STARC: Decentralized Coordination Primitive on Low-Power IoT Devices for Autonomous Intersection Management

N.B. When citing this work, cite the original published paper.

(article starts on next page)

Article

STARC: Decentralized Coordination Primitive on Low-Power IoT Devices for Autonomous Intersection Management

Patrick Rathje [1,*], Valentin Poirot [1,2] and Olaf Landsiedel [1,2]

1   Department of Computer Science, Distributed Systems, Kiel University, 24118 Kiel, Germany;
    vpo@informatik.uni-kiel.de (V.P.); ol@informatik.uni-kiel.de (O.L.)
2   Computer Science and Engineering, Computer and Network Systems, Chalmers University of Technology,
    412 96 Gothenburg, Sweden
*   Correspondence: pra@informatik.uni-kiel.de

Abstract: Wireless communication is an essential element within Intelligent Transportation Systems and motivates new approaches to intersection management, allowing safer and more efficient road usage. With lives at stake, wireless protocols should be readily available and guarantee safe coordination for all involved traffic participants, even in the presence of radio failures. This work introduces STARC, a coordination primitive for safe, decentralized resource coordination. Using STARC, traffic participants can safely coordinate at intersections despite unreliable radio environments and without a central entity or infrastructure. Unlike other methods that require costly and energy-consuming platforms, STARC utilizes affordable and efficient Internet of Things devices that connect cars, bicycles, electric scooters, pedestrians, and cyclists. For communication, STARC utilizes low-power IEEE 802.15.4 radios and Synchronous Transmissions for multi-hop communication. In addition, the protocol provides distributed transaction, election, and handover mechanisms for decentralized, thus cost-efficient, deployments. While STARC's coordination remains resource-agnostic, this work presents and evaluates STARC in a roadside scenario. Our simulations have shown that using STARC at intersections leads to safer and more efficient vehicle coordination. We found that average waiting times can be reduced by up to 50% compared to using a fixed traffic light schedule in situations with fewer than 1000 vehicles per hour. Additionally, we design platooning on top of STARC, improving scalability and outperforming static traffic lights even at traffic loads exceeding 1000 vehicles per hour.

Keywords: decentralized intersection management; Synchronous Transmissions; low-power vehicular ad hoc networks; internet of vehicles

1. Introduction

Intelligent Transportation Systems and the Internet of Vehicles (IoV) lay the foundations for new ways of achieving efficient mobility [1,2]. Their adoption reduces traffic delays, fuel consumption, and greenhouse gas emissions [3,4]. At the same time, vehicle-to-everything (V2X) communication minimizes infrastructure costs by replacing infrastructures like traffic signs and traffic lights [3,5]. In this field, protocols usually depend on cellular network coverage or central servers for coordination [3,4]. Yet, central management would be unreasonable for all intersections due to high infrastructure and maintenance costs [6,7]. Complementary, decentralized approaches rely solely on communication between the participants, requiring no auxiliary infrastructure [6,8]. However, most systems prioritize high-bandwidth technologies like IEEE 802.11p or 5G device-to-device (D2D), which require platforms with significant computing and energy resources.

Meanwhile, Internet of Things (IoT) devices cost a few dollars and run on AA batteries. For one, the TelosB Sky with an MSP430 chipset features a 4 MHz CPU with 10 kB of RAM and 250 kbps wireless communication bandwidth based on the IEEE 802.15.4 standard [9]. By implementing protocols that support power-efficient devices, everyone on

the road, including pedestrians, e-scooters, and bikes, can benefit from intelligent transportation's safety and efficiency advantages. As a result, we recognize the importance of developing coordination protocols that are distributed, inherently safe, and can operate on low-cost, battery-powered devices. Hence, this work brings safe and decentralized resource coordination to low-power wireless devices with restricted computing capabilities.

Challenges. Our target devices' limited resources starkly contrast the challenges posed by unreliable wireless transmissions, complex routing, and network membership in a time-sensitive environment. As safety is our top priority, we need to analyze and evaluate our solution thoroughly, ensuring it remains safe even in the event of radio failures and changes in topologies.

Approach. We introduce STARC: Synchronous Transmissions for Autonomous Resource Coordination. STARC uses inexpensive, low-power wireless radios to enable energy-efficient and safe coordination in a decentralized manner. For wireless communication, STARC operates on Synchrotron, an energy-efficient and low-latency synchronous transmission approach that enables seamless all-to-all communication. In addition, STARC includes transaction semantics, ensuring that all participants coordinate and commit to a shared access pattern with exclusive access to their resources. Our design and implementation for IEEE 802.15.4 radios bring STARC on devices with limited energy and processing power. Managing an intersection, we divide the intersection area into distinct areas and define each area as a separate resource. Traffic participants then use STARC to coordinate their access to these intersection parts, ensuring safe crossings due to STARC's inherent safety guarantees. After a car passes through the intersection, it releases its resources and exits the communication network, allowing the next vehicle to proceed.

Contributions. This paper finalizes our work which was partially presented at ITCVT 2020 [10]. Extending beyond the previous paper, this work features an expanded protocol description, an entirely new safety analysis, and in-depth evaluations that cover essential aspects of the communication primitive, like scalability and the effect of different platoon sizes. Beyond several extensions in Section 4, this work incorporates sections that are not present in the workshop version, namely Sections 3, 4.3, 4.4, 5.1, 5.2 and 5.5, as well as Section 6. Overall, its contributions are:

- In the context of autonomous intersection management, we introduce STARC, a decentralized coordination protocol tailored to resource-constrained devices.
- STARC offers fault-tolerant, decentralized coordination and transaction semantics in dynamic topologies and unreliable radio environments.
- We show that STARC is safe by design and does not allow concurrent access to the same resource, i.e., lanes of an intersection.
- We create STARC specifically for IoT platforms with limited resources, making it accessible to all users. Furthermore, we release our implementation and simulation suite to the public (Publicly available at: https://github.com/ds-kiel/starc, accessed on 10 July 2023).
- We assess the performance of STARC by simulating an intersection and analyzing its efficiency, network performance, and ability to handle interference and radio failures. Based on Synchronous Transmissions, STARC accommodates more nodes with minimal overhead and ensures safety despite radio failures. Compared to traditional traffic lights, STARC's users experience shorter delays, but only for traffic volumes up to 1000 vehicles per hour. However, with the addition of platooning, STARC scales to high-traffic scenarios.

Outline. This work covers the background information on Synchrotron and Synchronous Transmissions in Section 2. Section 3 describes the related work while Section 4 presents the design of STARC, followed by the evaluation results in Section 5. Our reflections on this work reside in Section 6, and the work concludes with Section 7.

2. Background

Intelligent Transportation Systems (ITS) employ computer systems to improve vehicle traffic, providing features such as brake assistants and crash and congestion warnings [1,11]. Vehicles can even form platoons to save fuel and increase road usage [11]. These advanced capabilities require communication and coordination between vehicles (V2V), infrastructure (V2I), and pedestrians (V2P) [5].

There are different protocols available for wireless communication. Protocols typically rely on IEEE 802.11p [12] (like WAVE), while others use cellular protocols like LTE or 5G [13,14]. In contrast, IEEE 802.15.4 is a wireless protocol designed explicitly for low-cost and low-bandwidth communication and is commonly used in the IoT domain [15]. This protocol operates within a range of 100 m in the 2.4 GHz band and has a bandwidth of 250 kbps. The payload size is typically 127 Bytes but can reach 2047 Bytes [15,16]. Beside its use in the IoT domain, IEEE 802.15.4 blends IoT and ITS, providing short-range communication for the Internet of Vehicles (IoV) [?]. The following section introduces Synchrotron, the communication substrate used by STARC.

Synchronous Communication Using Synchrotron

Synchrotron is a wireless communication method that combines all-to-all data sharing and in-network processing, enabling interaction between all nodes in low-power wireless networks [16]. STARC is based on Synchrotron's IEEE 802.15.4 implementation; however, other studies demonstrate that Synchronous Transmissions are possible for Bluetooth Low Energy [18,19] and Ultra-wideband [20]. Synchrotron's all-to-all communication inherently supports multiple hops and mobile devices and requires no need to address individual nodes. Furthermore, it integrates channel-hopping and cryptographic primitives for a robust protocol. Fundamentally, Synchrotron employs two techniques: Synchronous Transmissions and data aggregation.

Synchronous Transmissions. Using Synchrotron, nodes communicate in separate rounds divided into multiple slots. This synchronized structure lets nodes listen for or transmit packets during every slot. Even if several nodes transmit simultaneously, the precise time synchronization allows them to receive and decode packets accurately, thanks to the capture effect [21]. In IEEE 802.15.4, for decoding, a receiver must receive a signal at least 3 dBm stronger than the combined strength of all other signals. Additionally, the signal must arrive within 160 µs after the first signal is received [22]. To meet the timing requirement, the nodes constantly synchronize their clocks. Additionally, the nodes shut off their radio between rounds to conserve energy.

Data Aggregation. At the beginning of each round, a node initializes its local value. When a packet is received, the node combines the received and local information using an operator specific to the application. The node then marks its participation and transmits the result to others (see Figure 1). This approach offers a way to simultaneously perform collection, processing, and sharing. When the merging process finishes, the nodes rapidly retransmit. Because they transmit the same data in synchrony, the packets are less prone to causing destructive interference at the receiver's side, improving the spread of the result [23].
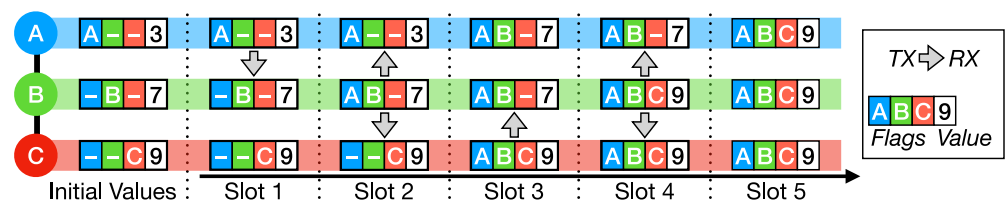
Figure 1. Example of a maximum calculation with Synchrotron: The nodes A, B, and C begin by setting their local value and flags. Node A starts the first round by transmitting in the first slot. In the subsequent slots, all nodes mark their participation and apply the maximum function to merge the values. Once all nodes have participated and incorporated their values, they learn the maximum.

3. Related Work

Managing an intersection can either be carried out in a centralized or decentralized manner. Centralized solutions require central agents or infrastructure. For example, Dresner and Stone propose a centralized approach for autonomous vehicles [3]. In their reservation-based protocol, vehicles rely on a central service to reserve their way over the intersection with a specific arrival time window. The resulting costs make centralized approaches at every intersection uneconomical [6,7]. Thus, decentralized techniques are developed in which the participants manage the intersection cooperatively.

