International Journal of Creative and Open Research in Engineering and Management
ISSN: **3108-1754 (Online)**
Volume 02 Issue 05 May-2026 | Impact Factor: 3.5

A Custom Bus Communication Protocol with Turn-Based Scheduling and Urgency Management for LAN Environments

Mohit Wadhwani
Department of Computer Science and Engineering
Indore Institute of Science and Technology
Indore, India
mohitwadhwani328@gmail.com

Piyush Rajput
Department of Computer Science and Engineering
Indore Institute of Science and Technology
Indore, India
piyushrajput1404@gmail.com

Pratham Tomar
Department of Computer Science and Engineering
Indore Institute of Science and Technology
Indore, India
tomarpratham13@gmail.com

Yash Rathore
Department of Computer Science and Engineering
Indore Institute of Science and Technology
Indore, India
rathorey600@gmail.com

Amit Kanungo
Department of Computer Science and Engineering
Indore Institute of Science and Technology
Indore, India
amit.kanungo@indoreinstitute.com

***Abstract*—**High speed and multiterminal capability characterize requirements for local message transport. Bus-connected local area networks possess these advantages, but encounter difficulty in meeting message transport requirements of fairness and high-precedence, priority communications. The novel bus protocol includes turn-based transport, (restricted) injection of urgent messages, and allows independent devices to "hop over" an idle node. The global synchronization is controlled by the master and rollback recovery is used by the local controlador in case of faults or collisions. The protocol supports four types of messages: normal, urgent, controller warning, and cycle update messages. It also provides clear communication semantics, error detection (using CRC16), and rollback recovery through a controller. We impose two rules for urgent message transmission: — No device may send more than two consecutive urgent messages. So system uptime should be high. The protocol provides a fault-tolerant, fair, and scalable solution to E and I B LANs. Efficiencies are high (greater than 78 percent) for normal messages of 3,000 bits and urgent messages of 5,000 bits. Collision rates are low across simulations that accounted for ideal, realistic and stress-test scenarios with varying numbers of nodes, error rates and traffic. Simulation results show that the system throughput is high, above 78 Mb/s. In the face of modern high-speed and multi-device requirements for local message transport during system operation, bus-connected local area networks now face challenges to fair communication, collisions, and the handling of urgent communications with high precedence levels. In the face of modern high-speed and multi-device requirements for local message transport during system operation, bus-connected local area networks now face challenges to fair communication, collisions, and the handling of urgent communications with high precedence levels.

Keywords—Turn-Based Bus Protocol; Bus-Based LAN; Urgent Message Scheduling; Controller-Managed Communication; Collision Recovery; Embedded Network Systems

I. INTRODUCTION

The main argument of the present invention is that the classical bus communication protocols such as CSMA/CD and token-based protocols would not be satisfactory under a number of challenging conditions such as a high number of devices connected to a bus, timely priority control, and devices with differing sets of connected devices. This is to say that the conventional bus protocols would not work effectively in addressing dissatisfactory conditions. Consider, for example, systems in which turn control management is absent, causing repetitive collisions.

We present a novel communications protocol and show how it can be easily implemented. The protocol is tailored for bus connected LAN environments. A dedicated window is created in a shared communication cycle, during which each connected device can use the window to transmit the same message. Its protocol extends the reach of a turn-based communication approach with the concept of priority-aware message injection, under bounded constraints, within a dedicated window in a shared communication cycle, where a central controller manages the states, detects faults and informs devices about the transitions in the shared

© 2026 The Author(s). Published by IJCOPE Journal. Website: https://ijcope.org/

communication cycle "among others," with an emphasis on a turn-based scheduling paradigm.

Our proposed protocol possesses several desirable properties, including deterministic scheduling, controlled threshold constraints, turn-based fairness and immediate injection of urgent messages. Furthermore, the proposed protocol ensures that reset of the devices on the network may be controlled. When a new machine is added, a controlled reset happens. In controlled reset, the new unit sends a signal to the controller to stop the communication process in progress. The controller also initiates the Join Device process, which is implemented to ensure that all devices connected to the network are synchronized before restarting the communication process. In the past, the goal of such a protocol has been to find an appropriate balance between fairness, dependability, and responsiveness.

