BPoL: A Disruption-Tolerant LoRa Network for Disaster Communication

Daniel Schmidt*, Franz Kuntke†, Maximilian Bauer†, Lars Baumgärtner†

Technical University of Darmstadt, D-64289 Darmstadt, Germany
*E-mail: daniel.schmidt.97@stud.tu-darmstadt.de
†E-mail: {lars.baumgaertner, franz.kuntke, maximilian_jochen.bauer}@tu-darmstadt.de

2023 IEEE Global Humanitarian Technology Conference (GHTC) | 979-8-3503-2132-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/GHTC56179.2023.10354717

***Abstract*—Information and Communication Technology (ICT) is vital for everyday life and especially during times of disaster. Relying on existing infrastructure is problematic as maintenance is expensive, and they can be disrupted in emergency scenarios. Cost is a major factor which limits the technologies that can be used in rural areas or for emergency response, as satellite uplinks or private cellular networks are very expensive and complex. LoRa is commonly used for IoT infrastructure worldwide in the form of LoRaWAN to cover larger distances with low costs. But it can also be used in a Device-to-Device (D2D) mode for direct communication. By combining LoRa with Disruption-tolerant Networking (DTN), we present an affordable and practical solution that can cope with challenging conditions and be used for a large variety of applications. In our evaluation, we show how adaptable our solution is and how it outperforms similar mesh-based applications for disaster communication.**

***Index Terms*—Emergency Communication, LoRa, Disruption-Tolerant Networking, Opportunistic Networking.**

I. Introduction

In 2021 about 86% of people in Europe subscribed to mobile services [1], and these numbers are expected to grow year by year. We rely on being connected to the Internet and having mobile reception for our daily lives. Various emergency and warning services are using mobile communication (e.g., cell broadcast, SMS and mobile Internet) too, which is also vulnerable in disaster situations where congestion, because of the increased volume of messages, can happen, or the infrastructure is damaged and rendered unusable. In such instances, we need communication solutions that work in highly stressed environments and are readily available as well as affordable.

In such challenging environments, Delay-Tolerant Networks (DTNs) offer a promising solution for communication. However, current open-source implementations of DTNs are limited in their support for long-range multicast and broadcast radio communication technologies, which hinders their effectiveness in emergency scenarios, as WiFi and Bluetooth have a too limited communication range to be effective for large scale disasters. Furthermore, these implementations are primarily based on unicast transmissions, which may not be suitable for situations where messages must be sent efficiently to multiple recipients or broadcasted to a wide area. Therefore, there is a need for small, cost-efficient technologies that can provide broadcasting functionality while still delivering messages as far as possible and are available worldwide.

One such radio technology is LoRa, which is often used in Internet-of-Things (IoT) setups in the form of LoRaWAN infrastructure. But LoRa can also work in a device-to-device (D2D) mode and provides comparatively long communication ranges from a few hunderd meters up to 16 km and has low energy requirements. The major limitations are, depending on the region, duty cycle restrictions and very limited bandwidth. Thus, protocols and communication patterns that are known to work on WiFi networks do not work or scale when directly used in LoRa setups.

Past approaches either only consider LoRa as a limited transport medium but ignore legal implications such duty cycle restrictions as well as the effects of long term usage with DTNs, or favor a mesh-based solution such as the *Disaster.Radio*[1] and *Meshtastic*[2] projects. While the latter works for low message numbers and a rather dense network of LoRa nodes, having the redundancy through a store, carry and forward architecture provided by a DTN has advantages in disaster settings. Here, a sparse network can compensate a lack of direct communication links through node mobility, trading a low end-to-end latency for delivery probability and infrastructure cost.

In this paper, we present our approach *Bundle Protocol over LoRa (BPoL)*, a practical LoRa-DTN overlay network that is extensible for different scenarios and common routing algorithms, supports duty-cycle restrictions and integrates with a well established implementation[3] of the *Bundle Protocol (BP)* as specified in RFC 9171 [2]. We not only provide a novel convergence layer for BP-over-LoRa but also designed an overlay network protocol based on *Protocol Buffers* (protobufs) for managing *BPoL* nodes, e.g. discovery and network diagnostics in addition to bundle delivery. Furthermore, by building upon the *rf95modem*[4] firmware, we can easily support a large variety of LoRa transceivers.

With our work, we make the following contributions:

- A novel and portable overlay network protocol for LoRa using *Protocol Buffers*

[1]https://disaster.radio
[2]https://meshtastic.org
[3]https://github.com/dtn7/dtn7-rs
[4]https://github.com/gh0st42/rf95modem

- A novel, real-world-deployable system to integrate our overlay network with an existing DTN implementation[5]
- Novel improvements and adaptations of DTN routing strategies for resource constraint broadcast channels
- A thorough evaluation of our system with common routing strategies, as well as a comparison with *Meshtastic* and complex urban scenarios.

II. Related Work

Meshtastic [3] is an open-source mesh network for low-powered devices, focusing on ESP32 microcontrollers. It uses LoRa as communication technology and employs protocol buffers to encode its messages. It lacks any DTN mechanics which are needed for highly stressed environments. Thus, its usefulness in sparse networks where no end-to-end mesh connectivity can be achieved is rather limited. Another project utilizing LoRa with a similar hardware stack for emergency communication is *disaster.radio* [4] which also mainly focuses on mesh communication with added peer-to-peer capabilities.

