

A Self-organizing Distributed Reinforcement Learning Algorithm to Achieve Fair Bandwidth Allocation for Priority-based Bus Communication

Tobias Ziermann, Nina Mühleis, Stefan Wildermann and Jürgen Teich
Hardware/Software Co-Design
Department of Computer Science
University of Erlangen-Nuremberg
Germany
tobias.ziermann@informatik.uni-erlangen.de

Abstract—Due to the raising complexity in distributed embedded systems, a single designer will not be able to plan and organize the communication for such systems. Therefore, it will get more and more important to relieve the designer in that task. Our idea is a communication system that is capable to organize itself to satisfy predefined properties. In this paper, we want to solve the problem of establishing fair bandwidth sharing on priority-based buses by using simple local rules on the distributed system to avoid a single point of failure and cope with online system changes.

Based on a game theoretical analysis, a multi-agent reinforcement learning algorithm is proposed that establishes fair bandwidth distribution. The main idea is to penalize nodes that claim too much bandwidth by the other nodes. We experimentally evaluated the algorithm with different parameter settings. The algorithm showed to converge to a fair solution in any experiment. This means the system is able to completely self-organize without global information for our assumptions. In addition, we could figure out that we can configure a trade-off between convergence speed and computation effort. We hope this is a small first step towards totally self-organizing real-time systems.

Keywords—self-organizing, bus-based communication, multi-agent reinforcement learning

I. INTRODUCTION

The number of components in distributed embedded systems is increasing steadily. For example, in a distributed embedded system of a modern car, more than 50 electronic control units (ECUs) need to be planned and organized. Going hand in hand, the number of signals and messages that need to be taken care of explodes. This communication is mainly processed by event-triggered priority-based communication. An example for such a communication protocol is the Controller Area Network (CAN) [2]. Here, a priority is assigned to each message, and, when requesting the resource, the message with highest priority is granted exclusive access.

Due to its predictability in case of collisions, priority-based arbitration mechanisms are commonly used in real-time systems with hard deadlines, and applied for scheduling

[8]. The design of such systems is mainly done using analysis of the worst-case execution times [9]. However, when the average-case execution times of tasks significantly vary from their worst-case execution times, this approach can cause a waste of resources [8]. For example, in multimedia applications time spent for transmitting compressed video images may change seriously for each frame. In practice, this waste of resources leads to an unusability of the worst-case approach. In addition, the requirements of the tasks may not be known in detail. In this case, the system needs to follow an as good as possible approach. Another drawback of offline designed systems is that they are not flexible and thus hard to adapt online to system changes, such as removal or insertion of communicating nodes.

Therefore, we prefer to approach the goal of a working communication system from a different perspective. Starting with minimal constraints on the individual communication nodes, the nodes themselves should be able to establish communication. This should be done by using simple local rules with as less global information as possible. As the first step in this development process, we want to take a look at bandwidth-oriented communication. In future work, these constraints will be extended to cover a larger field of applications.

The remainder of the paper is organized as follows. Section II explains the context of this work, introducing CAN and game theory. In Section III, the priority-based medium access game is formulated, which gives us the possibility to implement and explain our methods. Section IV presents a distributed reinforcement learning algorithm to establish fair bandwidth distribution and explains the working principle with an example. In Section V, different parameter settings are analyzed for time to converge and inaccuracy in the converged state. Section VI describes how the learning algorithm can be improved, while Section VII concludes our work.

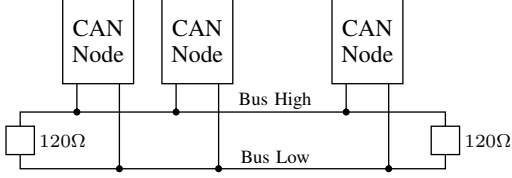


Figure 1: Setup of a Controller Area Network

II. OVERVIEW

A. Greater Goal

The work presented here is only a small piece of the greater goal to build *Organic Self-organizing Bus-based Communication Systems*. We will investigate an organic approach [7] for the analysis, design and optimization of bus-based communication systems. The goal of our approach is to overcome drawbacks of today’s pure offline designs that are based on worst-case estimations, are not expandable, and may easily degenerate when the environment or requirements change at run-time. In contrast, a decentralized approach using online self-organization would be able to monitor the actual traffic of the communication system and adapt either sending rates, probabilities, priorities, etc. accordingly.

This project intends to provide a) theoretical foundations on self-organization for bus-based communication architectures based on game theory and utility functions, b) models as well as a design methodology including learning techniques to implement such properties for conflicting requirements, such as deadlines and bandwidth, and c) a simulation testbed. Finally, d) a hardware demonstrator shall be developed in order to prove the benefits of the investigated approaches in a realistic environment.

B. CAN as Priority-based Medium Access

As described in the introduction, our target platform is CAN, even though any other event-triggered priority-based communication protocol could be used. An example setup is shown in Figure 1. CAN is an event-triggered message-oriented protocol. So, any node starts to transmit as soon as it has something to transmit and the bus is free. A bit-wise arbitration mechanism is used to avoid collisions when two nodes start transmitting at the same time. This arbitration mechanism guarantees a priority-based access depending on the identifier of the message. Without loss of generality, we assume in our experiments that each identifier is associated with exactly one node.

An event-triggered priority-based communication mechanism was chosen, because on the one hand it offers much more flexibility in changing the communication order than time-triggered approaches, but on the other hand offers the possibility to guarantee real-time properties using fixed priorities.

C. Introduction to Game Theory

Game theory is a branch of mathematics that can be used to analyze learning in multi-agent systems [1].

