Partitioning Algorithm for a Resource-Constrained Robotic Skin Sensor Network

Mikhail Vilgelm*, Gabriel Neamtu*, Philipp Mittendorfer†, Gordon Cheng†, Wolfgang Kellerer*
*Chair of Communication Networks, †Institute for Cognitive Systems
Technical University of Munich, Germany
Email: {mikhail.vilgelm, g.neamtu, philipp.mittendorfer, gordon, wolfgang.kellerer}@tum.de

***Abstract*—Sensor applications in robotics, such as artificial robotic skin, feature the shrinking size of nodes, and increasing scale of sensor networks. Together with a requirement for low complexity hardware for individual nodes, this leads to network optimization challenges. In this paper, we introduce a heuristic-based network partitioning algorithm for routing in a wired robotic skin sensor network, which aims at minimizing end-to-end latency and decreasing packet loss. Our algorithm is partitioning the network in sub-trees while balancing load on the root node connections. We benchmark its load balancing and evaluate its end-to-end latency and packet drops via simulation. The evaluation results show the superiority of our algorithm with respect to the state-of-the-art solutions.**

I. Introduction

Sensor network applications in robotic systems often require low end-to-end (E2E) latency and minimum packet loss in order to ensure control stability, satisfying user experience, and failure free operation [1]. Hence, establishing optimal data delivery routes is an important component for such systems. In this paper, we tackle the sensor network routing on an example of Cellul.A.R.Skin [2], [3]: artificial robotic skin, developed in order to introduce the sense of touch to machines.

Cellul.A.R.Skin consists of individual skin cells, interconnected to form a meshed sensor network. Compared to traditional computer networks, Cellul.A.R.Skin features large number of homogeneous nodes ($\approx 10^2$-10^3), each of them with limited processing power and memory. Thus, gathering the sensing data from all nodes requires a centralized routing algorithm, which is optimized for avoiding bottlenecks in the network or minimizing their negative effects.

The related work for our routing problem consists of two parts: 1) robotic skin networking solutions; and 2) general delay-optimizing sensor routing algorithms. As a relatively new research area, robotic skin networking does not have standard or well-established solutions. Hence, every existing implementation is very case-specific: some are not concerned with connectivity at all, while others [4], [5] do not need routing as no link redundancy is present. Both modularity and link redundancy are present in [6], however, as the data is meant to be used on the skin level, no scalability problem arises. For sensor network routing, the majority of topics in literature are aimed at wireless sensor networks, thus, research is focusing on wireless medium aspects, such as interference and resource management [7]. Algorithms, such as Capacitated Minimum Spanning Tree (CMST) [8] or Shortest Path Trees (SPTs),

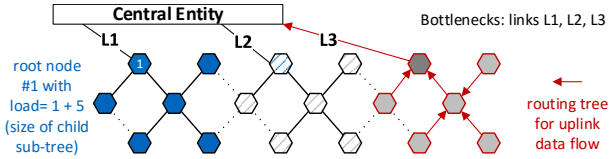

Fig. 1. Top-level balanced tree with three root connections.

could be applied in our case. However, no straightforward cost metrics for them are matching our use case. Other solutions, such as graph partitioning algorithms [9], are not applicable since a tree-based network is required for embedded routing simplification. Closest to the herein presented solution is a node-centric load balancing algorithm in [10], designed for wireless sensor networks.

In this paper, we propose a centralized algorithm for partitioning a skin sensor network (described in Sec. II). It decreases the E2E delay through balancing the load on the root node connections (see Fig. 1), and minimizes packet loss due to buffer overflow. The algorithm (refer to Sec. III) is designed for deployment in large scale sensor networks with limited processing power and buffer space on the individual nodes (the scarcity of buffer is even more relevant in case the cells are implemented as an Application-Specific Integrated Circuit). We benchmark the load balancing capabilities of the algorithm and its run times. Then, we simulate the Cellul.A.R.Skin network and show the delay and packet loss during network operation (refer to Sec. IV). A comparison to the currently deployed Cellul.A.R.Skin routing, as well as to the SPT solution is shown.

