Comparing the Viability of Different Communication Methods for Botball

Jeremy Sztavinovszki, Bernhard Klauninger, Sebastian Kawicher, Karl Dopplinger, Julian Kerer, Sven Oberwalder, Raphael Ackerl, Timon Koch
Höhere technische Bundes-Lehr- und Versuchsanstalt Wiener Neustadt
Department of Computer Science
2700 Wiener Neustadt, Austria

***Abstract*—This study focuses on comparing Wi-Fi and Bluetooth Low Energy for inter-robot communication in the educational robot competition *Botball*, with the aim of improving coordination and efficiency. Throughput, latency, and reliability under various conditions is critically examined, employing the Rust programming language for implementation due to its reliability and performance characteristics. This research aims to assist participants of the *European Conference on Educational Robotics* (ECER) in selecting the most suitable communication protocol for their robots, thereby improving their strategy effectiveness and competition performance.**

I. Introduction

Reflecting on past Botball tournaments shows that coordination between robots is a common issue. This can be as minor as one robot moving a game piece that the other robot needs later in the run, or as severe as the robots colliding with each other. Such issues often result in unexpected outcomes that may not satisfy the competing team. Over the years, numerous methods have been developed in Botball to solve these problems. Simple methods include synchronizing the robots by executing commands at a specified timestamp. More sophisticated ways of avoiding collisions use cameras [1]. The focus in this paper lies on the technologies that can be used in Botball. This study specifically compares the over-the-air communication methods Wi-Fi and Bluetooth Low Energy (BLE) in the environment of Botball. Other viable technologies for inter-robot communication include LoRa [2] and Zigbee [3], these are however not relevant in the Botball competition because the hardware on the Wombat only supports Ethernet, Wi-Fi, and Bluetooth [4]. Active communication between the robots enables better coordination. Which information is exchanged over these technologies is out of scope for this paper.

II. Methods

This paper heavily evolves around wireless communication technologies. For this reason most of the highlighted methods are related to the physical and data link layer of the OSI [5] model.

A. Wi-Fi

Wi-Fi is a vital technology in modern communication. It utilizes the 2.4 GHz and 5 GHz bands in the physical layer. Wi-Fi utilizes various technologies, including *Multiple-Input Multiple-Output* (MIMO) to enhance throughput and performance [6]. Additionally, it employs the family of *Wi-Fi Protected Access* (WPA) standards to ensure security [7].

B. Bluetooth Low Energy

Bluetooth Low Energy made its debut in the Bluetooth specification version 4.0, revolutionizing communications with energy-efficient, low-power connectivity [8]. BLE introduces key features ideal for battery-powered devices such as smart thermostats and the Wombat Controller. It utilizes a method called *advertising* to establish connections. Devices actively broadcast data packets to speed up discovery. *Frequency hopping*, a key innovation in over-the-air communication also used in BLE, involves rapid channel switching, reducing interference, and ensuring robust connections in diverse environments.

C. L2CAP

The *Logical Link Control and Adaptation Protocol* (L2CAP) is one of the protocols of the BLE stack [9]. It is responsible for multiplexing higher-layer BLE protocols over multiple connections and segmenting and reassembling data. It optimizes transmission efficiency while accommodating diverse application requirements. A program can directly use the L2CAP layer through BlueZ [10] and send data packets similarly to sending data through TCP sockets. This protocol has much less overhead than some of the higher-layer protocols.

D. TCP

The *Transmission Control Protocol* (TCP) [11] was developed by Vinton Cerf and Robert Kahn during the 1970s as part of the ARPANET [12] project. TCP is a connection-oriented transport protocol that guarantees reliable and ordered data transmission between devices. To establish a connection, TCP uses a three-way handshake. It maintains data integrity through segmentation, sequence numbers, and acknowledgments. TCP also includes flow control and congestion control mechanisms to optimize data transfer efficiency.

E. UDP

The *User Datagram Protocol* (UDP) was developed alongside TCP with a focus on simplicity and low overhead [13]. It operates as a connectionless protocol, enabling swift data transmission without the connection establishment and error recovery overhead of TCP. UDP is suitable for real-time applications such as multimedia streaming, however, it lacks sequence numbers and acknowledgments, meaning that it prioritizes speed over reliability.