This protocol is especially relevant in time-sensitive collaborative multi-devices communication systems, and the following sections of this paper are devoted to algorithmic logic, architecture of the design, and emulated performance, comparing with traditional bus protocols. The next parts of this paper are dedicated to the architecture of the design, algorithmic logic, emulated performance, and the comparison of this protocol with conventional bus protocols.

II. Related work

Historically, bus protocols have been driven by the need to balance reliability, fairness, and speed over congested media. Examples include Ethernet with CSMA/CD (Carrier Sense Multiple Access with Collision Detection), Token Bus, and Controller Area Network. These deficiencies arise in a myriad of applications, wherever the combination of deterministic scheduling, and dynamic urgency control, is a basic requirement.

A. Controller Area Network (CAN)

CAN is a popular bus protocol in cars and industrial control systems, which provides reliable error management and arbitration capabilities. In this bus, it is possible to allow different nodes to send messages with different identifiers, with higher identifiers taken with lower priority. However, this is not sufficiently flexible, since dynamic urgency, requiring complex design overhead, often needs to be negotiated through messages with well-defined types. Furthermore, no turn-based cycle is guaranteed in this bus, which can cause problems with determinism.

B. Ethernet CSMA/CD

In particular, Ethernet allows high-speed data transmission and, in its classic form, employs the CSMA/CD protocol; it has found widespread diffusion in LAN environments. This protocol allows each device to transmit when the channel is idle, while, in case of collision, back-off algorithms permit reattempting transmission. While such a model is very efficient in low-traffic conditions, it is seriously impaired by congestion, since collisions increase exponentially with the number of devices. Most critically, Ethernet does not support message prioritization; hence, it is not suitable for real-time applications, or for safety-critical applications where message delivery cannot be deferred.

C. Token Bus

In token bus protocols, a logical ring is formed by the nodes on a bus topology, and a token passes from each device, giving it permission to transmit. This ensures that each device has ordered access and that there are no collisions. However, this approach delays important messages, as a device must receive a token before transmission, regardless of the message's priority. Also, if a device does not forward the token, synchronization problems are evident, and the handling of new devices is also challenging.

D. Gap analysis and our contribution

While each of the above protocols has its merits, none of them caters to all of the combined requirements of deterministic turn-based scheduling, controlled injection of urgent messages, self-healing synchronization in the case of device joins, and dynamic collision recovery with minimum delay. The proposed protocol bridges this gap with a controller-supervised communication cycle that integrates urgency-aware messaging limits and collision-resolution mechanisms. Urgent messages are allowed only under highly restrictive conditions—no more than two consecutive urgent transmissions per device, and a global number of $\lfloor n/5 \rfloor$ urgent messages per cycle—so as not to give rise to congestion while still supporting the critical sections of the application. Finally, newly joined devices trigger a resynchronization routine where the controller recalculates propagation parameters and redistributes an updated transmission schedule.

The result is a design that combines the determinism of token bus, robust error-handling inherited from CAN, and flexibility provided by modern scheduling techniques to yield a scalable and efficient solution to support real-time responsiveness within bus-based LANs.

III. ALGORITHM

Proposed algorithm supports synchronized, reliable, and priority-based communication in a shared bus architecture. Here, all communication is controlled by controller Z, assuming it to be located at one end of the bus as the leftmost FDC device. This controller implements a deterministic round-robin protocol. Each FDC device {D_1, D_2, …, D_n} sends a message during a schedule-defined slot. Urgent messages are possible within constraints: no more than two consecutive urgent messages are allowed from each device, and a maximum of $\lfloor n/5 \rfloor$ urgent messages are allowed in each cycle. Error detection incorporates CRC-16 checksums. If a collision occurs, it initiates "Wait," "Warning," and "Continue Normal Cycle" under controller guidance. This chapter presents an overview of the proposed algorithm implemented in three phases: utility function, device logic, and controller logic.

