for  $k = i$ . When a node  $s$  runs KanGuRou, it may either duplicate and forward several times the message or just forward it once. Let us consider these two cases singly.

1. CASE 1: the message is sent on every branch originated at  $s$  in the  $k$ -MST, i.e. between 2 and

$k$  times. (On Fig. 1, the message is sent over 2 branches in  $s$ .) Since in KanGuRou:

- (i) when a sink is reached, it is removed from the list of available sinks,
- (ii) the set of available sinks is split over every branch such that an available sink is assigned to exactly one branch,
- (iii) the sum of the number of sinks to reach per branch is equal to  $k$ ,
- (iv) a message forwarding stops if and only if it has reached  $k$  sinks among sinks it has been assigned,

then node  $s$  runs independently KanGuRou with  $k$  such that  $1 \leq k \leq i - 1$ . for every branch. Thus, as Theorem 2 is true for  $k < i$ , the theorem is proven in this case.

2. CASE 2 (message is only forwarded), the message is forwarded in a repeated application of greedy and recovery phases. Greedy step is only applied if distance of current node  $u$  to the closest set of  $k$  actors/sinks  $W = ||T(s)||$  can be reduced (Lemma 1). The recovery step also has the same goal (reducing  $W$ ) after following a face. Gabriel graph preserves connectivity, and following very first face recover reduces distance  $W$  [6]. Distance  $W$  continues to decrease, and loop cannot be created until either delivery to a sink or a node on which a duplication of the message will be made. The delivery is guaranteed either to a sink in the set of available sinks assigned to the message or to a node which duplicates the message because its  $k$ -tree has several edges originating at itself since  $W$ , at each iteration, can always be reduced, until it eventually becomes 0.

- CASE 2.1 (a sink is reached): the sink will remove itself from the list of available sinks and runs KanGuRou with  $k = i - 1$ , which guarantees delivery to exactly  $i - 1$  sinks. Thus, in this case, KanGuRou eventually reaches exactly  $i$  sinks. Theorem 2 is true for  $k = i$ .
- CASE 2.2 (a node which will split the message is reached): the current node  $u$  initiates between 2 and  $i$  duplications and thus runs between 2 and  $i$  times KanGuRou with  $1 \leq k \leq i - 1$ . As shown for Case 1 of this proof, every branch guarantees delivery to the number of sinks they have been asked. Since a sink may be assigned to at most one branch, a same sink cannot be reached by several branches (and counted as it), thus, the overall

number of sinks reached is  $k = i$ . Therefore, in this case, KanGuRou reaches exactly  $i$  sinks.

Theorem 2 is true. □

## 5. Simulation Results

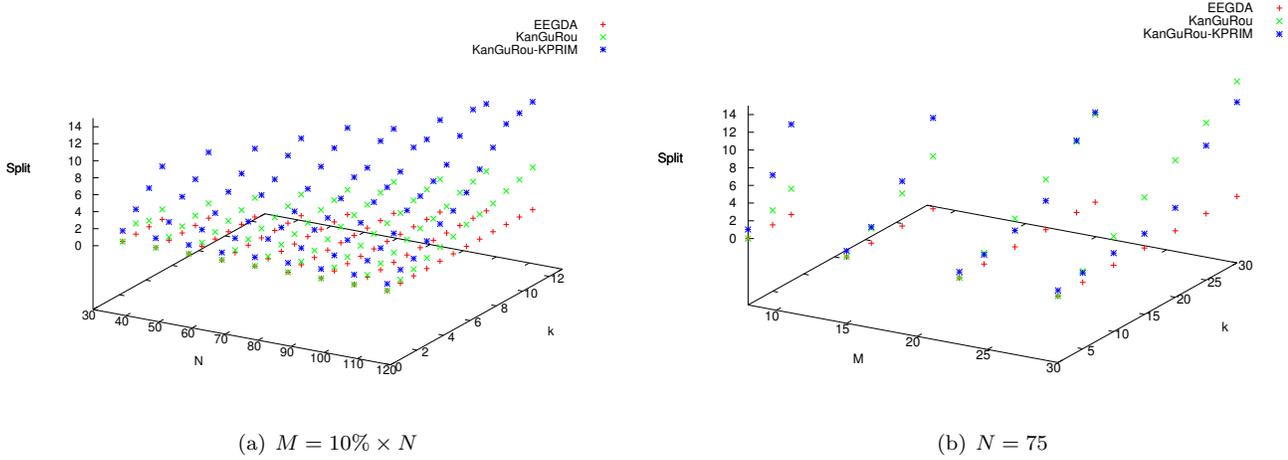
In this section, we evaluate the performances of KanGuRou under the WSN<sup>1</sup> simulator with an IEEE 802.15.4 MAC layer. As there is no comparable algorithm in the literature since KanGuRou is the first position-based algorithm from the literature, we compare the two variants KanGuRou and KanGuRou-kPrim to running  $k$  times the plain EEGDA anycast routing protocol [9] 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.