In general, a game G consists of a set of n players $N = \{1, 2, \dots, n\}$, a set of strategies S_i available for player i , and a specification of payoffs for each combination of player strategies. For an individual player i , the payoff is given by the utility function $u_i(\mathbf{s})$ where $\mathbf{s} = (s_1, \dots, s_n)$ are the strategies chosen by all players. If all players have chosen a strategy $s_i \in S_i$ and no one wants to change its strategy, an *equilibrium* is reached. A strategy vector $\mathbf{s} = (s_1, \dots, s_n)$ is called *Nash equilibrium* if no player can get a better response by changing its strategy unilaterally. This means that all players try to maximize their utility function while all other players keep their strategy. Defining a strategy vector $\mathbf{s}_{-i} = (s_1, s_2, \dots, s_{i-1}, s_{i+1}, \dots, s_n)$ that contains the strategies from all players aside from player i the Nash equilibrium can be formalized as follows.

Definition 1: A strategy vector \mathbf{s}^* is a Nash equilibrium iff

$$u_i(s_i, \mathbf{s}_{-i}^*) \leq u_i(s_i^*, \mathbf{s}_{-i}^*), \forall s_i \in S_i, \forall i.$$

D. Learning Games

Learning how to play games is almost as old as game theory itself [3]. One way is reinforcement learning, a learning method where the chosen actions are based on rewards that were received for previous actions. As multiple players are involved, the research field is called multi-agent reinforcement learning. A comprehensive survey can be found in [1].

Our priority-based access game falls into the field of fully cooperative static games. Fully cooperative means that the players try to jointly reach a common goal – In our case: fair bandwidth sharing. It is a static game, because for a given set of actions, the reward for each player is always the same and is not depending on any foreign environment or a system state. For these type of games, a centralized controller could learn an optimal joint-action using Q-learning [10]. However, when the players are independent decision makers, a coordination mechanism has to be introduced. Several communication-free methods have been developed to avoid this communication overhead [5], [4]. The problem with these algorithms is, that they assume that each player knows the rewards of all other players. In our scenario, this would cause a tremendous additional amount of communication.

For this reason, we chose to develop our own game-specific algorithm.

III. GAME THEORETICAL ANALYSIS

In this section, we describe how game theory can be used to organize priority-based medium access. This is an adaption of the Contention-based Medium Access Game described in [6].

A. Medium Access in Normal Form

For the further analysis, we assume given a system with several nodes that want to get access to a shared medium. In the following, we use the terms *node*, *user* and *player* interchangeably. Such a medium access can be described in *Normal Form* known from game theory. Here, for each strategy combination, the expected payoff of each user is given. For a two player game, this Normal form can be defined as a matrix as shown in Figure 2. Each cell contains the payoff $(u_1(s_1, s_2), u_2(s_1, s_2))$ of the players strategy dependent on the strategy selected by the other player. The strategy of player 1 is listed in the row, in the column the one of player 2.

The game presented in Figure 2 is a contention-based Medium Access Game introduced in [6]. For this scenario, valid strategies are sending data via the shared medium or waiting. The game assumes that each player wants to transmit data every time. If only one player wants to send data, he gets access granted to the medium and the payoff is 1. For the other player, the payoff is 0. If both try to send, none gets access and for both the payoff is 0.

		player 2	
		wait	send
player 1	wait	0, 0	0, 1
	send	1, 0	0, 0

Figure 2: Two player contention-based Medium Access Game in normal form

The primary goal of each player should be to maximize its payoff. However, this will lead to a conflict: On the one hand, if both try to send data, no player ever gets access to the medium. On the other hand, if both player choose to wait they will also get a payoff of 0.

B. The Contention-based Medium Access Game

To achieve fair bandwidth sharing, a mixed strategy for each player has been introduced in [6]. Instead of having just send and wait as possible strategies, a probability distribution on strategy space S_i is given. This means that player i sends with probability p_i and waits with probability $1 - p_i$.

Definition 2: The Medium Access Game G is defined as a tuple $G := (N, S_i, u_i)$ with $i \in N$ where

- $N = \{1, \dots, n\}$ is a set of n players.
- S_i is a set of mixed strategies available for player i .
- $u_i(\mathbf{p})$ is the utility function for player i dependent on the strategies chosen by all players $\mathbf{p} = (p_1, \dots, p_n)$.

The utility function $u_i(\mathbf{p})$ corresponds to the probability that a player tries to get access to the medium, while all other players decide to wait:

$$u_i(\mathbf{p}) = p_i \cdot \prod_{j \neq i} (1 - p_j). \quad (1)$$

In [6], it was shown that a Nash equilibrium exists for this game that leads to a fair bandwidth sharing.

C. Priority-based Medium Access Game

In priority-based communication systems, each participant has a unique priority. The medium access can be divided into rounds. Each round which corresponds to one bus access is divided into two phases: the arbitration phase and the transmission phase. This is illustrated in Figure 3. During the arbitration phase, the node with the highest priority that wants to send data is chosen. In the transmission phase, this node gets access to the shared medium. With this concept, collisions can be resolved as shown in Table 4. Here, we assume that player 2 has a higher priority than player 1. If only one player wants to access the medium, it gets a grant and the payoff is 1. A collision occurs when both players want to send. In this case, the player with the highest priority (here player 2) gets the grant.

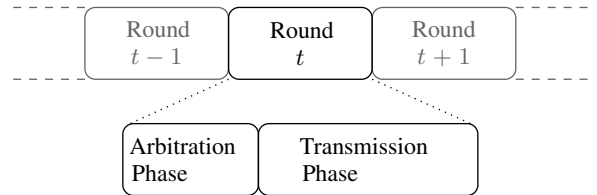


Figure 3: The two phases, *arbitration phase* and *transmission phase*, that define the rounds of the priority-based Access Game.

		player 2	
		wait	send
player 1	wait	0, 0	0, 1
	send	1, 0	0, 1

Figure 4: Two player priority-based Medium Access Game in normal form

For further analysis, we define a priority function $prior(i)$ that gives each player i a unique priority. This function is ordered as follows:

$$prior(1) < prior(2) < \dots < prior(n) \quad (2)$$

In the next step, we want to adapt Definition 2 to the priority-based Medium Access Game. Therefore, some assumptions about the system have to be made which are valid in the remainder of the paper:

- 1) Each node has only one message to send.
- 2) Each user has always a message to send.
- 3) Each user has a message of equal length.
- 4) The system is stable, so no technical failure occurs.