Höchst et al. [5] explored the possibilities of bringing DTN to LoRa by enhancing existing devices with a small cable and Bluetooth connectable LoRa module. Similarly, Sciullo et al. [6] use smartphones with LoRa and DTN for emergency communication services.

Wong et al. [7] surveyed the current landscape of multicast DTN routing mechanisms and opens a discussion on further issues that arise in contrast to unicast-based approaches, especially in regard to security considerations.

Solpico et al. [8] built a larger DTN system to collect GPS data and size-limited custom messages for affected persons in disaster situations. Their system includes smartphone-programmable LoRa beacons, but DTN communication is done via WiFi at a higher level, which includes mobile aggregation units and a mobile command center. Zguira et al. [9] introduced an Internet-of-Bikes protocol that exchanges data with a 802.11p based DTN and a tailored routing scheme similar to Binary Spray and Wait. Their evaluated scenario is sensor data propagation of a bike sharing system. Baumgärtner et al. [10] proposed a quadrant-based routing algorithm for LoRa-based DTNs which is used as evaluation routing strategy in *BPoL*. Udén et al. [11] and Grasic et al. [12] brought LoRa based DTN to harsh arctic environments to track reindeer. The key idea was to create DTN-based islands of connectivity with mobile masts and pocket nodes. Their approach achieves a far better cost factor than GSM- or satellite-based GPS solutions. The SHETLAND-NET research project also aims to build an IoT telemetry service for Antarctica, incorporating long distance wireless links and DTNs [13] with uplink nodes to a backbone network.

III. Design

In the following, we give a brief overview of the overlay network protocol and its features as well as the architecture of *BPoL*.

[5]https://github.com/BigJk/dtn7-rs-lora-ecla

A. Overlay Network Protocol

A well-defined lightweight, yet extendable protocol is needed to build a LoRa-based overlay network that handles the specifics of advertisement, bundle forwarding, configuration changes, and more. While designing the protocol, the following packet types and behavior surfaced as practical to solve the needs of the network. A brief overview of the currently supported packet types is given below:

- **Advertise**: information like node ID, position and additional data
- **BundleForward**: bundle data to be routed
- **Config**: *BPoL* configuration updates for remote management of nodes
- **PingPong**: diagnostics packet to quickly gather information about the surroundings of a node

To make it flexible enough for a large variety of routing protocols, the **Advertise** and **BundleForward** packets can contain additional optional data. For example, the *quadrant*-based routing strategy uses the optional data to transmit a hash of the locally stored bundles.

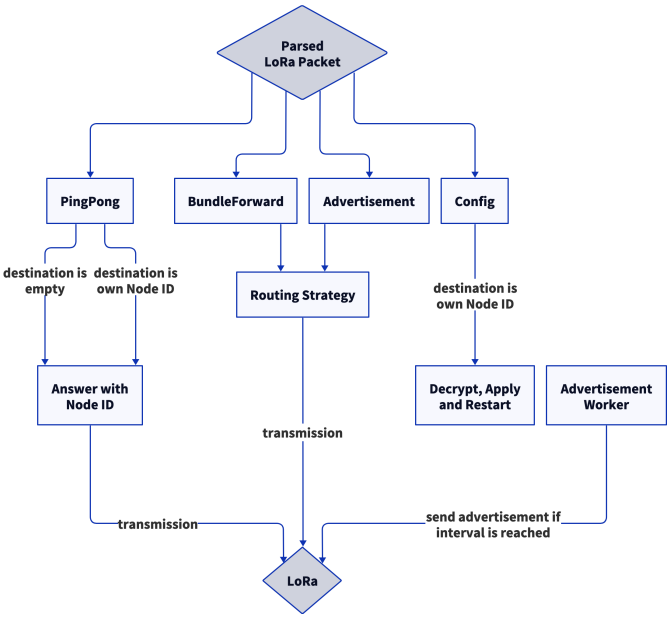

Fig. 1: Overlay Network Behavior

Each node in the overlay network should behave according to Figure 1. The **PingPong** functionality supports broadcast pings as well as probing only a specific node ID. The information received through **Advertise** and **BundleForward** packets is intended for the use by different routing strategies. Finally, the **Config** packet can be used to update a node's configuration, this applies for *dtnd* as well as *BPoL* itself. For security purposes, this packet uses a pre-shared key to deliver configurations.

B. BPoL Daemon

BPoL uses a flat architecture where decoupled actors pass messages via a single global message bus and have distinct

responsibilities as depicted in Figure 2. We need to interact with external services like the LoRa modem and the DTN process and be able to re-start and re-configure parts of the system at run-time without interrupting other parts. Therefore, an actor-based architecture with a message bus is a great fit.

