**Abstract**— Biofeedback systems and applications in sports have great potential in accelerated motor learning and injury prevention. Traditional methods with terminal feedback are now being complemented by concurrent feedback methods. In sports, wireless communication systems are preferred. Acquiring and processing sensor data in challenging wireless channel conditions may prove demanding; for example, wireless communication with sensors attached to the body of the swimmer. To address such issues, we have designed and implemented a reliable communication protocol for cyclically interrupted channel, named *Block Selective Repeat* protocol. By using this protocol in biofeedback systems, we achieve fast and reliable transmission of sensor data and signals to the processing device and back to the biofeedback device. This protocol may prove particularly useful for real-time feedback systems and applications working in challenging environments, especially in such with cyclically interrupted radio channel. The final goal of our work is the development of real-time feedback systems and user-friendly applications for athletes and coaches that will help accelerate motor learning and improve their performance.

**Index Terms**— biofeedback system, cyclically interrupted channel, modified selective repeat protocol, real-time communication, reliable communication protocol.

**I. INTRODUCTION**

The main goal of our research is to develop a real-time sports biofeedback system, which is able to work in challenging environments, such as swimming pools, where the radio communication channel is cyclically interrupted. Channel interruptions; for example, when the swimmer is underwater lead to loss and corruption of the transmitted data. The focus of this paper is to develop a protocol that provides reliable communication even under such demanding circumstances.

When we started with the development of a real-time biofeedback system for swimming, we came across the problem of the cyclically interrupted channel, which means that we cyclically faced random times and lengths of channel interruptions. After a study of the reliable data transmission protocols, we soon realized that no protocols exist that would adequately address this problem. To the best of our knowledge, the majority of the previously published papers tackle underwater communication by implementing either wireless systems using very low frequency or optical or acoustic systems. None of them is supporting real-time data transmission. However, as we are working with real-time biofeedback systems in sports, high sampling frequencies and high data rates are necessities and become even more significant when channel is cyclically unavailable [1-4].

**II. METHODOLOGY**

As we want to provide coaches in the swimming pool with real-time biofeedback information, we must use wireless technologies with high bit rate and an appropriate range. When using commercially available wireless technologies, such as Bluetooth, Wi-Fi, and Packet radio, we are limited to high-frequency radio communication, typically between frequencies of 433 MHz and 2.4 GHz. Consequently, we are faced with a problem of the cyclically interrupted channel as high frequencies, used in such wireless technologies, are significantly degraded when penetrating the water [5-9]. For example, the water penetration depths measured in [6] are less than 1 m. Similarly, it is reported in [7] that the Berkeley Mica 2 Motes, an experimental platform in sensor networking, has an underwater transmission range of 120 cm at 433 MHz. Evaluating the available bit rate and range of various commercially available wireless technologies, and at the same time ensuring the flexibility of our real-time biofeedback applications, we decided to choose Wi-Fi technology. We implemented our protocol in the application layer of the Internet protocol stack. On the network and the transport layer, we use IP and UDP protocols, respectively, which are both connectionless and unreliable. We could not use the reliable TCP transport protocol, as we are not able to control its operation in the needed detail over the cyclically interrupted channel, i.e. if the channel is unavailable and the connection is lost, we will not be able to restore the communication. Therefore, we are faced with the challenge of developing an application layer communication protocol, which can ensure a reliable and efficient transfer of sensor signals and data over the cyclically interrupted channel.

When addressing this problem, we first studied the different existing reliable data link protocols that deal with unreliable channels. We studied the set of ARQ (Automatic Repeat Request) data link protocols: Stop and Wait, Go back N, and Selective Repeat. They handle errors and manage the retransmission of lost data to achieve a reliable data transmission [10-12]. It is crucial to
consider that only the lost or corrupted packets should be retransmitted in order to optimize efficiency. In addition, it should be ensured that no information at the receiver is duplicated. For a real understanding, and in order to choose the best option between the available ARQ protocols, we implemented and tested them all. We identified Selective Repeat as the most efficient starting point for our high-speed real-time sensor data transmission. The advantages of the proposed protocol are especially noticeable in the case of heavy, bursty, and mid- to long-time channel losses, i.e. in swimming [13].