Definition 3: The priority-based Medium Access Game G is defined as a tuple $G := (N, S_i, u_i)$ with $i \in N$.

Like in the contention-based Medium Access Game, the utility function for a player i is equal to the probability to

successfully transmit data. In this game, player i sends a message if he chooses to send and all players with higher priority decide to wait. The nodes with lower priorities do not have an effect on the payoff. Thus, the utility function is given by:

$$u_i(\mathbf{p}) = p_i \prod_{j>i} (1 - p_j). \quad (3)$$

1) *Fairness*: Now, we want to give an overview how the priority-based Medium Access Game can be used to define and analyze fair bandwidth sharing. Detailed proofs can be found in [11]. Potentially, there are many different ways to define fairness. For the application presented in this paper, we define fairness as follows.

Definition 4: A strategy vector $\tilde{\mathbf{p}}$ is called fair if each user has the same probability to get access to the shared medium. This means that the following constraint always holds:

$$u_1(\tilde{\mathbf{p}}) = u_2(\tilde{\mathbf{p}}) = \dots = u_n(\tilde{\mathbf{p}}) \quad (4)$$

We now want to consider a medium where $b \in [0, 1]$ determines the available bandwidth. Then a strategy $\tilde{\mathbf{p}}$ is fair if each user gets $\frac{b}{n}$ bandwidth ($u_i(\tilde{\mathbf{p}}) = b/n$). To achieve this fairness, each node i has to choose as sending probability:

$$\tilde{p}_i = \frac{b}{n - (n - i) \cdot b} \quad (5)$$

The idea of choosing these probabilities is explained next. A detailed proof why this distribution leads to fair bandwidth sharing can be found in [11]. For simplification let $b = 1$. Then, Equation 5 reduces to $\tilde{p}_i = \frac{1}{i}$. Let's assume a scenario with two players ($n = 2$) with the probability distribution $p_1 = 1$ and $p_2 = 0.5$. Consequently, player 1 always tries to access the medium and player 2 tries every second round. Because of the priorities given in Equation 2, the players get grants alternately. In general, the idea is that the higher the priority the more rare the node tries to send data. Players with lower priorities try to send more often because they are blocked by higher priorities if they want to transmit data concurrently.

2) *Selfish users*: One problem that has to be considered is that if one player tries to maximize its own payoff by changing its strategy, one or more nodes can be blocked. We call those users selfish users. For example, the player n with the highest priority can set p_n to 1. This leads to the following utility function:

$$u_i(p_n, \mathbf{p}_{-n}) = p_i \cdot \prod_{i<j<n} (1 - p_j) \cdot (1 - p_n) = 0, \quad \forall i < n$$

This means the player with highest priority blocks all other users.

3) *Enhanced Priority-based Medium Access Game*: In section III-C1, we presented with Equation 5 a method to reach fair bandwidth sharing. To apply this strategy, each user needs global information, i.e., how many players joined the game (n) and which priorities (p_i) they have. This is

very inflexible in case new users join. However, we want to develop a self-organizing multi-agent framework that can solve the problem of fair bandwidth sharing with just local information. Furthermore this system should self-organize itself if the global situation changes, for example a new player joins.

The development of simple selfish strategies is impossible because nodes with lower priorities cannot influence the strategy of a higher-prior ones. Furthermore, there is no information available for a higher-prior node that a lower-prior one tries to access the medium and gets blocked. Therefore, in [11] a new parameter ϵ is introduced, which describes the amount of bandwidth that has to stay free. The probability that the medium is free is equal to the probability that all user decides to wait:

$$P(\text{Medium is free}) = \prod_{j \in N} (1 - p_j) \quad (6)$$

The condition in the enhanced priority-based medium access game that the amount ϵ stays free means that: $P(\text{Medium is free}) > \epsilon$. If this condition does not hold, the utility for all users is set to 0. This has a direct influence to the utility function that has to be modified as follows:

$$u_i(\mathbf{p}) = \begin{cases} p_i \cdot \prod_{j>i} (1 - p_j), & \text{if } \prod_{j \in N} (1 - p_j) \geq \epsilon \\ 0, & \text{else} \end{cases} \quad (7)$$

When all users try to maximize their utility function, the system ends in a Nash equilibrium. In such a system state, no player has an intent to switch the strategy because the payoff will not increase if the others do not change their strategy. In [11], it is proven that all users $i \in N$ are in a Nash equilibrium when choosing the strategies $\tilde{\mathbf{p}}$ such that

$$\prod_{i \in N} (1 - \tilde{p}_i) = \epsilon \quad (8)$$

holds. We mentioned before that Equation 5 leads to fair bandwidth sharing in the priority-based Medium Access Games. In [11], it was shown that this fair priority distribution is a Nash equilibrium. This means that as soon as the players reach this fair strategy no user has an appeal to differ from its strategy.

IV. PENALTY LEARNING ALGORITHM

In the following section, we will present a penalty learning algorithm that is based on the enhanced priority-based medium access game. Each node i adapts its strategy p_i in certain time intervals as shown in Algorithm 1.

A. Functional Principle

A pseudocode description of the penalty learning algorithm running on each node i is given in Algorithm 1. When playing the priority-based medium access game as described in Figure 3, after *intervalLength* rounds the

Algorithm 1 Penalty learning for each player

```
1:  $load = \frac{nOverallMsgSent}{intervalLength}$ ;  
2:  $success = \frac{nSuccess}{intervalLength}$ ;  
3: if ( $load > 1 - \epsilon$ ) then  
4:    $\Delta = -success$   
5: else  
6:    $\Delta = (1 - success)$   
7: end if  
8:  $p_i^{(t+1)} = p_i^{(t)} + \eta \cdot \Delta$ ;  
9:  $limit(p_i^{(t+1)})$ 
```

sending probability p_i of each player is locally updated. To do so, first the *load* of the last rounds, from now on called interval, has to be calculated (line 1) by dividing the number of rounds where any player has send a message by the total number of rounds of the interval. Additionally, the *success* (line 2) is the number of rounds player i has successfully transmitted divided by the total number of rounds of the interval. In line 3 to 6, depending on the load of the interval, the sending probability is lowered (*load* is greater $1 - \epsilon$), or increased (*load* is less or equal to $1 - \epsilon$). The amount by which the sending probability is changed depends on the success of the player. The more success the player had, the more it will decrease its sending probability in case of penalty and the less it will increase the sending probability in case of no penalty. This basic mechanism makes it possible that a fair bandwidth distribution will be established. The change Δ of the sending probability is multiplied by a learning rate $\eta < 1$. Finally, the sending probability is limited (line 9) to be between 0 and 1 to ensure it is within reasonable limits.

