A Robust and Efficient Dynamic Network Protocol for a large-scale artificial robotic skin

Christian Bader, Florian Bergner and Gordon Cheng

***Abstract*— Artificial robotic skins are continuously in contact with their environment, and therefore highly rely on proper connections in their skin cells' network. With a static network protocol approach, the affected skin area is unusable after a connection failure. Therefore, we developed a dynamic network protocol for large-scale artificial robotic skins, which re-routes the network upon connection failures to keep the whole skin in operation. Furthermore, the protocol balances the load for driving larger skins without packet loss. For verification, we validated the protocol on a large artificial robot skin we have developed and analyzed its performance with a skin network consisting of up to 204 cells. The failure recovery of the protocol converges in at most 50ms. We showed that the balancing method achieves a packet loss reduction of over 30% compared to the previously used protocol.**

I. Introduction

A. Motivation

Modular artificial robotic skins, like RoboSkin [1] or the skin for robots [2], are designed to extend a robot with a sense of touch and consist of a large number of homogeneous nodes ($\approx 10^2 - 10^3$) covering the whole body of a robot. Every skin cell is sensing the environment and forwards sensor data to a PC. With a growing number of nodes in a network, the probability of a connection failure increases, possibly leading to data loss of the affected skin area. If such a connection failure is not detected and resolved, the resulting data loss may lead to a safety deficit and loss of the deterministic behavior of the robot, and thus possibly causes damages to objects or even humans.

To the best of our knowledge, all modular robotic skin networks use a static network protocol, which configures the routing either off-line or once during start-up, and therefore have to be restarted after a broken connection. For instance, the skin of robots [2] takes around two seconds during start-up/re-start time, and the skin is unusable during this period. Such a long dead-time can be infeasible for critical robotic applications. Therefore, there is the need for a dynamic routing protocol, adapting on topology changes caused by broken connections during runtime, and thereby lowering any resulting dead time.

Furthermore, as the sensor data is used as input for control algorithms, the network requires low end-to-end latency and minimum packet loss to guarantee operations without failures. Due to the large number of skin cells in the modular artificial robotic skin and the limited processing power and memory resources of each cell, the nodes directly connected to the interface of the PC are processing close to their limits.

All authors are with the Institute for Cognitive Systems (ICS), Technische Universitt Mnchen, Munich, Germany, see http://www.ics.ei.tum.de

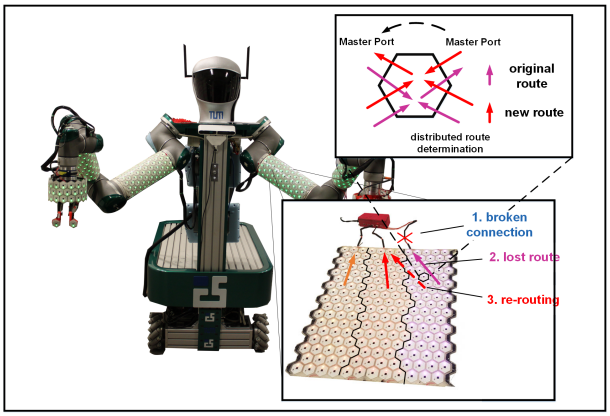


Fig. 1: Dynamic routing at an artificial robot skin. The network protocol constructs its routes in a distributed manner upon startup and reacts on connection failures. The image shows a single patch of a robot consisting of 204 skin cells. In case of a broken connection to the interface, it reroutes the network until every cell can reach the interface again.

If the network is unbalanced with respect to the root nodes, it may lead to additional packet loss or require a lower data gathering frequency, which reduces the reaction time.

B. Related Work

Several works exist on dynamic networking, mostly targeting wireless sensor or ad-hoc networks. Artificial robotic skins are flat based multi-hop networks, i.e. data from sources has to be forwarded to the next hop neighbor to reach the sink. Therefore, we first focus on this type of protocol. Many dynamic network solutions relying on shortest paths are out of three different types: i) Link-State; ii) Distance-Vector; or iii) Source Routing protocols.

