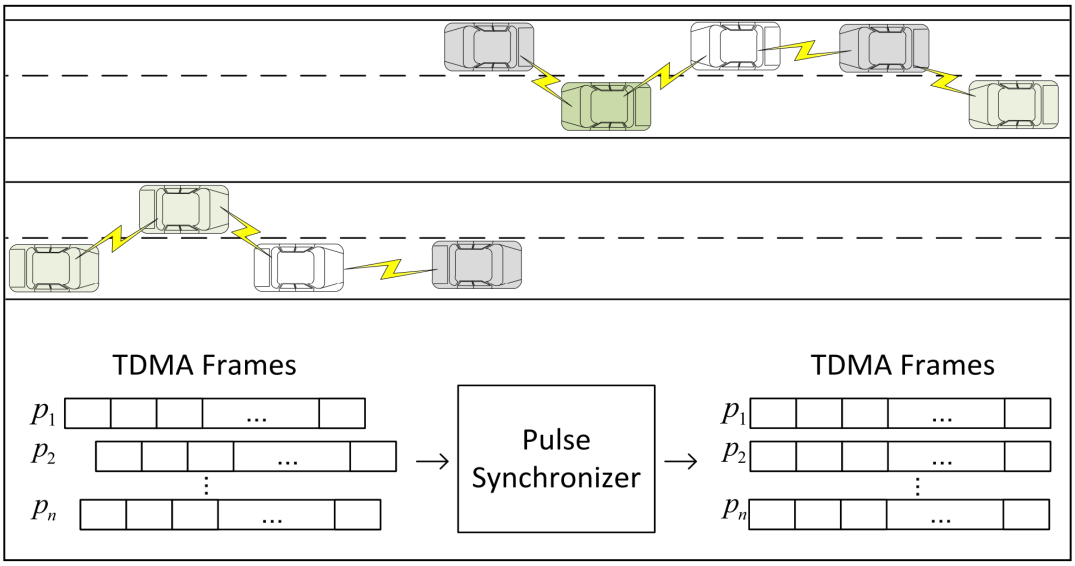
CHALMERS



Self-* Pulse Synchronization for Autonomous TDMA MAC in VANETs

Master of Science Thesis in Computer Science and Engineering

Mohamed Hassan Mustafa

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, January 2012

Self-* Pulse Synchronization for Autonomous TDMA MAC in VANETs

Mohamed Hassan Mustafa

Examiner: Elad Michael Schiller

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden January 2012

Abstract

The problem of local clock synchronization is studied in the context of media access control (MAC) protocols, such as time division multiple access (TDMA), for dynamic and wireless ad hoc networks. In the context of TDMA, local pulse synchronization mechanisms let neighboring nodes align the timing of their packet transmissions, and by that avoid transmission interferes between consecutive timeslots. Existing implementations for Vehicular Ad-Hoc Networks (VANETs) assume the availability of common (external) sources of time, such as base-stations or geographical positioning systems (GPS). This work is the first to consider autonomic design criteria, which are imperative when no common time sources are available, or preferred not to be used, due to their cost and signal loss.

We present self-$\star$ pulse synchronization strategies. Their implementing algorithms consider the effects of communication delays and transmission interferences. We demonstrate the algorithms via extensive simulations in different settings including node mobility. We also validate these simulations in the MicaZ platform, whose native clocks are driven by inexpensive crystal oscillators. The results imply that the studied algorithms can facilitate autonomous TDMA protocols for VANETs.

Key words: Pulse Synchronization, Clock Synchronization, TDMA Timeslot Alignment, MANETs, VANETs

Acknowledgments

I am heartily thankful to my supervisor, Elad Michael Schiller, whose encouragement, guidance and support from the initial to the final level enabled me to develop an understanding of the subject. I am also thankful to Marina Papatriantafilou and Philippas Tsigas for their valuable support and suggestions. And my thanks to Mitra Pahlavan and Amir Tohidi for many fruitful discussions.

Last but not least I wish to avail myself of this opportunity and express a sense of gratitude and love to my beloved family and friends for their priceless support, strength and prayers.

Gothenburg, Sweden

January 25, 2012

Contents

Chapter 1

Introduction

Recent work on vehicular systems explores a promising future for vehicular communications. They consider innovative applications that reduce road fatalities, lead to greener transportation, and improve the driving experience, to name a few. The prospects of these applications depend on the existence of predictable communication infrastructure for dynamic networks. We consider time division multiple access (TDMA) protocols that can divide the radio time regularly and fairly in the presence of node mobility, such as Chameleon-MAC [9]. The studied problem appears when neighboring nodes start their broadcasting timeslots at different times. It is imperative to employ autonomous solutions for timeslot alignment when no common (external) time sources are available, or preferred not to be used, due to their cost and signal loss. We address the timeslot alignment problem by considering the more general problem of *(decentralized) local pulse synchronization*. Since TDMA alignment is required during the period in which communication links are being established, we consider non-deterministic communication delays, the effect of transmission interferences and local clocks with arbitrary initial offsets, see Section 2. We propose autonomous and self-$\star$ algorithmic solutions that guarantee robustness and provide an important level of abstraction as they liberate the system designer from dealing with low-level problems, such as availability and cost of common time sources, see Section 3. Our contribution also facilitates autonomous TDMA protocols for Vehicular Ad-Hoc Networks (VANETs), see Section 5.

Let us illustrate the problem and the challenges of possible strategies using an example. Consider three neighboring stations that have unique timeslot assignment, but their timeslots are not well-aligned, see figure 1.1. Packet transmissions collide in the presence of such concurrent transmissions. Suppose that the stations act upon the intuition that gradual pairwise adjustments are most preferable. Station p_k is the first to align itself with its closest neighbor, p_j, see figure 1.2. Next, p_j aligns itself with p_i and by that it opens a gap between itself and p_k.

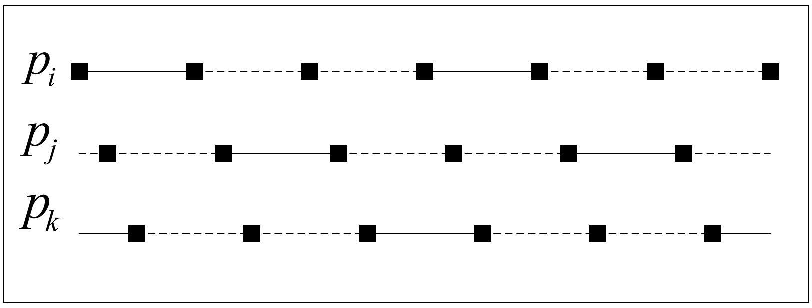


Figure 1.1: Unaligned TDMA timeslots. Solid and dashed lines stand for transmission, and respectively, idle radio times.

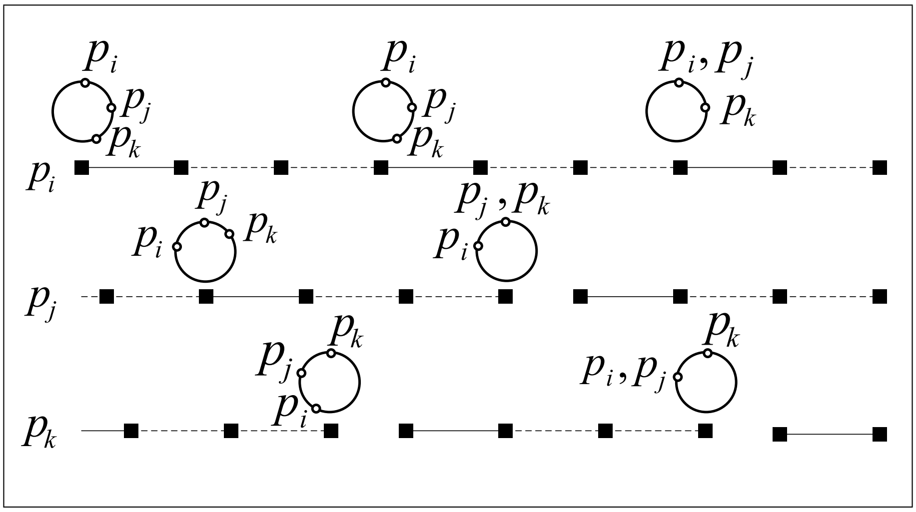


Figure 1.2: The cricket strategy. Solid and dashed lines stand for transmission, and respectively, idle radio times. The circles above the solid boxes represents the node's view on its neighbors' TDMA alignment at the start of its broadcasting timeslot. Gaps between two solid boxes represent alignment events.