To explain the functional principle of this simple distributed learning algorithm, we will show the behavior of the algorithm on a two player example. Using the graphical normal form representation of the game, we can explain why the algorithm will converge to a fair bandwidth distribution.

B. Two Player Example

The enhanced priority-based access game cannot be displayed with a two action normal form, because the reward calculated in a single round is either one or zero, but a more fine grained bandwidth is needed to use a reasonable ϵ . In consequence, it only makes sense to play a mixed strategy. However, if the game is played for several rounds the free bandwidth can be calculated by dividing the number of free slots by the number of rounds played. If this number of played rounds reaches infinity, then the free bandwidth can be calculated as described in Section III-C.

In Figure 5, the extended normal form representation of the enhanced two player priority-based access game with $\epsilon = 0.3$ is shown. On the left column, the strategy of player 1 is displayed and in the top row, the actions of player 2. The

strategy in this case is the probability p_i that player i is trying to send during each round. The probability p_i of course is a real number between 0 and 1, but for demonstration purpose we display only 11 possible strategies. The remaining entries show the amount of bandwidth (*success*) the players get when playing the strategy, the first number for player 1 and the second for player 2. The green (semi-dark) and yellow (light) entries in the upper left part denote the strategies where the amount of free bandwidth is below ϵ and therefore the reward u_i is equal to the *success*. The red (dark) entries denote the strategies where the ϵ -rule is violated and the reward is zero. In this table, the *success* is illustrated, because it is used by the learning algorithm. The yellow (light) entries show Nash equilibria, where a standard game learning algorithm would converge to. This example again shows that many unfair Nash equilibria do co-exist, which we try to avoid.

In the following, we assume $\eta = 0.2$. A discussion about η is presented in Section V. The thin small arrows show the change of action, i.e. probability, that is done according to the penalty algorithm given in Algorithm 1. The large arrows try to show the tendency of all small arrows. The tendency toward the Nash equilibria front, represented by the top left and bottom right arrow, can be read easily. However, to observe the tendency towards the fair solution on the front a closer look needs to be taken. The tendency from bottom left to top right can be read when looking at the arrows on the lower half of the Nash equilibria front. For example the resulting direction from arrow 1 and 2 point towards the fair solution and so do the other combined arrows in this region. When looking at the top half of the Nash equilibria front, the same conclusions can be drawn as for the lower half. As an example the resulting direction from arrow 3 and 4 point towards the fair solution.

Once reaching the fair solution, the strategies stay there. This is clarified by looking at arrow 5 and 6 close to the fair solution. They are parallel and point in opposite direction. So once reaching this point, the strategies are oscillating around this point with a maximum length of η .

V. SIMULATION RESULTS

Our simulation has five degrees of freedom: starting probability of each player p_i , free bandwidth ϵ , number of players n , *intervalLength* and learning rate η . The starting probability of each player is always set to $1 - \epsilon$, because experiments showed that it doesn't influence the converged state. These experiments are just not shown here because of space limitations. ϵ needs to be large enough as described in Section V-C. Choosing

$$\epsilon = \frac{1}{n+1} \quad (9)$$

showed to be large enough and will be used throughout this section. As this is a multi-agent game, the most simple

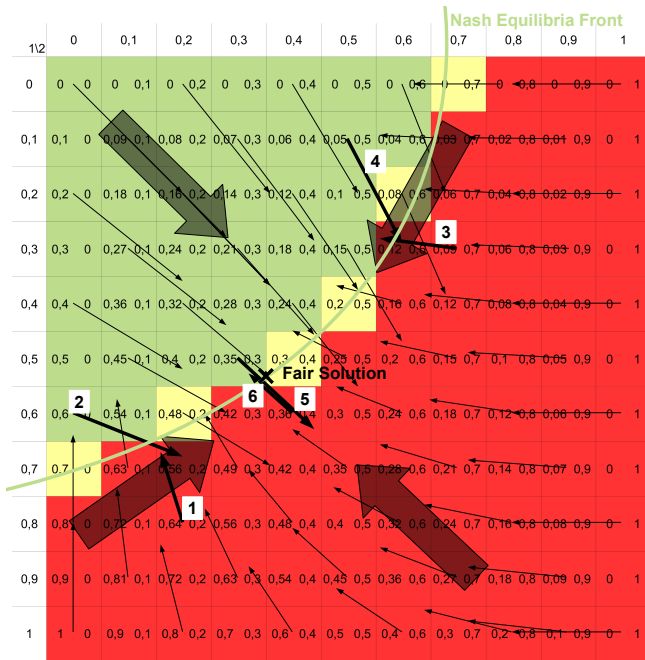


Figure 5: Discrete normal form representation of the two player enhanced priority-based access game with $\epsilon = 0.3$

example is to have two players. In the following, we assume that the lower the player id the higher is the priority. The *intervalLength* is first chosen quite large to ensure the law of large numbers is valid and the reward is very close to our theoretical calculations. The learning rate η will be analyzed in the next section.

During each simulation time step, the bandwidth is displayed to show the convergence and the stability of our algorithm. The bandwidth is similar to the variable *success* in Algorithm 1. A simulation time step equals one played round and therefore, the terms will be used interchangeably. The faster a fair solution is reached, the better the convergence. As defined in Equation 9, $\epsilon = \frac{1}{3}$, so the fair solution in the following experiments means both players have a bandwidth of $\frac{1}{3}$. The stability of our algorithm is defined as the deviation from the fair solution when convergence has been reached.