III. SYSTEM IMPLEMENTATION

The implemented system consists of an Arduino based sensor system as a sender and a LabVIEW based system as the receiver. The system architecture is shown in Fig. 1.

A. Sender operation

Fig. 2 presents the flow diagram describing the operation of the sender. Before explaining the flow diagram in detail, it is important to define the parameters of the sending side of the protocol:

- **timerSendAgain** is a timer controlling the sending of the RR (Receiver Ready) packets, which will trigger the retransmission of the last packet sent. When this timer times out, the sender sends the RR to check whether the receiver is still active, the channel is still available, and the sequence number of the last packet received.
- **NAK** is the negative acknowledgement that triggers the retransmission of the lost or corrupted packet. One or several NAKs, as many as packets missing in the receiver buffer, will be placed at the beginning of the receiver response packet.
- **ACK** is the acknowledgement of the received packets. It is placed after all the NAKs (if there are no NAKs to be sent, the response packet contains only an ACK). ACK represents which is the next expected packet after the one with the highest sequence number.
- **T_s** is the sensor sampling time, which defines the sensor sample acquisition and recording rate.

As it can be seen from Fig. 2, every **T_s**, Arduino code is run. The first step is the sensors data acquisition from the LSM6 sensor. The data is composed of the 3DoF from the gyroscope and 3DoF from the accelerometer. Next, time left until the next **T_s** is measured to check, if there is enough time to execute the protocol operations.

The protocol then checks for the presence of incoming packets from the receiver. When a new response is obtained, the protocol divides it into smaller frames, each of them containing a NAK or an ACK. Each NAK triggers the retransmission of the individual lost packet, by reading it from the sender buffer that stores the packets that were sent but have not been acknowledged yet. Besides, the first NAK in the response packet implicitly acknowledges all the packets before it and the sender buffer window is updated accordingly. If the packet contains only an ACK, is then this ACK which updates the sender buffer.

After the protocol has dealt with the incoming packets from the receiver, it should send one or more packets containing new (unsent) sensor data. A new packet can only be sent if the window is “open”; in this case one or more 6DoF are read from the sensor buffer. As protocol can dynamically create different length packets, they are composed in regard to the available number of sensor samples available.

Moreover, it should carefully be considered that, if for a certain time there is no response from the receiver, **timerSendAgain** will expire. When the timeout occurs, a RR packet will be sent to check, if the communication is still working and the sequence number of the last packet received.

B. Receiver operation

Fig. 3 shows protocol operation of the receiver LabVIEW code when a packet arrives at the receiver. Parameter definition:
• **Sequence Number** is the sequential number of the packet and it is defined by the sensing side of the protocol.
• **Last in Sequence** is the last packet received and stored in order in the receiver buffer, i.e., before this one there are no packets missing in the receiver buffer.
• **Last Received** is the packet with the highest sequence number in the receiver buffer. Between *Last in Sequence* and *Last Received* can still be missing packets in the receiver buffer.
• **Expected Sequence Number** is the next packet after *Last Received* that is expected by the receiver.

As it can be seen from Fig. 3, a conditional structure first checks, if the received packets content is *RR*, which would mean that the timerSendAgain timer expired in the sender. If this condition is true, the sender will not send additional frames and this timeout will trigger an automatic response in the receiver. The receiver response is composed of as many *NAK* as there are missing packets at the receiver that the sender has to retransmit and an *ACK i* that denotes the next packet expected by the receiver (*Last Received* + 1). However, if this comparison is false, a data packet arrives and the protocol continues to process it in the following way.