II. Robotic Skin Sensor Network

Example network topology of a Cellul.A.R.Skin is depicted in Fig. 1. Multiple cells are connected together in a grid-like network topology with up to four neighbors. A neighbor can be either the Central Entity (CE), or another cell. Every node in the network is responsible for forwarding its own data, as well as the data from its child sub-tree. Since all the cells generate equal amount of traffic, we define the **load** on a node as the size of child sub-tree + 1 (for own data). Fig. 2 shows the structure of an individual cell with its temperature, normal force, proximity and accelerometer sensors and additional infrastructure. Every cell has four port modules that can run with a speed of up to 4Mbit/s. The micro-controller periodically collects data from all sensors, filters and

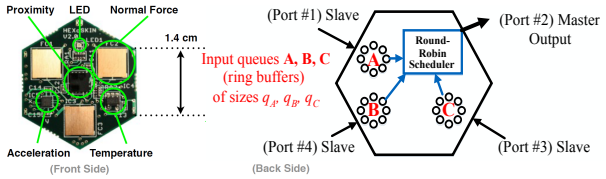

Fig. 2. Modular skin cell: multiple sensor modalities (front side) and local infrastructure (back side); with three slave ports, one master port, and a Round-Robin scheduler regulating output order [3].

encapsulates the preprocessed data into a packet. This packet is forwarded up in the hierarchy to the parent via a **master port**, predefined by the routing path. Eventually, all data arrives on one or multiple **root nodes**, which represent a connection to the CE, where the data can be processed and analyzed.

In the context of sensor networks, such uplink traffic is often referred to as convergecast (as an opposite of broadcast). There are several measures of efficiency for convergecast data delivery through the network. One of them is the E2E **delay (average and maximum)**, which has to be low for real time control. Second measure is the **packet drop ratio**, which primarily occur due to the insufficient buffer space. Some node connections are redundant, hence, in order to establish loop-free routes, the topology has to be partitioned into tree structures, each rooted at one of the root nodes.

The **current solution** uses Breadth-First-Search (BFS) to construct a routing tree. CE is injecting a token into the network, and every node, upon reception of the token, sets the reception port as master port, and re-sends the token further via all child ports. Subsequent token reception on this node is ignored. While this approach is simple to implement, it is highly dependent on the choice of the root nodes: some sub-trees are growing faster than others, thus, the uplink traffic on the root nodes is not evenly distributed.

In the next section we explain our solution, which (1) minimizes the latency and (2) minimizes the packet loss in the network. Since computational resources of individual cells are constrained, and large signaling overhead is to be avoided, we have chosen to implement a centralized algorithm: it is executed on a CE, and its result (which master port to select) is communicated individually to each cell.

III. Network Partitioning Algorithm

The three step work-flow of the algorithm is presented in Alg. 1. An *Initial Solution (IS)*, combined with a *Node Exchange (NE)* step, aims at decreasing the delay, while a *Buffer Overflow Avoidance (BOA)* step is applied to all nodes for ensuring load distribution to avoid packet loss.

Algorithm 1 Graph Balancing algorithm

```
1: Compute Initial Solution (IS)                    (Subsec. III-A1)
2: Perform Node Exchange (NE)                       (Subsec. III-A2)
3: for all nodes in the network, from root to leaves
4:     Apply Buffer Overflow Avoidance (BOA)        (Subsec. III-B1)
```

A. Delay minimization

There are four components for the end-to-end delay: propagation, transmission, packet processing and queuing delay. Propagation and processing delay can be safely ignored in our case, as they are much smaller than transmission delay. The transmission delay is the sending time t_s and depends on the packet size and the link bandwidth. Here, 20 bytes packets + 5 bytes spacing are used (with +2 bits per byte: start bit and stop bit), sent via a 4 Mbps interface. The queuing delay is due to the Round-Robin scheduling routine in each cell.