Vanmiddlesworth et al. present a decentralized coordination protocol based on claims [6]. The cars in this approach keep broadcasting their claims for the intersection. The car with the strongest claim is given priority to drive over the crossing. These claims include the car's intended path and the estimated arrival time. If a vehicle cannot arrive within the estimated time window, it cancels its claim using another message. Although this approach shows twice as large a throughput as ordinary stop signs in low-traffic scenarios, the protocol does not ensure safety in case of communication failures.

Wang et al. apply IEEE 802.15.4 in the context of IoV [24]. They investigate the successful delivery probability of non-synchronous broadcasts. Their proposed model dynamically adapts the broadcasting rate to achieve a successful delivery rate of over 90% but has no safety guarantees.

Virtual Traffic Lights (VTL) utilize wireless transmissions instead of physical traffic lights [8]. Vehicles with conflicting directions choose one car to act as the coordinator. This coordinator acts like a virtual traffic light and shares the schedules of the entire intersection. When it is permitted to drive, the coordinating vehicle hands over the coordination to another car. According to Ferreira et al., the VTL approach can potentially decrease CO_2 emissions by as much as 18% [25]. In addition, VTL enhances the driving experience by decreasing both waiting time and the average time spent traveling [26]. Sommer et al. enhance the VTL solution by adjusting the phase length in response to the number of cars waiting [27].

Hassan and Rakha's protocol reduces the messages sent [7]. Non-leading vehicles only communicate with their immediate neighbors in front and behind. Like the estimated arrival time, information on subsequent cars is gathered and merged for each lane separately to create a lane-specific schedule. One leading vehicle aggregates those partial schedules and combines them into an intersection-wide plan propagating back through the lanes. Priority is given to longer queues to avoid congestion. This approach reduces the number of needed messages, but the work does not evaluate the effect of failures.

Naumann et al. separate the intersection into critical regions. Their proposed algorithm uses semaphores to ensure that only one vehicle is allowed in each region [28]. A vehicle reserves all regions it wants to pass before entering the intersection and frees an area once it passed it. Only one car holds the send-token, i.e., the power to change the semaphore states, and only unreserved semaphores or the own semaphores may be altered. This restriction ensures that the access is, in fact, exclusive, and at most one vehicle believes that it reserved a region. Each car passes the token onto the next vehicle using a ring structure (token-ring) to guarantee that every participant eventually receives access to the token. They evaluate the performance with three simulated vehicles but do not describe how the vehicles would recover the token in case of network failures.

Within this work, participants employ STARC and assemble a crossing schedule based on a network-wide agreement similar to a two-phase-commit. STARC expects all participants to participate and approve the schedule, ensuring it is safe and valid. Also, STARC provides fairness properties by allowing each participant to cross the intersection eventually. Additionally, all road participants can benefit from the protocol as our implementation targets low-power wireless radios.

4. Design

In this section, we introduce the design of Synchronous Transmissions for Autonomous Resource Coordination, i.e., our coordination protocol STARC. At the coordination level, STARC is inherently resource-agnostic and applies to all coordination scenarios. Within this work, however, we introduce STARC in the context of autonomous intersection management and consider the problem of safe intersection management as the coordination of mutual access to shared resources. In this case, parts of an intersection represent the shared resources. We use vehicles for illustrative purposes, but STARC applies to all road entities, including cyclists and pedestrians.

In the following, Sections 4.1 and 4.2 cover the coordination process and the handover support in more depth. We then analyze the protocol's safety in Section 4.3. Finally, we illustrate the generalization of STARC with platooning in Section 4.4.

Overview. The STARC protocol ensures safe resource access, i.e., intersection crossing, without requiring a central infrastructure. Instead, STARC utilizes local, low-power wireless communication that relies on Synchronous Transmissions for efficient and reliable propagation. At the same time, we employ transaction semantics to guarantee safety even in the case of radio failures. The protocol is divided into two parts: (1) reservation-based movement coordination and (2) decentralized leader election and handover support (cf. Figure 2a).

System Assumptions. This work assumes all participants have IEEE 802.15.4 radios and a unique identifier (ID). For our road scenario, vehicles can access position measurement devices such as GPS and information about the intersections, including lane layout, size, and possible directions. Each car has some (limited) computing capabilities and can plan a trajectory to follow.

Failure Model. We assume that communication is unreliable: messages may have arbitrary delays or become lost entirely. Under this assumption, it is not possible to guarantee both safety and liveness. So instead, we design STARC to ensure safety despite radio failures but require that transmissions are eventually successful for liveness. In addition, we assume crash-recovery failures for the devices running the STARC protocol: devices may crash and are not reachable but eventually recover and participate again. However, we require the vehicles to follow their trajectories and stay in their lanes. The final and significant assumption is that all participants behave honestly and follow the protocol without any intention of disrupting it. This is known as non-Byzantine behavior.

(**a**)   (**b**)   (**c**)
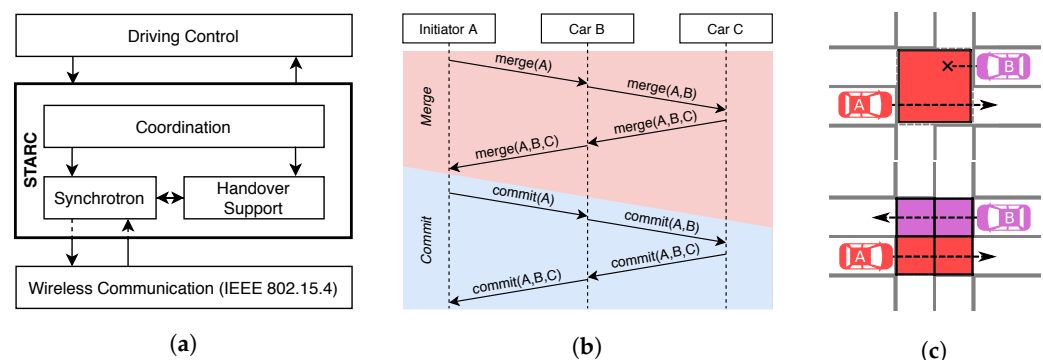
Figure 2. (**a**) STARC brings coordination atop Synchrotron and features mechanisms for joining, leaving, leader election, and handover. STARC serves as middleware; thus, autonomous and human-controlled vehicles and pedestrians can coordinate with STARC. (**b**) The elected leader initiates a coordination round. The first merge phase collects and merges individual resource requests. Afterward, when all devices participate, the leader transfers to the commit phase, where the participants disseminate and establish the final, merged schedule. (**c**) The intersection divides into a grid of tiles. Before crossing, vehicles reserve all tiles on their way. In this example, instead of a 1×1 grid (top), a 2×2 grid (bottom) enables parallel crossings. (**a**) Architectural overview. (**b**) Schematic coordination round. (**c**) Granularity example.

4.1. Distributed Resource Coordination

Using STARC, each vehicle tries to reserve its way over the intersection before crossing. Without a central service, the participants rely on STARC and exchange messages with their neighbors for distributed coordination. Therefore, STARC sends requests for resources, e.g., the required areas of the intersection, to all present entities and collects acknowledgments as confirmation to prevent conflicts and crashes. A resource becomes reserved when confirmed by everyone. This section introduces this coordination process.

Intersection Setup. To manage the intersection as different resources, we split it into a grid of $n \times n, n \in \mathbb{N}_+$ even tiles [3,29]. Each tile is labeled with a unique resource index and covers a specific area. We assume that every vehicle is aware of the tile layout of the intersection. Figure 2 illustrates this partitioning.

Tile Reservation. When a vehicle wants to pass through an intersection, it carefully plans its path. The car then identifies the parts of the intersection, i.e., the specific tiles that are passed when crossing, and attempts to reserve every tile on its path, see Figure 2. The resulting request contains the individual tile respectively resource indices, seeking unique access to the respective areas. In addition, a request includes the specific identifier and priority of the vehicle, defining a total order of the reservation inquiries. For conventional vehicles, our priority system is based on the arrival time, which promotes fairness by prioritizing those who arrive first [28]: if a car comes later, it receives a lower priority. At the same time, STARC inherently supports prioritizing emergency vehicles as priorities are set by the participants and not by the STARC protocol.

Because a granted reservation in STARC does not employ a notion of time, the vehicles do not require precise timings, i.e., coordination with STARC guarantees safety even when cars take longer than planned to pass through the intersection. However, a vehicle may only start its journey after successfully reserving its needed tiles [28]. As the car exits the intersection, it releases its reserved resources, allowing other vehicles to reserve them.

Our approach is based on Synchrotron, which is specifically designed to facilitate simultaneous data sharing and processing. Synchrotron divides communication into rounds and splits a round into synchronized slots for the actual data transmissions. At its heart, a vehicle applies a custom operator whenever new data are received in a Synchrotron slot, combining its local state with the received information. This operator is therefore applied every time a device receives a Synchrotron packet and defines the behavior of STARC. We devise a Synchrotron round that simultaneously gathers and merges resource requests. As Figure 2 illustrates, this coordination round is divided into two distinct phases:

1. Merge phase: The participants collect and merge their resource requests into a consistent schedule.
2. Commit phase: Once all members participate in the merge phase, this second phase contains the final resource assignment, i.e., the intersection-wide schedule, and distributes it through the network.

Merge Phase. During the merge phase, a specialized merge operator composes requests. Requests are collected and merged at the same time. Any resource conflicts that arise are resolved based on the respective priorities of each request. This operator is not affected by the order in which requests are received, and it builds partial resource assignments among the involved nodes in a decentralized manner. An elected node (see Section 4.2) starts the communication round in the merge phase with an initial transmission broadcasting its request. Participation flags track the progress within each packet. The merge phase concludes once all members participate, all requests are collected, and conflicts are resolved.