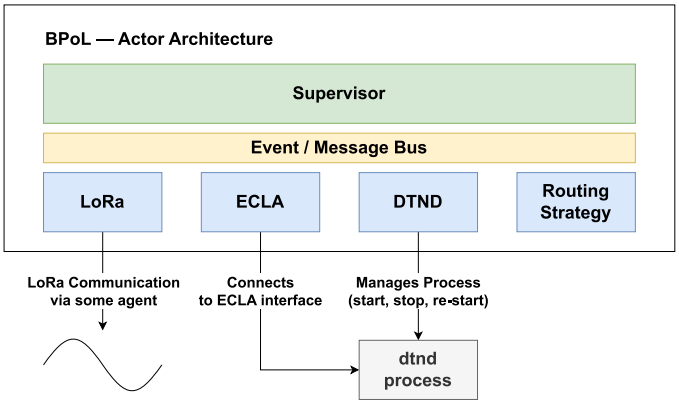

Fig. 2: *BPoL* Architecture

An alternative to this approach would be to explicitly weld sub-systems that must pass messages between them via separate communication channels for each relationship. However, the maintainability of this approach highly decreases the more sub-systems need to interact with each other, and introducing new interactions and sub-systems takes much work. In addition, re-starting parts of the system are also more complex in that approach.

1) Routing Strategies: Three example routing strategies were implemented for *BPoL* as a basic set for evaluation.

- **Random**: This strategy passes bundles to neighbors in random order. If no neighbor has been seen recently, no bundles are broadcasted.
- **Quadrant Based** This strategy is based on the routing algorithm described in *LoRAgent* [10]. It uses GPS locations of transmissions to assign a quadrant. This information is used to calculate priorities for bundle (re)transmissions.
- **Broadcasting Spray And Wait** This strategy is a new modified spray-and-wait [14] version that works in broadcast environments. In the typical spray-and-wait approach, it is essential to know if a transmission between two nodes was successful before reducing the number of copies left. Sending additional acknowledgments is not a viable solution, as we want to keep the network manageable with such messages. In our modified version, we assume that transmission to a node was successful if that node was seen recently. Thus, we spread to multiple nodes at once as all transmissions are broadcasted but reduce the number of copies only by one. The amount of time that is regarded as recent is configurable.

All routing algorithms have access to LoRa statistics such as used airtime for transmissions to change their behavior depending on limitations such as duty cycle restrictions, battery power, etc.

IV. Implementation

In this section we highlight a few implementation details of our solution. We start with the overlay network protocol and then go into details of *BPoL* itself.

A. Overlay Network Protocol

We want the protocol to be easily integratable into a multitude of languages and being portable while keeping the packet sizes as low as possible. LoRa can only transmit up to 256 bytes in a single frame. Therefore, widely popular but verbose string-based encodings like JSON or XML are unsuited for this task. On the other hand, there is CBOR [15], which BP7 already uses to encode bundles. While CBOR is very space efficient, it is less popular, and adopters of our protocol would need to manually implement the serialization for each new language they want to use.

In the end, the choice fell on *Protocol Buffers (protobuf)*. With *protobuf*, all the possible message types are described in a well-defined scheme. An example of such a message description can be seen for the **Advertise** packet in Listing 1. A custom compiler can then translate this scheme into language-specific serialization code. Because of its popularity, it is compatible with most commonly used programming languages. Furthermore, the resulting encoding is tightly binary encoded and tries to keep the overhead relatively low.

Listing 1: Advertise packet in protocol buffers

```
message Advertise {
  string node_name = 1;
  oneof position {
    LatLngPos lat_lng = 2;
    XYPos xy = 3;
    GenericPos generic = 4;
    NoPos no_pos = 5;
  }
  map<string, string> data = 6;
}
```

B. BPoL Daemon

BPoL is implemented in Rust. Rust is a system programming language known for being remarkably safe and fast. Its performance and overhead are comparable to other system programming languages like C or C++. This makes it a good fit for low-resource systems like the Raspberry Pi that this project targets. Additionally, the dtn7 implementation[6] we are using is also written in Rust. Thus, interfacing with *dtnd* and using the official client libraries becomes much easier. This way, no re-implementation of parts of RFC 9171 [2] is needed. *BPoL* also interfaces with the *rf95modem* firmware [16], which exposes a simple serial protocol to talk with these ESP32 LoRa transceiver boards and to fetch additional data like GPS.

[6]https://github.com/dtn7/dtn7-rs

Message Bus: The global message bus is realized using *tokio::sync::broadcast*, an asynchronous multi-producer multi-consumer broadcast queue. Because Rust enforces safety at compile time, including thread safety, code must be written safely immediately. The message bus helps to alleviate many thread safety problems. By using a message bus, memory sharing is done by communication, which avoids holding a lot of global states that would need to be wrapped by locks. We also get the benefit of using Rust pattern matching to only respond to the events a specific consumer is interested in a convenient way. A short example of the message bus usage can be seen in Listing 2.

Listing 2: Message bus example

```rust
// Fetch a sender and receiver of
// our global message bus
let (bus_sender, mut bus_receive) = get_channels
    ();

// Receive one message
if let Ok(msg) = bus_receive.recv().await {
    match msg {
        Event::ECLAPacketRx(Packet::ForwardData(
            packet)) => {
            // Do something with the packet
        }
        _ => {
            // Other events are not of interest
        }
    }
}
```

V. Evaluation

We evaluated *BPoL* in various scenarios to see how it performs in different circumstances. Besides baseline evaluations, we also compare it to mesh-based solutions and how it can be deployed in urban environments. To easily test with different parameters and in a reproducible environment, we used a real-time LoRa network emulator, *LoRaEmu*[7]. Finally, we also give a brief overview of how much such a setup costs. The whole evaluation setup with analysis and reporting scripts is available online[8]. All evaluations were done on a 2021 Apple M1 MacBook Pro, featuring 16 gigabytes of memory and a 10-core CPU with 8 performance and 2 efficiency cores.