Then, p_k aligns itself with p_i and p_j. The end result is an all aligned sequence of timeslots. We call this algorithmic approach the *cricket* strategy.

We observe that the convergence process includes a chain reactions, i.e., node p_k aligns itself before and after p_j's alignment. One can foresee the outcome of such chain reactions and let p_j and p_k to concurrently adjust their clock according to p_i. We name this algorithmic approach the *grasshopper* strategy, because grasshoppers jump further than crickets. We demonstrate that the latter strategy is faster, see Section 5. This improvement comes at the cost of additional memory and processing requirements.

We integrate the proposed algorithms with the Chameleon-MAC [9], which is a self-$\star$, mobility resilient, TDMA protocol. After extensive simulations and testbed experiments with and without mobility, we observe tight alignment among the timeslots, and high throughput of the

MAC protocol.

1.1 Related work

A number of biologically-inspired synchronization mechanisms are used for triggering periodic pulses [3, 12, 13, to name a few]. They do not consider wireless communication environments in which transmissions can be disrupted or have non-deterministic communication delays.

To the best of our knowledge, pulse synchronization mechanisms that do consider more practical communication environments [such as 6, 16, 17], do not study the problem in the context of TDMA-based MAC protocols. Namely, this work is the first to consider TDMA timeslot alignment during the period in which communication links are being established. For example, the authors of [12] consider a pseudo-random TDMA MAC in which packets are transmitted probabilistically, and in [17], the authors considered a p-persistent CSMA/CA MAC protocol.

We note that pseudo-random TDMA MAC and the p-persistent CSMA/CA MAC protocols provide a lower predictability degree than TDMA-based MAC protocols that follow the predictably scheduled approach [such as Chameleon-MAC 9]. We note that our algorithmic solutions can also facilitate TDMA protocols in which packets are transmitted probabilistically. Another interesting example appears in [6], where Byzantine-tolerance and self-stabilization properties are considered, although after communication establishment.

Chapter 2

Preliminaries

The system consists of a set, $N = \{p_i\}$, of n anonymous communicating entities, which we call *nodes*. The radio time is divided into fixed size TDMA frames and then into fixed size timeslots [as in 9]. The nodes' task is to adjust their local clocks so that the starting time of frames and timeslots is aligned. They are to achieve this task in the presence of: (1) a MAC layer that is in the process of assigning timeslots, (2) network topology changes, and (3) message omission, say, due to topological changes, transmission interferences, unexpected change of the ambient noise level, etc.

2.1 Time, clocks, and synchrony bounds

We consider three notations of time: *real time* is the usual physical notion of continuous time, used for definition and analysis only; *native time* is obtained from a native clock, implemented by the operating system from hardware counters; *local time* builds on native time with an additive adjustment factor in an effort to facilitate a neighborhood-wise clock.

Applications require the clock interface to include the READ operation, which returns a *timestamp* value of the local clock. Let C_k^i and c_k^i denote the value $p_i \in N$ gets from the k^{th} READ of the native or local clock, respectively. Moreover, let r_k^i denote the real-time instance associated with that k^{th} READ operation.

Pulse synchronization algorithms adjust their local clocks in order to achieve synchronization, but never adjust their native clocks. Namely, the operation ADJUST(add) adds a positive integer value to the local clock. This work considers solutions that adjust clocks forward, because such solutions simplify the reasoning about time at the higher layers. We define the native clocks *offset* $\delta_{i,j}(k,q) = C^i_k - C^j_q$, and the local clocks offset $\Lambda_{i,j}(k,q) = c^i_k - c^j_q$; where $\Delta_{i,j}(k,q) = r^i_k - r^j_q = 0$. Given a real-time instance t, we define the *(local clock) synchrony bound* $\psi(t) = \max(\{\Lambda_{i,j}(k,q) : p_i, p_j \in N \wedge \Delta_{i,j}(k,q) = 0\})$ as the maximal clock offset among the system nodes.

One may consider p_i's *(clock) skew*, $\rho_i = \lim_{\Delta_{i,i}(k,q)\to 0} \delta_{i,i}(k,q)/\Delta_{i,i}(k,q) \in [\rho_{\min}, \rho_{\max}]$, where $\rho_{\min}$ and $\rho_{\max}$ are known constants [5, 7]. The clock skew of MicaZ [15] nodes is bounded by a constant that is significantly smaller than the communication delays. Therefore, our simulations assume a zero skew. We validate these simulations in the MicaZ platform.

2.2 Pulses

Each node has hardware supported timer for generating *(periodic) pulses* every P (phase) time units. Denote by $c^i_{q_k}$ the $k - th$ time in which node p_i's timer triggers a pulse, immediately after performing the READ operation for the $q_k - th$ time. The term *timeslot* refers to the period between two consecutive pulses at times $c^i_{q_k}$ and $c^i_{q_{k+1}}$. We say that $t_i = c^i_{q_k} \bmod P$ is p_i's *(pulse) phase* value. Namely, whenever $t_i = 0$, node p_i raises the event $timeslot(s_i)$, where $s_i = k \bmod T$ is p_i's (broadcasting) timeslot number and $T > 1$ is the *TDMA frame size*.

2.3 The MAC layer

The studied algorithms use packet transmission schemes that employ communication operations for receiving, transmitting and carrier sensing. Our implementation considers merely the latter two operations, as in the Beeps model [2], which also considers the period prior to communication establishment.

We denote the operations' time notation (timestamp) in the format $\langle timeslot, phase\rangle$, where $timeslot \in [0, T-1]$ and $phase \in [0, P-1]$. We assume the existence of efficient mechanisms for timestamping packets at the MAC layer that are executed by the transmission operations, as in [4, 5]. We assume the existence of an efficient upper-bound, $\alpha \ll P$, on the communication delay between two neighbors, that, in this work, has no characterized and known distribution.

2.4 Task definition

The problem of *(decentralized) local pulse synchronization* considers the rapid reduction of all *local synchrony bounds* $\psi \geq \max(\{\Lambda_{i,j}(k,q) : p_i, p_j \in N \wedge p_j \in \mathcal{N}_i^T \wedge \Delta_{i,j}(k,q) = 0\})$, where $\mathcal{N}_i^T$ refers to p_i's recent neighbors, see figure 3.1 for definition. Given the synchrony bound $\psi \geq 0$, we look at the *convergence (rate bound)*, ℓ_ψ, which is the number of TDMA frames it takes to reach ψ. Recall that we consider only forward clock adjustments. We also study local pulse synchronization's relation to MAC-layer, network scalability and topological changes.

Chapter 3

Algorithmic Strategies for Pulse Synchronization

Pulse synchronization solutions require many considerations, e.g., non-deterministic delays and transmission interferences. Before addressing the implementation details, we simplify the presentation by first presenting *(algorithmic) strategies* in which the nodes learn about their neighbors' clock values without delays and interferences.

We present two strategies that align the TDMA timeslots by calling the function $\text{ADJUST}(aim)$ immediately before their broadcasting timeslot, see figure 1.2. The first strategy, named *Cricket*, sets aim's value according to neighbors that have the most similar phase values. The second strategy, named *Grasshopper*, looks into a greater set of neighbors before deciding on aim's value. Both strategies are based on the relations among the nodes' phase values, see figure 3.1 for definitions.

3.1 Cricket strategy

This strategy acts upon the intuition that gradual pairwise adjustments are most preferable. Node p_i raises the event timeslot(s_i), when $t_i = 0$, and adjusts its local clock according to Equation (3.1). At this time, $PhaseOrder_{\gamma_i}$'s first item has zero value, because it refers to p_i's own pulse, the second item refers to p_i's successor and the last item refers to p_i's predecessor.