In Link-State protocols, like OSPF [3], every node knows about the whole state of the network and resolves the shortest path problem with the Dijkstra algorithm. This requires large memory capacity and high computational power, both limited in artificial robotic skin networks. In contrast, Distance-Vector protocols use the distributed bellman ford algorithm to determine routes, which is quite efficient in terms of memory and processing requirements. The DSDV [4] is a proactive Distance-Vector protocol with every node periodically broadcasting its routing tables to its direct neighbors. Broken links are detected by the link layer and it guarantees loop freedom by the use of sequence numbers. An improved version of DSDV is shown by Chakeres et al. [5] called AODV. Here, routes are only determined if necessary by route requests via broadcast. Another Distance-Vector protocol is SGF [6]. Instead of only considering shortest path as a parameter for the bellman-ford algorithm, it also considers physical distance to sink and link-quality. In contrast to the former

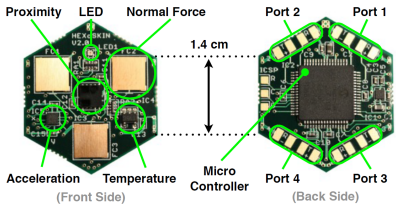


Fig. 2: A single skin cell, including micro-controller and peripherals [12].

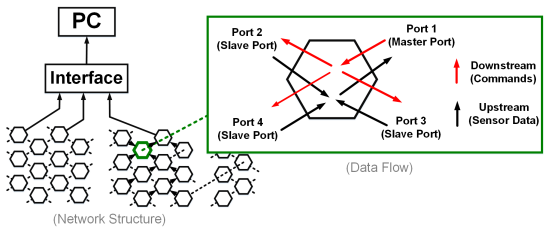


Fig. 3: Example network structure and its data flow with upstream and downstream direction. Upstream is the way from skin cells to the PC using the Master Port, downstream the opposite direction using Slave Ports.

protocols, they don't explain in [6] how they provide loop-freedom. Source Routing protocols like DSR, as described in [7], store routes in packets, which is insufficient for large-scale networks like artificial skins.

Dynamic routing protocols always require some bandwidth for route updates. To reduce the impact of this issue, overlying load balancing algorithms can increase the data-throughput. Some load balancing solutions like [8], [9], [10] try to avoid temporally occurring congestion, possibly leading to a frequent change of the routes. This may lead to packet-delay variations, which are unacceptable for artificial robotic skins due to control algorithms for the robot. One interesting load balancing solution is presented by Puccinelli *et. al* in [11]. Upon distance to sink and bottleneck link quality, they also use bottleneck load as routing criteria. Their approach could also be used for artificial skins to improve the packet throughput.

C. Our Approach

This work presents a robust network protocol for modular artificial robotic skins, running with an overlying balancing algorithm to increase the possible sensor data frequency. The protocol is a distance vector protocol with bottleneck load balancing. The design is implemented on a robot arm and verified on different skin cell patches and combinations.

II. System Description

A. The Robot Skin and skin cell network

The Robot Skin developed at the Institute for Cognitive Systems (ICS) is a modularized artificial skin for robotic systems [2], providing them with the sense of touch. It is built up of hexagon-shaped nodes (Skin Cells), each consisting of several peripheral modules like sensors for data gathering and a micro-controller for processing, as shown in Fig. 2.

These cells can be connected to up to four neighbors and are ordered in a honeycombed structure as patches with the size of up to hundred cells. Several patches may further be connected together building a large-scale network. Fig. 3 shows an example network structure built out of three patches.

In general, data is forwarded in two different directions. All data gathered by skin cells is forwarded via interface to the PC. This data flow is called *upstream* and its inverse is called *downstream*. If a port forwards data in upstream direction, it is called a *Master Port*. *Slave Ports* are the ports responsible for downstream data flow, for example, to broadcast commands.

Skin cells can only send packets to its directly connected neighbors. Therefore, most packets need to be forwarded multiple times until they reach the sink, which is also called a multi-hop network. This requires a method for determining routes to the sink.

In the previous network protocol solution [2] builds routes once upon startup via a breadth search first algorithm. The resulting routes are static and have some drawbacks:

1) If a connection breaks, the network may need to be restarted. This dead-time of up to a few seconds is unsatisfying for any robot control algorithm. 2) Differences in message forwarding delays lead to a non-deterministic routing, and 3) The network is not balanced with respect to the root nodes, reducing the maximal data throughput.

Therefore, there is the need for a new protocol, which dynamically resolves lost routes, is as deterministic as possible and balances the network with respect to the bottleneck load.