A. Collision Behavior

In a LoRa network, collisions are always possible and should be considered when evaluating communication strategies. To evaluate the performance of *BPoL* in environments with a high collision rate, a scenario was created where high collision rates occur frequently. In this scenario, 5 nodes are placed in a straight line so that each can only see its left and right neighbor, with the leftmost and rightmost nodes only having one reachable neighbor. Bundles are created periodically at the outer nodes and addressed from $node1$ to $node5$, from the leftmost node to the rightmost node, and vice versa.

[7]https://github.com/BigJk/LoRaEMU
[8]https://doi.org/10.6084/m9.figshare.23275532

Parameter	Value
Frequency	868 MHz
Preamble	6 symbols
Spreading Factor	7
Bandwidth	125 kHz
Coding Rate	8
Send Interval	10s, 15s, 20s, 15s, 10s (left to right)
Packets per Interval	2
Advertisement Interval	30s

TABLE I: Configuration of *Collision* scenario

The routing strategies are configured not to use any random delay when starting, and the (re)sending interval of each node is aligned so that periodic collisions occur. The scenario runs for 10 minutes, without further waiting for bundles to arrive after the last one is sent. The exact settings can be seen in Table I. For this scenario, the modified broadcast spray and wait was omitted, as the strategy is intended for environments with movement. When $node2$ and $node4$ receive a bundle their remaining copies are set to 1, which means only a direct delivery is allowed, but this is impossible as they will not move in the scenario.

In Figure 3, we can see that the closer the node is to the center, the more collision occurs, as one would expect if both sides try to send a packet to the other with semi-aligned sending intervals. In the middle of the chain, 50 to 80 percent of the transmissions result in a collision, a huge bottleneck for the overall throughput between the sides. The outer nodes only have one neighbor each and, thus, a significantly lower collision probability.

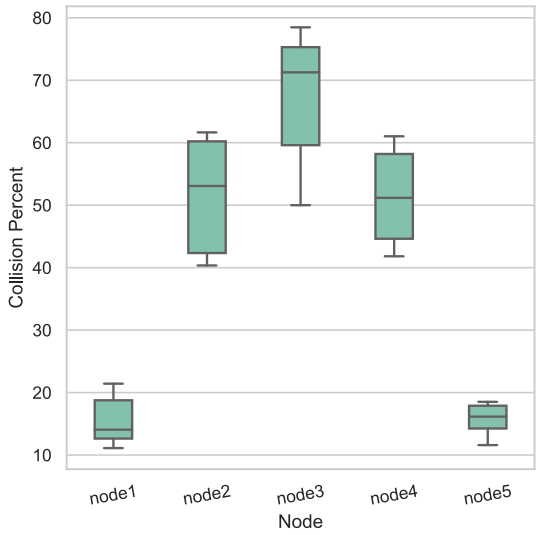

Fig. 3: Collisions in *Collisions* scenario

We then evaluated how different routing strategies handle and recover from such a high number of collisions. In Figure 4, we see that the random strategy is not well suited for such a case, as the delivery probability is very low, although not 0.

Parameter	Value
Frequency	868 MHz
Preamble	32 symbols
Spreading Factor	7
Bandwidth	250 kHz
Coding Rate	8
γ	1.55
Ref. Distance	30
Send Interval	20s
Packets per Interval	2
Advertisement Interval	30s

TABLE II: Configuration of random waypoint scenario

The quadrant-based approach, on the other hand, can still deliver 30 to 50 percent of the bundles. This shows that even in an environment with high collision rates with a good routing strategy, achieving good bundle delivery rates is possible.

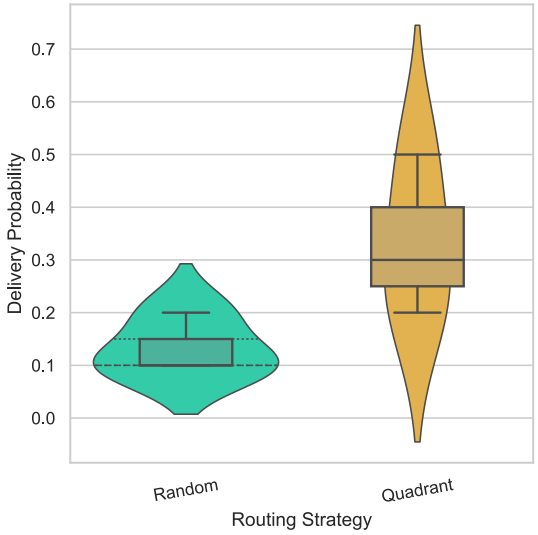

Fig. 4: Delivery probability in *Collision* scenario

B. Random Waypoint

In this scenario, 20 nodes are placed in a 3x3 km area. From these nodes, 15 are mobile and walking between random waypoints at $1.5m/s$, which is in the range of human walking speed, and the other 5 nodes are stationary. The scenario is run with 5 different mobility patterns. The path-loss parameter result in around 500 meters of transmission range. Therefore, some nodes form clusters that can reach each other. Due to the mobility pattern, these clusters keep breaking up and reforming. Bundles slowly spread through the whole network when nodes move between clusters. The exact parameters chosen for this scenario can be seen in Table II.