A. Analysis of learning rate η

Figure 6 shows the simulation results for *intervalLength* = 100000 and $\eta = 0.2$. It can be seen that the algorithm converges in 3 learning steps, i.e. 300,000 time steps, but then oscillates with an inaccuracy of 0.2. Interestingly, as Figure 7a shows, if $\eta = 0.02$, then it converges mainly in 30 learning steps as expected, but then the convergence slows down and the optimum is reached after 140 learning steps. In Figure 7b, the mean and the variance of the bandwidth are shown. Here, we can see that in the first 30 learning rounds, the mean is

changing to the desired $\frac{1}{3}$. After 140 learning steps, the variance is not decreasing anymore. At this converged state the inaccuracy is around 0.02. The results show that the learning process has two stages: 1. Reaching the Nash equilibria front and 2. Moving along the Nash equilibria front towards a fair solution. The time needed to reach the Nash equilibria front is inversely proportional to η , as shown in Figure 8. Moving along the Nash equilibria front then needs around double the time than reaching the front, but a formula could not be found. When $\eta > 0.05$, moving on the Nash equilibria front is not done because then, the step size leads to an overleaping of the Nash equilibria front. In addition, the inaccuracy during the converged state is directly proportional to η , as shown in Figure 8. In conclusion, the first experiments show that the algorithm converges and that there is a tradeoff between the convergence speed and the accuracy.

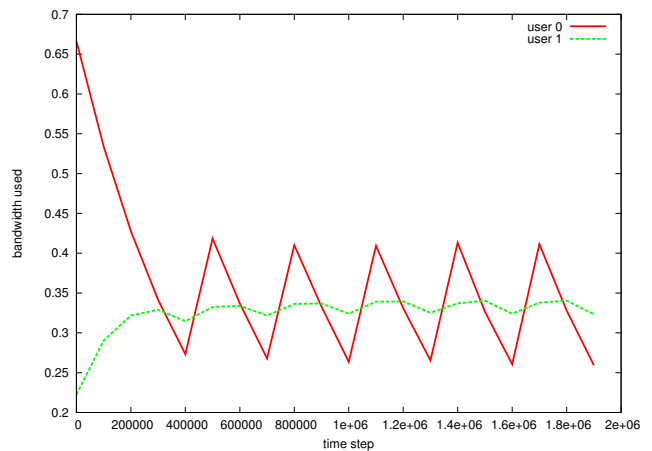


Figure 6: Simulation results of the penalty learning algorithm for $\eta = 0.2$

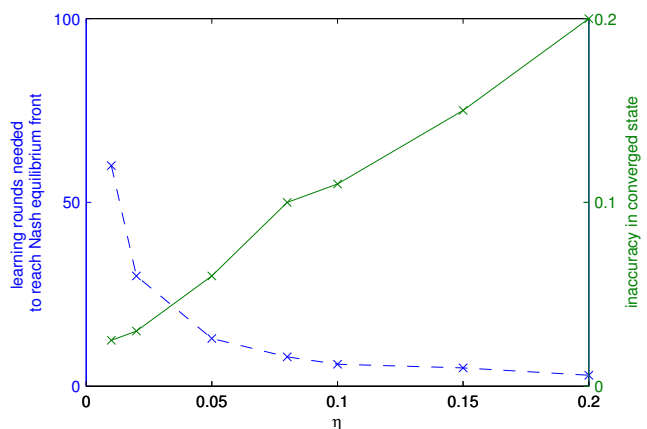


Figure 8: Convergence time and accuracy of the penalty learning algorithm for different η

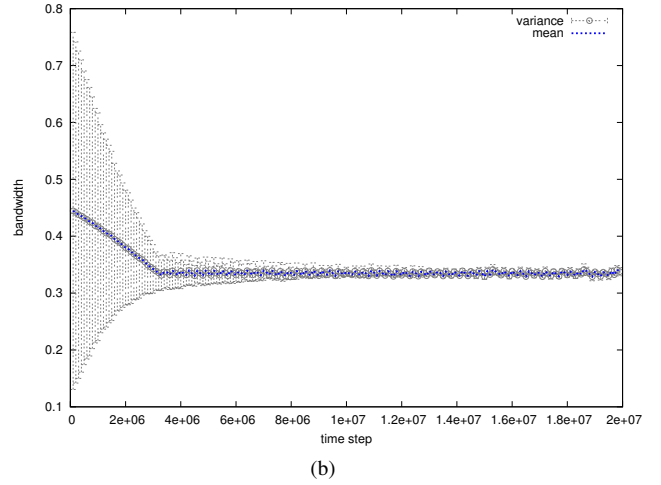
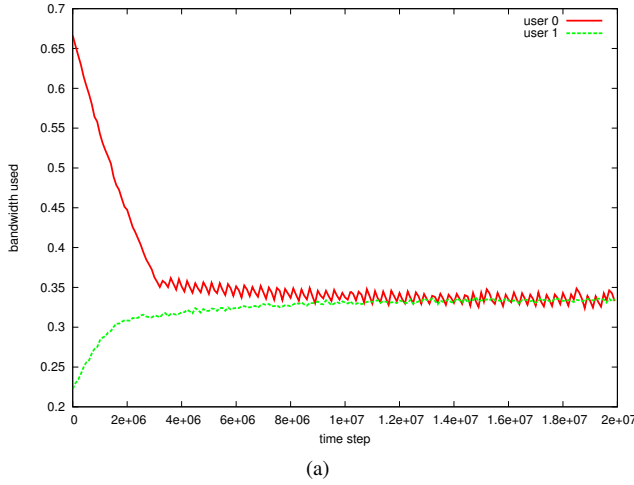


Figure 7: Simulation results of the penalty learning algorithm for $\eta = 0.02$

B. Analysis of the number of players n

In the following, we want analyze the behavior when the number of players is increased. Figure 9 shows the simulation results of the priority-based medium access game for $n = 20$ players. Compared to the two player scenario in Figure 7, the time to converge increases. This is in the nature of the priority-based access mechanism, because the high-priority messages get full access and need to learn by penalty that other low priority messages exist. However, after reaching the converged state, the inaccuracy is identical to the two player scenario. This shows the scalability of our approach.