B. Dynamic network protocol

The new dynamic network protocol is a distributed protocol adapting its routes during runtime upon topology changes. Additionally, it uses information from the sink to increase the data throughput and reduce the latency via load balancing.

Three tasks are running in parallel in the dynamic network: finding of potential paths to the skin driver, recognition of topology changes and determination of the final routes to the sink, as shown in Fig. 4. The information necessary for routing is shared between neighbors by periodically exchanged beacon packets.

The network protocol approach only slightly differs from conventional distance-vector protocols like AODV, DSDV. The main contribution of our algorithm is in reducing any

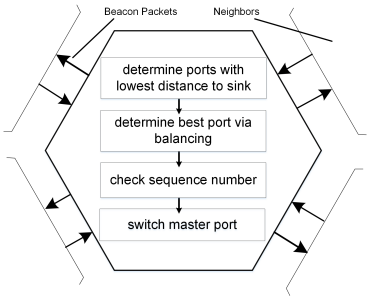


Fig. 4: Different steps in skin cell of the dynamic routing protocol. To share routing and network information (distance to the sink, load, etc.), skin cells periodically forward beacon packets to all at startup time bidirectionally connected neighbors. Furthermore, it periodically checks the connectivity to neighbors, determines the current best port, and if required, switches the master port.

Algorithm 1: Distributed routing algorithm

1: initial:
2: H(x) $\leftarrow \begin{cases} \infty & \text{for all skin cells} \\ 0 & \text{for the PC} \end{cases}$
3: given:
4: P_a as set of all active ports, P_s as set of slave ports
5: Beacon packet b_p
6:
7: $p_m \leftarrow \emptyset$
8: **EVERY PERIOD** T_b **DO:**
9:
10: **at every Skin Cell (Source)**
11: *find best master port* p_m
12: $P_b \leftarrow P_a / P_s$
13: $P_{min} \leftarrow min(H(P_b))$
14: $p_m \leftarrow$ determineMasterPort(P_{min})
15: **if** $p_m = \emptyset$ **or** $H(p_m) = \infty$ **then**
16: $H(b_{p_i}) \leftarrow \infty$
17: **else**
18: $H(b_{p_i}) \leftarrow H(p_m) + 1$
19: **end if**
20: $Sq(b_{p_i}) \leftarrow S(p_m)$
21: **send** bp_i to all ports $p_i \in P_a$
22:
23: **at the PC (Sink)**
24: $Sq \leftarrow Sq + 1$ *Increment Sequence Number*
25: $H(b_p) \leftarrow 0$
26: $S_q(b_p) \leftarrow S_q$
27: **send** b_p to all ports P

Algorithm 2: Determine MasterPort

1: given: P_{min}: Set of ports with shortest path to the sink
2: **if** $m_p = \emptyset$ **or** $H(P_{min}) < H(mp)$ **then**
3: $p_{m_{new}} \leftarrow random(P_{min})$
4: **else if** $|P_{min}| > 1$ **then**
5: $p_{m_{new}} \leftarrow balancingFunction(P_{min})$
6: **if** $Sq(p_{m_{new}}) \geq Sq(p_m)$ **then**
7: $p_m \leftarrow p_{m_{new}}$
8: **end if**
9: **else**
10: $H(p_m) \leftarrow \infty$
11: **end if**
12: return p_m

overhead, which is unnecessary for wired networks and could increase the convergence time. To further reduce the convergence, the detections of broken connections is modified. Furthermore, a balancing algorithm reduces the overall bandwidth cost of the dynamic network protocol or even improves the maximal sensor data bandwidth compared to the previous static solution.

1) Algorithm for Route Determination: The protocol determines routes in a distributive manner, i.e. without knowledge about the networks topology. Possible routes to the sink are determined by a gradient based on the hops to the sink to keep the resulting routes and network depth as deterministic as possible. In this work, we use the distributed Bellman-Ford algorithm [13] for gradient determination, implemented in the same way as in most proactive distance-vector routing protocols, e.g. like in DSDV [4].