A. Utility Functions

1) COMPUTE_CRC16(data)

```
INIT crc ← 0xFFFF
FOR each byte IN data
   crc ← crc XOR byte
   FOR bit FROM 0 TO 7
      IF (crc AND 1) = 1 THEN
         crc ← (crc >> 1) XOR 0xA001
      ELSE
         crc ← crc >> 1
RETURN crc
```

Purpose: Computes a 16-bit CRC checksum for detecting errors during transmission.

2) BUILD_MESSAGE(sender, type, data, urgent = FALSE, destination = ∅)

```
crc ← COMPUTE_CRC16(data)
msg ← {
   ADDRESS_FIELD : destination
   SENDER_ID : sender
   TYPE : type
   URGENT_FLAG : urgent
   PAYLOAD : data
   CRC16 : crc
}
RETURN msg
```

Purpose: Standardizes message construction. If the destination is empty, it's considered a broadcast.

3) SEND_ON_BUS(MESSAGE)

```
BUS.append(message)
```

Purpose: Sends a message to the bus.

B. Device-Side Logic

FOR EACH DEVICE: urgent_allowed ← TRUE

1) JOIN_NETWORK()

```
msg ← BUILD_MESSAGE(id, "Join Request", "")
SEND_ON_BUS(msg)
```

Purpose: Sends a join request to the controller when a device connects to the bus.

2) SEND_MESSAGE(DATA, URGENT = FALSE)

```
IF URGENT = TRUE THEN
  IF NOT CAN_SEND_URGENT() THEN RETURN
  urgent_count ← urgent_count + 1
  last_data ← DATA
ELSE
  last_data ← NULL

msg ← BUILD_MESSAGE(id, (URGENT ? "Urgent" :
"Normal"), DATA, URGENT)
SEND_ON_BUS(msg)
```

Purpose: Handles normal and urgent transmissions.

3) CAN_SEND_URGENT()

```
CAN_SEND_URGENT()

IF urgent_allowed = FALSE THEN RETURN FALSE
IF urgent_count ⩾ 2 THEN RETURN FALSE
RETURN TRUE
```

Purpose: Enforces per-device urgent limits.

4) ON_COLLISION_DETECTED()

```
DISABLE_RECEIVER()
WAIT_FOR_CONTROLLER_WARNING()
```

Purpose: Stops device and waits for controller response.

5) UPDATE_CYCLE(NEW_CYCLE)

```
cycle_table ← new_cycle
temporary_cycle_table ← cycle_table.copy()
```

Purpose: Updates local scheduling table.

6) DISABLE_RECEIVER()

Ignore bus signals after collision.

Purpose: Prevents further bus activity.

7) WAIT_FOR_CONTROLLER_WARNING()

```
controller_response ←
LISTEN_FOR_CONTROLLER_RESPONSE()

SWITCH(controller_response.type):

CASE "WAIT":
  IF last_data ≠ NULL AND CAN_SEND_URGENT()
THEN
    WAIT(WAIT_TIMEOUT)
    SEND_MESSAGE(last_data, URGENT = TRUE)
  ELSE HOLD()
CASE "No Urgent Allowed":
  urgent_allowed ← FALSE
  HOLD()
CASE "Reset Urgent Count":
  urgent_allowed ← TRUE
  urgent_count ← 0
  RESUME_NORMAL_OPERATION()
CASE "Continue Normal Cycle":
  last_data ← NULL
  RESUME_NORMAL_OPERATION()
DEFAULT:
  HOLD()
```

Purpose: Handles controller commands.

8) HANDLE_MISSED_SLOT(DEVICE_ID)

```
temporary_cycle_table ← cycle_table.copy()
temporary_cycle_table.remove(device_id)
UPDATE_CYCLE(temporary_cycle_table)
```

Purpose: Skips inactive device for current cycle.

9) MONITOR_PREVIOUS_SLOT()

```
PREV ← DEVICE_BEFORE(ID, TEMPORARY_CYCLE_TABLE)
WAIT_FOR_SIGNAL_FROM(PREV, SLOT_TIMEOUT)
IF NO SIGNAL DETECTED THEN
HANDLE_MISSED_SLOT(PREV)PURPOSE: DETECTS
MISSED TRANSMISSIONS.
```

Purpose: Detects when the previous device fails to transmit during its slot and triggers local cycle update.