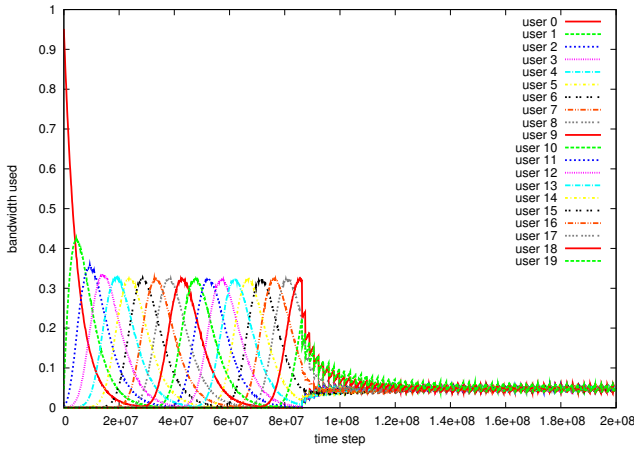


Figure 9: Simulation results of the penalty learning algorithm for 20 players and $\eta = 0.02$

C. Analysis of $intervalLength$ and ϵ

First, we will look at the influence of the $intervalLength$ on the convergence speed and the stability. Again, we will

compare the following results to the two player scenario in Figure 7 and therefore use $\eta = 0.02$. On the previous plots, we always plotted the bandwidth of the players by dividing the number of successful messages during one learning interval by the $intervalLength$. If we decrease now the $intervalLength$, the influence of the actions chosen can differ a lot from the resulting bandwidth distribution. This effect can be seen, when two players are behaving fair from start with $intervalLength = 100$. Two fair players on the priority-based access game have $p_0 = \frac{1}{3}$ and $p_1 = \frac{1}{2}$. The simulation results are shown in Figure 10. However, for many applications, this short-term variations can be neglected and only the average over several messages is interesting.

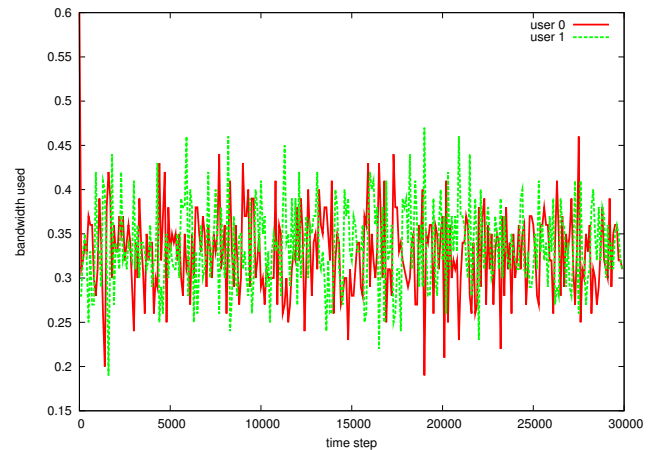


Figure 10: Simulation results for two fair players from start with $intervalLength = 100$

Apart from the difference between action and result, two aspects for lowering the $intervalLength$ have to be

considered: 1. the amount of computation and 2. the ϵ -behavior.

Lowering the *intervalLength* means that learning is done more often. This in consequence requires more computation time. So, when looking at the following results we always have to keep this in mind.

The ϵ -behavior denotes the ability to keep exactly ϵ bandwidth free. Our penalty algorithm decides whether to lower or to raise the sending probability on the *load* during the last interval. Therefore, it is essential that the *load* can take enough values between 0 and 1. For example, if we set *intervalLength* = 1, the *load* would only take the values 0 or 1. So, no matter what ϵ is set to, the algorithm will raise the probability if no one sends and lower it if anyone sends. The results with a plotting interval of 100 are shown in Figure 11. Amazingly, the system converges very fast to a state where the bandwidth of each player is $\frac{1}{n+1}$. This behavior can be observed for any number of players. We are still working on finding out the reasons for this behavior. Still, it has to be noted that our goal is to use an ϵ as small as possible, which is not possible if the granularity of the load doesn't represent it.

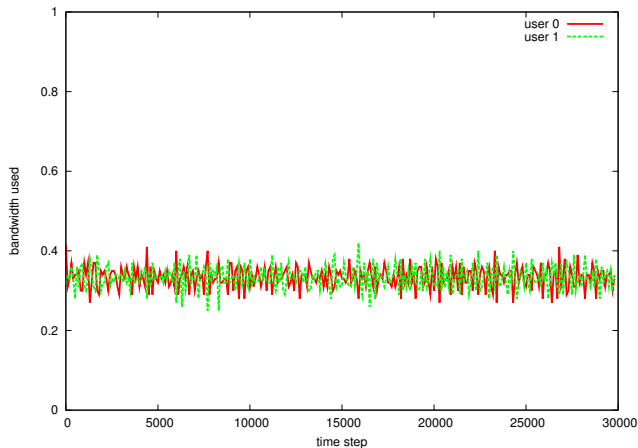


Figure 11: Simulation results for *intervalLength* = 1 and plotting interval of 100 time steps

Figure 12 shows the simulation results for the penalty learning algorithm for *intervalLength* = 100. Even though the variance is much higher than in Figure 7a, we can see the first stage of the learning process takes also about 30 learning steps, i.e. 3000 time steps. The second stage can not be identified, because of the large variation of the bandwidth. As the *intervalLength* is much shorter, the total number of time steps to converge is reduced tremendously. The inaccuracy in the converged state is also improved, because the period of the oscillation around the fair solution decreases. This is shown in Figure 13, where we used the same plotting interval length as in Figure 7.

In conclusion, as long as the ϵ -behavior is still given, we

can tradeoff computation time with convergence speed and accuracy.