We exemplify our average delay estimations with a network of N nodes with r root connections, each root serving a sub-tree with the loads $n_1 \leq n_2 \leq \cdots \leq n_r$. Assume that all nodes generating a packet at time $t = 0$, and no new packet is generated until all previous are delivered. Hence, each node sends a packet at an interval equal to the sending time t_s until it runs out of packets to send. In this case, the root node connection to the central entity becomes the bottleneck. All root nodes' connections are fully utilized until the node servicing n_1 child cells finishes its last packet. After that, for the time $(n_2-n_1)t_s$ only $(r-1)$ links are utilized, and so forth. Considering that the end-to-end delay for every packet is the time of arrival to the sink, we obtain the following estimation of the average delay:

$$d_{avg} = (\sum_{j=1}^{r}\sum_{i=1}^{n_j} i \cdot t_s)/N = \frac{t_s}{2}\sum_{j=1}^{r} n_j(n_j+1)/\sum_{j=1}^{r} n_j \quad (1)$$

Thus, the average delay is determined by the sending time and the root load distribution. It is straightforward to see that d_{avg} is minimal when $n_1 = n_2 = \cdots = n_r$, or the top level is balanced. The root-level balancing, thus, is the target of our algorithm. It is achieved during the first two steps of the algorithm, namely *IS* and *NE*.

Algorithm 2 Initial Solution

```
1: Sets of processed nodes P = ∅, and of subtrees ST = ∅
2: for all root nodes r_i do
3:     initialize new subtree ST_i ← r_i
4:     ST ← ST_i
5: F ← neighbor nodes of r_i ∀ root nodes r_i
6: while F ≠ ∅ do
7:     Initialize a heap C ← F ordered by increasing node degree
8:     F = ∅
9:     while C ≠ ∅ do
10:        nextNode = PopFrom(C) (Rule A applied)
11:        if |ST_i| = min_{∀ST_i∈ST} |ST_i| and nextNode has a link to ST_i then
12:            |ST_i| ← nextNode (Rule B applied)
13:        I - set of neighbors of nextNode
14:        for i ∈ I do
15:            if i ∉ P then
16:                F ← F + {i}
17:        P ← P + {nextNode}
```

1) Initial Solution (IS): The first step is used to explore the network and create an initial tree solution. The step is similar to BFS with additional custom tiebreakers (see Alg. 2). For the selection of the next node to proceed the following rules are applied. **Rule A**: a node with the smallest degree (number of links) is processed before a node with a larger degree. Nodes with under four connections are situated at the edge of the skin patch and have little degree of freedom. Therefore, we process them first. **Rule B**: in case of an equal node degree, a node

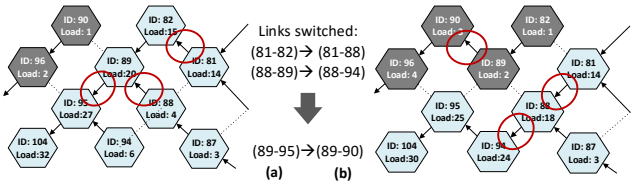

Fig. 3. Sample run of the second NE subroutine. Before (a) and after (b).

with fewer sub-tree neighbors is processed before a node with more sub-tree neighbors. The latter nodes have the greatest degree of freedom as they can connect to any sub-tree.

2) Node Exchange (NE): Because the IS top level load balance is usually sub-optimal, we propose the additional NE step. It iteratively selects the root-level sub-tree pair with the highest load difference. For the chosen pair, NE attempts to even out the difference by passing nodes from the larger sub-tree to the smaller one (refer to Alg. 3). The step consists of two subroutines outlined below.

Algorithm 3 Node Exchange step

do
  $balancing \leftarrow$ **false**
  Heap $\mathcal{P}$ of adjacent sub-tree pairs
  Ordered by decreasing node difference
  do
    $\{ST_1, ST_2\} \leftarrow PopFrom\,(\mathcal{P})$
    $\mathcal{P} \leftarrow (\mathcal{P} - \{\{ST_1, ST_2\}\})$
    $D_c \leftarrow ComputeNodeDifference(ST_1, ST_2)$
    $success = FirstNESubroutine(ST_1, ST_2)$
    if not $success$ **then**
      $SecondNESubroutine(ST_1, ST_2)$
    $D_n \leftarrow ComputeNodeDifference(ST_1, ST_2)$
    if $D_n < D_c$ **then**
      $balancing \leftarrow$ **true**
  while $\mathcal{P} \neq \emptyset$ **and** $balancing$ is **false**
while $balancing$ is **true**

a) First NE subroutine: The approach is to transfer nodes together with their child sub-trees. To determine the new load difference D_n when passing a node x from one sub-tree to another we use following equation: $D_n = |D_c - 2L\,(x)\,|$, where D_c is the current difference, $|\cdot|$ is the modulo operator and $L\,(x)$ is the load of node x. The first NE subroutine goes through nodes in the larger sub-tree neighboring the smaller sub-tree and finds minimum D_n. If $\min D_n \geq D_c$ the step fails and the second subroutine is attempted. Otherwise, the selected node is transfered to the smaller sub-tree.