The first step compares the **Sequence Number** of the received packet to the **Last in Sequence** parameter of the receiver. If the **Sequence Number** is lower than the **Last in Sequence**, the packet was already received and processed before, so it is already stored in the receiver buffer and it is discarded. Otherwise, it must be further processed. The next step in the flow diagram compares the **Sequence Number** in the packet with the **Expected Sequence Number**. Regarding this comparison result, there are two different scenarios:

a. **Sequence Number and Expected Sequence Number** are equal. The received packet is the one expected, so it should be stored in the receiver buffer, and all the data should be processed. **Last in Sequence** parameter should be updated.

b. **Sequence Number and Expected Sequence Number** are not equal. It is necessary to consider two different cases:

i. **Sequence Number** is higher than **Last Received**. One or more packets were lost or corrupted in the channel before the arrival of the actual packet. For this purpose, the protocol should first write *NE* (Non-Existent) in the missing packet(s) indexes of the receiver buffer and then the received packet is inserted into the appropriate index of the receiver buffer, regarding its packet sequence number. Then, **Last Received** parameter should be updated.

ii. **Sequence Number** is lower than **Last Received**. A packet that was lost before arrived to the receiver. The protocol should insert it into the correct receiver buffer index and update **Last in Sequence** to the next missing packet - 1, or in case that there are no any other missing packets, to the **Last Received**.

After that, the receiver will wait until the next packet arrival.

Fig. 4 shows the diagram of the receiver response construction and transmission process. Parameter definition:

- **Acknowledge (ACK) period** defines the response sending interval, i.e. how often a response packet will be sent from the receiver to the sender.
Initially, it should be mentioned that the response is not sent for every received packet. Such operation is not efficient and uses sender resources and channel time that will be necessary for sensor data packet transmission. Instead, the response is sent every ACK period. Every time that ACK period is reached, a new response packet is composed at the receiver and sent to the sender. To create the packet, the protocol checks various parameter values.

First, Last in Sequence is compared to Last Received. If Last in Sequence is lower than Last Received, one or more receiver buffer indexes contain NE, which means that one or more packets are missing. Then, starting at Last in Sequence position in the receiver buffer every index position \( i \) will be checked and when equal to NE, a NAK \( i \) will be appended to the response packet. When the counter \( i \) becomes equal to the Last Received, an ACK \( i \) will be appended to the end of the packet to indicate the last packet sequence number stored in the receiver buffer.

Otherwise, if Last in Sequence is equal to Last Received, there are no missing packets in the receiver buffer. In this situation, the response will be ACK \( i \), where \( i \) denotes the next packet expected.

Lastly, the response packet will be sent back to the sender and its structure will be NAK \( i \)SNACK \( j \)...SACK \( k \), as shown in Fig. 5.

\[
\text{NAK} \ i \ \$ \ \text{NAK} \ j \ \$ \ \ldots \ \$ \ \text{ACK} \ k
\]

Fig. 5 Response packet structure. There are as many NAKs as missing packets in the receiver buffer and they might or not be in a consecutive order (\( i \), \( j \) and \( k \) represent sequence numbers).

To be able to check the results in an easy and friendly way and to make sure of the correct protocol operation, we designed a testing tool (see Fig. 6) that shows the different parameter values of the receiver in real-time. In Fig. 6, packets 25, 26 and 27 are lost or corrupted, so the value for Last in Sequence is 24, while Last Received is 28. It can be easily checked that, the implemented buffer is circular, and its indexes are overwritten when new packets are received. Moreover, it can be observed the different packets length that are the consequence of dynamic packet creation process at the sender.

![Fig. 6 An example of the state of the receiver buffer after the packet with sequence number 29 is received. The figure also shows values of Last in Sequence, Last Received and Expected Sequence Number.](image)

IV. SIMULATION RESULTS AND DISCUSSION

As it was previously mentioned, the Selective Repeat protocol had the best performance for sports biofeedback systems with cyclically interrupted channel and therefore we used it as the base for the development of our own Block Selective Repeat protocol [10-12]. We implemented several modifications and added some functionality to be able to transmit the sensor data over the interrupted channel fast and efficiently using the underlying Wi-Fi, IP, and UDP protocols.