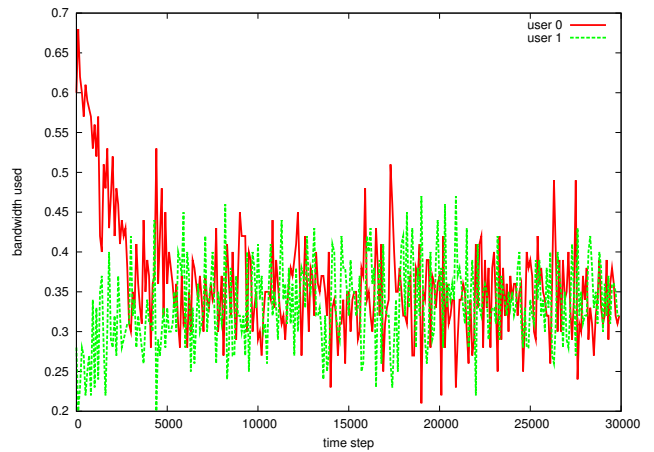


Figure 12: Simulation results of the penalty learning algorithm for *intervalLength* = 100

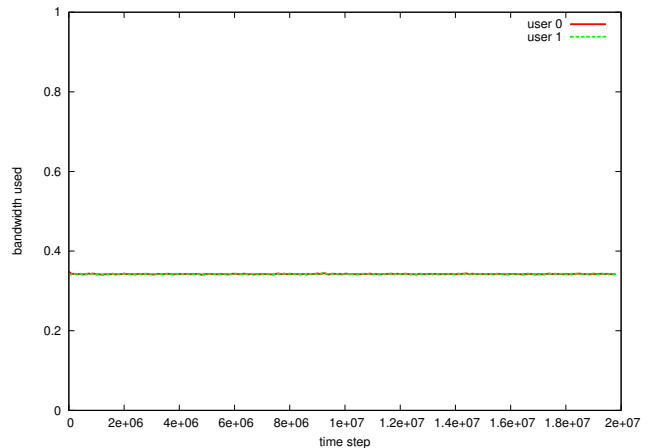


Figure 13: Simulation results of the penalty learning algorithm for *intervalLength* = 100 and plotting interval of 100000 time steps

In the following, we will analyze the influence of the size of ϵ . On the one hand, it is desirable to use as much available bandwidth as possible, therefore we try to reduce ϵ to a minimum. On the other hand, a certain amount of ϵ needs to be kept free such that the players with the lower priority can penalize the ones with higher priority. In theory, it would be sufficient to force one round to be free per learning interval. In practice, the access mechanism produces some variations, and therefore, ϵ needs to be large enough to compensate those, otherwise the time to converge is increasing as seen in Figure 14, which has the same setup as Figure 7 except for $\epsilon = 0.1$.

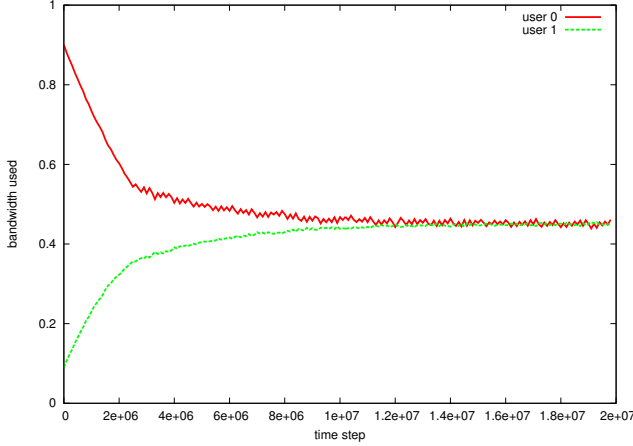


Figure 14: Simulation results of the penalty learning algorithm for $intervalLength = 100000$ and $\epsilon = 0.1$

VI. EXTENDED PENALTY LEARNING ALGORITHM

A. Adapting the Learning Rate

As seen in the results from Section V, it would be desirable to have a high learning rate at the beginning, and when the system has converged, to decrease the learning rate to better stabilize the system. We picked up an idea from Q-learning [10], to enable this property. In Q-learning, an expected reward R at each time step is calculated:

$$R^{t+1} = success^t \cdot \gamma + R^t \cdot (1 - \gamma)$$

Here, $success^t$ is the *success* calculated in Algorithm 1 at time step t . γ is called the discount factor which decides how much the expected reward is influenced by the current reward. Experiments showed that $\gamma = 0.5$ gives good results. We can now calculate a measure of convergence θ :

$$\theta = R^t - success^t$$

The lower θ , the more stable the system is. We can include that in our penalty learning algorithm as described in Algorithm 1 by replacing η in line 8 by θ :

$$\eta = \theta$$

The result of the extended penalty learning algorithm with the same setup as in Figure 7 is shown in Figure 15. Here, the time to converge is much lower, but it doesn't converge to the fair solution. To prevent the system from getting stuck in an unfair solution, we finally introduce a minimum learning rate η_{min} . If η is below η_{min} we use η_{min} . This, of course, creates inaccuracy in the converged state. However, our extended penalty learning algorithm provides the same inaccuracy in the converged state as the previous penalty learning algorithm, but reduces the time to converge. The results are shown in Figure 16.