b) Second NE subroutine: While the first approach works most of the time, there are situations in which all available nodes to be exchanged have too many children and cannot be passed over. Such a situation is depicted in Fig. 3a. The approach is to reduce the load on the nodes neighboring the smaller subtree, in order to make them eligible for exchange as in first subroutine. For each candidate node x_i the algorithm traverses its sub-tree from parent to children. For each node x_c below x_i it searches for replacement parents such that the routing path does not include x_i. By replacing the master port to point to this new root, $L\,(x_i)$ is reduced. If x_c has a valid exchange parent, the graph traversal for that branch stops. A replacement parent link is only valid if the two following conditions hold true: (1) the node it is pointing at does not forward its data through x_i, meaning that the new parent must not be below x_i to prevent a forwarding loop; and (2) the node does not belong to another sub-tree.

If the algorithm manages to reduce the load on any cell eligible for exchange such that the new difference is reduced, the transfer is performed and the NE step exits. In Fig. 3 a snapshot of the second NE subroutine is depicted. The pseudocode for it is presented in Alg. 4.

Algorithm 4 Second NE subroutine

Set of nodes eligible for exchange X_e
for $x_i \in X_e$ **do**
  Set of replacement parent mappings M_i
  $C \leftarrow$ neighboring children of x_i
  while $C \neq \emptyset$ **do**
    $x \leftarrow FirstElement(C)$
    $C \leftarrow C - \{x\}$
    $r_n \leftarrow ReplacementRoots(x)$
    if $r_n \neq \emptyset$ **then**
      $M_i \leftarrow M_r + \{\{x, r_n\}\}$
    else
      $C \leftarrow C+$ all child nodes of x
  Record new load difference:
  $D_n^{x_i} \leftarrow |D_c - 2NewPotentialLoad(x_i)|$
$x_b \leftarrow \min_{\forall x_r \in X_e} D_n^{x_i}$ (choose minimum load difference)
if $D_n^{x_b} < D_c$ (difference has been reduced) **then**
  Perform all recorded parent changes in M_b
  Exchange node x_b

B. Packet loss minimization

The objective of this step is to reduce the packet loss by avoiding buffer overflow on the nodes. The detection mechanism for buffer overflow is illustrated in network calculus concepts. We consider the node with three children, and hence, three active input queues A, B, C (assume $A \geq B \geq C$), where max queue size correspond to length of the child sub-tree connected to it. There is exactly one output port, and the nodes are scheduled in a Round-Robin fashion, resulting in the arrival and serving curves for the queue A as illustrated in Fig. 4. Note that the buffer overflow occurs in case the backlog of queue A, b_A, grows larger than the buffer size q_A. As it can be seen from the illustration, there are exactly three cases for calculating the maximum backlog: (1) $t_Q^A < t_s^C$: queue A arrivals end before queue C is fully served, (2) $t_s^C \leq t_Q^A < t_s^B$: A arrivals end after C is served, but before B is served, and (3) $t_Q^A \geq t_s^B$: A arrivals end after B is fully served. The conditions (1)-(3) can be expressed in terms of the queue sizes, and for every of this three cases, maximum backlog $b_A^{\max}$ can be computed as:

$$b_A^{\max} = \begin{cases} 2A/3, & \text{if } A < 3C \\ (C+A)/2, & \text{if } 3C \leq A < C + 2B \\ C + B, & \text{if } A \geq C + 2B \end{cases} \quad (2)$$

The computation is straightforward from the illustration in Fig. 4. Hence, the node risks a buffer overflow if the maximum backlog, corresponding to the longest queue is larger than the