Alg. 1 shows the basic procedures, executed every period T_b at skin cells or PC. At initialization, every skin cell starts with a distance to sink $H(x)$ of ∞. The PC, as the sink itself, starts with a distance of 0. Now every node i shares $H(i) + 1$ with its neighbors by sending beacon packets b_{p_i} every period T_b to every active port $p_{a_i} \in P_a$. In the next period, every skin cell received one packet at every p_{a_i}, setting the port with the lowest distance $min(H(P_a))$ to its master port. After determination of the current best master port using Alg. 2, it forwards a distance $H(m_p) + 1$ to its direct neighbors. If the skin cell itself and all its neighbors still have a distance of ∞, it can not distinguish a master port and therefore continuous forwarding ∞ to its neighbors. Let's say N is the set of all nodes in the network, then after $max(H(N)) * T_b$, the algorithm converges and all nodes know a potential route to the sink.

As shown in Alg. 2, the function *determineMasterPort(P_{min})* chooses a random node out of P_{min} during initialization, since there, $m_p = \emptyset$. If balancing is enabled, it uses the balancing algorithm to possibly switch its master port, if it improves the network's balancing.

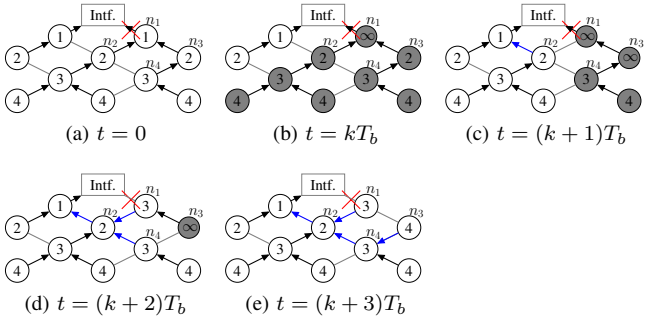
2) Sequence numbers for loop avoidance: As explained in [13], the Bellman-Ford algorithm is not loop-free. In theory, a loop will solve itself if all nodes have a possible way to the sink [13]. However, if sensor data packets are forwarded via loops in the robot skin, it is possible that skin cells are too occupied by data packets to process beacon packets, which are necessary for resolving loops. This leads to an unstable and nondeterministic network behavior.

One solution for a loop prevention is the use of sequence numbers (Sq), as shown in DSDV[4]. Here, sequence numbers express the freshness of a route and are incremented by sinks after every route update (in the skin network every period T_b). A new port p_i can only be chosen as master port if $Sq(p_i)$ is equal or larger than the sequence number of the current master port $Sq(p_m)$, executed in line 7 of Alg. 2.

3) Recognition of broken connections: All nodes validate the consistency of routes by frequent packet hand-shake at any connection. To achieve that, every node in the network examines, whether it received the last k consecutive beacon packets at any port. If none of the k packets arrived on port p_i, it sets p_i as broken. If p_i is the current master port, the skin cell will now forward a distance to sink of ∞, and Alg. 1 is going to reroute the network until every node has a potential route, as described in Fig. 5. The threshold of k packets is necessary since packets can be missed due to timer differences between nodes or too high processing occupancy of the micro-controller. Furthermore, they can be corrupted due to electromagnetic noise or other environmental influences.

It is possible that some connections only partially work. This kind of connections have to be avoided as a route to prevent unnecessary frequent route changes. DSDV damps these switches by waiting for a fixed delay before any route switch. Since the skin cell network should convergence as fast as possible, a different approach has to be used.

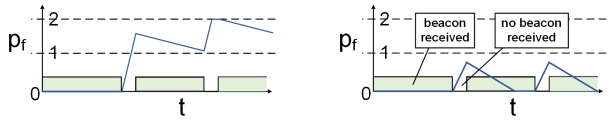
The new dynamic protocol detects these connections via failure penalty p_f and two penalty rates, r_f and r_r, whereby r_f specifies the increase of p_f after a missed packet and r_r the reduction after a detected recovery, i.e. after receiving a packet.



(a) $t = 0$ (b) $t = kT_b$ (c) $t = (k+1)T_b$

(d) $t = (k+2)T_b$ (e) $t = (k+3)T_b$

Fig. 5: Worst case scenario of a single broken connection: Failure on the connection (n_1,$Intf.$). At $t = 0$, the failure occurs (a), then the effected node (n_1) will recognize this at kTb, and sets its own distance to the sink to ∞ (b). The first node which switches its master port is n_2, upon receiving a distance of ∞ from n_1(c). n_4 switches after being informed about the lost route by n_3 (d). This gives node n_4 the ability to resolve its route to the sink. The algorithm converged after $(k+3) * T_b$ (e).