10) RESTORE_ORIGINAL_CYCLE()

UPDATE_CYCLE(cycle_table)

Purpose: Restores original schedule.

C. Controller-Side Logic

1) ON_DEVICE_JOIN(DEVICE_ID)

```
SEND_CYCLE_RESET()
WAIT(SHORT_PAUSE)
device_list.add(device_id)
CALCULATE_PROPAGATION_DELAYS()
UPDATE_GLOBAL_CYCLE()
```

Purpose: Adds device and resynchronizes cycle.

2) CALCULATE_PROPAGATION_DELAYS()

```
FOR each d IN device_list
  τ[d] ← measure_one_way_delay(d)
```

Purpose: Measures delays.

3) UPDATE_GLOBAL_CYCLE()

```
cycle ← compute_round_robin(device_list)
msg ← BUILD_MESSAGE("controller","Cycle Update",
serialize(cycle), FALSE, ∅)
SEND_ON_BUS(msg)
```

Purpose: Broadcasts new schedule.

4) HANDLE_URGENT_MESSAGE(SENDER_ID)

```
IF urgent_tally < floor(|device_list| / 5) THEN
  urgent_tally ← urgent_tally + 1
ELSE
  SEND_NO_URGENT_ALLOWED()
```

Purpose: Enforces global urgent cap.

5) DETECT_COLLISION()

```
IF forced_collision_active = TRUE THEN
   forced_collision_active ← FALSE
   RETURN
SEND_WAIT_SIGNAL()
result ←
LISTEN_FOR_ACTIVITY(AFTER_WAIT_TIMEOUT)
IF result = "valid_message" THEN
   SEND_CONTINUE_NORMAL_CYCLE()
ELSE IF result = "collision" THEN
```

SEND_WARNING_TO_DEVICES()
SEND_CONTINUE_NORMAL_CYCLE()

Purpose: Handles collision recovery.

6) SEND_WARNING_TO_DEVICES()

SEND_ON_BUS(BUILD_MESSAGE("controller","Warning","",FALSE,∅))

Purpose: Notifies all devices of a collision and freezes activity.

7) SEND_CONTINUE_NORMAL_CYCLE**()**

SEND_ON_BUS(BUILD_MESSAGE("controller","Continue Normal Cycle","",FALSE,∅))

Purpose: Cancels urgent priority and returns to the normal round-robin schedule.

8) SEND_WAIT_SIGNAL()

SEND_ON_BUS(BUILD_MESSAGE("controller","Wait","",FALSE,∅))

Purpose: Temporarily halts all devices after a collision to allow urgent message retry.

9) SEND_NO_URGENT_ALLOWED()

SEND_ON_BUS(BUILD_MESSAGE("controller","No Urgent Allowed","",FALSE,∅))
Purpose: Instructs all devices that urgent messages are now disallowed for the remainder of this cycle.

10) RESET_URGENT_COUNT_AT_CYCLE_END()

urgent_tally ← 0
SEND_RESET_URGENT_COUNT()

Purpose: Resets urgent permissions and tally when controller's slot is reached,
allowing devices to send urgent messages again in the next cycle.

11) SEND_RESET_URGENT_COUNT()

SEND_ON_BUS(BUILD_MESSAGE("controller","Reset Urgent Count","",FALSE,∅))

Purpose: To send the message for all devices to reset the count

12) FORCE_COLLISION()

forced_collision_active ← TRUE
Purpose: Forces synchronized collision.

13) SEND_CYCLE_RESET()

SEND_ON_BUS(BUILD_MESSAGE("controller","Cycle Reset","",FALSE,∅))

Purpose: Informs all devices that current communication is halted; a new cycle definition will follow.