Learning about neighbors' clock values

At any real-time instance t, p_i's *reach set*, $R_i(t) = \{p_j\} \subseteq N$, represents the set of nodes, p_j, that receive p_i's transmissions. At the MAC layer, the real-time instance t refers to the time in which p_j raises the carrier sense event. The set *recent neighbors*, $\mathcal{N}_i^T = \{p_j \in N :$ starting-time$(s_j) \in [t, t'] \wedge p_i \in R_j(t(s_j))\}$, refers to nodes whose broadcast in timeslot s_j, arrive to node p_i, where t is a real-time instance that happens T timeslots before the real-time instance t' and starting-time$(s_j) \in [t, t']$ refers to the starting time of p_j's timeslot.

Locally observed pulse profiles

Given a real-time instance t and node $p_i \in N$, we denote the *locally observed pulse profile* by $\gamma_i(t) = (\langle s_j, t_j \rangle)_{p_j \in \mathcal{N}_i^T}$, as a list of p_i's recently observed timestamps during the passed T timeslots before t. We sometimes write γ_i, rather than $\gamma_i(t)$, when t refers to the starting time of p_i's timeslot.

Phase orders

Let $Order = (p_{i_k})_{k=0}^{n-1}$ be an ordered list of nodes in N, where p_i's predecessor and successor in N are $p_{i_{k-1 \bmod n}}$, and respectively, $p_{i_{k+1 \bmod n}}$. The ordered list, $PhaseOrder_{\gamma_i}$, of the pulse profile, γ_i, is sorted by the phase field of γ_i's timestamp $\langle timeslot_j, phase_j \rangle \in \gamma_i$.

Predecessors, successors, heads, and tails

Given a node, p_i, and its view on the pulse profile, γ_i, we define the $predecessor_i$ and the $successor_i$ to be p_i's predecessor, and respectively, successor in $PhaseOrder_{\gamma_i}$. Moreover, we define $head_{\gamma_i} = (t_i - t_{pr}) \bmod P$ and $tail_{\gamma_i} = (t_{su} - t_i) \bmod P$ as the phase difference between p_i's phase value, t_i and $predecessor_i = p_{pr}$, and respectively, $successor_i = p_{su}$. These imply that $predecessor_i$ is pulsed $head_{\gamma_i}$ units of time before node p_i and $successor_i$ is pulsed $tail_{\gamma_i}$ units of time after node p_i.

Figure 3.1: Pulse profiles and the relations among the nodes' phase values

$$aim_{\gamma_i} = \begin{cases} head_{\gamma_i} & : head_{\gamma_i} < tail_{\gamma_i} \ \text{JUMP} \\ 0 & : head_{\gamma_i} > tail_{\gamma_i} \ \text{WAIT} \\ head_{\gamma_i} \text{or } 0\text{; each with probability } \frac{1}{2} & : head_{\gamma_i} = tail_{\gamma_i} \ \text{MIX} \end{cases} \tag{3.1}$$

The cricket strategy considers two types of steps: Pure deterministic actions (JUMP and WAIT) and a non-deterministic one (MIX).

∘ JUMP**:** Whenever node p_i is closer to its predecessor than to its successor ($head_{\gamma_i} < tail_{\gamma_i}$), it catches up with its predecessor by adding $head_{\gamma_i}$ to its clock value, which is the phase difference between itself and its predecessor.

◦ WAIT: Whenever p_i is closer to its successor than to its predecessor ($head_{\gamma_i} > tail_{\gamma_i}$), p_i simply waits for its successor to catch up.

◦ MIX: Node p_i needs to break symmetry whenever it is as close to its predecessor as it is to its successor ($head_{\gamma_i} = tail_{\gamma_i}$). In this case, p_i randomly chooses between JUMP and WAIT.

3.2 Local dominant pulses

Let us look into a typical convergence of the cricket strategy, see figure 3.2. Given two nodes, $p_i, p_j \in N$, and p_i's locally observed pulse profile, $\gamma_i(t)$, we say that p_j's pulse (phase value) *locally dominates* the one of p_i, if $head_{\gamma_i} < tail_{\gamma_i}$ and p_j is p_i's predecessor in γ_i. Observe that clock updates can result in a chain reaction, see figure 3.2-left. Lengthy chain reactions can prolong the convergence up to $\mathcal{O}(n)$ TDMA frames, see figure 3.2-right.

3.3 Global dominant pulses

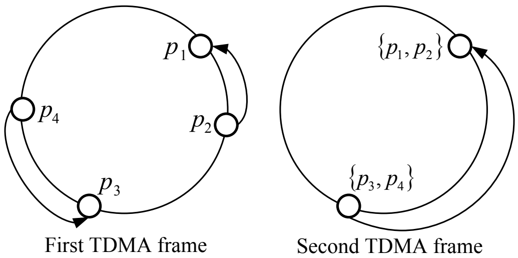
In figure 3.2-right, all nodes align their timeslots with the one of p_1, because p_1's pulse immediately follows the maximal gap in γ_i. Pulse gaps provide useful insights into the cricket strategy convergence. Given node $p_i \in N$, its pulse profile γ_i and $k \in [1, \mid \mathcal{N}_i^T \mid]$, we obtain the *(pulse) gaps* between γ_i's consecutive pulses, $Gap_{\gamma_i}(k) = (PhaseOrder_{\gamma_i}[k].phase - PhaseOrder_{\gamma_i}[k-1].phase)$. For the case of $k = 0$, we define $Gap_{\gamma_i}(0) = (P - PhaseOrder_{\gamma_i}[\mid \mathcal{N}_i^T \mid].phase)$. The set, $MaxGap_{\gamma_i}$, of pulses that immediately follows the maximal gap in γ_i are named *global dominates*, see Equation (3.2).

$$MaxGap_{\gamma_i} = \underset{k \in [0, |\mathcal{N}_i^T|]}{\operatorname{argmax}} (Gap_{\gamma_i}(k)) \tag{3.2}$$

Given three nodes, $p_i, p_j, p_\ell \in N$, p_i's locally observed pulse profile, $\gamma_i(t)$, $j \in MaxGap_{\gamma_i}$ and $i, \ell \notin MaxGap_{\gamma_i}$, we say that p_j's pulse *globally dominates the one of* p_i, if at least one of the following holds: (1) $i = j$ (2) p_j's pulse locally dominates the one of p_i, or (3) p_j's pulse globally dominates the one of p_ℓ and p_ℓ's pulse locally dominates the one of p_i. We define node p_i's clock offset towards its preceding global dominant pulse as $DominantPulse_i = P - PhaseOrder_{\gamma_i}[k].phase$, where $k \in MaxGap_{\gamma_i}$ refers to p_i's global dominant pulse, see Equation (3.2).

We define $OneGlobal(\gamma_i) = (\mid MaxGap_{\gamma_i} \mid = 1)$ to be true whenever γ_i encodes a single global

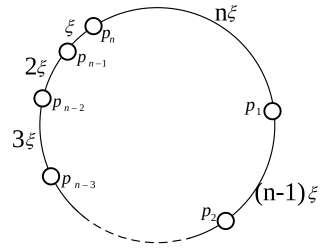
Cricket strategy convergence during the first two TDMA frames.



Set up: Given that the broadcasting schedule of nodes $N = \{p_1, \ldots, p_4\}$ is by their index value, the pulse profiles, $\{\gamma_i\}_{p_i \in N}$, encode pulses such that initially p_1 and p_3 have local dominant pulses.

Convergence: In the first TDMA frame, nodes $N(1) = \{p_2, p_4\}$ converge towards their respective local dominant pulses in $p_y \in \{p_1, p_3\}$. In the second TDMA frame, the local dominant pulses are nodes $p_y \in \{p_1, p_2\}$. Note that node p_3's pulse is no longer a local dominant and it adjusts its phase according to Equation (3.1), i.e., $N(2) = \{p_3, p_4\}$.