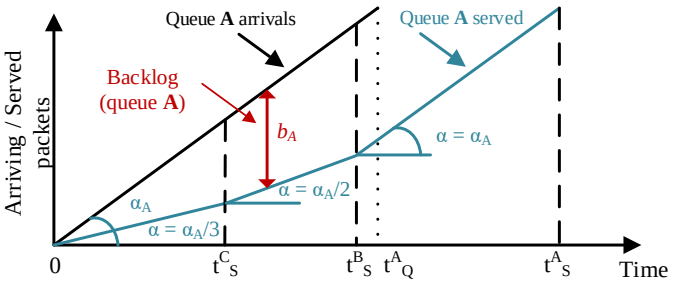

Fig. 4. Arrival and serving curves for the queue A, corresponding to the longest sub-tree of a node. During the interval $[0, t_s^A)$ all three queues are active, hence the serving rate α is trice lower than arriving rate α_c. In $[t_s^C, t_s^B)$, queue C is already empty, hence $\alpha = \alpha_c/2$. Only queue A is served during $[t_s^B, t_s^A]$. Buffer overflow occurs if the backlog (difference between the arrival and sending curve) grows larger than the buffer size.

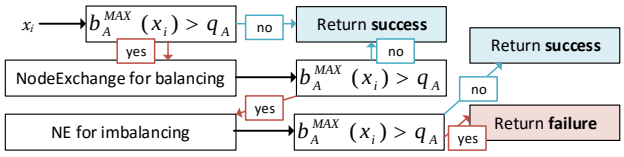

Fig. 5. Decision tree for the Buffer Overflow Avoidance step.

buffer size: $b_A^{\max} > q_A$. By analogy, for the case with two child sub-trees (with queues B and A, A>B), we obtain:

$$b_A^{\max} = \begin{cases} A/2, & \text{if } A < 2B \\ B, & \text{if } A \geq 2B \end{cases} \tag{3}$$

1) Buffer Overflow Avoidance (BOA): Next, we use Eqns. (2) and (3) in the Buffer Overflow Avoidance step of the algorithm (see Fig. 5). The intuition behind *BOA* is the following: if the node risks a buffer overflow, we use the Node Exchange routines developed above in order to create a load distribution, for which the maximum backlog for longest queue is sustained less than the buffer size. Note that the balanced load distribution (optimizing delay) is not necessary optimal for *BOA* (this does not contradict to the balancing steps, since they are equalizing only the root-level connections from root nodes to the central instance). The procedure is applied iteratively for all nodes starting from roots to leaves.

IV. Performance Evaluation

This section presents evaluation of the algorithm (w.r.t. run times and the load balancing performance), and the results of network simulation that includes delay and packet drop ratio. For the evaluation, two topology types are considered: rectangular and cylinder skin patches. The choice of topology is justified by the standard use case of the Cellul.A.R.Skin: covering the parts of the robots. To show the robustness against the root node placement, root nodes are chosen randomly for every run. We compare our algorithm against the output of the current routing implementation as described in Sec. II, referred to as "Current routing" and Shortest-Path-Tree (SPT) routing.

Run Times. The test is performed on a development PC: Intel Core i7 with four cores, 8Gb RAM notebook. Fig. 6 shows the dependency between execution time and the number of nodes in the network, for rectangular topologies. Run time measurements are performed for different root node placements, chosen randomly. As we can see, run times within a second were obtained for networks of 3600 nodes, thus, making the algorithm feasible to use for envisioned Cellul.A.R.Skin application in practice. It is observed that the run time is influenced by the network topology and the root placement. A cylinder network performed better since worst case root placement scenarios, such as all roots in one corner, are not possible for cylinders. We acknowledge that this run time evaluation is only valid for our particular use case, where the node degree is architecturally bounded by four. As a future work, we plan a thorough complexity analysis for a generic meshed network, with higher possible node degrees. Also note that the both current solution and SPT perform faster than our algorithm in cases.

Balance Factor. As a performance metric, we use the balance factor defined by Jain's Fairness Index [11]:

$$\beta = \frac{\left(\sum_{i=1}^{k} L\left(x_i\right)\right)^2}{k \sum_{i=1}^{k} L(x_i)^2}. \tag{4}$$

The nodes x_i are the k root nodes and $L(x_i)$ represents the load of node x_i (size of the sub-trees). The balance factor β takes values from $(0, 1]$, and is equal to one if loads are equal. Fig. 7 shows the balance factor for two setups: (a) for three rectangular topologies of increasing size; (b) dependency β from the number of root nodes. Fig. 7a), it is observed that the developed algorithm achieves perfect balancing in all cases, and the balance factor is independent from the network sizes. This is due to the common rectangular shape and the constant number of root nodes. Fig. 7b) shows how efficiently the algorithm uses added root links. As we can see, as the number of roots increases, it is getting more difficult to ensure perfect balancing in all cases. However, our algorithm is ensuring a close to perfect balancing significantly outperforming comparative solutions.