In Figure 5, we can see that the *random* and *broadcasting spray and wait* strategies underperform, but can still deliver bundles. The *quadrant* strategy, on the other hand, performs well, even reaching a 100% delivery rate in some cases.

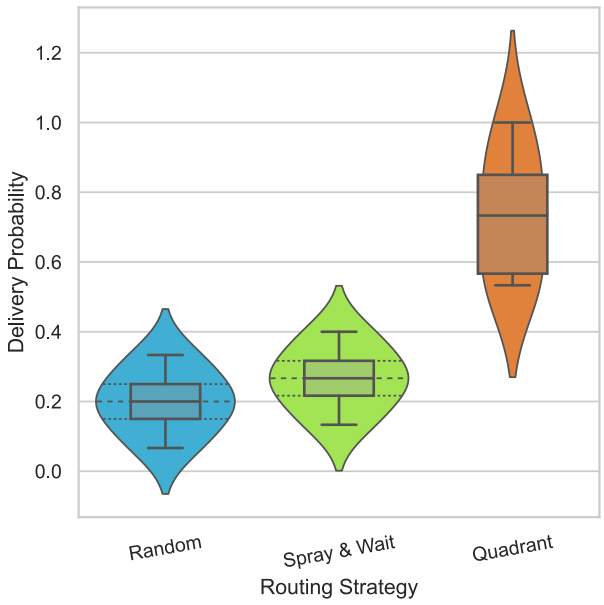

Fig. 5: Delivery probability in *Random Waypoint* scenario

The *broadcasting spray and wait* in its naive implementation does not work well in a broadcasting context where mobility and collision occur. The strategy as it is implemented at the moment treats a transmission to a node as successful if it saw that node. In case of a collision or if the node has already left transmission range, it would still count as successfully transmitted, although the bundle would need to be resent. Implementing an acknowledgment mechanisms would increase the chances for collisions even further as more packets need to be sent plus precious airtime is used, which can be problematic depending on duty-cycle restrictions. One possible solution is to reset the available number of copies of a message at each node after a timeout. Thus, starvation would not be a problem anymore as after a while retransmissions would happen and lessen the effect oft collisions.

C. Comparison: Meshtastic

Meshtastic has a discrete event simulator called *Meshtasticator* [17] that simulates n nodes in a LoRa environment using the mesh routing algorithm present in *Meshtastic*. For the evaluation, 10 scenario runs of the simulator, each with 20 nodes, were generated and converted to equivalent *LoRaEMU* scenarios. The same Log-Distance Path Loss settings are used, and packets are created at identical timestamps as in the original scenario. Each run has a length of 200 seconds, and no mobility is involved in the evaluation scripts of *Meshtastic*. The *random* and *quadrant* strategies are evaluated in two different ways. First, by letting it run as long as the meshtasticator simulation, and the other (called *Trail*) is given extra time to keep delivering bundles after the last send has occurred. Resulting in a total simulation time of 400 seconds. The parameters used for the simulation are shown in Table III.

Parameter	Value
Frequency	868 MHz
Preamble	32 symbols
Spreading Factor	7
Bandwidth	250 kHz
Coding Rate	8
γ	1.5
Ref. Distance	0.15
Send Interval	10s
Packets per Interval	2
Advertisement Interval	30s

TABLE III: Configuration of *Meshtastic* scenario

The evaluation shows that, in terms of latency (Figure 6), the *random* and *quadrant* are similar to the *meshtasticator*. This can be explained with the fact, that all systems have similar strategies for direct retransmission of incoming bundles

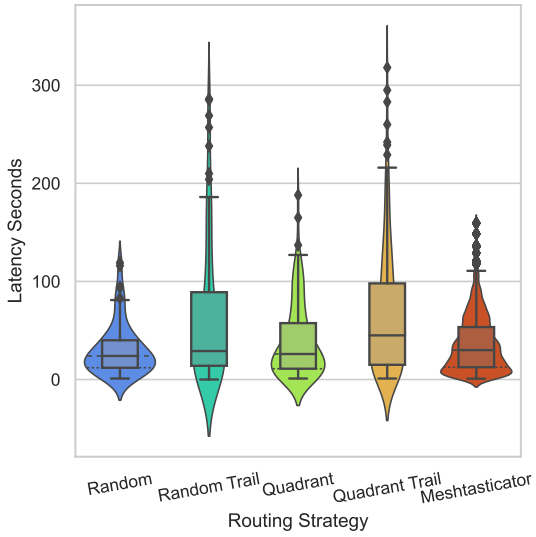

Fig. 6: Latency in *Meshtasticator* comparison

Regarding delivery probability, as shown in Figure 7, the *random* strategy performs poorly with around 20% to 30%, while the *quadrant*-based is considerably better with 40% to 60%. In the case of the runs with trail time, we see that the delivery probability of the *quadrant* strategy increases considerably and surpasses the *meshtasticator* one, reaching a median of 80%. This is expected behavior of a DTN system that, with enough time, the bundles will eventually arrive at their destination, albeit with higher latency (Figure 6) unless collisions, mobility or lifetime issues prevent delivery. Bundles get more chances to be retransmitted and, thus, the delivery rate increases. A real-world deployment would also run continuously, so that the runs with trail time give a more realistic impression of the expected performance of *BPoL* with different routing strategies.