(a) $r_f \approx r_r$ (b) $r_f > r_r$

Fig. 6: Two examples for the detection of a loose connection. If p_f exceeds 1, the connection is marked as not working. If beacons are received, p_f is reduced by r_r, if no beacons are received, it is increased by p_f. Fig. a) shows the example for r_f similar to r_r, whereby it does not detect the loose connection, since beacon misses are too rare. If r_f is increased and r_r is reduced, the connection is detected (p_f is larger than 1), shown in b)

$$p_f = \begin{cases} p_f - r_r, & \text{if packet received} \\ p_f + r_f, & \text{if no packet received} \end{cases}$$

If p_f exceeds the value 1 at a port, the connection is set as broken, otherwise, it is expected not to be broken. By choosing p_f and r_f, the allowed ratio between lost and received packets can be set. p_f can only have values in the range of $[0, 2]$, to guarantee that the cell is able to fast detect working connections which previously were broken (e.g. after replugging a cable). Fig. 6 shows two examples of a partly working connection and the resulting change of p_f during the time.

4) Route selection for load balancing: Additional methods for picking the best out of all possible routes can provide a more balanced network.

One possible balancing method without topology knowledge is the Shortest Path Balancing (SPB), which is among other balancing methods also used in Arbutus [11]. Here, every route is always the shortest path to the sink, to guarantee, that the Bellman-Ford algorithm can resolve routes as quickly as possible.

In SPB, a node chooses its master port depending on the load of the interface port used in that route, as described in Alg. 3.

This requires additional data, carried by the beacon packets: *Load* and *Subnetwork Size*.

The *Load* $L(i)$ of node i is calculated as:

$$L(i) = 1 + \sum_{j \in P_{s_i}} L(j) \tag{1}$$

Algorithm 3: Balancing Function

```
given:
P_min (Ports with shortest path to sink), S(x) (Subnetwork Size for port x)
L(x) (Load for port x), L (Load of Skin Cell)

find new best master port p_mnew
p_mnew ← ∅
Q ← P_min/p_m
while |Q| > 0 do
    q_i ← min(S(Q))
    if S(q_i) ≥ S(p_m) and L < L(q_i) − L(p_m) then
        p_mnew ← q_i
        break
    else
        Q ← Q/q_i
    end if
end while
if p_mnew == ∅ then
    return p_m
else
    return p_mnew
end if
```

with P_{s_i} as the set of all slave ports of node i. The *Subnetwork Size* $S(x)$ of a node k refers to the load of the interface port used by k to forward data to the PC.

To avoid unnecessary oscillations, a node in the network should only switch its subnetwork, if the new routing solution decreases the overall balance factor. This can be achieved by forbidding node i to switch from subnetwork k to j for balancing if:

$$L(i) < L(j) - L(k) \tag{2}$$

Additionally, to guarantee that the balancing will converge in finite time, the balancing frequency has to be lower than the beacon packet frequency to satisfy the following conditions: 1) Subnetwork sizes have to be up-to-date before a node balances, and 2) Only one node should balance at the same time.

The maximum number of beacon periods until the balancing is up to date (n_{val}) can be calculated as: $n_{val} \leq 2 * max(H(N))$, with $H(x)$ as the distance to the sink for node x and N as the set of all nodes in the network.

It is important, that there is only one cell balancing every t_{val}. Therefore, f_b has to be reduced further by the number of nodes, which possibly balance at the same time (n_{bal}). Lets say the distance of a node i to the PC is d_i. Since the network has a two dimensional meshed structure, only one node of the network can have the distance d_i to two root nodes, i.e. the ability to balance.

Therefore, n_{bal} is bound by the maximum distance to the sink in the network d_{max} and the number of root nodes $|N_R|$ via: $n_{bal} \leq d_{max} * |N_R|$.

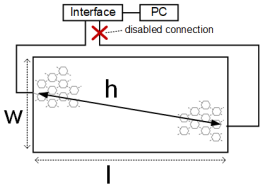
This leads to a maximum balancing frequency f_{bal} of

$$f_{bal} = \frac{f_b}{n_{bal} * n_{val}} = \frac{f_b}{2 * d_{max}^2 * |N_R|} \tag{3}$$

However, this is a worst-case scenario. Experiments with the real network were showing, that higher balancing frequencies also lead to a stable balancing.