Requests are tracked for each tile separately; this creates a grid of requests where each tile is assigned a vehicle ID with the highest priority, if available. Each conflicting request is resolved on a per-tile basis, based on its assigned priority. Figure 3 illustrates an example merge phase and the commutative property of the merge operator.
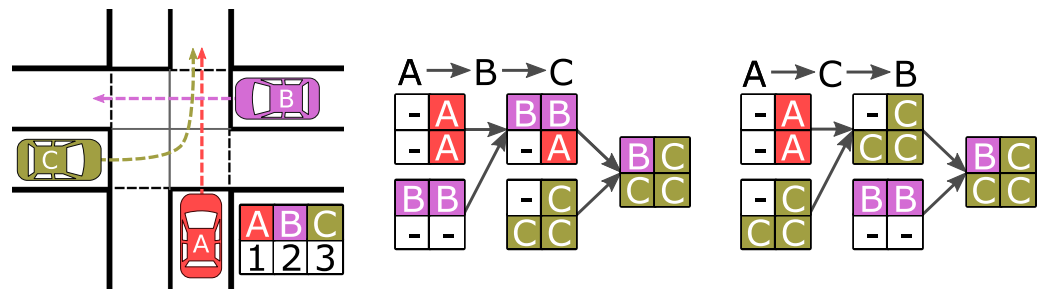
Figure 3. Merge operator example application: Cars A, B, and C try to reserve their paths over the intersection using STARC (left). The priority is given to vehicle C, which overrides the other's requests during the merge phase. STARC applies a commutative merge operator to handle conflicting reservations based on priorities. The merge order A-B-C results in the same grid as A-C-B. The merge process is carried out for each tile individually; no boundaries need to be checked.

Commit Phase. Once the individual requests are collected and merged, the devices should commit to the pre-computed grid of reservations. While the participation flags suggest that all nodes were involved in the merging phase, Synchrotron cannot guarantee that they are all aware of the final schedule. For example, two vehicles with conflicting directions can only be certain that their paths are safe by knowing each other's request. Due to this reason, we include a second phase: the commit phase. The leader in charge initiates the commit phase and acknowledges all members' participation. The leader marks the packet as a commit phase transmission for this second phase and clears participation flags. In this phase, the final resource assignment, i.e., the schedule for the entire intersection, propagates through the network, and devices cannot change the requests.

When a node that is currently in the merge phase receives a commit packet, it transitions to the new commit phase and takes on the received reservation grid, disregarding any incomplete or local state. Again, the node acknowledges its participation by setting a flag to indicate that it has received the commit. Afterward, nodes disseminate the packet. The marked flags serve as a progress indicator. Using the commit phase, STARC ensures that all members participate in the chosen, unique schedule. Due to retransmissions and constructive interference within Synchrotron, all nodes are probably notified about the plan toward the end of the round. Afterward, the participants turn off their radios until the beginning of the next round to save energy.

In general, those transaction semantics serve as the basis for ensuring safety with STARC: for a vehicle to proceed, it must obtain agreement from all other cars on the path. Monotonically increasing commit numbers detect missed commits on every node [16].

Crossing Schedule. At the end of each round, vehicles check the status of their reservations as provided by STARC. If there was a commit, the car proceeds to verify the status of its resource request, i.e., the status of each reserved tile along the designated route. Only if all are granted to the vehicle is its request accepted. Based on the prioritization, it follows that all other requests either attempted to book different tiles or had a lower level of importance. As the second phase only initiates once all parties have participated, the commit guarantees that all requests are combined, resulting in a unified schedule encompassing the entire intersection.

If a request for a vehicle is approved, the node still participates in merge phases but marks its requests as "passing" for the following rounds. In any conflicts that may arise in subsequent rounds, this type of request priority holds the highest priority and takes precedence. Passing vehicles thus request and reserve their path in each round repeatedly as they have the highest priority. This way, the request grid merges the reservation requests and approved reservations of both passing and waiting vehicles. Vehicle conflicts are avoided as long as they abide by the protocol and do not reserve additional tiles.

Once the vehicle has cleared the intersection, it releases the tiles by not reserving with high priority; other vehicles can reserve them again. Moreover, each passing vehicle can directly free already passed tiles. This process necessitates the vehicle's involvement in each subsequent round, with the identical requests being sent every time. Yet, there is no requirement to retain any information about the requests of other vehicles. In the worst case, a car can reconstruct its state based on its position on the road. Handling device or radio failures in the reservation process is uncomplicated as there is no need to save reservation states, ensuring a fail-safe protocol.

4.2. Handover Support, Leader Election, and Group Membership

Vehicles arrive and eventually leave the intersection area. STARC requires that every relevant participant joins an intersection-specific network to participate in the coordination process. This way, the network members know all vehicles they might collide with.

In STARC, a leader manages the network membership and assigns network-specific IDs to entities that want to join the network, e.g., the vehicles. STARC expects the participation of all members and requires them to free all resources before leaving the network, i.e., when vehicles have passed the intersection. If a vehicle departs without leaving the network, the protocol cannot proceed to the commit phase, hindering further progress and ultimately blocking the protocol. Leaving the network is also performed explicitly; otherwise, the network could not decide whether a node left or, e.g., crashed.

Group Membership. Nodes need to join and leave the STARC network dynamically. An arriving device starts listening for packets, as each packet contains low-level parameters, such as the beginning of the next round. With this knowledge, the node acts as a forwarder in the next round but is not yet a member: the device follows the designated policy for transmission and combines incomplete schedules. However, it cannot request any resources until it has joined. If no network is present, the node starts the STARC protocol as the leader node. Once the vehicle crosses the intersection, it attempts to exit the network.

We integrate the join and leave protocol into the coordination phase: during the merge phase, nodes can join and leave the network using designated slots in the packet. A node issues join requests by placing its unique identifier into an empty join slot inside the packet, while leaving is handled by simple flags. If the number of join slots is insufficient for the number of concurrently joining nodes, the merge operator discards excessive join requests with smaller vehicle IDs. Those slots are disseminated and merged automatically, eventually reaching the leader. The leader checks for potential join and leave requests at the beginning of the commit phase. It then assigns free network IDs to the individual join requests and simultaneously removes leaving nodes from the network. The resulting commit packet explicitly confirms the join and leave and assigns the resulting network ID for each join slot. As soon as the nodes receive the commit packet, they extract and directly use their obtained network ID to acknowledge the reception. The leader executes joining nodes before removing any other, allowing accurate process tracking throughout the commit phase. The network then expects the participation of the freshly joined nodes in upcoming rounds, whereas the left nodes should refrain from participating anymore. Ultimately, the network configuration may only change within a commit: thus, STARC detects inconsistencies in the membership using commit numbers.

Rejoining. Even though nodes retransmit packets in Synchrotron, packets can become lost due to unreliable or unpredictable wireless environments. Non-participation in the merge phase blocks the round completely or delays the join process for nodes that still need to join the network. However, not receiving the commit message can lead to inconsistencies in the network configuration. STARC, therefore, employs monotonically increasing commit numbers. The detection is a straightforward comparison: if the received number is higher than the node's locally saved one, it missed a commit. While a node can recover its reservation state, it might have joined or left the network but missed the corresponding confirmation; it hence still tries to join or leave the network waiting for the leader's confirmation.

A successful leave without the reception of its confirmation is especially dangerous: as soon as two nodes participate with the same network ID STARC can not guarantee that the reservations are safe: two vehicles could think that they reserved the same tiles. We thus require nodes to rejoin the network if they detect inconsistencies. They try to join the network, as usual. Then, the leader additionally checks if a join slot contains a vehicle ID already part of the network. The leader propagates the assigned network ID in separate rejoin slots through the network. On reception, the nodes use this network ID to take part in the merge phase again. This way, STARC handles nodes that operate in an outdated network configuration or are recovering from failure.

Leader Election and Handover. The leader manages the network but needs to handover to a different node as soon as it wants to leave the network. This handover data contain the assignment of node IDs to network IDs of the joined nodes. Whenever a leader wants to hand over, it starts election rounds instead of coordination rounds to determine the succeeding leader. If another leader is found, the new leader takes over and confirms the handover from its predecessor and the leave of the previous leader. An election round contains two phases: The network first determines the potential leader, then, the new leader starts a commit phase, accepting its election. After that, the old leader starts election rounds until it is the only member in the network or the elected one confirms the take-over and the leave. If no other member in the network exists, the leader leaves without a handover.

The members do not participate in those election rounds with their usual reservation priority. Instead, we introduce a separate, so-called election priority. By default, STARC uses the latest arrival time as the highest election priority (plus the respective number of desired resources). The current leader and other leaving nodes participate with the lowest priority possible. All nodes merge received packets and keep the ID with the highest election priority (using the ID as a tie-breaker, highest ID first; see Figure 1 for an example of maximum computation in Synchrotron). After the participation of all members, the packets contain the ID of the new potential leader. When the newly elected node receives the corresponding packet, it starts the commit phase of the election round. The old leader issues a leave request with the election round, allowing it to leave the intersection once the new leader confirms it with the commit. If the network re-elects the current leader, all other joined nodes must also be trying to leave. The elected leader then handles the leaving of nodes (except its own) but does not support joining new nodes (rejoining is still handled). When a different leader is elected, it again increases the commit number. With this increase, all aware nodes ignore packets with lower numbers, including the commit number from the former leader; this prevents any progress of the old network configuration [30]. The network thus converges to the new commit number as the new leader eventually rejoins every node that did not receive the election commit; this includes any old leaders that did not receive their leave acknowledgment. The new leader continues with regular coordination rounds after the election.

4.3. Safety Analysis

Safety is critical in the vehicle domain and a significant concern for STARC. However, STARC can only guarantee safety if the participants stick to the protocol and reserved resources: willful crashes or driving errors can not be prevented. To analyze the safety properties of STARC, we assume a communication model in which the messages have arbitrary delays or could be lost entirely. Also, the communicating devices may crash but recover eventually. In this context, safety means that two participants never enter the same part of the intersection at any point. In the presence of network failures, the STARC protocol builds on a two-phase commit with monotonically increasing commit numbers. Algorithm 1 presents an overview of STARC's slotted behavior and commit numbering. For reference, we add the algorithmic details within the Appendix A. We first show the safety property assuming all vehicles have joined the network and then further argue that dynamically joining and leaving the network does not harm this safety property.