We first evaluate the behavior of different algorithms. Fig. 4 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. 4(a)). We can also note (Fig. 4(b)) that the number of duplications is not really impacted by the overall number of available sinks  $M$  in the network (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  $N$ . We simulate the performance of three algorithms (EEGDA, KanGuRou, KanGuRou-kPrim) for 100 times in terms of  $N$  as well as  $k$ , and calculate the average values of results. We generate in random a new distribution of  $N$  nodes for each simulation. Fig.5 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

<sup>1</sup>WSNet: <http://wsnet.gforge.inria.fr/>



**Figure 4.** Number of splits for each algorithm.  $M$  = number of sinks.

for  $k = 1$ , results are the same for all three algorithms since KanGuRou comes to EEGDA independently of the tree construction. 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. 4. A high number of splits implies shorter paths.

Figure 6 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. 7), 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 network. Obviously, when  $k$  increases for a given number of available sinks  $M$ , the path and the energy consumption increase since there

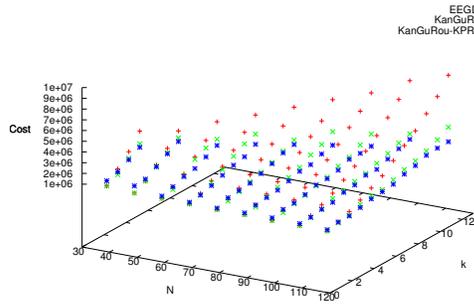
are more sinks to reach. Similarly, when the number of sinks to reach  $k$  is fixed and that the number of available sinks  $M$  increases, the path and the energy consumption decrease since algorithms have more choice among sinks and can join closer ones. An important feature is that results show that 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.

Figure 8 shows the impact of the number of sinks to reach for a fixed number of nodes and a constant side of sink set. It shows that for a low  $k$ , KanGuRou outperforms other solutions while it is not so efficient when  $k$  increases.

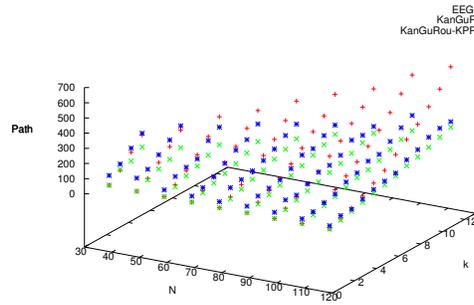
To sum up, the simulation results of different scenarios clearly show that (i) KanGuRou variants result in a significant gain of energy consumption and path length compared to the traditional algorithm EEGDA, (ii) depending of the percentage of sinks to be reached, one variant of KanGuRou performs better 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  $k > 30\% \times M$ ), KanGuRou-kPrim brings a significant gain compared to KanGuRou especially when  $M$  is important. This is also highlighted by Fig. 9 which has a closer look at this feature. Figure clearly shows that up to a given number of available sinks, KanGuRou-kPrim performs better than KanGuRou ( $M = 23$  on figure).