Chain reactions of pulse updates: An example with $\mathcal{O}(n)$ TDMA frames before convergence.



Set up: For $\xi > 0$ and $p_i \in N$, the pulse profiles $\{\gamma_i\}_{p_i \in N}$ encode pulses in which node p_{i+1}'s pulse occurs $(n - i)\xi$ clock units after p_i's pulse.

Convergence: In the first TDMA frame, only node p_n can take JUMP action to align with its neighbor local clock, p_{n-1}, because its $head_{\gamma_n} < tail_{\gamma_n}$. In the second TDMA frame, nodes $N(2) = \{p_n, p_{n-1}\}$ adjust their clocks to be aligned with p_{n-2}. Thus, in the $(n - 1)$-th TDMA frame, nodes $N(n - 1) = \{p_n, p_{n-1}, \ldots, p_2\}$ align with node p_1. Therefore, $n - 1$ TDMA frames are needed before convergence.

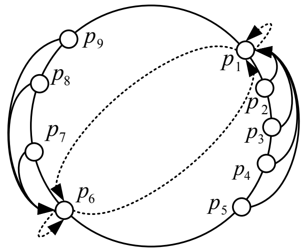
Figure 3.2: Typical convergence process of the cricket strategy.

dominant pulse. For the cases in which there is more than one, we define the term next (global) dominant pulse for node p_i, where $i \in MaxGap_{\gamma_i}$ refer to p_i's global dominant pulse. In this case, p_i's *next (global) dominant pulse*, $NextDominantPulse_i = DominantPulse_{pr}$, is p_i's predecessor's global dominant pulse, where $predecessor_i = p_{pr}$.

Next we present the grasshopper strategy, which uses the notion of global dominant pulses to avoid lengthy chain reactions in order to achieve a faster convergence.

3.4 Grasshopper strategy

This strategy is based on the ability to see beyond the immediate predecessor and local dominant pulses. The nodes converge by adjusting their local clocks according to the phase value of



Set up: Given nodes $N = \{p_1, \dots, p_9\}$, the pulse profiles, $\{\gamma_i\}_{p_i \in N}$, encode pulses, such that initially p_1 and p_6 are global dominants.
Convergence: During the first TDMA frame, nodes $p_x \in \{p_2, \dots, p_5\}$ and $p_y \in \{p_7, p_8, p_9\}$ take the JUMP action to align their local clocks with their respective preceding global dominant pulse, p_1 and p_6 (solid lines). Whereas, node p_1 and p_6 take the MIX action to either stay or align to next dominant pulse (dotted lines).

Figure 3.3: Typical convergence process of the grasshopper strategy.

their global dominant pulses, and by that avoid lengthy chain reactions of clock updates.

Equation (3.3) defines the adjustment value, $aim(\gamma_i)$, for the grasshopper strategy. Whenever node $p_i \in N$ notices that its clock phase value is dominated by the one of node $p_j \in N$, node p_i aligns its clock phase value with the one of p_j, see the JUMP step. Thus, whenever a single global dominant pulse exists, the convergence speed-up is made simple, because all nodes adjust their clock values according to the dominant pulse of p_j. Thus, there are no chain reactions of clock updates. Note that node p_j does not adjust its clock, see the WAIT step. For the possible case of many global dominant pulses, we take the mixed strategy approach, see the MIX step. Here, chain reactions of clock updates can occur. However, they occur only among the nodes whose clock phase values are global dominants, see figure 3.3.

$$aim_{\gamma_i} = \begin{cases} DominantPulse_i & : DominantPulse_i < P & \text{JUMP} \\ 0 & : DominantPulse_i = P \ \wedge OneGlobal(\gamma_i) & \text{WAIT} \\ NextDominantPulse_i \text{ or} & : \textbf{else} & \text{MIX} \\ 0; \text{ each with probability } \frac{1}{2} & & \end{cases} \quad (3.3)$$

Chapter 4

Strategies Implementation

This chapter delineates the challenges and techniques related to the implementation of the proposed pulse synchronization algorithms.

4.1 Platform and architecture

We implement the proposed pulse synchronization strategies using the wireless sensor network platform Tiny Operating System (TinyOS 2.1.0) [11]. In this platform, different components interact by means of delegated interfaces to achieve an event-driven programming model. We use TinyOS Simulator (TOSSIM) to evaluate our algorithmic design in a simulated radio environment [10]. Our implementation is then deployed into MicaZ motes for validation. The components of our simulation and platform implementation are depicted in figure 4.1.

4.2 Pulse overshooting

Pulse synchronization solutions require many considerations, e.g., non-deterministic delays and transmission interferences. Clock adjustments in the presence of non-deterministic communication delays can result in overshooting the targeted clock values. Therefore, it is crucial to obtain accurate timestamps at the MAC layer, cf. [4]. Moreover, we employ an adaptive clock adjustment technique that takes into consideration the aimed clock value, aim, in order to avoid overshooting. Namely, when the target is greater than the bound on the communication delay, $aim_{\gamma_i} > \alpha$, the local clock is adjusted by $aim_{\gamma_i} - \alpha$. However, when $aim_{\gamma_i} < \alpha$, we use

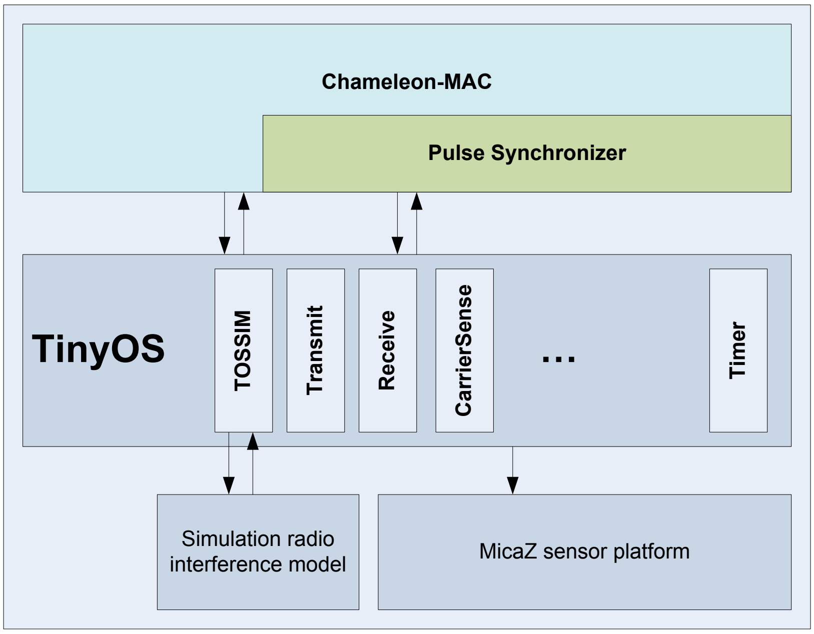


Figure 4.1: The implementation architecture and components interaction.

$aim_{\gamma_i} \cdot \beta^{1-(\beta)^{\epsilon}}$ for gradual adjustment of clock values, where $\beta = aim_{\gamma_i}/P$ and $\epsilon = 0.1$, see Equation (4.1)

$$AdjustClock(aim) = \begin{cases} aim_{\gamma_i} - \alpha & : aim_{\gamma_i} > \alpha \\ aim_{\gamma_i} \cdot \beta^{1-(\beta)^{\epsilon}} & : aim_{\gamma_i} \leq \alpha \end{cases} \tag{4.1}$$

4.3 Mitigating synchrony bound jitter

The proposed pulse synchronization algorithms are required to maintain the low synchrony bound achieved. Figure 4.2 displays the synchrony bound level and its progression in time. We notice that the synchrony bound decreases to 1% $(50\ \mu sec)$ of the timeslot size, P, however, this is followed by series of oscillations between $1\% - 10\%$. The synchrony bound jitters in time as pulses approach a synchronized state. We observe that this undesirable behavior appears when the local clock offsets are smaller than the non-deterministic communication delay, α.

