Competition: Keep it Simple, Let Flooding Shine.

Jan Mueller, Anna-Brit Schaper, Romain Jacob, Reto Da Forno
ETH Zurich
{muelljan, schapera}@student.ethz.ch ; {jacobr, rdaforno}@ethz.ch

Abstract

This abstract presents our solutions to the EWSN 2019 Dependability Competition. We designed generic yet performant solutions by leveraging the flexibility of *Baloo*, a design framework tailored to synchronous transmissions. Instead of crafting subtle optimizations to match the Competition requirements, we used some well-known concepts and combined them efficiently to propose simple yet robust and efficient protocols. Keep it simple, let flooding shine!

1 Introduction

The Dependability Competition traditionally includes many nodes, external interference, and variable traffic. In 2019, the scenarios also feature a large range of input parameters (traffic pattern, load, node types), which calls for flexible network stack designs, compile-time tuning to fit the input set, and runtime adaptation to fight interference.

Synchronous Transmissions (ST), popularized by Glossy [3], is a very powerful technology in such scenarios, as best illustrated by the winning solutions of past competitions [2, 7, 8] – all based on ST. The time synchronization requirements for ST are very tight (from 0.5 μs to 100 μs), which makes it challenging to design flexible protocols and network stacks relying on ST; A slight delay on the radio control may break the whole protocol. However, if it can be handled properly, ST provides a very efficient broadcast communication primitive, which greatly simplifies the design of the upper-layers of the network stack.

From the original Glossy protocol [3], many extensions, modifications, or improvements have been proposed to influence the protocol performance in three dimensions: reliability, latency, and energy consumption. We can cite channel hopping, in-network processing, forwarder selection, and contention slots as a few examples.

To work reliably, ST requires to precisely control the timing of the radio, which involves low-level programming, very close to the hardware – a notoriously hard, time-consuming, and error-prone task. Consequently, the possible design space exploration for a network stack based on ST is usually limited by the time and effort required to simply try out something. Implementing any new idea involves a painstaking cycle of debugging and compatibility check with previously included features, before even starting to evaluate the possible benefits.

2 A Framework to Design Them All

We recently designed *Baloo* [5], a flexible network stack design framework tailored to ST primitives like Glossy [3]. *Baloo* provides a well-defined interface between the radio (executing ST primitives at the physical layer) and the protocol implementation (*i.e.*, the network layer), as illustrated in Fig. 1. Communication is organized in rounds, composed of a sequence of Glossy floods[1]. Each round starts with a transmission of a central node, called the *host*, that sends a *control packet*. This packet includes the *configuration* information for the round (*i.e.*, schedule, protocol parameters, etc.) which is used by the whole network to adapt its behaviour at runtime.

Most importantly, all parameter settings and features are available to the system designer *without any direct interaction with Glossy*. The radio management and execution of Glossy floods is under the full control of a middleware layer which guarantees that the protocol running at the network layer does not compromise the timing requirements for successful synchronous transmissions. *Baloo* lets us focus on the core of the task: design protocols.

3 Protocol Design

Our strategy consists in using a simple but sound protocol logic. We intentionally leave out advanced optimizations, which are typically hard to "get right", in particular in a very dynamic and challenging context (various input parameters and strong interference).

Analysing the scenarios. One particularity of the 2019 edition of the competition is that the same firmware must adapt and perform well across a large range of input parameters. The abstraction offered by *Baloo* naturally helps.

[1] Glossy is the default ST primitive; others can also be used [5].

International Conference on Embedded Wireless Systems and Networks (EWSN) 2019
25–27 February, Beijing, China

ISBN: 978-0-9949886-3-8

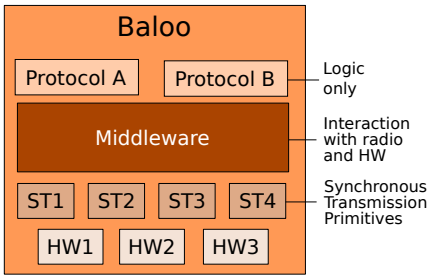

Figure 1. ***Baloo*** **[5] is a design framework for low-power network stack based on Synchronous Transmissions (ST). It is based on a middleware layer that enables to separate the concern of the timely execution of ST primitives from the implementation of the protocol logic.** *Thanks to this additional layer of abstraction,* Baloo *flexibly supports multiple ST primitives and significantly reduces the effort required to implement network layer protocols compared to a traditional stack design.*

All scenarios feature relatively low bandwidth requirements: up to 8 source nodes, generating packets of at most 64 bytes, at most once every 5 s. Moreover, packet generation is not synchronous across source nodes (*i.e.*, packets are released at different times for different nodes). These offsets between packet releases are random and unknown before runtime.