Algorithm 1 Round overview with configuration handling

[H]

Require: $isLeader \in \{true, false\}, numCommits \in \mathbb{N}_0$

$txData \leftarrow begin\_round()$ ▷ Initialize local data
$tx \leftarrow isLeader$ ▷ leader starts each round
for each slot in round **do**
    Turn on radio at synchronized slot time
    if tx **then**
        Broadcast $(numCommits, txData)$
    else
        Listen for packet
        if $(rxNumCommits, rxData)$ received **then**
            if $rxNumCommits = numCommits$ **then**
                $handle\_packet(txData, rxData)$
                $tx \leftarrow true$ if $txData$ changed   ▷ merged into txData
            else if $rxNumCommits < numCommits$ **then**
                $tx \leftarrow true$ ▷ RX packet outdated, inform about new state
      ▷ Forwarders adapt lower numbers in first reception as they would otherwise disrupt
new networks.
            else if $rxNumCommits > numCommits$ **then**
                $memberID \leftarrow null$ ▷ Outdated network view, force rejoin
                $shouldJoin \leftarrow true$
                $numCommits \leftarrow rxNumCommits$
                $txData \leftarrow rxData$
                $tx \leftarrow true$
            end if
        else if Unsuccessful receptions exceed random threshold **then**
            $tx \leftarrow true$
        else if $txData.commit$ and all members received packet **then**
            Retransmit for some slots, then turn off radio
        end if
    end if
    Turn off radio until next slot
end for
if $txData.commit$ **then**   ▷ Commit is set by leader during round
    $numCommits \leftarrow numCommits + 1$ ▷ Increased at end for easy comparison
    if $txData.type = "coordination"$ **then**   ▷ Coordination successful
        Check $txData.roundData$ for state of reservations
        Notify about granted reservations
    end if
end if

Scheduling. Our merge operator (see Section 4.1) is order-invariant and idempotent. It assigns the IDs with the highest priority to each resource/tile, preventing duplicate assignments. Because all nodes must participate in the merge phase with distinct priorities (ID as tie-breaker) and the conflicts are directly resolved, the resulting resource assignment is conflict-free and inherently safe. However, we must also ensure that no other vehicle is present at the reserved location. We hence require that cars may only start once they have secured their full path over the intersection. Conflicts between passing vehicles are impossible; we forbid passing vehicles to reserve new tiles. Further, crossing vehicles have a higher priority than any waiting vehicle, preventing them from entering the intersection. If we further assume that messages arrive eventually, the priority scheme based on the arrival time assures that, eventually, every car may cross the intersection; this prevents deadlocks and enables fairness.

Handling of Communication Failures. Communication failures, such as losing messages, may occur in all protocol phases. For example, if a node does not participate in the merge phase, the network cannot commit and does not propagate a potentially dangerous resource assignment. Losing commit messages is detected by non-matching commit numbers and is handled by simple rejoining. At no time does a node believe that something wrong was agreed on.

Join and Leave. Join and leave are explicit inquiries that need confirmation from the network through a commit. The commit numbers detect resulting inconsistencies as both share the commit mechanism. When a leaving node does not hear such a confirmation but misses a commit, it directly rejoins and leaves again. A joining node keeps trying to join. As the used network IDs stay unique, the coordination remains safe.

Leader Election and Handover. Unique IDs serve as a tie-breaker when electing a new leader for the network and guarantee a unique election. Only with the participation of all nodes does the new leader commit and increase its commit number. With the commit, the new leader ignores packets with the old commit number and thus prevents commits of the old leader as it is still part of the network. Because election commits are separate from coordination commits, they do not affect the safety of the coordination.

Node Failures. The leader is the only node that requires persistent storage to restore the network state. All other nodes can restore their state based on their position in the intersection and the rejoin mechanism. Without the node's participation, the network does not progress as STARC prefers to block and prevent collisions under failures.

4.4. Generalization of STARC: Platooning

Until now, we introduced STARC using paths of single cars. STARC, however, is designed as a foundation for the decentralized coordination of resources and applies to various application scenarios. In this section, we introduce STARC for platooning to underline this aspect. Apart from cars or pedestrians, STARC supports all kinds of vehicles. It can further coordinate virtual entities like vehicle platoons: when vehicles are traveling in the same direction, they can form platoons to make better use of the road or, as in this case, the intersection.

We design platooning with STARC as follows: as vehicles wait in line, they form platoons in their respective lanes (we assume a platoon-forming system is available [31,32]). A designated platoon head participates in the STARC coordination and attempts to reserve the path for the entire platoon. The platoon crosses as a whole once the reservation is accepted. While passing, the platoon leader updates the reserved tiles according to the positions of all platoon members. The platoon only releases a tile if the last vehicle in the platoon passes it. Only the head of the platoon needs to join the network, while the other vehicles participate as forwarders. Characteristics of the platoon (like the platoon size or arrival time) yield the priority value for the whole platoon. When leaving the intersection, the platoon head hands over the coordination to the next vehicle. The last car in the platoon is then responsible for the leave.

5. Evaluation

In this segment, we examine the performance of STARC in complex wireless environments and its effectiveness in various traffic conditions. Additionally, we showcase our findings on network usage and overall performance.

Methodology. We obtained the measurements by conducting three simulation runs, each lasting 30 min. Each vehicle went through the following steps: Queueing, waiting for the reservation, crossing, and, finally, leaving the network. The amount of time spent in line, waiting, crossing, and leaving the network all contribute to the delay experienced by a vehicle—the average delay gauges efficiency. We determine safety by the number of collisions [3]. We evaluate STARC with varying traffic loads and introduce the essential parameter of vehicles per hour, determining the level of traffic at an intersection as the number of cars that pass through it every hour in all lanes combined.

Setup. The simulated intersection has four ways with three lanes each. The granularity has been adjusted to 6, which ensures that the tile borders align with the lanes; Figure 4 displays the intersection layout with the corresponding tile grid. To comply with IEEE 802.15.4 packet restrictions, we have limited our network size to 16. Consequently, we exclusively allow the foremost vehicle per lane to join the network. The sensors installed in cars help them to identify the presence of other vehicles on the road ahead. Additional vehicles assist in maintaining the network by forwarding the packets following the Synchrotron transmission scheme. With a maximum network size of 16, the network has ample slots to accommodate all twelve lanes and four extra slots for vehicles crossing or exiting. The implementation further includes four slots for joining and one slot for rejoining. We set the length of a Synchrotron slot to 6 ms, providing enough time to process incoming packets. Shorter slot durations exhibit a loss of time synchronization within the network. We configured the maximum number of slots to 200 per round based on initial testing. The Synchrotron rounds occur every 2 s as a base interval. Nodes wait 5 s to detect existing networks before initiating their STARC instance. It is important to note that creating parallel networks is prohibited, as discussed in Section 6.
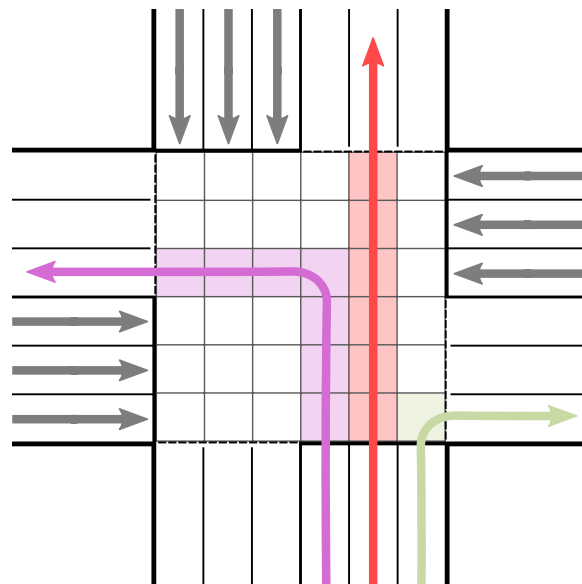
Figure 4. The simulated intersection has four directions with three lanes, allowing only left turns, straight passings, or right turns. The arrows indicate the restrictions, while the tiles that need to be reserved by each car in the 6 × 6 grid are colored in.

Vehicles reach the intersection at constant intervals, while their respective arrival lane is randomized as follows: We assume a 15% rate for both right and left turns; 70% of the cars are thus trying to move straight across the intersection. Each starting lane has a corresponding end lane. Lane swaps are prohibited. All the cars have identical physical specifications, which Table 1 lists. The rate at which turning can occur is restricted to 90 degrees per second. Hence, vehicles must slow down and decelerate when they encounter sharp curves. As mentioned in the design chapter, the vehicles adhere to a predetermined path when crossing the intersection. This path is simplified to align with the tiles used in the intersection. To simplify the simulation and collision checking, the car body is defined as a circle. The circle's diameter is 2 m, whereas the lanes are 3 m wide.

Implementation. We utilize the Synchrotron implementation on TelosB sensor nodes [16]. The TelosB has a Texas Instruments MSP430 chipset that provides 10 kB of RAM and a 4 MHz CPU [9]. The devices use their IEEE 802.15.4 radios to perform and transmit according to the STARC protocol. The devices utilize Contiki OS, an open-source operating system specifically designed for the Internet of Things [33]. To create a near-realistic radio simulation, we use the Cooja simulator to emulate the execution of

nodes' code and simulate their radio messages through a Multi-path Ray-tracing radio medium. The STARC protocol is implemented in the C programming language on top of Synchrotron. For a maximum network of size 16 and a 6×6 tile grid, STARC's coordination occupies 92 of the 100 Bytes available for application-specific behavior in Synchrotron. The rest of the IEEE 802.15.4 packet is reserved for round synchronization, round and slot numbers, and additional integrity checks; Table 2 lists the content of a coordination packet. We emphasize that the STARC protocol implementation is agnostic of the type of resources. STARC handles the coordination of mutual access to shared resources. For the actual vehicle behavior, we developed a Java-based plugin for the simulation that interfaces with Cooja and moves the emulated radio nodes according to the vehicle positions. Figure 5 illustrates the resulting simulation architecture.

Table 1. Evaluation parameters.

Category	Parameter	Value
Intersection	Number of lanes	12
	Tile grid	6×6
	Turn rate (left and right)	15 % each
	Lane width	3 m
	Maximum speed	$50 \, \mathrm{km \, h^{-1}}$
Vehicles	Acceleration	$2 \, \mathrm{m \, s^{-2}}$
	Deceleration	$4 \, \mathrm{m \, s^{-2}}$
	Vehicle diameter	2 m
STARC	Round Interval	2 s
	Slot length	6 ms
	Slot number	200
	Max. network size	16
	Join/leave/rejoin slots	4/16/1
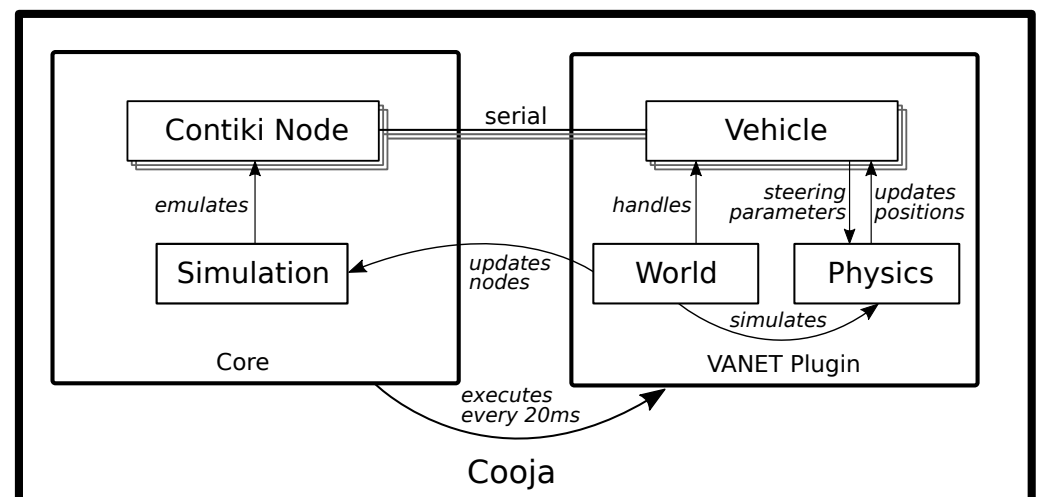
Figure 5. Simulation architecture: The simulation architecture extends Cooja with a VANET plugin that handles vehicles' behaviors and physics. Cooja emulates a resource-constrained device per vehicle and simulates its radio communication. Those devices run the actual STARC coordination protocol, which is agnostic of the vehicle scenario and decentrally handles the reservation of abstract resources. We restrict communication between vehicles and their respective coordination node to a serial connection.