Network Simulation. In order to estimate the delay and packet loss, Cellul.A.R.Skin network is modeled and simulated

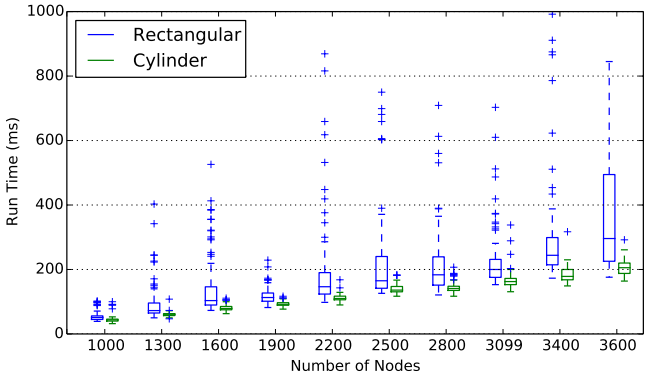

Fig. 6. Run times vs. number of nodes in the network for rectangular and cylinder networks. Boxplot for 100 runs with 4 randomly selected root nodes.

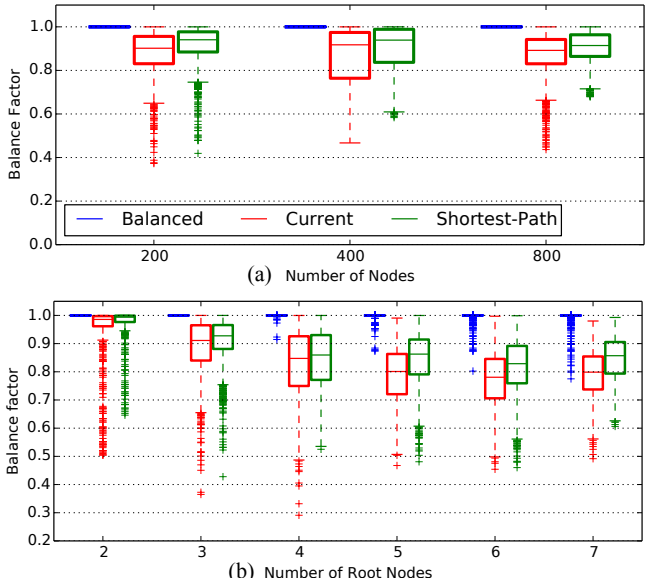

Fig. 7. Balance Factor vs. (a) network size and (b) number of root nodes (random placement), boxplots for 3000 runs. For (a) two random root nodes are used; for (b) 200 nodes topology.

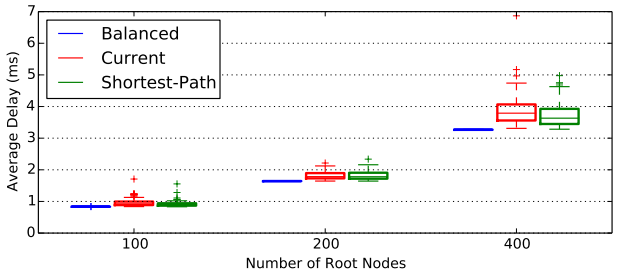

Fig. 8. Average delay vs. network size in the no packet loss scenario: 5 root nodes, 2000 sampling periods per run, and boxplots for 100 runs.

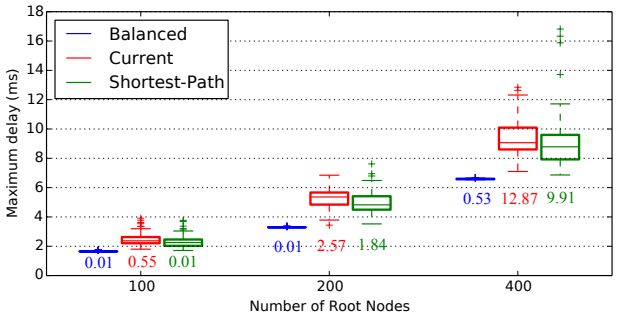

Fig. 9. Maximum delay and drop ratio vs. network size (5 root nodes, 2000 sampling periods per run, boxplots for 100 runs). The drop ratio percentage is displayed under each box-plot. Buffer size is 20, 30, 40 packets for 100, 200 and 400 nodes networks respectively.