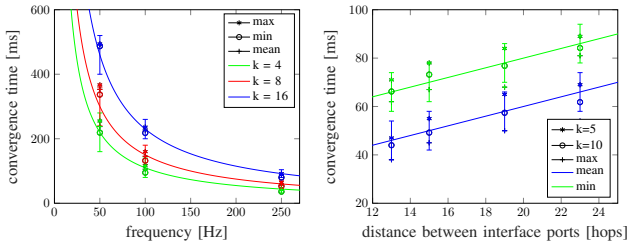
III. Experiments

The experiments consist of two parts: Analysis of the dynamic network protocol itself and analysis of the impact of a connection failure on a running system.



setup	l	w	h	N
1	11	9	13	99
2	11	12	15	132
3	11	12	19	132
4	11	12	23	132

Fig. 7: Setups for measurement. Two different patches were used with the interface connected to ports on opposite sides. l and w are the length and width of the patch in number of cells. h are the number of cells between the interface connections. The red cross marks the connection disabled at t_0.



(a) C vs f_b and k (b) C vs h and k

Fig. 8: Convergence time vs beacon frequency b_f and beacon threshold k (a) as well as distance h in hops 2 sink (b). Experiment in (a) is taken with setup 1 by disabling one interface port at every measurement. The lines show $C_1(f_b, k)$ and its error $\Delta C_{err}(f_b)$. Higher frequency f_b than 250 Hz for the beacon packets were not tested, since this may result in an unstable network routing. For the measurement in (b), setup 1-4 were used, with varying distances between the interface ports. Lines represent the function $C_2(k, f_b)$, error bars $\Delta C_{err2}(f_b)$.

A. Analysis of the dynamic routing protocol

1) Route recovery convergence time: The worst case convergence time appears by connecting the ports to cells of opposite sides on patches, to force every skin cell, which is forwarding to the affected port, to switch its route.
It is measured by taking the difference between the occurrence of the failure (t_0) and the last master port switch (t_f), reported to the PC via a packet delayed by Δt_{d_i}. The convergence time c_m is calculated as: $c_m = max(t_{f_i} - t_0 - \Delta t_{d_i})$.

Instead of physically breaking the connection, the failure is simulated by stopping to forward beacon packets at the PC, to get precise measurements.

The first experiment studies the dependency of the recovery convergence time on the beacon frequency f_b and the beacon threshold k. The measurements are taken with setup 1, as shown in Fig. 7. There was an average delay Δt_{d_i} of 6 ms in the network. This results in the following function for the mean convergence:

$$C_1(f_b, k) = \frac{7}{f_b} + \frac{k}{f_b} \tag{4}$$

for a network with a distance between the interface ports of 13 hops and a total size of 99 skin cells.
The error of the function is given by:

$$\Delta C_{err}(f_b) = \pm\frac{3}{f_b} \tag{5}$$

This variance is most likely caused by timing differences between nodes, varying the routing information propagation delay.

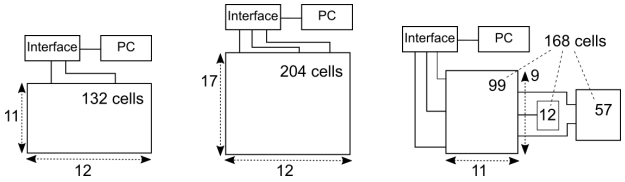
Furthermore, the convergence time may depend on the distance between the nodes h. For the study of this relationship, we measure the convergence time of setup 1 to 4, with a beacon frequency f_b of 250 Hz. Fig. 8b shows its results. It is possible to distinguish, that the convergence time almost linearly depends on the hop-distance between interface ports. The error applied in the figure, $C_{err2}(f_b)$, is $\frac{2}{f_b}$. This leads to the following function:

$$C_2(h, k) = (\lceil \frac{1}{2}h \rceil + k) * \frac{1}{250Hz} \pm \frac{2}{f_b} \tag{6}$$

The factor $\lceil \frac{1}{2}h \rceil$ equals the maximal subnetwork depth d_m when connecting the interface to skin cells placed at the opposite side of a patch. By combining formula 6 with the results of formula 4 and 5, we can determine the dependency of the convergence time on d_m, k and f_b as:

$$C(d_m, k, f_b) \leq \frac{d_m}{f_b} + \frac{k}{f_b} \pm \frac{3}{f_b} \tag{7}$$

This equals the theoretical example as mentioned in Fig. 5. Further measurements with different setups were showing, that the convergence time for any connection break in the network is bounded by $C(d_m, k, f_b)$.