**Table 2.** Packet structure of coordination round packets of STARC.

Field	Length [Byte]
Round Type and Phase	1
Commit Number	2
Node Count	1
Number of Join Requests	1
Join Slots	12
Rejoin Slot	3
Priority Mapping	32
Reservation Grid	36
Participation Flags	2
Leave Flags	2
Overall Length	92

5.1. Round Completion Time

For STARC to function effectively, every member in the network must participate. At the same time, all present vehicles act as forwarders. However, the implementation limits the number of transmission slots to 200 per round. We evaluate the influence of both factors on the number of slots required until all joined nodes acknowledge the final commit packet.

Scenario. We analyze the change in round completion time for different amounts of joined and forwarding nodes. Because a single initiator does not commit, we filter out rounds with the initiator as the only member.

Results. Overall, the round completion time slightly increases as more nodes join (see Figure 6). Fewer data points were obtained for larger network sizes. The 97.5 Percentile is below 125, leaving more than 75 more slots in 97.5% for additional communication within a round. The average completion time and 97.5 Percentile seem stable even with an increased number of forwarders. We thus conclude that forwarders do not harm the protocol, and additional members are handled with little overhead.

(**a**) (**b**)
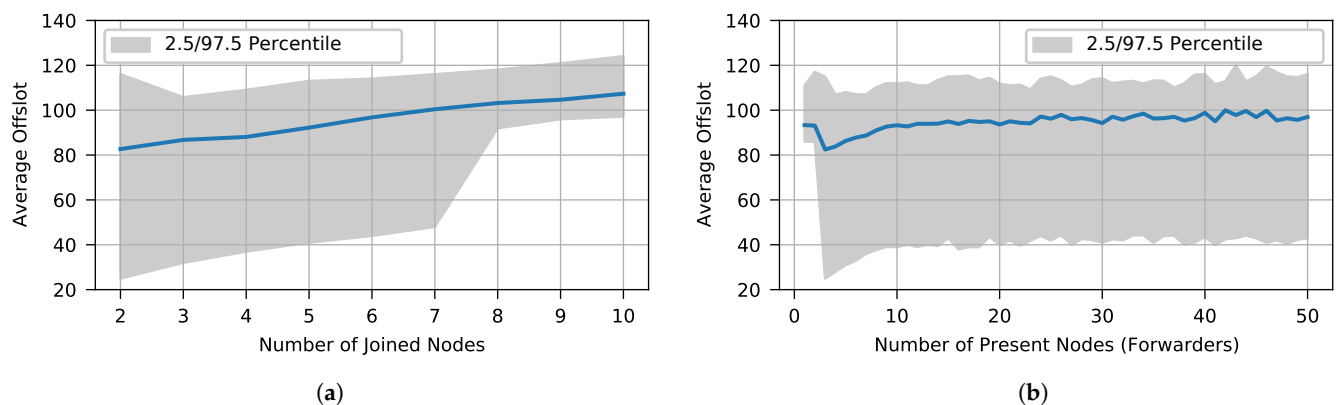
Figure 6. Effect of the round completion with different sizes of (**a**) joined vehicles and (**b**) all present vehicles that forward packets. While the network needs just a few more slots to accommodate another joined node, the forwarding by all present vehicles does not seem to harm the coordination in STARC. (**a**) Effect of the network size on the round completion. (**b**) Influence of the total number of forwarders on the round completion time.
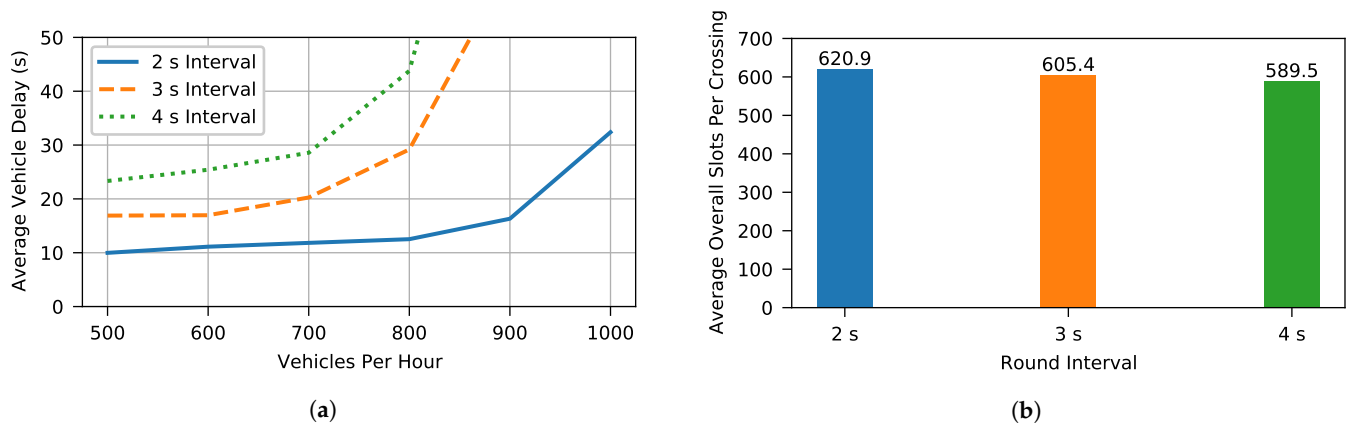
5.2. Round Interval Length

The round interval length specifies the time between Synchrotron rounds. Reducing this interval directly reduces the time between STARC coordination rounds and increases the radio usage and, thus, the power consumption.

Scenario. We are examining how coordination efficiency is affected by the round interval. We, therefore, compare 2, 3, and 4 s of spacing over multiple traffic loads. Furthermore, we evaluate how many slots a vehicle participates in when crossing the intersection.

Results. Figure 7a depicts the resulting efficiencies. All three configurations show increased delays when confronted with higher traffic intensities. The additional coordination rounds reduce the average delay. Notably, the round interval of 2 s generates the lowest average delay in our test.

Figure 7 displays the overall slots needed to cross the intersection. At 700 vehicles per hour, with round intervals of 2 s, 3 s, and 4 s, each car needs approximately 621, 605, and 590 overall slots on average. We conclude that lower efficiency and the resulting elongated stay partially compensate for the smaller intervals' initial overhead.

(**a**)　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　(**b**)

Figure 7. Effect of the round interval length on efficiency and the overall number of transmission slots: (**a**) A decreased interval shows increased efficiency due to more coordination rounds at the price of more slots per crossing and thus higher power consumption. (**b**) The increase in efficiency partially compensates for the power consumption as the crossing needs fewer slots in total. (**a**) Round interval influence on efficiency. (**b**) Number of Synchrotron slots per vehicle with different round intervals at 700 vehicles per hour.

5.3. Radio Failures

Due to its nature, wireless communication is often unreliable and prone to errors. The communication quality can be significantly impacted by interference from other technologies, multi-path reflection, and antenna orientation. For this reason, we are assessing STARC's ability to withstand failures and maintain safety even when certain cars are unable to communicate with the rest of the network.

Scenario. We introduce random failures that impede communication for specific nodes in the network: during each round, there is a possibility that a node may fail at any given slot. If the node fails, it will not be able to communicate for the rest of the round: until the end of that round, it cannot receive nor transmit any packets. Once the next round begins, all failed nodes regain their communication capabilities. However, the leader is not subject to failure injection; this ensures that the leader can always initiate the round. We configure the traffic level to 1000 vehicles per hour. In this setting, we record the success rate of commits and monitor the number of collisions.

Results. Table 3 contains the success rate starting the commit phase and the number of crashes for different failure rates. With a failure rate of only 0.01%, most rounds are successful and cars can generally agree on a crossing schedule. The commit rate declines to 63.7% with failure levels of 0.1%, i.e., a 0.1% probability to stop communicating every 6 ms. Out of 10 rounds, there were only 6 instances where an agreement was reached on which cars should cross the intersection, while in the remaining 4 rounds, the network did not establish a new schedule. However, during these four rounds, no cars took an unsafe route and all waiting cars agreed not to cross. It is worth noting that there were no collisions

between cars during any of these rounds. STARC, designed for safety, prioritizes blocking to avoid conflicting paths and potential collisions in case of failures.

Table 3. Commit success, recorded collisions in the presence of radio failures. Although STARC's success rates and vehicle delay can be impacted by failures, it is safe by design and collisions never occur.

| Failure Rate	Commit Success Rate	Car Collisions
0%	99.8%	0
0.001%	99.4%	0
0.01%	96.1%	0
0.1%	63.7%	0

5.4. Delay Induced by Crossing

The additional delay that the vehicles experience when passing the intersection specifies the efficiency of the coordination mechanism and is thus an important metric.

Scenario. We assess the effectiveness of STARC concerning various traffic loads. The user's delay is the sum of the times for joining the network, waiting for the reservation, and leaving the network. We simulate a traffic flow of 500 to 1200 vehicles per hour, similar to the traffic at a typical city intersection during morning rush hour, with an average of one car passing every 3 s. Later, we will evaluate platooning for even higher arrival rates.

Results. Figure 8a presents the mean delays encountered when passing through the intersection using STARC. Initially, the average delay is less than 20 s. However, the delay increases rapidly when the number of vehicles per hour exceeds 1100. Nevertheless, the 2.5 Percentile suggests that some cars still experience shorter delays.

Figure 8b provides a breakdown of vehicle delays for 1100 cars per hour, categorized by direction. At the intersection, the 70% of vehicles that move straight spend the most time (38.2 s) waiting in their lanes. The waiting time in line is shorter for cars turning left (3.9 s) and right (0.3 s). The waiting times for acceptance depend on the number of tiles required for the reservation. Vehicles turning right have the shortest waiting time of 6.2 s, while those turning left have the longest waiting time of 18.9 s. Cars driving straight experience a waiting time of 14.6 s. The time it takes to exit the network is almost the same for all directions, ranging from 1.3 to 1.5 s.