The effect of synchrony bound jitter is mitigated by defining a *dismissal interval* $\omega < \alpha$ around every pulse. Timestamps falling in this interval are ignored by corresponding nodes, cf. figure 4.3. In the figure, node p_1 ignores p_2's timestamp because p_2's timestamp falls in p_1's ω interval. However, p_1 accepts p_3's timestamp because p_3's timestamp does not fall in p_1's ω interval. This simple technique refrains the nodes from basing adjustments on less accurate

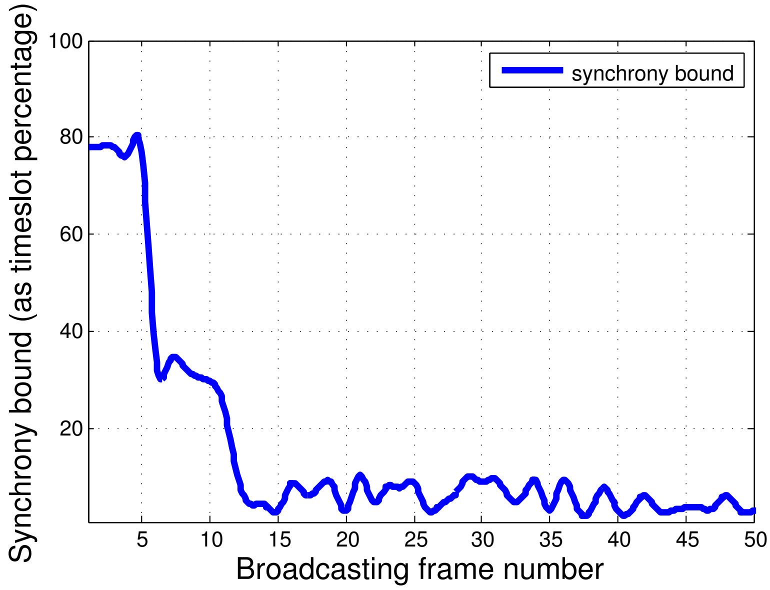


Figure 4.2: Synchrony bound jitter effect as pulses approach a synchronized state.

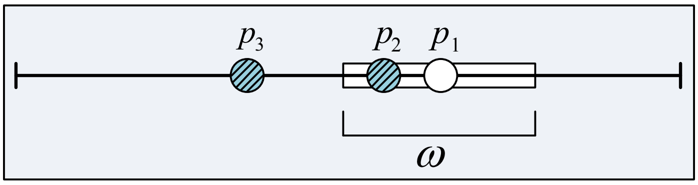


Figure 4.3: Timestamp jitter mitigation using dismissal interval ω, such that node p_1 ignores p_2's timestamp because it falls in its interval, while it accepts p_3's timestamp in the pulse synchronization process.

timestamps. In our simulation and platform experiments, we consider $\omega = 1\%$ of P.

4.4 Memory consumption

The grasshopper strategy considers sorted pulse profiles of $\mathcal{O}(n(\log(P) + \log(T)))$ memory bits. Note in the cricket strategy, the values of $head_{\gamma_i}$ and $tail_{\gamma_i}$ can be updated on the fly, i.e., sorting can be avoided. Thus, the cricket strategy requires using $\mathcal{O}(\log(P) + \log(T))$ memory bits.

4.5 Algorithm pseudocode

Algorithm 1: Pulse Synchronization Algorithms

```
Common variables and functions
P = phase size
T = TDMA frame size
α = upper bound on communication delay
ω = dismissal interval

phase: [0,P-1] = phase value
aim:[0,P-1] = phase adjustment value
str:{0,1} = define strategy to execute
s:[0, T-1] = next timeslot to broadcast
Timestamps:{[0,P-1] × [0,T-1]} = a set of timestamps

Upon carrier_sense(timestamp)
    if timestamp mod P ∉ ω then
        Timestamps ← Timestamps ∪ {timestamp mod P} (* Record timestamp *)
end

Function AdjustClock(aim)
    if aim > α then  return (aim - α)
    return ⟨ aim · β^(1-β^ε) ⟩, where β = aim/P, ε > 0
end

Upon timeslot(t)
    if t mod T = 0 then
        (* New broadcasting frame *)
        phase ← phase + Strategy(str) (* Adjust phase according to strategy *)
        Timestamps ← ∅
    if t = s then transmit() (* My timeslot *)
end
Function Strategy(str)
    if str = 0 then
        (* Run cricket strategy *)
        ⟨head, tail⟩ ← FindHeadAndTail() ()
        if IsAdjusting(head, tail) then Return ⟨AdjustClock(head)⟩
    else if str = 1 then
        (* Run grasshopper strategy *)
        aim ← FindDominantPulse()
        return ⟨AdjustClock(aim)⟩
end
```

Cricket variables and functions
tail = successor's phase offset
head = predecessor's phase offset

Function FindHeadAndTail()
  $head \leftarrow P - \max(Timestamps)$ (* Offset towards predecessor pulse *)
  $tail \leftarrow \min(Timestamps)$ (* Offset towards successor pulse *)
  return ⟨ *head, tail* ⟩
end

Function IsAdjusting($head$**,** $tail$**)**
  if $head = tail$ **then return** ⟨ *random_number* > $\frac{1}{2}$ ⟩
  return ⟨ $head < tail$ ⟩
end

Grasshopper variables and functions
DominantPulse = preceding global dominant pulse
NextDominantPulse = second preceding global dominant pulse
Gap = set of gap sizes in the order of timestamps reception
MaxGap = set of timestamps preceded by maximal gap

Function FindDominantPulse()
  (* Function returns offset towards the nearest global dominant pulse *)
  for $timestamp_i \in Timestamps$ **do**
    $Gap \leftarrow Gap \cup \{timestamp_i - timestamp_{i-1}\}$ (* Find gaps between pulses *)
  $MaxGap \leftarrow \{timestamp_i\} : timestamp_i - timestamp_{i-1} = \max(Gap)$
  $DominantPulse \leftarrow P - \max(MaxGap)$
  if *(DominantPulse = P)* $\wedge$ *(| MaxGap | > 1)* **then**
    (* More than one global dominant pulse might exist *)
    $MaxGap \leftarrow MaxGap \setminus \max(MaxGap)$
    if *random_number* > $\frac{1}{2}$ **then**
      (* Find offset towards nearest global dominant pulse *)
      $NextDominantPulse \leftarrow P - \max(MaxGap)$
      return $\langle NextDominantPulse \rangle$
  return $\langle DominantPulse \rangle$
end

Chapter 5

Experimental Evaluation

We use computer simulations and MicaZ platform experiments to show that: (1) both proposed algorithms achieve a small synchrony bound, and (2) the grasshopper, which has a higher resource consumption cost, converges faster than the cricket.

5.1 Experiments design

The proposed algorithms aim at aligning the TDMA timeslots during the process of communication establishment. Namely, during the period in which the MAC layer assigns the TDMA timeslots. Since communication interruptions can occur, the nodes might not correctly observe their local pulse profiles.

In other words, the convergence process of pulse synchronization algorithms inherently depends on: (1) the MAC layer ability to guarantee (eventually) interruption-free communications, and (2) the system ability to infer on pulse profiles during periods in which there are no guarantees for interruption-free communications. In order to overcome these inherent dependencies, our tests consider benchmarks that are based on common (external) sources of synchronization that serve as control tests. We consider the result parameters, ψ, and ℓ_ψ, and compare between the experiments and their control tests, where ψ is the synchrony bound and ℓ_ψ is the convergence time.

5.1.1 Experiment and control test for dependency (1)

The former dependency is mitigated by experimenting with Chameleon-MAC, a self-$\star$ TDMA protocol [9], and considering a control test that uses a *preassigned TDMA*. Here, the external common source of synchronization is provided by a MAC layer that assigns its timeslots before any communication. The preassigned TDMA guarantees that the convergence process of the pulse synchronization algorithms does not include communication interruptions due to timeslot assignment of the MAC layer. Thus, the preassigned TDMA serves as a baseline from which we can estimate the degree of improvement that would result from perhaps employing better MAC algorithms.