(a) Setup 5 (b) Setup 6 (c) Setup 7

Fig. 9: The network setups for the balancing performance analysis. The setups were chosen in a way that there is some data loss with the static network protocol with data gathering period of 250 Hz.

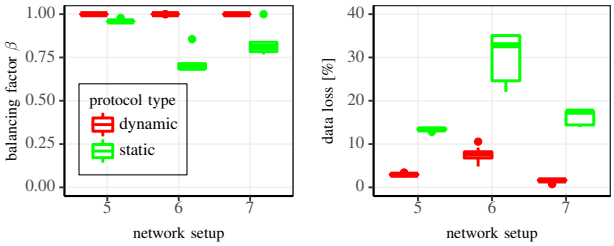


Fig. 10: Balance factor and data loss studies for setups 5-7. The dynamic protocol reaches always a balance factor of 1, in contrast to the static network protocol. For the data loss study, the networks have been set up in a way, that every network is running beyond its limits to force data loss, by choosing a data gathering rate of 250 Hz and using a too large network setups for the number of interface ports. The dynamic protocol could decrease the data loss for every network compared to the previous used protocol by balancing the bottleneck packet load, i.e. the load at the connections between network and interface).

2) Load balancing performance: The balance factor β is an often used performance metric to compare the load of nodes in the networks and is calculated by:

$$\beta = \frac{(\sum_{i=1}^{k} L(x_i))^2}{k \sum_{i=1}^{k} L(x_i)^2} \tag{8}$$

with $L(x)$ as the load of k different ports x_i. β is in the range of $(0, 1]$ with 1 if the load of every port at a node is the same. Three different network setups were used to compare the balancing and the data loss improvement, as shown in Fig. 9. It was important to construct the networks in a way that data loss occurs when using the previous network protocol. Every measurement was taken with a data gathering rate of

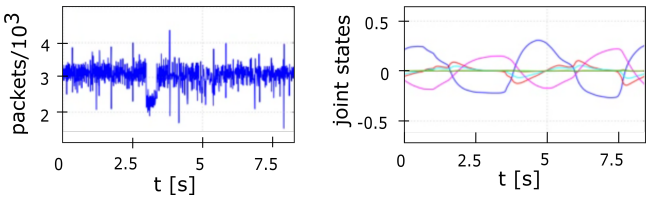


Fig. 11: Failure at a controlled robot arm [14]. One connection to the interface port were un-plugged by hand at around 2.6 seconds. It can be seen, that the joint states are smooth, without any unpredicted behavior during the connection loss. Therefore, the algorithm resolves the network routing fast enough for a stable control application.

250 Hz and a beacon frequency of 10 Hz. The balance factor results are shown in Fig. 10. The dynamic protocol solution reaches almost balance factors close to 1, in contrast to the previous protocol. Nevertheless, the balancing performance depends on the network structure since the algorithm can only balance in the range of shortest paths.

Data loss could be reduced by about 10% in every network setup, as shown in Fig. 10, with a maximum of about 30 % reached in setup 2. Furthermore, it can be distinguished, that the dynamic protocol leads to more deterministic results. The remaining data loss is caused by the network setup. Reduction of the network size or increase of the number of interface ports prevents this data-loss.

3) Cost analysis: The dynamic network requires additional packet transmission bandwidth B_{rel}, which can be calculated with knowledge about packet-size P_s, beacon frequency f_b, intermission gap I_g, as well as data rate D_r by

$$B_{rel}(f_b) = \frac{f_b * (P_s + I_g)}{D_r} = \frac{f_b * 240bit}{4Mbit/s} \tag{9}$$

leading to an bandwidth usage of 0.6% for $10Hz$, 6% for $100Hz$. This has an effect on the maximum subnetwork size vs interface port ratio, which is decreased by B_{rel} when using the dynamic network.