F. Rust

Rust [14] is the programming language used to streamline the development and testing process of the benchmarks this paper covers. It is widely adopted across diverse programming domains and offers a multitude of compelling features. Its robust support for various architectures, efficient cross-compilation capabilities, and extensive library ecosystem make it particularly well-suited for our application. In this work, cross-compiling is significant due to the use of disparate CPU architectures. The Wombat Controller, which uses *aarch64/armv7-l* architecture, is different from *the x86_64* architecture typically used in modern computers. Rust provides the necessary versatility to handle these diverse architectural requirements seamlessly.

III. Useful Software Design Patterns for Network Programming

Network programming is a very difficult task because of the amount of possible errors to take into consideration and the process of synchronizing procedures taking place on multiple devices. There are, however, software design patterns that provide a way to handle this complexity. Following are some of the patterns made use of for the experiments in this paper.

A. The Actor Pattern

The Actor Pattern [15] is a pattern often used in the context of concurrent programming. An actor is a building block of a concurrent computation and can send and receive messages. Based on these messages, an actor can create more actors, mutate its internal state, and send a response. A handle is often implemented alongside the actor. The handle constructs the actor object and provides wrapper functions to hide the complexity of the message parsing going on in the background. An example would be hiding the complex process of sending a stream of data through UDP or handling requests with a TCP listener. This pattern is used in all following experiments to push outgoing data onto a message queue and process a queue of received packets.

B. The Factory Pattern

Factories [16] are a way of removing the complexity associated with instantiating objects. The Factory Pattern is either implemented by providing a function or creating an interface through which objects can be created. This pattern is used to reduce code duplication.

C. Result Monad and Railway Oriented Programming

A monad is a structure that wraps a state or an object and provides ways to interact with the contained state while handling possible errors [17]. The failing behavior can be represented by wrapping the state into a *Result Monad*. It can adopt either the error state or the intact state. To interact with the state, any function must go through a check to determine if the contained state is intact. If the inner object is intact, the function is applied, otherwise it is ignored. Many other programming languages offer similar implementations, such as *std::optional* in C++. This pattern is also known as railway-oriented programming because the running program can be thought of as two railways, with each function being able to switch from the intact track to the error track, but not the other way around.

IV. Testing Data Throughput and Reliability

In an ideal world, a communication method should provide a reliable and efficient means of transmitting data. In reality, either reliability or throughput has to be prioritized. To determine the best method of communication for a specific need and identify an all-rounder, two tests will be conducted. Performance and reliability are determined by testing under optimal and more realistic conditions. Both tests use an echo server that sends back any data sent to it. As many packets as possible are exchanged for 60 seconds. The packets contain a 64-bit sequence number in the first 8 bytes and are then filled with random data until the package reaches the size of the maximum transmission unit. During this process, the time at which the packet is sent and received is recorded. If the packet fails to return, it is considered lost. Additionally, packets that become corrupted during transit and do not contain a valid sequence number also contribute to packet loss.

A. Benchmarking Throughput and Reliability in an Optimal Environment

To ensure optimal test conditions, two Wombats are placed one game table length (or 240 centimeters) apart from each other. The experiment is conducted in a room with thick concrete walls and no other devices to prevent interference and ensure accurate results. The Wombat's Wi-Fi module operates on the 2.4 GHz frequency band and uses a channel width of 20 MHz. One Wombat acts as a Wi-Fi access point, and after the other Wombat connects to it, both the TCP and UDP benchmarks are run. Then, the Wi-Fi interface on both Wombats is turned off and the BLE tests are run using L2CAP.