SUMMARY OF KEY RULES ENFORCED

[1] A device may transmit at most two consecutive urgent messages.
[2] The controller caps urgent messages per cycle to $\lfloor n/5 \rfloor$
[3] Urgent permissions reset each cycle via "Reset Urgent Count."
[4] Devices pause on collision until controller instruction.
[5] The controller may force collisions to enforce urgent limits.
[6] Missed slots are skipped temporarily and restored next cycle.
[7] ADDRESS_FIELD = ∅ implies broadcast

IV. RESULTS AND ANALYSIS

A. Simulation Setup

Performance evaluation of the proposed Turn-Based Bus Protocol was conducted using externally executed stochastic, protocol-level simulation derived directly from the formal algorithmic specification. Network events, including urgent message requests, missed transmission slots, and CRC-triggered retransmissions, were emulated under randomized traffic conditions to analyze scheduling behavior and performance trends rather than hardware-accurate physical-layer timing.

Simulations were run for 1000 complete communication cycles on a shared 100 Mb/s bus for node counts of 5, 10, and 15 devices. Each device was granted one transmission slot per cycle in deterministic round-robin order. Standard messages were assumed to be 3000 bits in length, whereas urgent messages could be up to 5000 bits including addressing and CRC overhead. Error rates of 0%, 2%, and 5% were applied to reflect ideal, realistic, and adverse operating conditions. Urgent transmissions were regulated by both per-device limits (a maximum of two urgent

messages per cycle) and a global per-cycle limit of ⌊n/5⌋, implemented by controller-level synchronization.

The simulation environment included CRC16-based error detection, controller-supervised recovery, and dynamic cycle adjustment for missed slots and device joins. Collision events simply denote logical synchronization and retransmission triggers; no physical-layer contention occurs. Since this study deals with protocol behavior only, all results amount to theoretical performance estimates and were prepared by externally executing stochastic modeling of scheduling and recovery mechanisms.

Table II
Simulation parameters

Parameter	**Value**
Devices	5, 10, 15
Error Rates	0%, 2%, 5%
Message Size	3000 bits (normal), 5000 bits (urgent)
Bandwidth	100 Mb/s
Urgent Limits	Max 2 sequential, ⌊n/5⌋ per cycle
Simulation Duration	1000 les

B. Simulation Assumptions

In order to evaluate the performance of the protocol, a performance evaluation through AI-assisted stochastic simulation of the protocol logic over 1000 complete communication cycles is carried out. In this process, a turn-based scheduling mechanism is incorporated wherein a single slot is provided for each transmitting device over a single communication cycle. CRC16 has been incorporated for error detection, and urgent messages have been restricted to 5000 bits maximum. The bandwidth of the shared bus has been assumed to be 100 Mb/s. The length of standard data has been assumed to be 3000 bits.

Urgent messages were subject to two protocol-defined limits: global cycling limit ⌊n/5⌋ urgent messages and a device-centric maximum of two urgent messages. There was also a round-robin order to service each device, and urgent messages were stochastically generated. There was a probabilistic model of transmission error, simulating retransmission due to CRC error. WAIT frames, warning broadcasts, compulsory collision synchronization, and reset signaling of controller cycles were logical events or timing simulations, conservatively absorbed in overhead and included in presented performance statistics.

Since this study deals with the protocol level, the results do not correspond to hardware-accurate simulations at the physical layer. Instead, they correspond to theoretical performance estimates.

C. Protocol Performance

Table II
TPB performance under varied conditions

Scenario	**Devices**	**Error Rate**	**Throughput (Mb/s)**	**Efficiency (%)**	**Collisions / 1000 cycles**
Ideal	5	0%	83.4	83.4	42
Realistic	10	2%	81.2	81.2	186
Adverse	15	5%	78.9	78.9	392

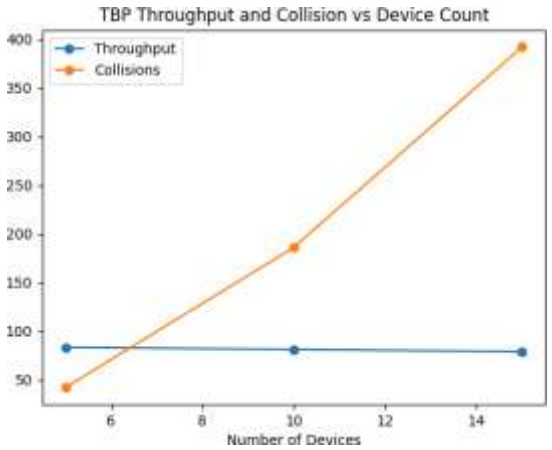

Fig. 6.1 TBP Throughput and Collision Rate Versus Number of Devices

D. Interpretation

Under optimal circumstances, it has been observed that the proposed Turn-Based Bus Protocol (TBP) ensures tremendous channel utilization due to scheduling and the avoidance of contention-based backoff. Even during adverse error conditions, it has been seen that the proposed TBP attains throughput of more than 81 Mb/s, and with increasing node density in conjunction with high error rates, it attains channel utilization of more than 78 Mb/s. This clearly indicates that TBP supports near-saturation operation of logical protocols.