## 6. Conclusion and Future Work

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

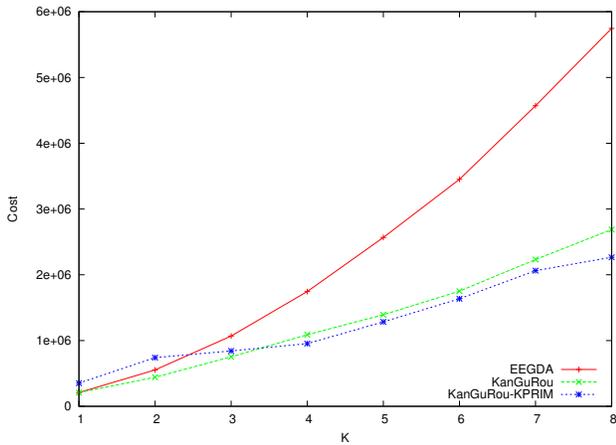


(a) Energy consumption

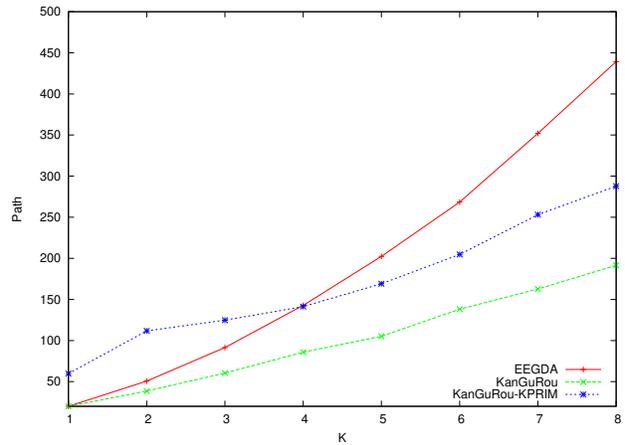


(b) Path length

**Figure 5.** Algorithms performances with regards to  $N$  and  $k$  for  $M = 10\% \times N$ .

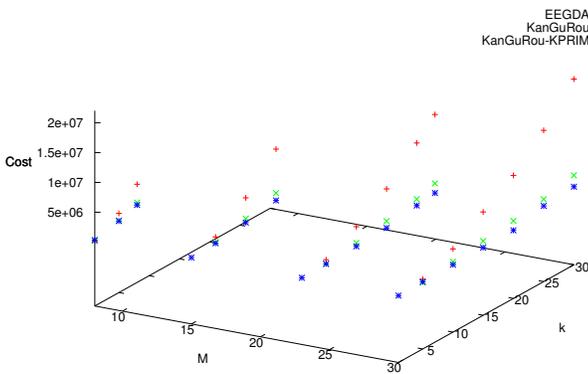


(a) Energy consumption

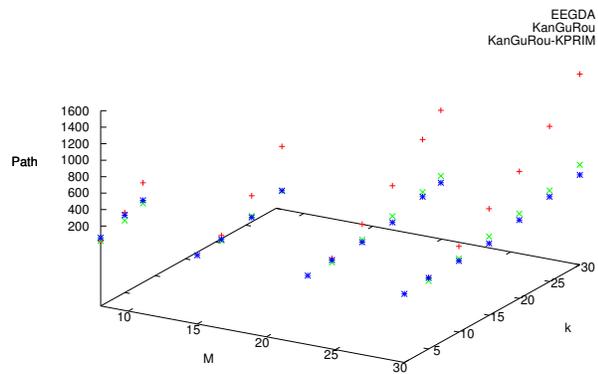


(b) Path length

**Figure 6.** Algorithms performance in terms of  $k$  over  $M = 8$  sinks among  $N = 75$  nodes.



(a) Energy consumption



(b) Path length

**Figure 7.** Algorithms performances with regards to  $M$  and  $k$  for  $N = 75$  nodes.

variants are proposed for the construction of the tree. KanGuRou performs well when the number of sinks