B. Testing the Effects of Interference on Throughput and Reliability

Botball events often have many participants, which lead to interference when multiple devices broadcast using a shared medium. Interference occurs when signals collide, corrupting the data sent. While Wi-Fi and BLE have methods to counteract these errors, they come at a cost. To measure the effects of interference on Wi-Fi and BLE, we simulate it. This requires either many clients or a large antenna to saturate the entire signal spectrum. Due to the availability of Wi-Fi and BLE-enabled devices, such as the Wombat, the former method is more practical and cost-effective. The setup of the interference test is shown in Figure 1. Each device records sent and received data over time, providing insight into transfer rates and packet loss experienced by the devices.

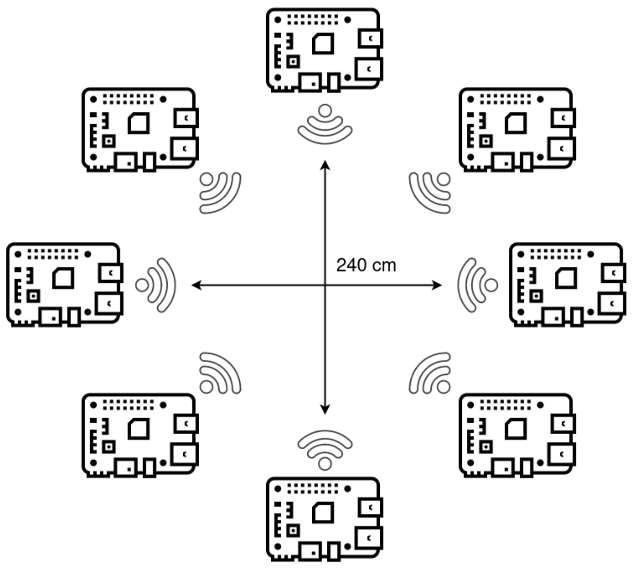

Fig. 1: Setup of the interference test

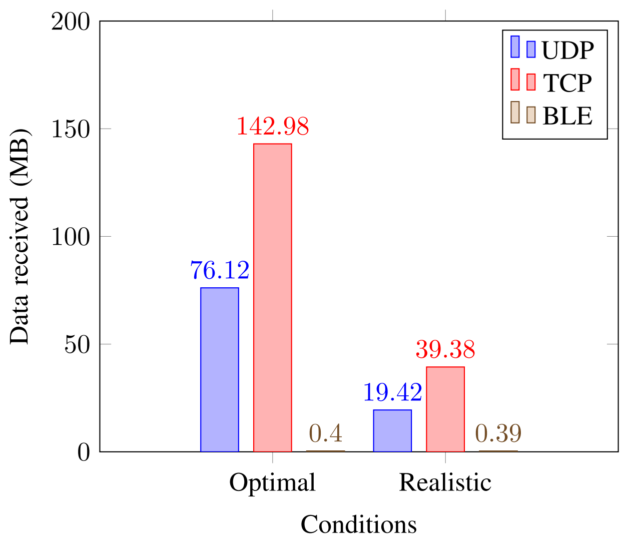

Fig. 2: Total data successfully transmitted in 60 seconds

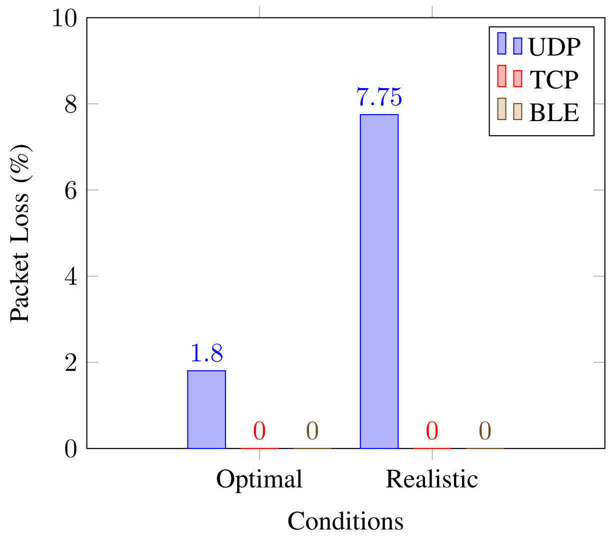

Fig. 3: Packet loss over 60 seconds

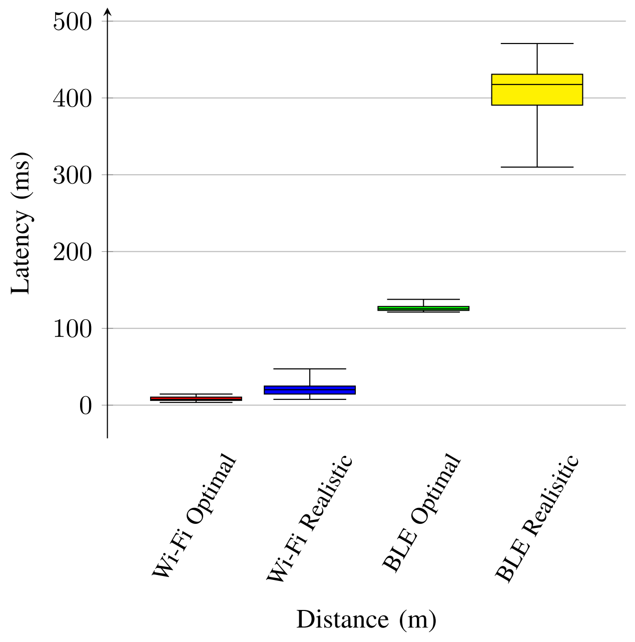

Fig. 4: Latency distribution for Wi-Fi and BLE over 100 packets each

V. Results

In Figure 2, Figure 3, and Figure 4 the results of the conducted experiments delineate the strengths and weaknesses of each communication method under various conditions.

A. Data Throughput

The data in Figure 2 show that TCP achieved the highest throughput, transmitting approximately 143 MB over 60 seconds, in contrast to BLE's 0.39 MB in the same time. This highlights the superior data handling capacity of TCP in optimal conditions. Under more realistic conditions with interference simulated, the throughput for both TCP and UDP declines, with TCP achieving 39.38 MB and UDP 19.42 MB. BLE's performance remained nearly constant at 0.39 MB, signifying its lower susceptibility to interference affecting its throughput.

B. Packet Loss

Figure 3 illustrates packet loss in optimal and realistic conditions. While UDP experiences a minor packet loss of 1.80%, TCP and BLE, on the other hand, show exceptional reliability with zero packet loss. Under the stress of simulated interference, UDP's packet loss increases to 7.75%, while TCP and BLE maintain their resilience with no packet loss.