using event-based Omnet++ framework [12]. The routes as an output of the algorithm were preset statically, no protocol related aspects were in consideration. Here, we have two base modules: the node and the central entity. The central entity is the final destination for all packets and is used to compute the delay for each received packet. The node module simulates the behavior of a Cellul.A.R.Skin cell, with four input and four output ports, as well as four internal packet queues: one for each child input port and one to hold the self-generated packets. Round-Robin scheduler is regulating the sending order. In order to obtain a fair delay comparison, we first consider the buffer size to be large enough to ensure no overflow. From Fig. 8, it is observed that if no packet loss occurs, our algorithm achieves lowest average delay.

In the second setup, we intentionally limit the buffer size, such that buffer overflows are expected to occur for all algorithms, and measure the ratio of dropped packets along with the maximum delay. Fig. 9 shows that our solution has ten times less packet drops compared to the other two algorithms while still maintaining a competitive delay.

V. Conclusions

In this paper, we introduced a network partitioning algorithm for routing in a meshed skin network consisting of hundreds to thousands of nodes with limited processing power and memory. It reduces the E2E delay and minimizes the packet loss. Delay minimization has been achieved via root-level load balancing, while buffer overflow is avoided considering the nodes' maximum backlog constraints. Practical evaluations and an event-based simulation show that our algorithm achieves significantly lower delay values and low packet drop ratios than competitive solutions. Practically, this allows a higher data reporting frequency in a Cellul.A.R.Skin for a full network of cells without risking loss of packets. Although the algorithm is tailored to the deployment with an artificial skin, with the appropriate adjustments of link and node costs, it can be applied in any large scale meshed sensor network scenario – wired or wireless.

References

[1] W. jong Kim, K. Ji, and A. Srivastava, "Network-based control with real-time prediction of delayed/lost sensor data," *Control Systems Technology, IEEE Transactions on*, vol. 14, no. 1, pp. 182–185, Jan 2006.

[2] P. Mittendorfer and G. Cheng, "Self-organizing sensory-motor map for low-level touch reactions," in *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, Oct 2011, pp. 59–66.

[3] P. Mittendorfer, E. Yoshida, and G. Cheng, "Realizing whole-body tactile interactions with a self-organizing, multi-modal artificial skin on a humanoid robot," *Advanced Robotics*, vol. 29, no. 1, pp. 51–67, 2015.

[4] T. Hoshi, A. Okada, Y. Makino, and H. Shinoda, "A Whole Body Artificial Skin Based on Cell-Bridge Networking System," in *Proc. 3rd Int. Conference on Networked Sensing Systems*, 2006, pp. 55–60.

[5] A. Buchan, J. Bachrach, and R. S. Fearing, "Towards a minimal architecture for a printable, modular, and robust sensing skin," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 33–38.

[6] D. Hughes and N. Correll, "A soft, amorphous skin that can sense and localize textures," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014, pp. 1844–1851.

[7] O. Incel, A. Ghosh, B. Krishnamachari, and K. Chintalapudi, "Fast Data Collection in Tree-Based Wireless Sensor Networks," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 1, pp. 86–99, Jan 2012.

[8] Y. Lee and M. Atiquzzaman, "Exact algorithm for delay-constrained capacitated minimum spanning tree network," *Communications, IET*, vol. 1, no. 6, pp. 1238–1247, Dec 2007.

[9] K. Andreev and H. Racke, "Balanced graph partitioning," *Theory of Computing Systems*, vol. 39, no. 6, pp. 929–939, 2006.

[10] H. Dai and R. Han, "A node-centric load balancing algorithm for wireless sensor networks," in *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, vol. 1, Dec 2003, pp. 548–552 Vol.1.

[11] R. Jain *et al.*, "The Art of Computer Systems Performance Analysis: Techniques," 2010.

[12] A. Varga *et al.*, "The OMNeT++ discrete event simulation system," in *Proceedings of the European simulation multiconference (ESM2001)*, vol. 9, no. S 185. sn, 2001, p. 65.