Figure 9 gives detailed insights about the delay composition of left-turning vehicles using STARC over different traffic loads of 500, 1000, and 1500 vehicles per hour. STARC affects joining, leaving, and the main waiting component, the time each vehicle has to wait for its successful reservation. As expected, the times for queueing (0.1 s, 1.5 s, 7.8 s) and waiting (8.7 s, 15.4 s, 22.6 s) increase with higher traffic loads. However, the time to join the network reduces with higher traffic loads (5.2 s, 3.9 s, 3.1 s). The crossing times stay nearly identical (6.6 s, 6.5 s, 6.4 s). Leaving times are low for all traffic levels (0.7 s, 1.3 s, 1.3 s). Overall, the times for joining and leaving the STARC network do not increase with higher traffic loads.
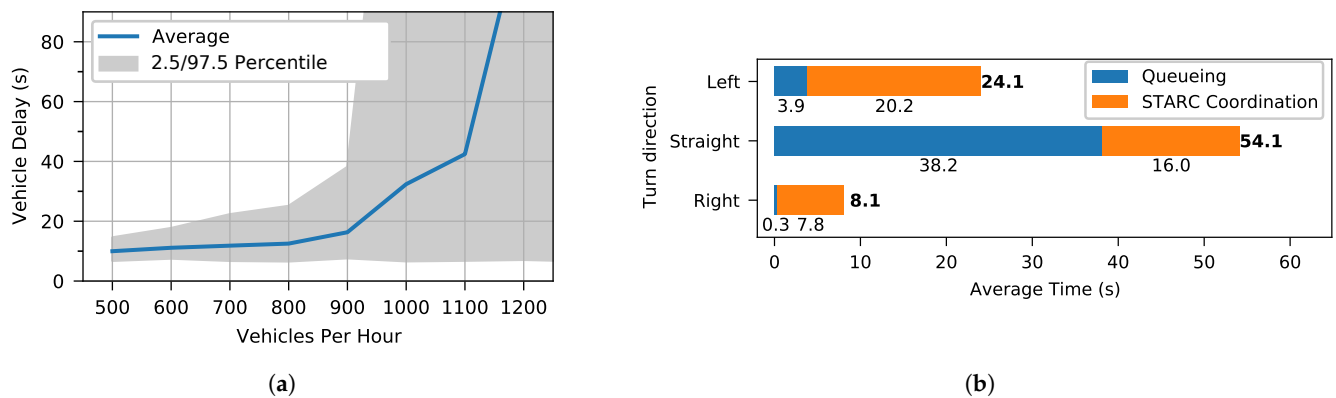
Figure 8. Efficiency evaluation of the basic STARC protocol without platoons: (**a**) Simulations exhibit mean delays of less than seconds under 1000 vehicles per hour. Afterward, delays indicate congestion. (**b**) Being part of the coordination process, joining and leaving the network, and waiting for reservations are necessary steps. The waiting times vary based on path lengths (excluding crossing times). Out of all the cars, 70% go straight ahead, while 15% turn left and another 15% turn right. (**a**) STARC protocol delay without platoons. (**b**) Delay composition for each turn direction with STARC simulation runs with 1100 vehicles per hour and no platoons.
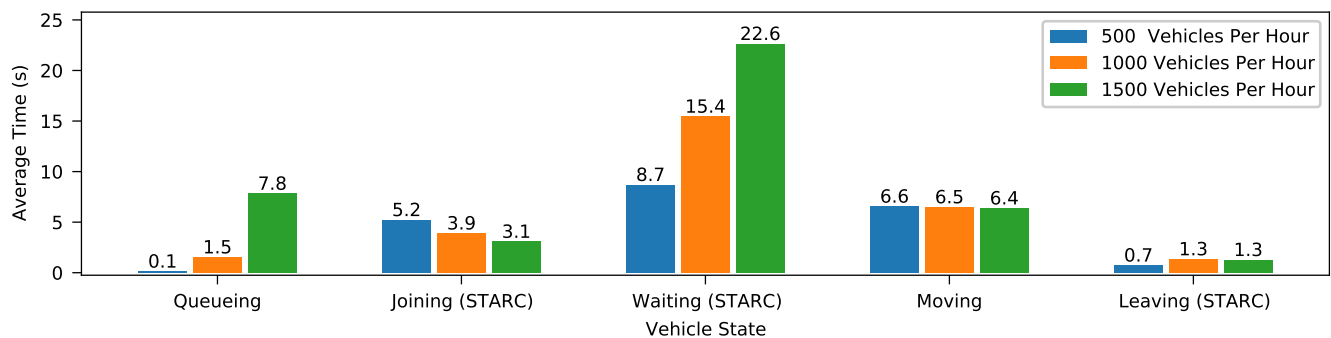

Figure 9. Comparison of delays for traffic loads of 500, 1000, and 1500 vehicles per hour broken down for each vehicle state of left-turning vehicles. The times for queueing and waiting increase with higher traffic loads. Joining becomes faster with more vehicles around. Moving and leaving times stay similar.

5.5. Platooning

We introduce a platoon extension in Section 4.4. Each platoon head coordinates the path for its whole platoon. Using STARC, they reserve their path as one vehicle. The platoon's head's priority stands for the priority of the full platoon.

Scenario. In this scenario, we analyze the impact of different platoon sizes on efficiency. We assume that the vehicles form platoons while queueing and simulate STARC with different platoon size limits.

Results. Figure 10a displays the simulation results for a limit of 10 and 25 vehicles per platoon as well as no limit. All variants have comparable delays until the platoons reach the individual size limits. The limit of 10 saturates at 5000 vehicles per hour while the limit of 25 vehicles starts to saturate with 6000 vehicles per hour. With unlimited platoon lengths, the delay increases slightly and does not seem to saturate. Additionally, Figure 10b displays the average platoon size in the unlimited setting for each direction. The results indicate an expected increase in platoon size with higher traffic volumes. As 70% of the vehicles drive straight, this direction uses the longest platoons. In the simulations with 6000 cars per hour, the average platoon sizes are 15.6 vehicles for straight crossings, 3.97 for left turns, and 1.2 for right turns.
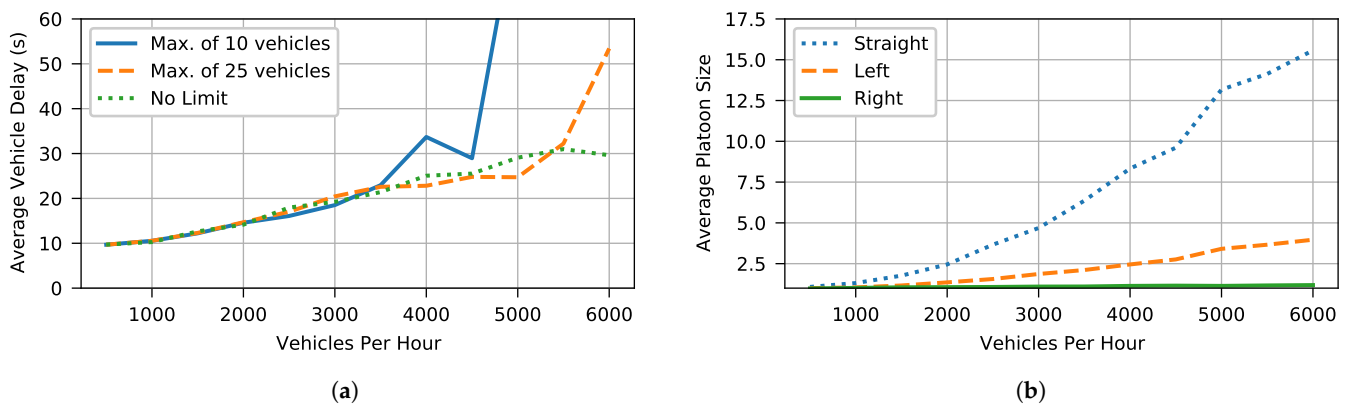
Figure 10. Platoon extension with STARC: (**a**) Larger platoons support higher traffic loads. (**b**) Bigger platoons are required for longer paths (left turns) and for the 70% of vehicles that want to drive straight. (**a**) Average delays for different platoon size limits. (**b**) Average platoon sizes for directions without size limitations.

5.6. Traffic Lights Comparison

Traffic lights are an established coordination mechanism. This section compares the efficiency of a fixed schedule with STARC.

Scenario. Although the scenario now includes traffic lights, the fundamentals remain the same. However, when approaching an intersection with traffic lights, cars can only be in one of two states: waiting in a queue or moving through the intersection, i.e., vehicles must line up and wait for a green signal to pass the intersection. The traffic lights operate on an all-lane model, allowing all lanes in one direction to drive at the same time [3]. To ensure the safe passing of left-turning vehicles, each direction is scheduled to display a green light for 9 s, followed by a yellow and then a red light, each lasting for 3 s. This cycle repeats every minute, resulting in 15 s for each direction. In the traffic lights scenario, the delay is determined solely by the time spent waiting in the queue. We also assess the platoon extension, which permits up to 25 vehicles to join before the platoon begins crossing.

Results. Figure 11 displays the mean delay with traffic lights and the STARC protocol, both with and without platooning enabled. The platoon size limit is 25, which is approximately the same number of cars that can cross a straight road during a single green light phase. Our protocol's recorded delay gradually increases when there are no platoons and spikes when the number of vehicles exceeds 900 per hour. Without platoons, the delay surpasses traffic lights at around 1000 vehicles per hour. It is not unexpected that there is a limit to the protocol without platoons since it only permits one car per lane to move. On the other hand, the platooning version manages these values effectively while handling less than 1000 vehicles per hour more efficiently compared to STARC without platoons. As expected, no tested approach operates at arbitrarily high traffic volumes. However, by adjusting the traffic lights to adapt to changing traffic volumes or increasing the limit on platoon sizes, we can reduce delays experienced by drivers (see Section 5.5).
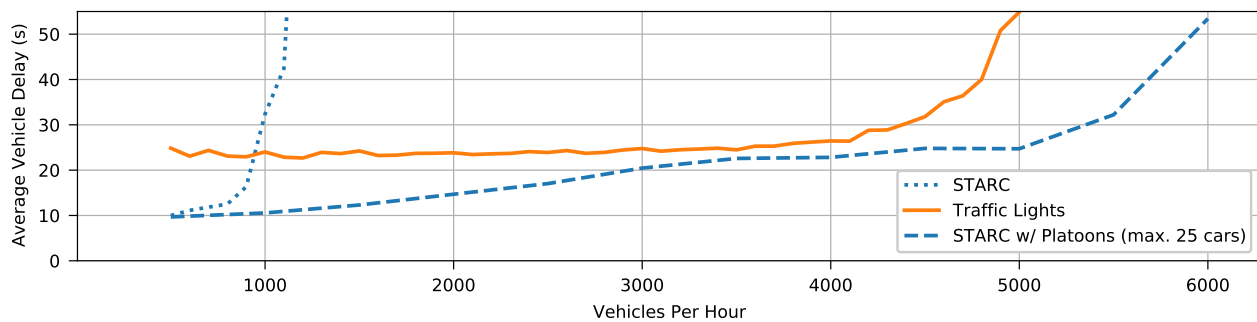
Figure 11. STARC's Efficiency for Autonomous Intersection Management: when STARC's coordination is used without platoons, there is less delay than relying solely on traffic lights. However, this is only true when the number of vehicles per hour exceeds 1000. Beyond this point, the system becomes saturated. Coordinating entire platoons, however, results in better scalability even at higher traffic levels. Traffic lights employ a static all-lane model and are not optimized for the different traffic levels. Although this schedule is not specifically designed for the intersection layout or adaptable, it is a reference point for comparing the coordination performance of STARC in managing intersections.