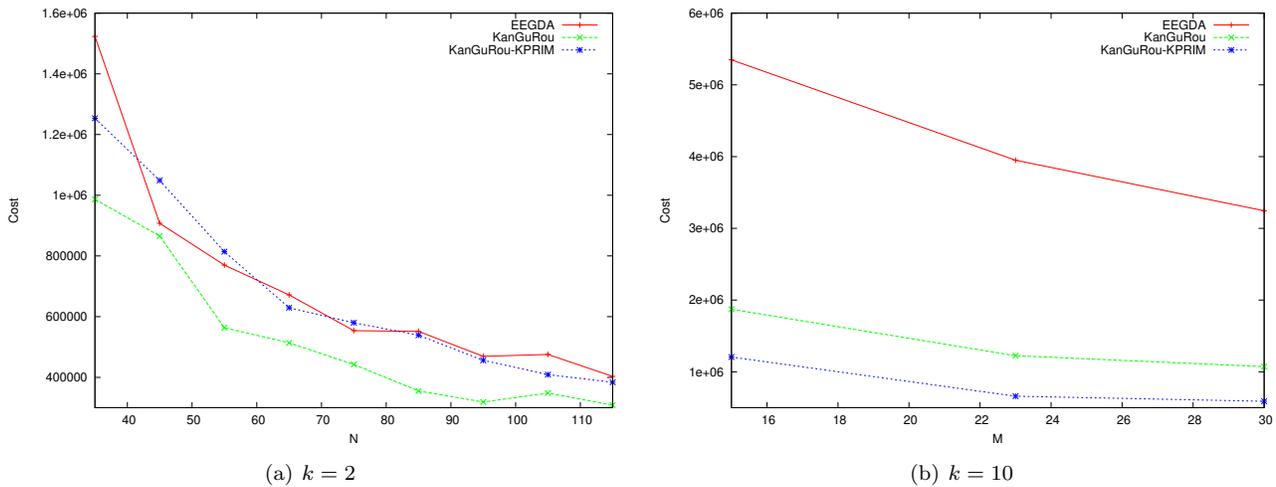


Figure 8. Algorithms performances for  $M = 20$  sinks and  $N = 75$  nodes.

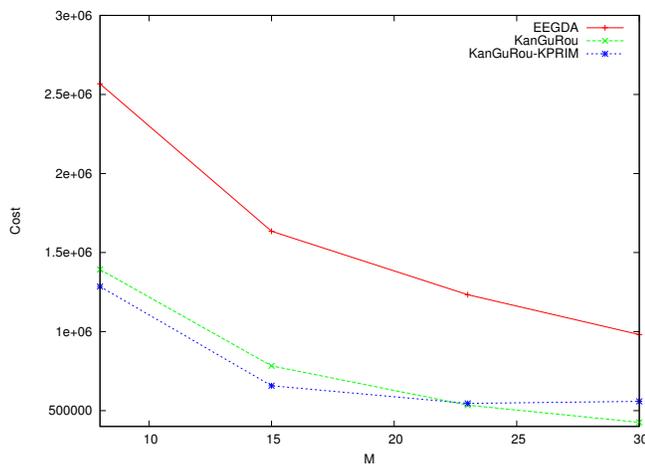


Figure 9. Algorithms performances for  $k = 5$  and  $N = 75$  nodes.

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] B. Awerbuch, A. Brinkmann and C. Scheideler. Anycasting and multicasting in adversarial systems. Dept. of Computer Science, Johns Hopkins University, 2002
- [2] 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.
- [3] S.C. Chen, C.R. Dow, S.K. Chen, J.H. Lin and S.F. Hwang. An efficient anycasting scheme in ad-hoc wireless networks. In *Consumer Com. and Networking Conf.*, 178:183, 2004.
- [4] 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.
- [5] H. Frey, F. Ingelrest, and D. Simplot-Ryl. Localized mst based multicast routing with energy-efficient guaranteed delivery in sensor networks. In *WOWMOM*, 2008.
- [6] H. Frey and I. Stojmenovic. On delivery guarantees of face and combined greedy-face routing in ad hoc and sensor networks. *ACM MOBICOM*, 2006
- [7] W. Hu, N. Bulusu and S. Jha. A Communication Paradigm for Hybrid Sensor/Actuator Networks. In *Journal of Wireless Information Networks*, 12(1):47–59, 2005.
- [8] T. Melodia, D. Pompili, V. C. Gungor and I.F. Akyildiz, A Distributed coordination Framework for Wireless Sensor and Actor Networks. In *ACM Mobihoc*, 2005,
- [9] 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.
- [10] R.C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.
- [11] V. Rodoplu and T. Meng. Minimizing energy mobile wireless networks. *IEEE JSAC*, 17:1333–1347, 1999.
- [12] W. Wang, XY Li, and O. Frieder. k-anycast game in selfish networks. In *ICCCN*, 2004.
- [13] X. Wang. Analysis and design of a k-anycast communication model in ipv6. *Comput. Commun.*, 31:2071–2077, June 2008.
- [14] C.-J. Wu, R.-H. Hwang and J.-M. Ho. A scalable overlay framework for Internet anycasting service. In *Symp. on Applied Computing (SAC)*, 2007.
- [15] B. Wu and J. Wu. k-anycast routing schemes for mobile ad hoc networks. In *IPDPS*, 2006.

- 
- [16] X. Xu, Y-L Gu, J. Du, and H.-y. Qian. A distributed k-anycast routing protocol based on mobile agents. In *WiCOM*, 2009.