However, if the network is built in a way, such that it can be balanced with the constraint of shortest distances, the improvement of the balancing algorithm exceeds the maximal data bandwidth reduction caused by the periodic beacon packets, whereby the dynamic protocol still improves the overall network performance. This can be seen in Fig. 10, as the dynamic network decreases the data loss.

B. Failure at a controlled robot arm

In the final experiment, we disconnected an interface port connection by hand at a currently controlled robot and analyze the joints' behavior as well as the data loss, caused by the failure. Fig. 11 shows the result with a beacon frequency b_f of 20 Hz and a k value of 8. Although there is a total convergence of around 500 ms, the joint states don't show any negative effect caused by the failure and rerouting of the network.

IV. CONCLUSION

This work presents a dynamic routing protocol for large-scale meshed skin networks with constrained memory and processing power. It uses the Bellman-Ford algorithm to determine possible routes and guarantees a loop-free convergence by the use of sequence numbers. If the protocol detects a connection failure, it reroutes the network when necessary, until every node has a valid route to the sink.

The protocol can achieve maximum convergence time of 40 ms when using a beacon frequency of 250 Hz. Furthermore, the protocol implements an additional shortest path load balancing algorithm allowing larger skin networks and leading to more deterministic routing results. Depending on the network topology, it could reduce the largest subnetwork size for about 20% compared to the previously used protocol. The design was validated and analyzed on a real robot skin network.

REFERENCES

[1] G. Cannata, M. Maggiali, G. Metta, and G. Sandini, "An embedded artificial skin for humanoid robots," in *2008 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2008, pp. 434–438.

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

[3] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot, "Optimized link state routing protocol for ad hoc networks," in *Proceedings. IEEE International Multi Topic Conference, 2001. IEEE INMIC 2001. Technology for the 21st Century.*, 2001, pp. 62–68.

[4] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, pp. 234–244, 1994. [Online]. Available: http://doi.acm.org.eaccess.ub.tum.de/10.1145/190809.190336

[5] I. D. Chakeres and E. M. Belding-Royer, "Aodv routing protocol implementation design," in *24th International Conference on Distributed Computing Systems Workshops, 2004. Proceedings.*, 2004, pp. 698–703.

[6] P. Huang, H. Chen, G. Xing, and Y. Tan, "Sgf: A state-free gradient-based forwarding protocol for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 5, no. 2, pp. 14:1–14:25, Apr. 2009. [Online]. Available: http://doi.acm.org/10.1145/1498915.1498920

[7] R. Poonia, A. K. Sanghi, and D. Singh, "Dsr routing protocol in wireless ad-hoc networks: Drop analysis," *International Journal of Computer Applications (0975-8887)*, vol. 14, no. 7, 2011.

[8] A. Hossein Mohajerzadeh and M. Hossien Yaghmaee, "Tree based energy and congestion aware routing protocol for wireless sensor networks," *Wireless Sensor Network*, vol. 02, no. 02, pp. 161–167, 2010.

[9] D. Kumar, T. C. Aseri, and R. B. Patel, "Eehc: Energy efficient heterogeneous clustered scheme for wireless sensor networks," *Computer Communications*, vol. 32, no. 4, pp. 662–667, 2009.

[10] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "Pump-slowly, fetch-quickly (psfq): A reliable transport protocol for sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 4, pp. 862–872, 2005.

[11] D. Puccinelli and M. Haenggi, "Arbutus: Network-layer load balancing for wireless sensor networks," in *2008 IEEE Wireless Communications and Networking Conference*, 2008, pp. 2063–2068.

[12] F. Bergner, P. Mittendorfer, E. Dean-Leon, and G. Cheng, "Event-based signaling for reducing required data rates and processing power in a large-scale artificial robotic skin," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2015, pp. 2124–2129.

[13] C. Cheng, R. Riley, S. P. R. Kumar, and J. J. Garcia-Luna-Aceves, "A loop-free extended bellman-ford routing protocol without bouncing effect," *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 4, pp. 224–236, 1989. [Online]. Available: http://www.cs.cmu.edu/~srini/15-744/papers/_Cheng.pdf

[14] E. Dean-Leon, B. Pierce, P. Mittendorfer, F. Bergner, K. Ramirez-Amaro, W. Burger, and G. Cheng, "Tomm: Tactile omnidirectional mobile manipulator," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2441–2447.