6. Discussion

The evaluation results underline STARC's inherent safety and potential for decentralized intersection management. This section reflects on several aspects beyond the main protocol.

Technology Independence. We demonstrate STARC on top of IEEE 802.15.4 radios. However, the feasibility of Synchronous Transmissions and thus the primary building block of STARC are shown for other technologies such as Bluetooth Low Energy and Ultra-wideband [18,20,34]; this fosters a flexible deployment of STARC, allowing other radio technologies beyond IEEE 802.15.4.

Scalability. The evaluation results demonstrate an approximately linear growth in the number of slots for an increasing number of joined nodes and no significant overhead caused by nodes acting as forwarders. For STARC, the packet size dictates the supported network size and the number of tiles, i.e., resources, to manage. Hence, the limiting factor of scalability remains in the size of the packets as participation flags, priorities, and resource assignments are all packed into a single IEEE 802.15.4 packet. With just 16 joined devices and 36 resources, the illustrated intersection example fits tightly into the standard IEEE 802.15.4 packet size of 128 Bytes (cf. Table 2). According to the IEEE 802.15.4 standard, some PHY configurations allow packets of up to 2048 bytes, allowing STARC to scale to more complex coordination scenarios [15,16].

Security. Own efficiency gains and traffic regulations strongly incentivize vehicles to follow the protocol. Still, without security mechanisms, STARC is highly susceptible to packet injection attacks, compromising its safety guarantees. Therefore, beyond basic message integrity checks, deployments of STARC should use additional authentication and possibly encryption mechanisms in the synchronous transmission layer [35].

Fault Tolerance. STARC prioritizes safety over liveness, blocking whenever nodes are unreachable. However, if a single node does not recover or the respective vehicle leaves the intersection without leaving the network first, STARC blocks indefinitely. Thus, the protocol is not fault-tolerant. Instead of relying on a single point of failure, fault-tolerant protocols such as Paxos ensure progress as long as a majority of participants are involved. Poirot et al. introduced Paxos atop Synchronous Transmissions [36]. Hence, Paxos could replace the leader-based commit functionality within STARC. With additional sensors, vehicles could further determine if a vehicle is still at the intersection and remove it from the network if this is not the case.

Network Creation. We assume arriving nodes join any existing network and create one only if none exists. Targeting decentralized deployments, this is both a crucial but non-trivial assumption: in the worst case, two nodes could start different STARC instances

in parallel, trying to coordinate the very same intersection. Consequently, their independent coordination rounds could grant fatal conflicting reservations, contrasting STARC's safety guarantees. A careful network initialization process is, therefore, required [37].

7. Conclusions

Intelligent Transportation Systems and the Internet of Vehicles rely on connected traffic as a foundation. When vehicles and other road users communicate, they can work together to efficiently and safely navigate intersections. Central approaches to intersection management require the deployment and availability of costly infrastructure. Decentralized solutions only rely on local wireless communication between the participants, but most existing approaches require expensive and energy-hungry platforms or need more detailed safety analyses.

This work introduces STARC decentralized protocol for resource assignment in the context of intersection management. STARC, designed for safety, uses inexpensive, low-power wireless radios. Built on top of IEEE 802.15.4, STARC allows for energy-efficient communication without additional infrastructure, making it a cost-effective deployment option. With STARC, vehicles, cyclists, and pedestrians can quickly establish or join a low-power network to coordinate their movements. We implement STARC for resource-constrained IoT devices and simulate vehicles at an intersection with small IoT devices that run STARC. Our results show that STARC enables safe, scalable resource coordination in challenging wireless conditions—despite the absence of a central authority. Applied to the setting of intersection management, STARC, decreases the average waiting time by up to 50% compared to traditional traffic lights for traffic volumes below 1000 vehicles per hour. Further, STARC can also efficiently manage traffic loads of more than 1000 vehicles per hour by utilizing platoons—all while ensuring no collisions. Based on the abstract coordination mechanism, STARC is not restricted to the proposed roadside application. Other domains, such as drones, could equally benefit from decentralized coordination.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Respective programming code and data generation are available at: https://github.com/ds-kiel/starc (accessed on 10 July 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ID	identifier
IoT	Internet of Things
IoV	Internet of Vehicles
ITS	Intelligent Transportation Systems
STARC	Synchronous Transmissions for Autonomous Resource Coordination
VTL	Virtual Traffic Lights

Appendix A

Algorithm A1 *begin_round*()

Require:
  global *netState* = (*memberID*, *shouldJoin*, *shouldLeave*)
  global *ressourceInfo* = (*ID*, *resourceList*, *priority*, *electionPriority*)
  global *leaderState* = (*isLeader*, *count*, *memberIDs*)

  txData ← *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)
  txData.type ← "*unknown*"
  txData.commit ← *false*
  txData.rejoins ← []
  txData.joins ← []
  txData.leaves ← {}
  txData.flags ← {}
  txData.roundData ← ([], [])

  if *shouldJoin* and *memberID* = *null* **then**
    joins ← [(*ID*, *null*)]
  else if *shouldLeave* and *memberID* ≠ *null* **then**
    leaves ← {*memberID*}   ▷ This also holds for the Leader
    electionPriority ← 0   ▷ Leaving nodes least desired for leadership
  end if

  if *memberID* ≠ *null* **then**
    txData.roundData ← ([*memberID*, *priority*], *resourceList*)
    flags ← {*memberID*}
  end if

  if *isLeader* **then**
    if *shouldLeave* **then**
      txData.type ← "*election*"   ▷ Leader starts election rounds until alone
      txData.roundData ← (*memberIDs*, [(*memberID*, *electionPriority*)])
    else
      txData.type ← "*coordination*"
    end if
  end if
  return *txData*

Algorithm A2 *handle_packet*(*txData*, *rxData*)

Require:
  global *netState* = (*memberID*, *shouldJoin*, *shouldLeave*)
  global *ressourceInfo* = (*ID*, *resourceList*, *priority*, *electionPriority*)
  global *leaderState* = (*isLeader*, *count*, *memberIDs*)
  txData of type *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)
  rxData of type *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)

  if *txData.commit* = *false* **then**
    if *memberID* = *null* and (*ID*, *oldMemberID*) ∈ *rxData.rejoins* **then**
      memberID ← *oldMemberID*
    end if
    txData.leaves ← *txData.leaves* ∪ *rxData.leaves*
    Merge *rxData.joins* into *txData.joins* based on IDs as tie-breaker
    if *isLeader* and ∃*joinID* ∈ *txData.joins* : (*joinID*, *oldMemberID*) ∈ *memberIDs* **then**
      Append (*joinID*, *oldMemberID*) to *txData.rejoins* if space permits
    end if
    Merge *rxData.rejoins* into *txData.rejoins* if slots available
  end if

Algorithm A2 *Cont.*

---

  if *rxData.type* = *"coordination"* **then**
    if *txData.type* = *"election"* **then**
      txData ← *rxData*   ▷ Coordination takes preference, local state is ignored.
      ▷ A newly elected initiator increases the configuration number and advances the
network eventually.
      return
    end if
    handle_coordination_packet(*txData*, *rxData*)
  else if *rxData.type* = *"election"* **then**
    if *txData.type* = *"unknown"* **then**   ▷ Election needs initialization
      if *memberID* ≠ *null* **then**
        txData.roundData ← ([], [(*memberID*, *electionPriority*)])   ▷ No member IDs
known
      else
        txData.roundData ← []
      end if
    else if *txData.type* = *"coordination"* **then**
      Ignore coordination packet
      return
    end if
    handle_election_packet(*txData*, *rxData*)
  end if

Algorithm A3 *handle_coordination_packet*(*txData*, *rxData*)

Require:
  global *netState* = (*memberID*, *shouldJoin*, *shouldLeave*)
  global *ressourceInfo* = (*ID*, *resourceList*, *priority*, *electionPriority*)
  global *leaderState* = (*isLeader*, *count*, *memberIDs*)
  txData of type *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)
  rxData of type *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)

  if *txData.commit* = *rxData.commit* **then**
    txData.flags ← *txData.flags* ∪ *rxData.flags*
    Merge priorities of *rxData* into *txData*
    Merge resource assignment of *rxData* into *txData* based on priorities and memberIDs
tie-breaker
    if *isLeader* and *txData.commit* = *false* and all members participated **then**
      txData.commit ← *true*   ▷ Leader commits
      txData.flags ← {*memberID*}   ▷ Reset participation flags
      Add nodes according to *txData.joins*, save new member IDs for each
      Remove nodes according to *txData.leaves*, free respective memberID
      Update overall member count
    end if
  else if *txData.commit* = *false* and *rxData.commit* = *true* **then**
    txData ← *rxData*   ▷ Node is behind, copy received data
    if (*ID*, *newMemberID*) ∈ *txData.joins* **then**
      memberID ← *newMemberID*   ▷ Join confirmed
    end if
    if *memberID* ∈ *txData.leaves* **then**
      memberID ← *null*   ▷ Leave confirmed
      Notify about successful leave
    end if
    if *memberID* ≠ *null* **then**
      txData.flags ← *txData.flags* ∪ {*memberID*}
    end if
  end if

Algorithm A4 *handle_election_packet*(*txData*, *rxData*)