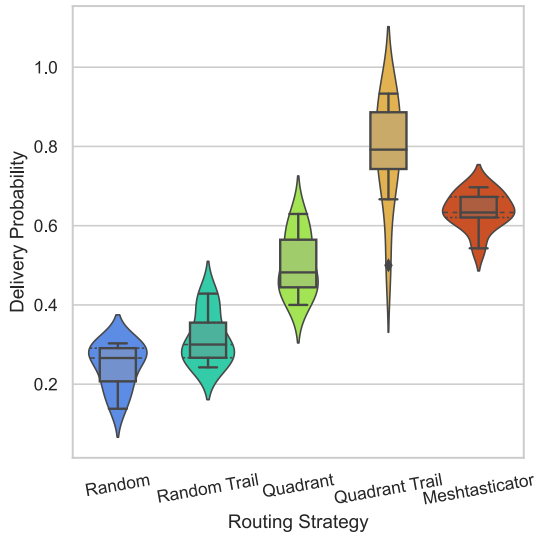

Fig. 7: Delivery Probability of *Meshtasticator* comparison

D. Performance

To evaluate the performance, the CPU and RAM usage of the simulations running on our previously described evaluation machine were tracked. Figure 8 and Figure 9 show the performance stats of the *Meshtasticator* evaluation running 20 nodes, each with an active *dtnd* and *BPoL* process. Additionally, the *LoRaEMU* is running and managing the simulation. We can see that the complete simulation with all sub-processes consumes about 2.5% CPU resources in the median. There are infrequent spikes up to about 25%. The memory utilization stays a bit above 4% in the median. Divided by the number of processes running, these metrics seem promising as each node has less than 0.2% CPU usage, including the *dtnd* as well as *BPoL*. The infrequent spikes are related to situation where a lot of sends or mobility are overlapping and higher amounts of performance is needed for a short time.

E. Case-Study: Darmstadt City

To evaluate *BPoL* in a setup more akin to the real world, a scenario was created based on the city of Darmstadt, Germany. It contains 2 moving cars around the city, 4 pedestrians walking through the city, and 3 clusters of static nodes (e.g., north, center, and south), resulting in 17 nodes in total. The transmission and receiving strength of the static nodes is higher, so situations where a static node can transmit to a pedestrian, but the pedestrian lacks the transmission power to reach the node back also occurs. This scenario is run for 1 hour, and every 2 minutes, a new bundle is created from a random starting node to another node. All parameters can be seen in detail in Table IV.

In Figure 11, we can see that the quadrant-based routing works well in this scenario with a delivery probability of over 60% while the naive random and broadcasting spray and wait underperform.

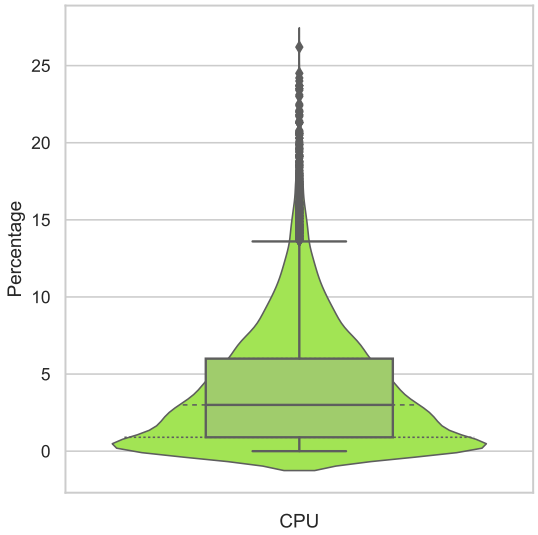

Fig. 8: Total CPU usage of all 20 nodes combined in *Meshtasticator* scenario

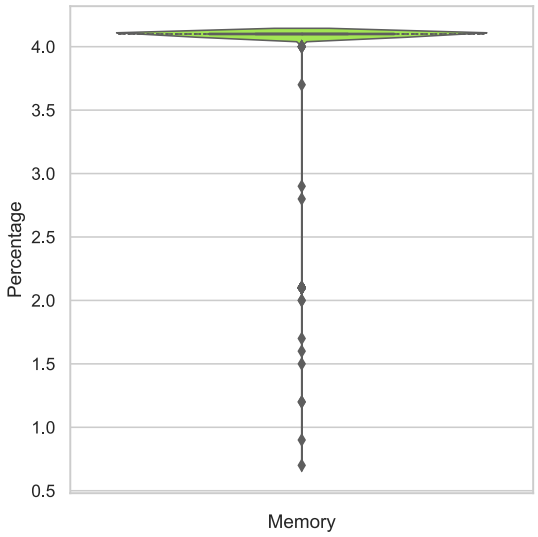

Fig. 9: Total memory usage of all 20 nodes combined in *Meshtasticator* scenario

Parameter	Value
Frequency	868 MHz
Preamble	6 symbols
Spreading Factor	7
Bandwidth	125 KHz
Coding Rate	8
γ	3
Ref. Distance	40
Send Interval	20s
Packets per Interval	2
Advertisement Interval	30s