5.1.2 Experiment and control test for dependency (2)

The latter dependency is mitigated by experimenting with Chameleon-MAC, a self-$\star$ TDMA protocol [9], and considering a control test that uses a *centralized pulse synchronizer*. Here, the external common source of synchronization is provided by a (base-station) node that broadcasts a distinguished alignment message once in every TDMA frame. The other nodes never broadcast before receiving the base-station's message, and thus, all nodes can infer on their pulse profiles. Thus, the centralized pulse synchronizer serves as a baseline for estimating the overheads imposed by the autonomous design.

Next we present the simulation and testbed results. For each setting, we average the performance of 8 executions. Furthermore, our simulation and testbed experiments consider a timeslot size of, $P = 5\ msec$, and respectively, $P = 20\ msec$, frame size of d_i timeslots, where d_i refers to node p_i's interference degree, and an upper-bound on communication delay of, $\alpha = 5\%$ of P.

5.2 Simulation experiments

The proposed pulse synchronization algorithms are simulated using TOSSIM [10] on single-hop, multi-hop and mobile ad hoc networks. We observe the synchrony bound and convergence time, and study the proposed algorithms' relation to MAC-layer, network scalability and topological changes.

5.2.1 Single-hop Ad Hoc Network

Both algorithms reduce the synchrony bound down to 1% of the timeslot size, see figure 5.1. Moreover, the synchrony bounds of the cricket and grasshopper are 24%, and respectively, 62% lower when using preassigned TDMA rather than Chameleon-MAC [9]. However, these values drop to 0.04%, and respectively, 0.4% after convergence. Furthermore, the grasshopper convergence is 5.4 times faster than of the cricket. In addition, the cricket and grasshopper converge 6.8%, and respectively, 40% times faster when using preassigned TDMA rather than Chameleon-MAC.

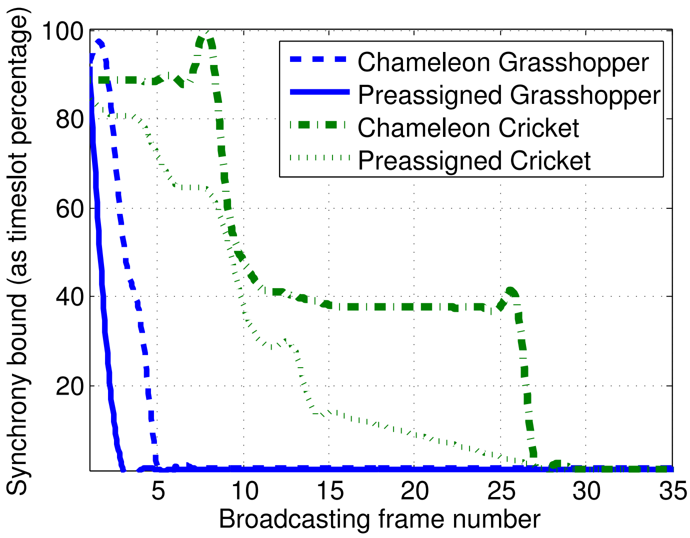


Figure 5.1: The studied algorithms in a single-hop network of 20 nodes using preassigned TDMA and Chameleon-MAC. Both algorithms, for both MAC settings, drop the synchrony bound to 1% (50 μsec). The convergence time (in number of frames) for the cricket and grasshopper is 3, and respectively, 27 for preassigned TDMA, and 5, and respectively, 29 for Chameleon-MAC.

We also study the algorithms' scalability by considering a variable number of nodes, $n \in \{10, 20, 30, \ldots, 100\}$. The grasshopper converges faster than the cricket as the number of nodes increases, cf. figure 5.2 (a) and (b). The convergence depends on the number of nodes. E.g., for 10% synchrony bound, $0.3n + 6.4$ and $0.0062n + 10.86$ are linear interpolations of the convergence time for the cricket, and respectively, grasshopper strategies. Moreover, $2(log_2(n) + 0.1)$ is a logarithmic interpolation of the grasshopper convergence time. The proposed algorithms affect the MAC throughput, which is the radio time utilization percentage, cf. figure 5.2 (c) and (d). Both algorithms eventually reach a throughput of 70%.

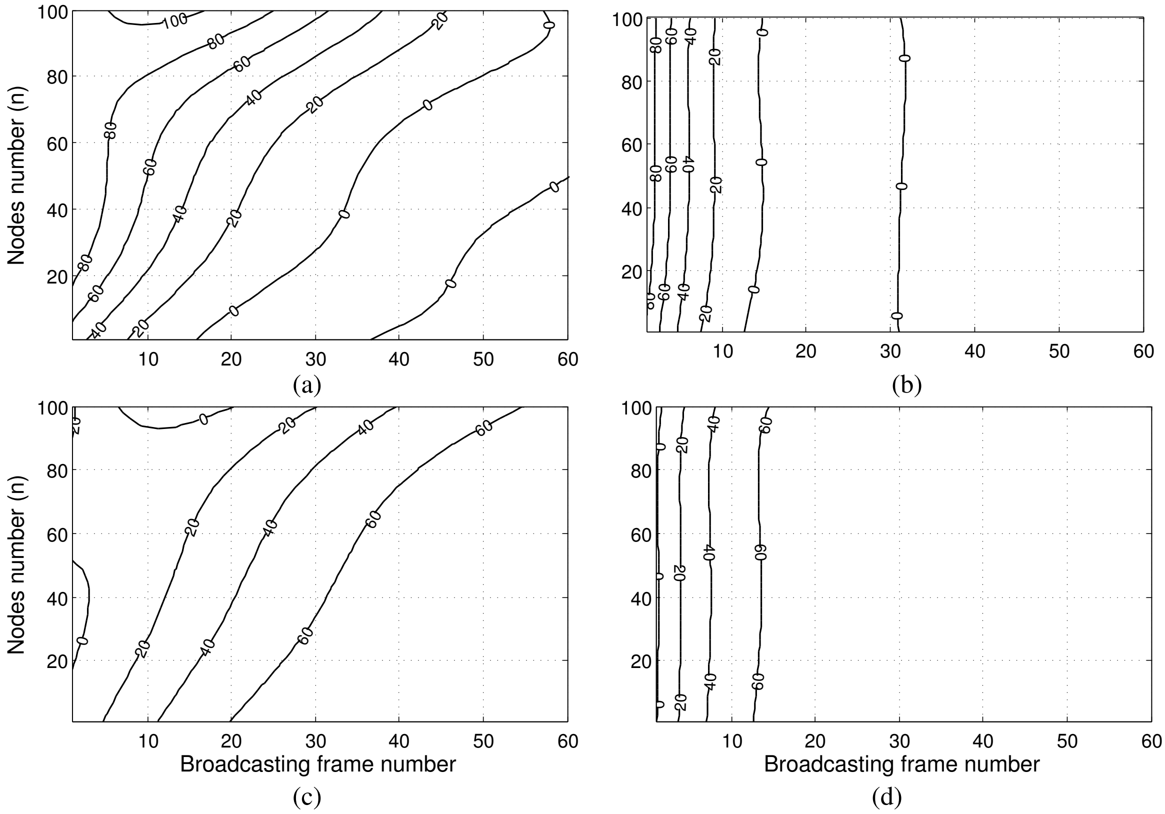
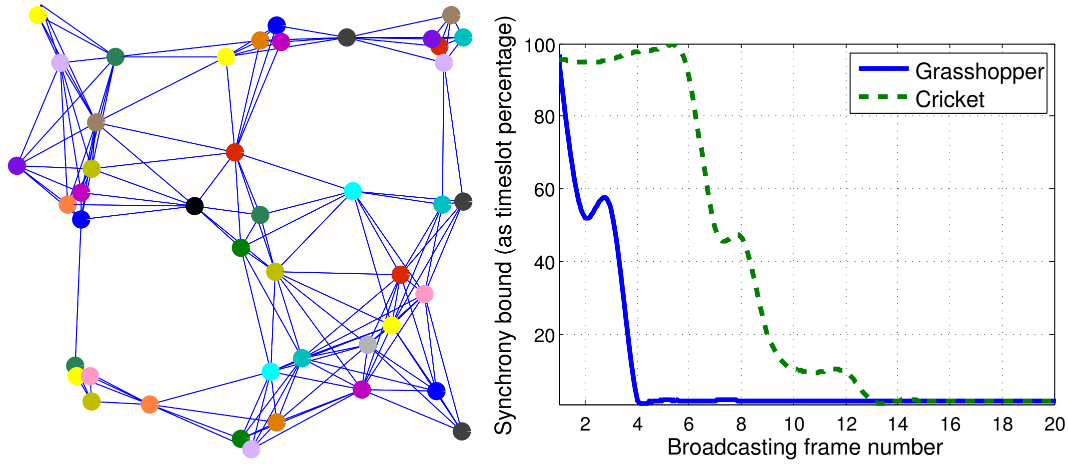