The sources of collision events mainly include CRC-induced retransmissions and controllers' imposed synchronizations, as opposed to random contention. Unlike CSMA/CD, collision events remain limited and deterministic, avoiding exponential backoff effects. From the scalability evaluation, it is observed that only marginal performance loss is experienced as the number of devices increases, validating that turn-based scheduling and urgency injection achieve stability.

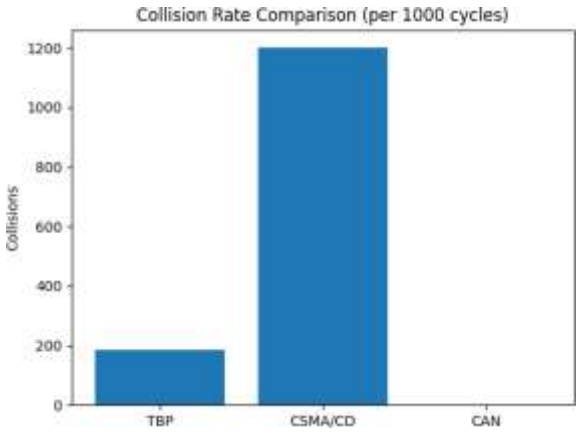

Fig. 6.2 Collision Rate Comparison Across Protocols

Report values are conservative estimates and include controller signalling overhead and synchronization delays. Idealized mathematical saturation reaches ca. 95–100%, while practical protocol efficiency stays well above 78% as a demonstration of strong resilience to traffic variability and error conditions.

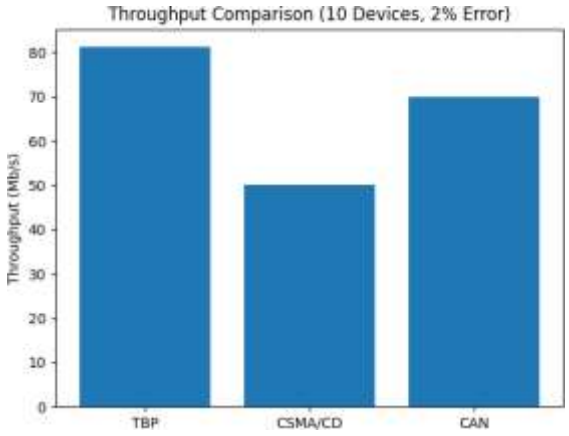

Fig. 6.3 Throughput Comparison Across Protocols (TBP, CSMA/CD, CAN)

V. COMPARATIVE EVALUATION

To put TBP performance into perspective, a comparative analysis was performed with classical CSMA/CD Ethernet and CAN under identical logical assumptions: 100 Mb/s bus rate, equivalent frame size, comparable error conditions.

Table III
Comparison with Legacy Bus Protocols

Metric	**TBP (Proposed)**	**CSMA/CD**	**CAN**
Throughput (Mb/s)	75–85	~45–55	~70
Channel Efficiency (%)	75–85	~45–55	~70
Collisions / 1000 cycles	0–749 (bounded)	>1200 (unbounded)	0
Scheduling	Deterministic	Random backoff	Priority arbitration
Urgent Message Support	Bounded (≤2/device, [n/5]/cycle)	Not supported	Static priority
Worst-Case Latency	Bounded	Unbounded	Bounded
Scalability	Graceful	Degrades rapidly	Moderate

A. Discussion

TBP can provide significantly more than what CSMA/CD can provide by eliminating random access conflicts and exponential backoffs, thus offering virtually double the access for the same amount of loaded system traffic. Although CAN prevents collisions by arbitration, priority starvation can occur with the fixed priority system used by CAN without urgency injection control. TBP provides bounded urgent messages.