TABLE IV: Configuration of *Darmstadt* city scenario

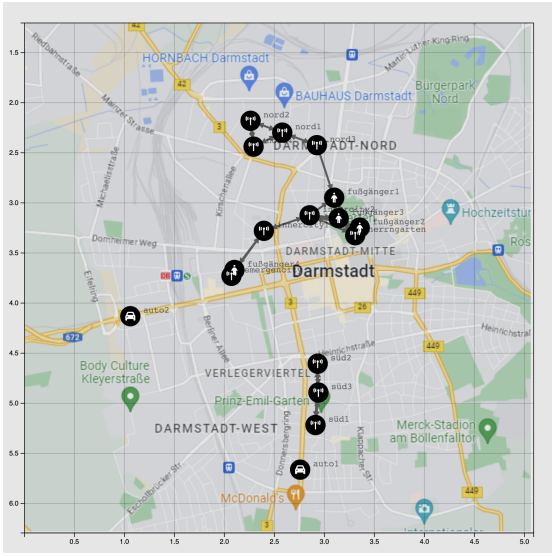

Fig. 10: Overview of *Darmstadt* city scenario

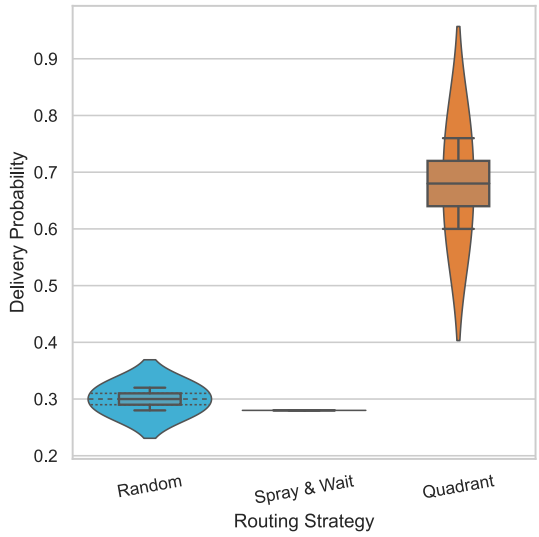

Fig. 11: Delivery probability in *Darmstadt* city scenarion.

F. Deployment Cost

For real-world deployments, it is important that the cost per unit is low, and the different parts can easily be purchased anywhere on earth. Using affordable and readily available hardware like the Raspberry Pi and LILYGO® TTGO LoRa boards, overall cost for a single node can be kept to a minimum. At the time of writing, the cost for a Raspberry Pi 4 starts at 35$ and the Raspberry Pi Zero at 15€. The basic TTGO LoRa board without GPS can be bought for around 20 €. The GPS variant currently costs about 43€. So in a minimal configuration, an RPi Zero, basic TTGO board, and micro SD card (around 5€) would be around 40€. Additionally, a suitable battery pack or solar setup is needed if mobility or extra redundancy is needed. For use cases

that require mobility it is also important to use small form factor, low weight and energy efficient hardware. Furthermore, antennas can be upgraded for better communication range. This factor is excluded from the price calculation because there are many available options and price points for various scenarios. Another cost factor to consider is a case for the *BPoL* box, but here any outdoor junction box or even a plastic food container can be used for short-term deployments and only cost very few euros. Some of our boxes assembled with battery packs and different ESP-LoRa boards and Raspberry Pis can be seen in Figure 12.

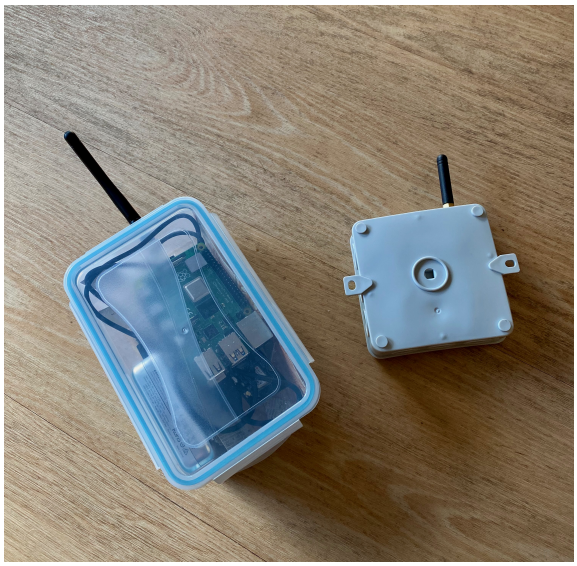

Fig. 12: *BPoL* boxes assembled for deployment

As our *Darmstadt* scenario has shown, we do not need many of these boxes to cover a large area. Using the LoRa nodes in combination with WiFi access points to integrate mobile users with their smartphones, easily lets average users benefit of ad-hoc long range DTN.