Figure 5.2: Synchrony bounds (as timeslot percentage) and throughput levels (as radio time utilized percentage) for single-hop networks of $n \in \{10, 20, 30, \cdots, 100\}$ nodes. Top and bottom contour plots show the synchrony bounds, and respectively, throughput levels for cricket (a) and (c), and grasshopper (b) and (d). Given these plots, the number of TDMA frames needed to reach to a particular synchrony bound (or throughput) by a given number of nodes can be estimated. E.g., 60 nodes reach 20% synchrony within 25 frames using the cricket strategy.



(a) (b)

Figure 5.3: A connected graph with 45 nodes and a 6-hops diameter in (a), Multi-hop network synchrony bound (as timeslot percentage) and convergence time (in number of TDMA frames) in (b) of the cricket and grasshopper algorithms using Chameleon-MAC.

5.2.2 Multi-hop Ad Hoc Network

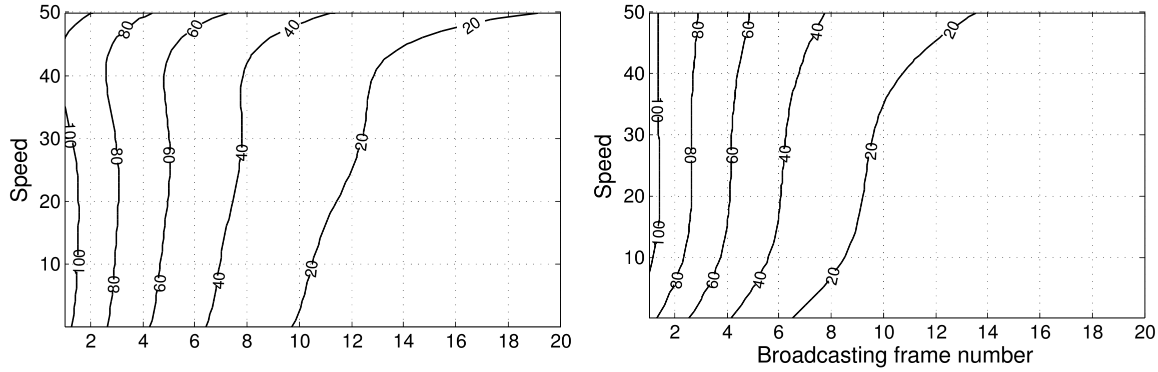
We consider a connected graph with 45 nodes, 6 hops diameter and an interference degree $d_i \in [5, 15]$. We use a set of 8 graphs similar to the one in figure 5.3 (a). Both algorithms reduce the synchrony bound down to 3% of the timeslot size. Thus, the achieved synchrony bounds in multi-hop networks are higher than in single-hop ones. This phenomena is well-known [see 8, for relevant lower bounds]. The grasshopper converge 3.25 times faster than the cricket, cf. figure 5.3 (b).

5.2.3 Mobile Ad Hoc Network

We borrow two mobility models from [9] for studying the proposed algorithms. In the first model, the nodes are traveling on a grid and thus their radio interferences follow a regular pattern. In the second model, we consider mobile node clusters that pass by each other and thus they experience transient radio interferences.

The first model places 72 mobile nodes on a grid with an interference degree of $d_i \in [2, 4]$ and a diameter of 12 hops. Nodes in even and odd rows travel in opposite directions by a constant *speed* $\in \{2, 4, \cdots, 50\}$ units every TDMA frame. We considers a transmission (interference) radius of 22 units, and study the effect of topological changes on the proposed algorithms.

The simulations show that both proposed algorithms reach to a synchrony bound of 10% of the timeslot size, cf. figure 5.4. Furthermore, we observe that the grasshopper is more resilient to



(a) (b)

Figure 5.4: The synchrony bound (as timeslot percentage) for the cricket (a) and grasshopper (b) using Chameleon-MAC and considering regular interferences. The neighborhood change rate increases with speed, causing the algorithms to spend longer time for convergence.

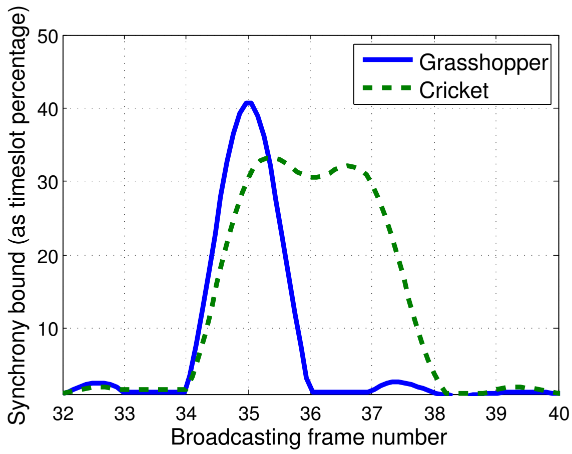


Figure 5.5: Cricket and grasshopper convergence with mobile clusters.

topological changes than the cricket.

We also study the algorithms re-convergence time when the synchrony bound is compromised due to transient radio interferences. We consider two mobile node clusters that, in the beginning of the simulation, do not interfere with each other. When the mobile clusters approach each other, the communication within each cluster is interrupted because of timeslots misalignment.

Within the first TDMA frames, both clusters converge. When the two clusters start to pass by each other, we observe transient interferences, cf. figure 5.5. Note that the grasshopper converges in half the time consumed by the cricket.

5.3 Testbed experiments

We validate the single-hop simulation results on MicaZ [15] motes and estimate the cost of our autonomous solutions. The testbed experiments consider a single-hop network of $n \in \{5, 10, 15\}$ nodes, and use Chameleon-MAC to establish communication. We validate the simulation synchrony bound and convergence time, and evaluate the communication success rate during and after convergence.

The grasshopper converges 6.5 times faster than the cricket, cf. figure 5.6. Furthermore, the transmission success rate increases during the convergence process.

We estimate the overheads of our autonomous design via control tests that use a centralized pulse synchronizer. By executing both experiments, centralized and autonomous, we estimate the synchrony bound ratio between the two approaches. The centralized and autonomous pulse synchronizers drop the synchrony bound down to 0.8%, and respectively, 2.0%, cf. figure 5.6. The autonomous design overheads are merely 2.5 times higher than the centralized one; fixed ratio for any n.