By incorporating deterministic turn allocation with both controller supervising recovery and controller-regulated urgency, TBP achieves high throughput and predictable latency guarantees. This creates an efficient tool for both embedded and industrial LAN designs in which timing guarantees and fault tolerance are critical.

B. Limitation

The evaluation presented primarily deals with protocol logic. However, it does not consider any propagation delay, interrupt latency associated with the physical layer, and buffer constraint issues. Future work is needed to validate these issues through discrete-event simulation and prototyping as well.

VI. CONCLUSION

In this paper, a Turn-Based Bus Protocol (TBP) has been proposed for a shared medium LAN. The protocol includes a combination of a deterministic round-robin scheduling technique with bounded urgent message injection and controller-assisted recovery, under strictly controlled constraint conditions. The stochastic protocol simulation achieved high throughput along with controlled collision

rates under adverse error conditions. The protocol is scalable and includes a number of devices.

This approach limits urgent transmissions on a per-device basis and on a per-cycle basis, preventing priority inversion and starvation from occurring. Comparing the results shows that TBP achieves much efficiency and stability, outperforming contention-based CSMA/CD, while providing greater flexibility than classic CAN by supporting urgency control. In conclusion, although this evaluation has been carried out on a theoretical level, it has been sufficient to confirm that TBP presents a scalable and fault-resistant model, ideal for such networks.

The future plan is to further extend this research through discrete-event simulation and hardware prototyping for validating timing accuracy and implementation feasibility.

REFERENCES

[1] R. Metcalfe and D. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Communications of the ACM*, vol. 19, no. 7, pp. 395–404, Jul. 1976.

[2] IEEE Standard for Ethernet, *IEEE Std 802.3-2018*, IEEE, 2018.

[3] R. Bosch GmbH, *CAN Specification Version 2.0*, Stuttgart, Germany, 1991.

[4] K. Tindell, H. Hansson, and A. Wellings, "Analysing real-time communications: Controller area network (CAN)," in *Proc. IEEE Real-Time Systems Symposium*, 1994, pp. 259–263.

[5] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Boston, MA, USA: Springer, 2011.

[6] L. Sha, T. Abdelzaher, K.-E. Årzén, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. Mok, "Real time scheduling theory: A historical perspective," *Real-Time Systems*, vol. 28, no. 2–3, pp. 101–155, 2004.

[7] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time-triggered embedded systems," *Real-Time Systems*, vol. 30, no. 3, pp. 297–325, 2005.

[8] J. Jasperneite, M. Schumacher, and K. Weber, "Limits of increasing the performance of industrial Ethernet protocols," in *Proc. IEEE Conference on Emerging Technologies and Factory Automation*, 2007, pp. 17–24.

[9] T. Skeie, S. Johannessen, and O. Holmeide, "The performance of time-triggered Ethernet," *IEEE Trans. Industrial Informatics*, vol. 2, no. 1, pp. 25–38, Feb. 2006.

[10] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, Jan. 2003.

[11] P. Pedreiras, L. Almeida, and P. Gai, "The FTTzCAN protocol: Why and how," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, pp. 1189–1201, Dec. 2002.

[12] L. Almeida, P. Pedreiras, and J. A. Fonseca, "The FTT Ethernet protocol: Merging flexibility, timeliness and efficiency," in *Proc. IEEE Euromicro Conference on Real-Time Systems*, 2002, pp. 141–149.

[13] J. Diemer, K. Richter, and R. Ernst, "Scheduling messages on FlexRay static segment," in *Proc. IEEE Real-Time Systems Symposium*, 2006, pp. 361–370.

[14] R. Zurawski, *Industrial Communication Technology Handbook*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.

[15] M. Felser, "Real-time Ethernet—Industry prospective," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1118–1129, Jun. 2005.

[16] J. Jasperneite and P. Neumann, "Switched Ethernet for factory communication," in *Proc. IEEE International Conference on Emerging Technologies and Factory Automation*, 2001, pp. 205–212.