Require:
  global *netState* = (*memberID*, *shouldJoin*, *shouldLeave*)
  global *ressourceInfo* = (*ID*, *resourceList*, *priority*, *electionPriority*)
  global *leaderState* = (*isLeader*, *count*, *memberIDs*)
  txData of type *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)
  rxData of type *struct*(*type*, *commit*, *rejoins*, *leaves*, *joins*, *flags*, *roundData*)

  if *txData.commit* = *rxData.commit* **then**
    txData.flags ← *txData.flags* ∪ *rxData.flags*
    if *txData.commit* = *false* **then**
      Merge leader candidate *rxData.roundData* into *txData.roundData* according to highest election priority and member ID as a tie-breaker
      Merge member IDs *rxData.roundData* into *txData.roundData*
      if *memberID* is leader candidate and all members participated **then**
        isLeader ← *true*   ▷ This node was elected
        Copy member IDs from *txData* to *memberIDs*   ▷ Copy state
        txData.commit ← *true*   ▷ New leader commits
        txData.flags ← {*memberID*}   ▷ Reset participation flags
        Remove nodes according to *txData.leaves*, free member IDs
        Update member count
      end if
    end if
  else if *txData.commit* = *false* and *rxData.commit* = *true* **then**
    txData ← *rxData*   ▷ Node is behind, copy received data
    if *txData.commit* = *false* and *rxData.commit* = *true* **then**
      if *memberID* ∈ *txData.leaves* **then**
        memberID ← *null*   ▷ Leave confirmed
        Notify about successful leave
      end if
    end if
    if *memberID* ≠ *null* **then**
      txData.flags ← *txData.flags* ∪ {*memberID*}
    end if
  end if

References

1. Winter, S.; Sester, M.; Wolfson, O.; Geers, G. Towards a computational transportation science. *J. Spat. Inf. Sci.* **2011**, *2*, 119–126. [CrossRef]
2. Lu, N.; Cheng, N.; Zhang, N.; Shen, X.; Mark, J.W. Connected vehicles: Solutions and challenges. *IEEE Internet Things J.* **2014**, *1*, 289–299. [CrossRef]
3. Dresner, K.; Stone, P. A Multiagent Approach to Autonomous Intersection Management. *J. Artif. Intell. Res.* **2008**, *31*, 591–656. [CrossRef]
4. Bashiri, M.; Fleming, C.H. A platoon-Based Intersection Management System for Autonomous Vehicles. In Proceedings of the 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, 11–14 June 2017; pp. 667–672. [CrossRef]
5. Hobert, L.; Festag, A.; Llatser, I.; Altomare, L.; Visintainer, F.; Kovacs, A. Enhancements of V2X communication in support of cooperative autonomous driving. *IEEE Commun. Mag.* **2015**, *53*, 64–70. [CrossRef]
6. Van Middlesworth, M.; Dresner, K.; Stone, P. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, Estoril, Portugal, 12–16 May 2008.
7. Hassan, A.A.; Rakha, H.A. A Fully-Distributed Heuristic Algorithm for Control of Autonomous Vehicle Movements at Isolated Intersections. *Int. J. Transp. Sci. Technol.* **2014**, *3*, 297–309. [CrossRef]
8. Ferreira, M.; Fernandes, R.; Conceição, H.; Viriyasitavat, W.; Tonguz, O.K. Self-organized traffic control. In Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM, Chicago, IL, USA, 20–24 September 2010; pp. 85–89. [CrossRef]
9. MSP430 Ultra-Low-Power MCUs. Texas Instruments. Available online: https://www.ti.com/microcontrollers-mcus-processors/msp430-microcontrollers/overview.html (accessed on 10 July 2023).

10. Rathje, P.; Poirot, V.; Landsiedel, O. STARC: Low-power Decentralized Coordination Primitive for Vehicular Ad-hoc Networks. In Proceedings of the NOMS 2020—2020 IEEE/IFIP Network Operations and Management Symposium, Budapest, Hungary, 20–24 April 2020; pp. 1–6.

11. Segata, M.; Bloessl, B.; Joerer, S.; Sommer, C.; Gerla, M.; Lo Cigno, R.; Dressler, F. Toward communication strategies for platooning: Simulative and experimental evaluation. *IEEE Trans. Veh. Technol.* **2015**, *64*, 5411–5423. [CrossRef]

12. Arena, F.; Pau, G.; Severino, A. A review on IEEE 802.11 p for intelligent transportation systems. *J. Sens. Actuator Netw.* **2020**, *9*, 22. [CrossRef]

13. Cao, H.; Gangakhedkar, S.; Ali, A.R.; Gharba, M.; Eichinger, J. A 5G V2X testbed for cooperative automated driving. In Proceedings of the 2016 IEEE Vehicular Networking Conference (VNC), Torino, Italy, 27–29 November 2017. [CrossRef]

14. Rasool, I.U.; Kim, S.W. A review of wireless access vehicular environment multichannel operational medium access control protocols: Quality-of-service analysis and other related issues. *Int. J. Distrib. Sens. Netw.* **2017**, *13*, 2017. [CrossRef]

15. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*; IEEE Standard for Low-Rate Wireless Networks. IEEE: Piscataway, NJ, USA, 2020; pp. 1–800. [CrossRef]

16. Nahas, B.A.; Duquennoy, S.; Landsiedel, O. Network-wide consensus utilizing the capture effect in low-power wireless networks. In Proceedings of the SenSys 2017—15th ACM Conference on Embedded Networked Sensor Systems, Delft, The Netherlands, 6–8 November 2017. [CrossRef]

17. Benalia, E.; Bitam, S.; Mellouk, A. Data dissemination for Internet of vehicle based on 5G communications: A survey. *Trans. Emerg. Telecommun. Technol.* **2020**, *31*, e3881. [CrossRef]

18. Al Nahas, B.; Duquennoy, S.; Landsiedel, O. Concurrent Transmissions for Multi-Hop Bluetooth 5. In Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks, Beijing, China, 25–27 February 2019; pp. 130–141.

19. Baddeley, M.; Alberto Boano, C.; Escobar-Molero, A.; Liu, Y.; Ma, X.; Raza, U.; Römer, K.; Schuß, M.; Stanoev, A. The Impact of the Physical Layer on the Performance of Concurrent Transmissions. In Proceedings of the 2020 IEEE 28th International Conference on Network Protocols (ICNP), Madrid, Spain, 13–16 October 2020. [CrossRef]

20. Vecchia, D.; Corbalan, P.; Istomin, T.; Picco, G.P. Playing with Fire: Exploring Concurrent Transmissions in Ultra-wideband Radios. In Proceedings of the Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops, Boston, MA, USA, 21–25 June 2019. [CrossRef]

21. Landsiedel, O.; Ferrari, F.; Zimmerling, M. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In Proceedings of the SenSys 2013—11th ACM Conference on Embedded Networked Sensor Systems, Rome, Italy, 11–15 November 2013. [CrossRef]

22. Dutta, P.; Dawson-Haggerty, S.; Chen, Y.; Liang, C.J.M.; Terzis, A. Design and evaluation of a versatile and efficient receiver-initiated link layer for low-power wireless. In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems, Zurich, Switzerland, 3–5 November 2010; pp. 1–14.

23. Ferrari, F.; Zimmerling, M.; Thiele, L.; Saukh, O. Efficient network flooding and time synchronization with glossy. In Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks, Chicago, IL, USA, 12–14 April 2011; pp. 73–84.

24. Wang, C.H.; Chou, C.T.; Lin, P.; Guizani, M. Performance evaluation of IEEE 802.15.4 nonbeacon-enabled mode for internet of vehicles. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 3150–3159. [CrossRef]

25. Ferreira, M.; D'Orey, P.M. On the impact of virtual traffic lights on carbon emissions mitigation. *IEEE Trans. Intell. Transp. Syst.* **2012**, *13*, 284–295. [CrossRef]

26. Hagenauer, F.; Baldemaier, P.; Dressler, F.; Sommer, C. Advanced Leader Election for Virtual Traffic Lights. *ZTE Commun.* **2014**, *12*, 11–16. [CrossRef]

27. Sommer, C.; Hagenauer, F.; Dressler, F. A networking perspective on self-organizing intersection management. In Proceedings of the 2014 IEEE World Forum on Internet of Things (WF-IoT), Seoul, Republic of Korea, 6–8 March 2014; pp. 230–234. [CrossRef]

28. Naumann, R.; Rasche, R.; Tacken, J.; Tahedl, C. Validation and simulation of a decentralized intersection collision avoidance algorithm. In Proceedings of the IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC, Boston, MA, USA, 9–12 November 1997; pp. 818–823. [CrossRef]

29. Choi, M.; Rubenecia, A.; Choi, H.H. Reservation-based cooperative traffic management at an intersection of multi-lane roads. In Proceedings of the International Conference on Information Networking. IEEE Computer Society, Philadelphia, PA, USA, 18–20 October 2018; pp. 456–460. [CrossRef]

30. Birman, K.; Joseph, T. Exploiting Virtual Synchrony in Distributed Systems. In Proceedings of the Eleventh ACM Symposium on Operating Systems Principles, Austin, TX, USA, 8–11 November 1987; pp. 123–138. [CrossRef]

31. Santini, S.; Salvi, A.; Valente, A.S.; Pescape, A.; Segata, M.; Lo Cigno, R. A Consensus-Based Approach for Platooning with Intervehicular Communications and Its Validation in Realistic Scenarios. *IEEE Trans. Veh. Technol.* **2017**, *66*, 1985–1999. [CrossRef]

32. Lioris, J.; Pedarsani, R.; Tascikaraoglu, F.Y.; Varaiya, P. Platoons of connected vehicles can double throughput in urban roads. *Transp. Res. Part C Emerg. Technol.* **2017**, *77*, 292–305. [CrossRef]

33. Contiki: The Open Source Operating System for the Internet of Things. Available online: https://github.com/contiki-os/contiki (accessed on 10 July 2023).

34. Herrmann, C.; Zimmerling, M. RSSISPY: Inspecting Concurrent Transmissions in the Wild. In Proceedings of the International Conference on Embedded Wireless Systems and Networks, Linz, Austria, 3–5 October 2022.

35. Hewage, K.C.; Raza, S.; Voigt, T. Protecting glossy-based wireless networks from packet injection attacks. In Proceedings of the 2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS), Orlando, FL, USA, 22–25 October 2017; pp. 37–45.
36. Poirot, V.; Al Nahas, B.; Landsiedel, O. Paxos Made Wireless: Consensus in the Air. In Proceedings of the 2019 International Conference on Embedded Wireless Systems and Networks, Beijing, China, 25–27 February 2019; pp. 1–12.
37. Mager, F.; Biri, A.; Thiele, L.; Zimmerling, M. BUTLER: Increasing the Availability of Low-Power Wireless Communication Protocols. In Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks, Linz, Austria, 3–5 October 2022; pp. 108–119.