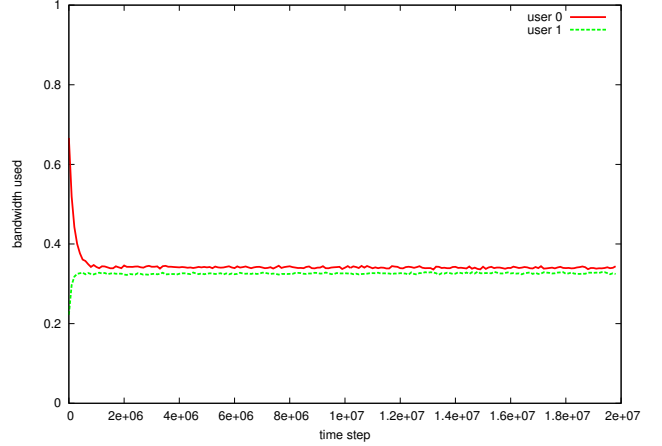


Figure 15: Simulation results of the extended penalty learning algorithm for $intervalLength = 100000$

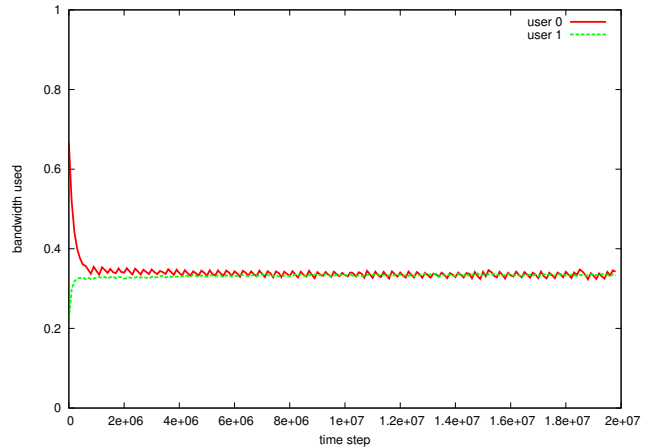


Figure 16: Simulation results of the extended penalty learning algorithm for $intervalLength = 100000$ with $\eta_{min} = 0.02$

B. Period Access Scheme

In this section, we propose to use a sending period instead of a sending probability to control the access to the medium. Even though this access scheme is very difficult to describe formally, and we are not able to provide a prove of Nash equilibria, the period access scheme gives better results in terms of time to converge and inaccuracy in the converged state. Using the period access scheme, the actions each player has is the length of the period. The player is trying to send at every period slot. The main difference to the probability access scheme is, that if the player is not granted access when trying to send, he remembers this message and tries to send it in the next round. This could of course lead to a buffer overflow. To prevent this, these stored messages are cleared after one learning step. The penalty learning algorithm described in Algorithm 1 needs to be adapted

slightly. Instead of producing a negative Δ in line 4, the minus has to be removed. In line 6 the minus has to be added. This will cause the period to increase when the load is above $1 - \epsilon$ and therefore decrease the bandwidth used by all players. The main positive aspect is that the period access scheme is deterministic, which means, each learning interval played with the same periods will return the same reward to all players.

The simulation results for $intervalLength = 100$ are shown in Figure 17. The setup is chosen similar to those of Figure 12 to demonstrate the improvement.

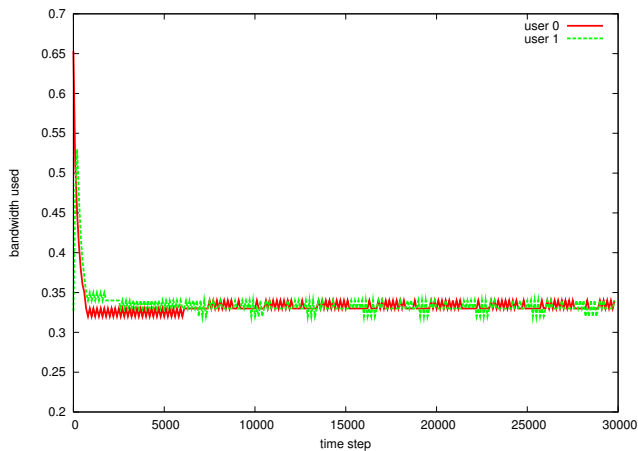


Figure 17: Simulation results of the extended penalty learning algorithm with period access scheme for $intervalLength = 100$

VII. CONCLUSION

In this paper, we presented a penalty learning algorithm, which establishes a fair bandwidth distribution on priority-based buses. The algorithm has been tested experimentally by simulation. We could show that the algorithm converges in any case. Additionally, we could improve the algorithm to reduce the time to convergence significantly while keeping the inaccuracy in the converged state at the same level.

At the moment, we are trying to extend the algorithm to support weighted bandwidth sharing. First experiments have been made which show some way of weighting is possible, but we are not yet able to control the distribution perfectly. In addition, a proof for the convergence of the penalty algorithm is at work.

For future work, we are planning to go in several directions: On the one hand, we try to extend the current approach to also support other constraints, such as deadlines. On the other hand, the currently developed algorithms need to be tested on more realistic simulations. We even think about implementing the algorithms on reconfigurable hardware to look at area and time overheads of the presented self-organizing mechanism.

ACKNOWLEDGMENT

Supported in part by the German Science Foundation (DFG) under contract TE 163/15-1.

REFERENCES

- [1] L. Busoniu, R. Babuska, and B. De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(2):156–172, March 2008.
- [2] CAN. Controller Area Network. <http://www.can.bosch.com/>.
- [3] D. Fudenberg and D.K. Levine. *The theory of learning in games*. The MIT press, 1998.
- [4] M. Lauer and M. Riedmiller. An algorithm for distributed reinforcement learning in cooperative multi-agent systems. In *In Proc. 17th International Conf. on Machine Learning*, 2000.
- [5] M.L. Littman. Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.
- [6] Sudipta Rakshit and Ratan K. Guha. Fair Bandwidth Sharing in Distributed Systems: A Game-Theoretic Approach. *IEEE Transactions on Computers*, 54(11):1384–1393, 2005.
- [7] H. Schmeck. Organic computing—a new vision for distributed embedded systems. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005*, pages 201–203, 2005.
- [8] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok. Real time scheduling theory: A historical perspective. *Real-Time Syst.*, 28(2-3):101–155, 2004.
- [9] Ken Tindell, Hans Hansson, and Andy Wellings. Analysing Real-Time Communications: Controller Area Network (CAN). In *Proc. 15th IEEE Real-Time Systems Symposium*, San Juan, Puerto Rico, December 1995. IEEE Society Press.
- [10] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.
- [11] Stefan Wildermann, Tobias Ziermann, and Jürgen Teich. Self-organizing bandwidth sharing in priority-based medium access. In *Proceedings of the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 09)*, pages 144–153, San Francisco, USA, 2009.