Given the sparsity of the packets to be transmitted, waiting to bundle packets into periodic rounds (*e.g.*, round period equals packet generation) would lead to an expected latency of about half the period (*i.e.*, multiple seconds). This is clearly not optimal.

Periodic case. Given our analysis, we propose the following protocol logic for the periodic scenarios:

- We schedule one *Baloo* round per source nodes per application period (*i.e.*, at most 8 rounds every 5 s).
- Our *Baloo* rounds resemble Crystal rounds [4]; first send a data packet, then send an acknowledgement. Such pair of slots is repeated until the data packet is successfully acknowledged.
- In the data dissemination scenarios, acknowledgements are collected efficiently using Chaos floods [6].
- To reduce latency, the *Baloo* host learns the source node offsets when the first packets are being sent. Each source node timestamps the release time of its data packet and the start time of the next round. The time difference is piggy-backed onto the data packet, which allows the host to adjust the epoch of the forthcoming rounds.
- Since each source gets its round per period, exactly one packet is expected in each round. Thus, all nodes terminate the round when they see the acknowledgement.
- To increase resilience to interference, we hop between a fixed list of channels following a predefined sequence, similarly as in Crystal [4].

Aperiodic case. From a protocol design perspective, the aperiodic case significantly differs from the periodic one. In the aperiodic case, the design decision essentially boils down to a trade-off between latency and energy-consumption: as one cannot predict when there might be a packet to send, one usually picks a period T at which the network wakes-up. This yields an expected latency of $T/2$. The smaller T, the better the latency and the more energy consumed.

For the data collection scenarios, we propose the following design: Each round starts with a "flag" slot. This slot is used by source nodes to announce whether or not they have a packet to send. If so, a data slot is executed; otherwise, the round terminates immediately. The host adds the (non)-acknowledgement in the control packet of the next round. Source nodes buffer their data packets until an ACK is received. As data generation is sparse, the flag slot allows significant energy savings for a small round period T.

In the data dissemination scenario, a dedicated ACK slot is required as there are multiple destinations. Like in the periodic case, Chaos floods [6] can be used.

4 Conclusions

The EWSN 2019 Dependability Competition has been the first "real" test case for the usability, flexibility, and performance of *Baloo* for designing novel network layer protocols based on synchronous transmissions. Our proposed solutions have been implemented within 10 weeks by master students (the two lead authors of this abstract) *whom had no prior knowledge of Glossy nor Baloo*. The code is open source and available through the *Baloo* project webpage [1].

5 References

[1] Baloo. http://www.romainjacob.net/research/baloo/, Dec. 2018.

[2] A. Escobar, F. Moreno, A. J. Cabrera, J. Garcia-Jimenez, F. J. Cruz, U. Ruiz, J. Klaue, A. Corona, D. Tati, and T. Meyerhoff. Competition: BigBangBus. In *Proceedings of the 2018 International Conference on Embedded Wireless Systems and Networks*, EWSN '18, pages 213–214, USA, 2018. Junction Publishing.

[3] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 73–84, Apr. 2011.

[4] T. Istomin, M. Trobinger, A. L. Murphy, and G. P. Picco. Interference-resilient Ultra-low Power Aperiodic Data Collection. In *Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '18, pages 84–95, Piscataway, NJ, USA, 2018. IEEE Press.

[5] R. Jacob, J. Bächli, R. Da Forno, and L. Thiele. Synchronous Transmissions Made Easy: Design Your Network Stack with Baloo. In *Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks*, To appear.

[6] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and Efficient All-to-all Data Sharing and In-network Processing at Scale. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, SenSys '13, pages 1:1–1:14, New York, NY, USA, 2013. ACM.

[7] R. Lim, R. Da Forno, F. Sutton, and L. Thiele. Competition: Robust Flooding Using Back-to-Back Synchronous Transmissions with Channel-Hopping. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, EWSN '17, pages 270–271, USA, 2017. Junction Publishing.

[8] P. Sommer and Y.-A. Pignolet. Competition: Dependable Network Flooding Using Glossy with Channel-Hopping. In *Proceedings of the 2016 International Conference on Embedded Wireless Systems and Networks*, EWSN '16, pages 303–303, USA, 2016. Junction Publishing.