VI. Conclusion

In this paper, we have presented *BPoL*, a novel overlay network using LoRa for long range peer-to-peer communication using a DTN built upon the *Bundle Protocol* [2]. *BPoL* supports different routing protocols and provides convenient management functions for real world deployments. Furthermore, it is designed for resource-efficiency and constraint devices. We provide convergence layer that are optimized for broadcast channels and take LoRa specifics such as duty-cycle restrictions into account. By improving and adapting existing routing algorithms, they can be fitted to broadcast based technologies such as LoRa. Through a thorough evaluation and various scenarios, we have shown the practicality and the advantages of our solution compared to mesh-based approaches.

In the future, we want to further optimize the routing to better cope with long-term scenarios where more bundles need to be stored and transferred from the nodes. Another approach to be investigated is the mobility of the nodes using unmanned aerial or ground vehicles to further improve the performance. Furthermore, due to the design decision for protobufs and the portability of our approach, we want to research relay nodes purely on embedded devices such as ESP32, making middle nodes even cheaper and more energy efficient.

Acknowledgment

This work has been co-funded by the LOEWE initiative (Hessen, Germany) within the *emergenCITY* center, as well as the German Research Foundation (DFG) in the Collaborative Research Center (SFB) 1053 MAKI.

References

[1] GSMA, "GSMA Mobile Economy," https://www.gsma.com/mobileeconomy/europe/, 2023, [Online; accessed 24-March-2023].
[2] S. Burleigh, K. Fall, and E. J. Birrane, "Bundle Protocol Version 7," Internet Requests for Comments, Internet Engineering Task Force, Request for Comments 9171, 2022.
[3] M. LLC., "Meshtastic," https://meshtastic.org/, 2023, [Online; accessed 24-March-2023].
[4] "disaster.radio," https://disaster.radio/, 2023, [Online; accessed 24-March-2023].
[5] J. Höchst, L. Baumgärtner, F. Kuntke, A. Penning, A. Sterz, and B. Freisleben, "Lora-based device-to-device smartphone communication for crisis scenarios," in *International Conference on Information Systems for Crisis Response and Management (ISCRAM)*, 2020.
[6] L. Sciullo, F. Fossemo, A. Trotta, and M. Di Felice, "Locate: A lora-based mobile emergency management system," in *IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–7.
[7] K.-S. Wong and T.-C. Wan, "Current state of multicast routing protocols for disruption tolerant networks: Survey and open issues," *Electronics*, vol. 8, no. 2, p. 162, 2019.
[8] D. Solpico, M. I. Tan, E. J. Manalansan, F. A. Zagala, J. A. Leceta, D. F. Lanuza, J. Bernal, R. D. Ramos, R. J. Villareal, X. M. Cruz, J. A. Dela Cruz, D. J. Lagazo, J. L. Honrado, G. Abrajano, N. J. Libatique, and G. Tangonan, "Application of the V-HUB Standard using LoRa Beacons, Mobile Cloud, UAVs, and DTN for Disaster-Resilient Communications," in *IEEE Global Humanitarian Technology Conference (GHTC)*, 2019.
[9] Y. Zguira, H. Rivano, and A. Meddeb, "IoB-DTN: A lightweight DTN protocol for mobile IoT applications to smart bike sharing systems," in *Wireless Days (WD)*. IEEE, 2018.
[10] L. Baumgärtner, P. Lieser, J. Zobel, B. Bloessl, R. Steinmetz, and M. Mezini, "LoRAgent: a DTN-based location-aware communication system using LoRa," in *IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE, 2020.
[11] M. Udén, S. Grasic, K. Kerstin Kemlén, and A. Annelie Päiviö, "NomaTrack and the LoRa-DTN protocol: DTN in innovation for reindeer husbandry," 2021.
[12] S. Grasic, "LoRa-DTN: Tailor-made IoT platform for extreme Arctic conditions," https://www.youtube.com/watch?v=srTsFzihRWc, 2022, [Online; accessed 27-March-2023].
[13] A. Mallorquí, A. Zaballos, and A. Briones, "DTN Trustworthiness for Permafrost Telemetry IoT Network," *Remote Sensing*, vol. 13, no. 2222, 2021.
[14] T. Spyropoulos, K. Psounis, and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," in *ACM SIGCOMM workshop on Delay-tolerant networking*, 2005.
[15] C. Bormann and P. E. Hoffman, "Concise Binary Object Representation (CBOR)," Internet Requests for Comments, Internet Engineering Task Force, Request for Comments 8949, 2020.
[16] L. Baumgärtner, A. Penning, P. Lampe, B. Richerzhagen, R. Steinmetz, and B. Freisleben, "Environmental Monitoring Using Low-Cost Hardware and Infrastructureless Wireless Communication," in *IEEE Global Humanitarian Technology Conference (GHTC)*, 2018.
[17] GUVWAF, "Meshtasticator Event Simulator," https://github.com/GUVWAF/Meshtasticator, 2023, [Online; accessed 24-March-2023].