The first enhanced system feature is the ability to dynamically create packets in the sender. This characteristic improves efficiency and speed, as it enables sending a variable number of sensor readings in one packet. Every sampling time (\( T_s = 200 \) ms) we read data from sensors and store it in the sensor buffer. Regarding how many data from sensors are waiting to be sent in the mentioned buffer, we are able to include a different number of sensor samples in one packet. By this, we are able to create different packets lengths up to the maximum length of 548 bytes that is the limitation of the standard UDP receiver block in LabVIEW. 548 bytes allows us to send up to 17 DoF sensor samples in the one packet. Sender buffer holds all the packets that have not been acknowledged yet.

Moreover, sending a response every time when a new packet is received at the receiver, will not accomplish the real-time requirements in terms of delay. Such operation leads to the unnecessary waste of processing time in the sender, which is a scarce resource. Because of the above-mentioned, we decided to modify the response pattern of the standard Selective Repeat protocol. The receiver now generates a response packet, which contains the combination of negative and positive acknowledgements, and periodically sending it back to the sender. The response packet structure is shown in Fig. 5.

We tested our protocol by using different values for the response or ACK period. We realized that this parameter is directly related to the window and buffer size in the sender. This means that the larger the ACK period is in the receiver, the larger the window and buffer size should be in the sender. As we are working with real-time systems, time is critical and we cannot deal with large delays. Shorter delays require smaller window sizes. The most suitable value for the window length and ACK period should therefore be chosen very carefully.

In Fig. 7 we present results for the scenario when the channel is constantly available. It can be noticed that the delay for different window lengths \{2, 4, 8, 16\} is bounded until a certain value of the ACK period. A real-time performance of the protocol can be expected only for the ACK periods below those values. For example, for the window length of 8 packets, the delay is bounded below the ACK period of 1.6 s.

Fig. 8 presents the testing results when the channel is cyclically unavailable. This testing was performed at different window lengths \{2, 4, 8, 16\}. As we were trying to get results similar to the expected ones at the field tests, we made the channel available for three seconds and then unavailable for the same period. It can be noticed that the best efficiency is achieved for ACK period of around one second. For longer ACK periods the efficiency decreases. We also confirmed that the efficiency is very similar for
the different window size values. However, as it was mentioned before, low window sizes may lead to unacceptable delays.

![Fig. 7 Delay at different window sizes {2, 4, 8, 16} and ACK period for the scenario when the channel is constantly available.](image)

Our protocol outperforms the protocols proposed in previous works in terms of efficiency and restoration after channel losses on the cyclically interrupted channel, what makes it the best choice for real-time biofeedback systems and applications operating in challenging conditions.

## V. CONCLUSIONS

The final aim of this work is to develop and implement a device, which could be used in real-time biofeedback systems in sports, even in hard conditions when the channel is cyclically unavailable. Developing this fully working system may lead to future manufacturing of real-time sports, rehabilitation, and health-care devices used not only for the augmentation of the professional athlete’s performances but also for amateurs who might find it as a powerful tool to boost the performance of their daily sports activities.

The main benefits of our protocol are the fast and reliable transmission of sensor data and signals to the processing device and back to the biofeedback device along with the capability of operating in challenging environments, especially in such with cyclically interrupted radio channel.

Another idea we conceive is developing a mobile app for coaches (trainers) and athletes to receive relevant information about the performed activities and interact with it in a friendly way, without the need for specialized technical knowledge. For example, they would only need to set up parameters and configuration values in an easy and straightforward way to get a transcript of the athlete’s current performance and possible historical records.

Our future research will also be directed into the improvement of the efficiency and performance of the system.

## REFERENCES