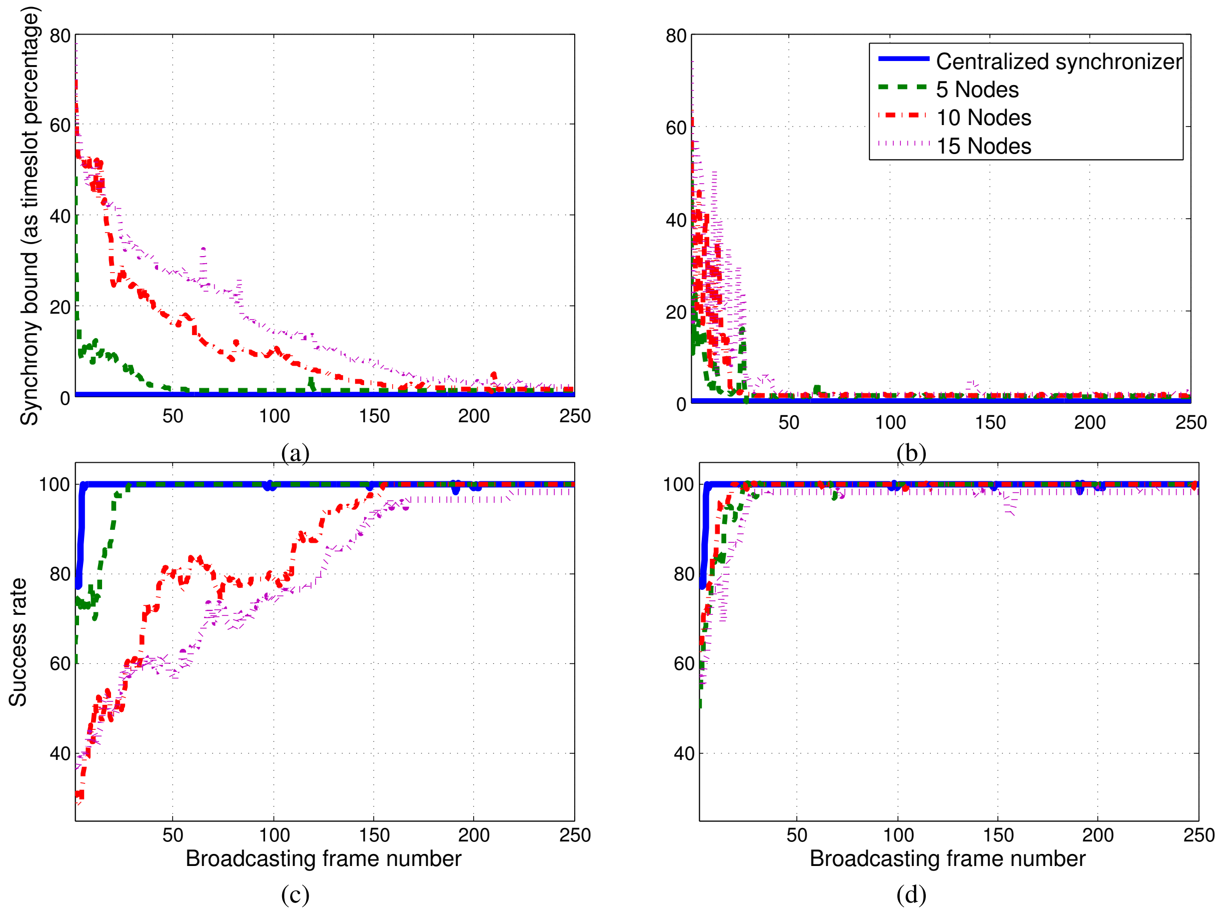


Figure 5.6: Testbed synchrony bounds and success rates. The cricket and grasshopper converge $n \in \{5, 10, 15\}$ nodes within $\{50, 165, 200\}$, and respectively, $\{9, 20, 32\}$ TDMA frames to a synchrony bound of 2% $(400\ \mu sec)$ of P, see (a) and (b). The packet delivery success rate is presented in (c) and respectively (d).

Chapter 6

Discussion

The prospects of safety-critical vehicular systems depend on the existence of predictable communication protocols that divide the radio time regularly and fairly. This paper presents autonomous and self-$\star$ algorithmic solutions for the problem of TDMA timeslot alignment by considering the more general problem of (decentralized) local pulse synchronization. The studied algorithms facilitate autonomous TDMA-based MAC protocols that are robust to transient faults, have high throughput and offer a greater predictability degree with respect to the transmission schedule. These properties are often absent from current MAC protocol implantations for VANETs, see [1, 14].

We saw that avoiding clock update dependencies can significantly speed up the convergence and recovery processes. In particular, the grasshopper algorithm foresees dependencies among the clock updates, which the cricket cannot. However, dependency avoidance requires additional resources.

Existing vehicular systems often assume the availability of common time sources, e.g., GPS. Autonomous systems cannot depend on GPS services, because they are not always available, or preferred not to be used, due to their cost. Arbitrarily long failure of signal loss can occur in underground parking lots and road tunnels. Moreover, some vehicular applications cannot afford accurate clock oscillators that would allow them to maintain the required precision during these failure periods.

By demonstrating the studied algorithms on inexpensive MicaZ motes, we have opened up the door for *hybrid-autonomous* designs (cf. centralized pulse synchronizer in Section 5). Namely, nodes that have access to GPS, use this time source for aligning their TDMA timeslots, whereas nodes that have no access to GPS, use the studied strategies as dependable fallback for catching up with nodes that have access to GPS.

We expect applicability of the hybrid-autonomous design criteria to other areas of VANETs and vehicular systems. For example, spatial TDMA [14] protocols base their timeslot allocation on GPS availability. As future work, we propose dealing with such dependencies by adopting the hybrid-autonomous design criteria.

Bibliography

[1] K. Bilstrup, E. Uhlemann, E. G. Ström, and U. Bilstrup. Evaluation of the ieee 802.11p mac method for vehicle-to-vehicle communication. In *VTC Fall*, pages 1–5. IEEE, 2008.

[2] A. Cornejo and F. Kuhn. Deploying wireless networks with beeps. In *Distributed Systems and Networks (DISC)*, pages 148–162, 2010.

[3] A. Daliot, D. Dolev, and H. Parnas. Self-stabilizing pulse synchronization inspired by biological pacemaker networks. In *Stabilization, Safety, and Security of Distributed Systems - 6th International Symposium*, pages 32–48, 2003.

[4] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. In *USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[5] T. Herman and C. Zhang. Best paper: Stabilizing clock synchronization for wireless sensor networks. In *Stabilization, Safety, and Security of Distributed Systems*, pages 335–349, 2006.

[6] E. N. Hoch. Self-stabilizing byzantine pulse and clock synchronization. Master's thesis, School of CSEat The Hebrew Univ. of Jerusaelm, 2007.

[7] J.-H. Hoepman, A. Larsson, E. M. Schiller, and P. Tsigas. Secure and self-stabilizing clock synchronization in sensor networks. In T. Masuzawa and S. Tixeuil, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 4838 of *LNCS*, pages 340–356. Springer, 2007.

[8] C. Lenzen, T. Locher, and R. Wattenhofer. Tight bounds for clock synchronization. *J. ACM*, 57(2), 2010.

[9] P. Leone, M. Papatriantafilou, E. M. Schiller, and G. Zhu. Chameleon-mac: Adaptive and self-* algorithms for media access control in mobile ad hoc networks. In *Stabilization, Safety, and Security of Distributed Systems*, pages 468–488, 2010.

[10] P. Levis, N. Lee, M. Welsh, and D. E. Culler. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *ACM SenSys*, pages 126–137, 2003.

[11] P. Levis, S. Madden, J. Polastre, R. Szewczyk, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[12] D. Lucarelli and I.-J. Wang. Decentralized synchronization protocols with nearest neighbor communication. In *ACM SenSys*, pages 62–68, 2004.

[13] R. E. Mirollo, Steven, and H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM*, 50:1645–1662, 1990.

[14] K. Sjöberg, E. Uhlemann, and E. G. Ström. Delay and interference comparison of CSMA and self-organizing TDMA when used in VANETs. In *IEEE Wireless Communications and Mobile Computing Conference*, pages 1488–1493, 2011.

[15] Crossbow Technology Inc. MicaZ specs. `http://bit.ly/roPGqJ`, 2009.

[16] A. Tyrrell, G. Auer, and C. Bettstetter. Fireflies as role models for synchronization in ad hoc networks. In *ICST BIONETICS*. ACM, 2006.

[17] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In J. Redi, H. Balakrishnan, and F. Zhao, editors, *ACM SenSys*, pages 142–153. ACM, 2005.