This highlights the reliability of both TCP and BLE despite challenging conditions.

C. Latency

In terms of latency, as depicted in Figure 4, Wi-Fi employing the Internet Control Message Protocol (ICMP), situated one layer below TCP and UDP, demonstrates significantly lower average latency under ideal conditions, with an average of 8.62 ms compared to BLE's 122.20 ms. Noteworthy is that the 95th percentile of Wi-Fi's latency is 13.64 ms, while BLE's latency is 133.62 ms. The worst case performance that can be expected is highlighted by this metric. This pattern persists under practical conditions, where Wi-Fi's latency increases to 19.92 ms, and BLE's latency experiences a more pronounced rise to 412.81 ms. Under practical conditions, the 95th percentile of Wi-Fi's latency is 38.28 ms, while BLE's latency is 445.37 ms. The substantial increase in BLE latency under practical conditions underscores the impact of interference. It is important to note that Wi-Fi's low-latency performance is comparatively less affected by the simulated interference.

VI. Conclusion

In ideal circumstances, Wi-Fi shows to have superior throughput and lower latency compared to BLE, making it the preferred choice for tasks that require high data rates or time-sensitive operations. Both the TCP protocol and the BLE protocol exhibits negligible packet loss, highlighting their reliability for transmitting critical information despite BLE's lower data transmission capabilities.

Under conditions with realistic interference levels, TCP demonstrates resilience by maintaining packet integrity. In environments where interference is prevalent, TCP over Wi-Fi is a reliable method of communication. While BLE has limited throughput, it maintains reliability, making it useful in scenarios where energy efficiency and minimal data transmission are important. In Botball, where the amount of energy used for Wi-Fi can be neglected, it is recommended to not use BLE to transmit data. Since BLE uses the same 2.4 GHz bands as Wi-Fi it is also affected by interference. Because of the high amount of interference at tournaments, it is critical for teams to create backup plans for when data cannot be transmitted successfully.

For participants in the ECER, the findings encourage teams to carefully weigh the benefits and drawbacks of implementing bot-to-bot communication in their competitive designs. Specifically, teams can discern the optimal protocols to employ based on the specific requirements of their Botball strategies and the likely environmental conditions during the conference. This anticipation for interference is vital for enhancing the reliability and effectiveness of their robots in the dynamic environment of robotics competitions.

This study underlines the significance of considering environmental factors such as interference when designing and testing robot communication systems. As Botball teams strive for precision and efficiency in their robotic endeavors, the insights derived from this research offer a pathway to more effective and reliable robot-to-robot communication, ultimately enhancing the overall performance in competitions. While this study delineates the comparative efficacy of Wi-Fi and BLE in Botball, future research should explore the integration of adaptive protocols that could dynamically switch between these methods based on environmental conditions.

Acknowledgement

The authors would like to thank Michael Stifter, Harald R. Haberstroh, Jakob Eichberger, the HTBLuVA Wiener Neustadt, and all members of robo4you for their support throughout the process of writing this paper.

References

[1] Bernard Schmidt and Lihui Wang
"Depth camera based collision avoidance via active robot control"
(April 2014)

[2] The Things Network
"LoRa and LoRaWAN"
www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/
(accessed 11.02.2024)

[3] Connectivity Standards Alliance
"Zigbee"
csa-iot.org/all-solutions/zigbee/
(accessed 12.02.2024)

[4] Broadcom
"Wi-Fi 7 and dual-core Bluetooth combo chipset"
www.broadcom.com/products/wireless/wireless-lan-bluetooth/bcm4398
(accessed 11.02.2024)

[5] Day, J.D. and Zimmermann, H.
"The OSI reference model"
(December 1983)

[6] IEEE
"802.11n"
https://standards.ieee.org/ieee/802.11n/3952/
(accessed 12.02.2024)

[7] Wi-Fi Alliance
"WPA3™ Specification"
https://www.wi-fi.org/system/files
(accessed 12.02.2024)

[8] Bluetooth SIG
"Bluetooth Low Energy Specification"
https://www.bluetooth.com/specifications/specs/core-specification-5-3/
(accessed 12.02.2024)

[9] Bluetooth SIG
"Logical Link Control and Adaptation Protocol Specification"
https://www.bluetooth.com/wp-content/uploads/Files/Specification/HTML/Core-54/out/en/host/logical-link-control-and-adaptation-protocol-specification.html
(accessed 12.02.2024)

[10] BlueZ Project
"BlueZ - Official Linux Bluetooth protocol stack"
https://github.com/bluez
(accessed 12.02.2024)

[11] W. Eddy, Ed. MTI Systems
"RFC 9293: Transmission Control Protocol (TCP)"
https://www.ietf.org/rfc/rfc9293.html
(accessed 12.02.2024)

[12] Advanced Research Projects Agency
"ARPANET"
https://en.wikipedia.org/wiki/ARPANET
(accessed 12.02.2024)

[13] J. Postel
"RFC 768: User Datagram Protocol (UDP)"
https://www.ietf.org/rfc/rfc768.txt
(accessed 12.02.2024)

[14] Matsakis, Nicholas D. and Klock II, Felix S
"The rust language"
(published 2014)

[15] Carl Hewitt
"Actor Model of Computation: Scalable Robust Information Systems"
arXiv:1008.1459 [cs.PL]
(accessed 12.02.2024)

[16] Ross Harmes and Dustin Diaz
"The Factory Pattern"
https://doi.org/10.1007/978-1-4302-0496-1_7
(accessed 12.02.2024)

[17] Eric Silverberg
"Better architecture with Railway Oriented Programming"
https://medium.com/geekculture/better-architecture-with-railway-oriented-programming-ad4288a273ce
(accessed 12.02.2024)